causalml Documentation

Someone at Uber

Apr 25, 2024

CONTENTS

Contents:

CHAPTER

ONE

ABOUT CAUSALML

`CausalML` 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), also known as **Individual Treatment Effect** (ITE), from experimental or observational data. Essentially, it estimates the causal impact of intervention **W** on outcome **Y** for users with observed features **X**, without strong assumptions on the model form.

1.1 GitHub Repo

https://github.com/uber/causalml

1.2 Mission

From the CausalML Charter:

> CausalML is committed to democratizing causal machine learning through accessible, innovative, and well-documented open-source tools that empower data scientists, researchers, and organizations. At our core, we embrace inclusivity and foster a vibrant community where members exchange ideas, share knowledge, and collaboratively shape a future where CausalML drives advancements across diverse domains.

1.3 Contributing

Contributing.md

1.4 Governance

- Charter
- Contributors
- Maintainers

CHAPTER

TWO

INTRO TO CAUSAL MACHINE LEARNING

2.1 What is Causal Machine Learning?

Causal machine learning is a branch of machine learning that focuses on understanding the cause and effect relationships in data. It goes beyond just predicting outcomes based on patterns in the data, and tries to understand how changing one variable can affect an outcome. Suppose we are trying to predict a student's test score based on how many hours they study and how much sleep they get. Traditional machine learning models would find patterns in the data, like students who study more or sleep more tend to get higher scores. But what if you want to know what would happen if a student studied an extra hour each day? Or slept an extra hour each night? Modeling these potential outcomes or counterfactuals is where causal machine learning comes in. It tries to understand cause-and-effect relationships - how much changing one variable (like study hours or sleep hours) will affect the outcome (the test score). This is useful in many fields, including economics, healthcare, and policy making, where understanding the impact of interventions is crucial. While traditional machine learning is great for prediction, causal machine learning helps us understand the difference in outcomes due to interventions.

2.2 Difference from Traditional Machine Learning

Traditional machine learning and causal machine learning are both powerful tools, but they serve different purposes and answer different types of questions. Traditional Machine Learning is primarily concerned with prediction. Given a set of input features, it learns a function from the data that can predict an outcome. It's great at finding patterns and correlations in large datasets, but it doesn't tell us about the cause-and-effect relationships between variables. It answers questions like "Given a patient's symptoms, what disease are they likely to have?" On the other hand, Causal Machine Learning is concerned with understanding the cause-and-effect relationships between variables. It goes beyond prediction and tries to answer questions about intervention: "What will happen if we change this variable?" For example, in a medical context, it could help answer questions like "What will happen if a patient takes this medication?" In essence, while traditional machine learning can tell us "what is", causal machine learning can help us understand "what if". This makes causal machine learning particularly useful in fields where we need to make decisions based on data, such as policy making, economics, and healthcare.

2.3 Measuring Causal Effects

Randomized Control Trials (RCT) are the gold standard for causal effect measurements. Subjects are randomly exposed to a treatment and the Average Treatment Effect (ATE) is measured as the difference between the mean effects in the treatment and control groups. Random assignment removes the effect of any confounders on the treatment.

If an RCT is available and the treatment effects are heterogeneous across covariates, measuring the conditional average treatment effect(CATE) can be of interest. The CATE is an estimate of the treatment effect conditioned on all available experiment covariates and confounders. We call these Heterogeneous Treatment Effects (HTEs).

2.4 Example Use Cases

- **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**: A company might have multiple options to interact with its customers such as different product choices in up-sell or different messaging channels for communications. One can use CATE to estimate the heterogeneous treatment effect for each customer and treatment option combination for an optimal personalized engagement experience.

CHAPTER

THREE

INSTALLATION

Installation with `conda` is recommended.

`conda` environment files for Python 3.7, 3.8 and 3.9 are available in the repository. To use models under the `inference.tf` module (e.g. `DragonNet`), additional dependency of `tensorflow` is required. For detailed instructions, see below.

3.1 Install using `conda`

3.1.1 Install `conda`

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b
source miniconda3/bin/activate
conda init
source ~/.bashrc
```

3.1.2 Install from `conda-forge`

Directly install from the `conda-forge` channel using `conda`.

```
conda install -c conda-forge causalml
```

3.1.3 Install from the `conda` virtual environment

This will create a new `conda` virtual environment named `causalml-[tf-]py3x`, where `x` is in `[7, 8, 9]`. e.g. `causalml-py37` or `causalml-tf-py38`. If you want to change the name of the environment, update the relevant YAML file in `envs/`.

```
git clone https://github.com/uber/causalml.git
cd causalml/envs/
conda env create -f environment-py38.yml    # for the virtual environment with Python
↪3.8 and CausalML
conda activate causalml-py38
```

3.1.4 Install `causalml` with `tensorflow`

```
git clone https://github.com/uber/causalml.git
cd causalml/envs/
conda env create -f environment-tf-py38.yml # for the virtual environment with Python
↪3.8 and CausalML
conda activate causalml-tf-py38
pip install -U numpy                        # this step is necessary to fix [
↪#338](https://github.com/uber/causalml/issues/338)
```

3.2 Install from `PyPI`

```
pip install causalml
```

3.2.1 Install `causalml` with `tensorflow` from `PyPI`

```
pip install causalml[tf]
pip install -U numpy                            # this step is necessary to fix [
↪#338](https://github.com/uber/causalml/issues/338)
```

3.3 Install from source

Create a clean `conda` environment.

```
conda create -n causalml-py38 -y python=3.8
conda activate causalml-py38
conda install -c conda-forge cxx-compiler
conda install python-graphviz
conda install -c conda-forge xorg-libxrender
```

Then:

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

with `tensorflow`:

```
pip install .[tf]
```

3.4 Windows

See content in https://github.com/uber/causalml/issues/678

3.5 Running Tests

Make sure pytest is installed before attempting to run tests.

Run all tests with:

```
pytest -vs tests/ --cov causalml/
```

Add `--runtf` to run optional tensorflow tests which will be skipped by default.

You can also run tests via make:

```
make test
```

CHAPTER

FOUR

API QUICKSTART

Working example notebooks are available in the example folder.

4.1 Propensity Score

4.1.1 Propensity Score Estimation

```
from causalml.propensity import ElasticNetPropensityModel

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

4.1.2 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_cols=score_cols,
                             groupby_col=groupby_col)

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

4.2 Average Treatment Effect (ATE) Estimation

4.2.1 Meta-learners and Uplift Trees

In addition to the Methodology section, you can find examples in the links below for *Meta-Learner Algorithms* and *Tree-Based Algorithms*

- Meta-learners (S/T/X/R): meta_learners_with_synthetic_data.ipynb
- Meta-learners (S/T/X/R) with multiple treatment: meta_learners_with_synthetic_data_multiple_treatment.ipynb

- Comparing meta-learners across simulation setups: benchmark_simulation_studies.ipynb
- Doubly Robust (DR) learner: dr_learner_with_synthetic_data.ipynb
- TMLE learner: validation_with_tmle.ipynb
- Uplift Trees: uplift_trees_with_synthetic_data.ipynb

```
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, treatment, y, e)
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]))
```

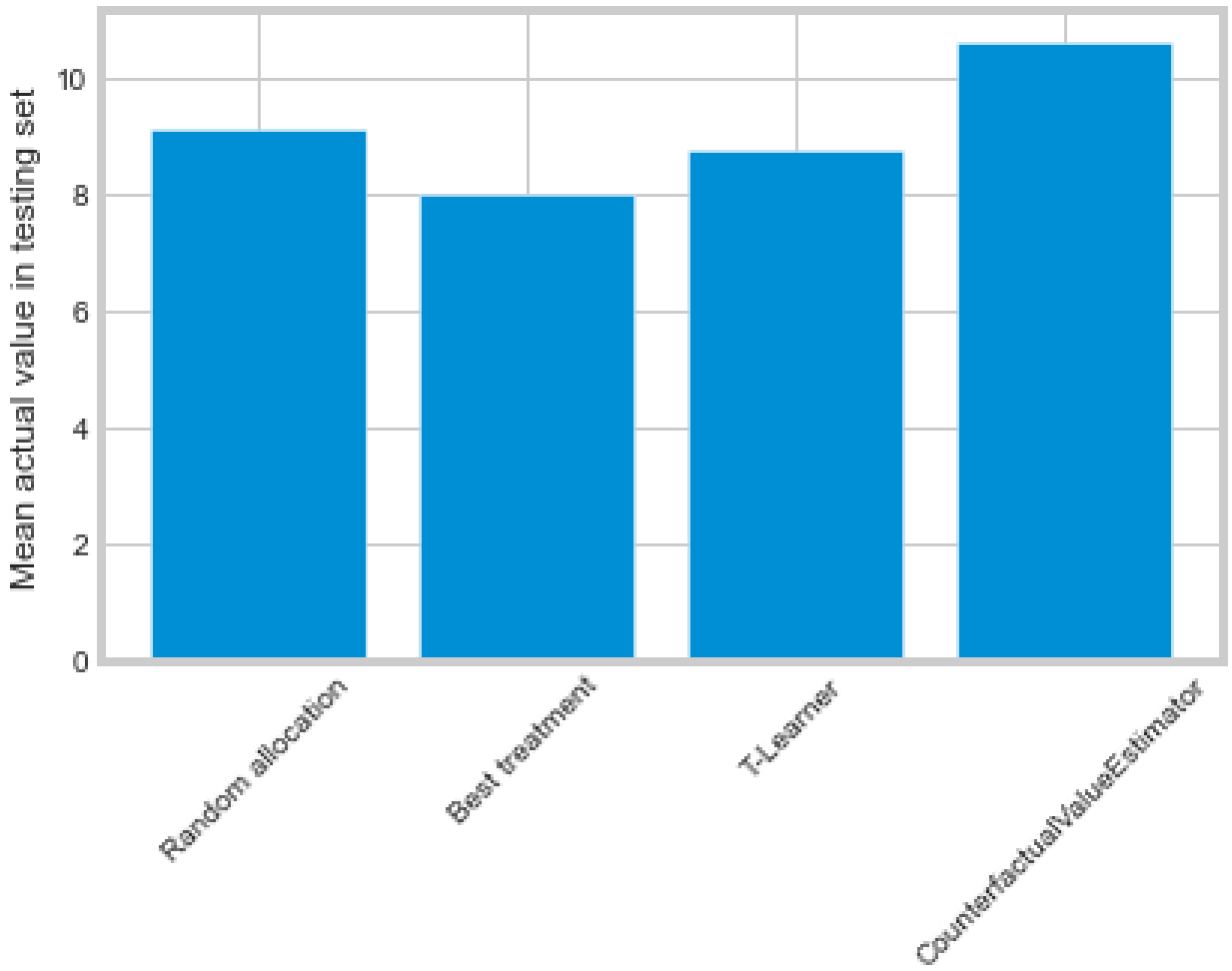
4.3 More algorithms

4.3.1 Treatment optimization algorithms

We have developed *Counterfactual Unit Selection* and *Counterfactual Value Estimator* methods, please find the code snippet below and details in the following notebooks:

- counterfactual_unit_selection.ipynb
- counterfactual_value_optimization.ipynb

```
from causalml.optimize import CounterfactualValueEstimator
from causalml.optimize import get_treatment_costs, get_actual_value

# load data set and train test split
df_train, df_test = train_test_split(df)
train_idx = df_train.index
test_idx = df_test.index
# some more code here to initiate and train the Model, and produce tm_pred
# please refer to the counterfactual_value_optimization notebook for complete example

# run the counterfactual calculation with TwoModel prediction
cve = CounterfactualValueEstimator(treatment=df_test['treatment_group_key'],
                                   control_name='control',
                                   treatment_names=conditions[1:],
                                   y_proba=y_proba,
                                   cate=tm_pred,
                                   value=conversion_value_array[test_idx],
                                   conversion_cost=cc_array[test_idx],
                                   impression_cost=ic_array[test_idx])

cve_best_idx = cve.predict_best()
cve_best = [conditions[idx] for idx in cve_best_idx]
actual_is_cve_best = df.loc[test_idx, 'treatment_group_key'] == cve_best
cve_value = actual_value.loc[test_idx][actual_is_cve_best].mean()

labels = [
    'Random allocation',
    'Best treatment',
    'T-Learner',
    'CounterfactualValueEstimator'
]
values  = [
    random_allocation_value,
    best_ate_value,
    tm_value,
    cve_value
]
# plot the result
plt.bar(labels, values)
```



4.3.2 Instrumental variables algorithms

- 2-Stage Least Squares (2SLS): iv_nlsym_synthetic_data.ipynb

4.3.3 Neural network based algorithms

- CEVAE: cevae_example.ipynb
- DragonNet: dragonnet_example.ipynb

4.4 Interpretation

Please see *Interpretable Causal ML* section

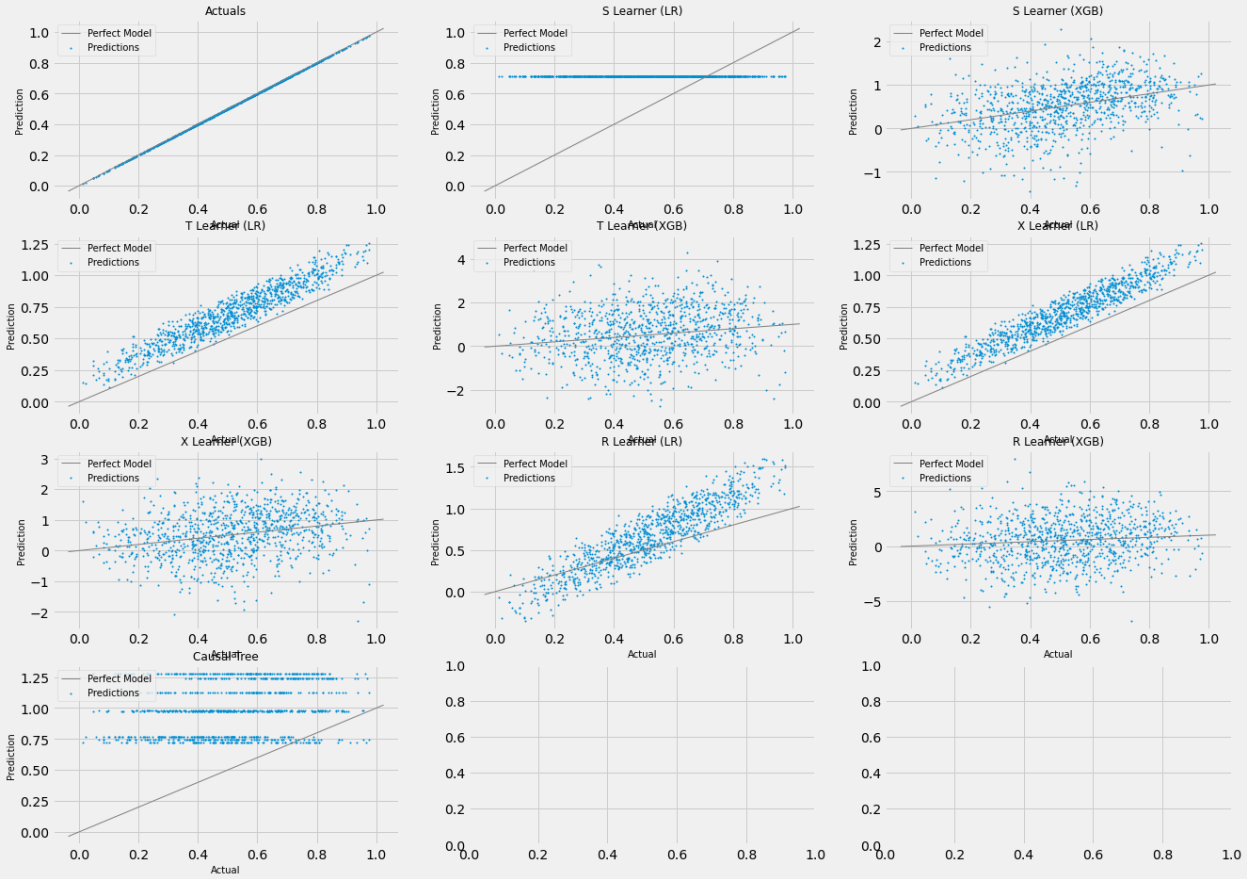
4.5 Validation

Please see *Validation* section

4.6 Synthetic Data Generation Process

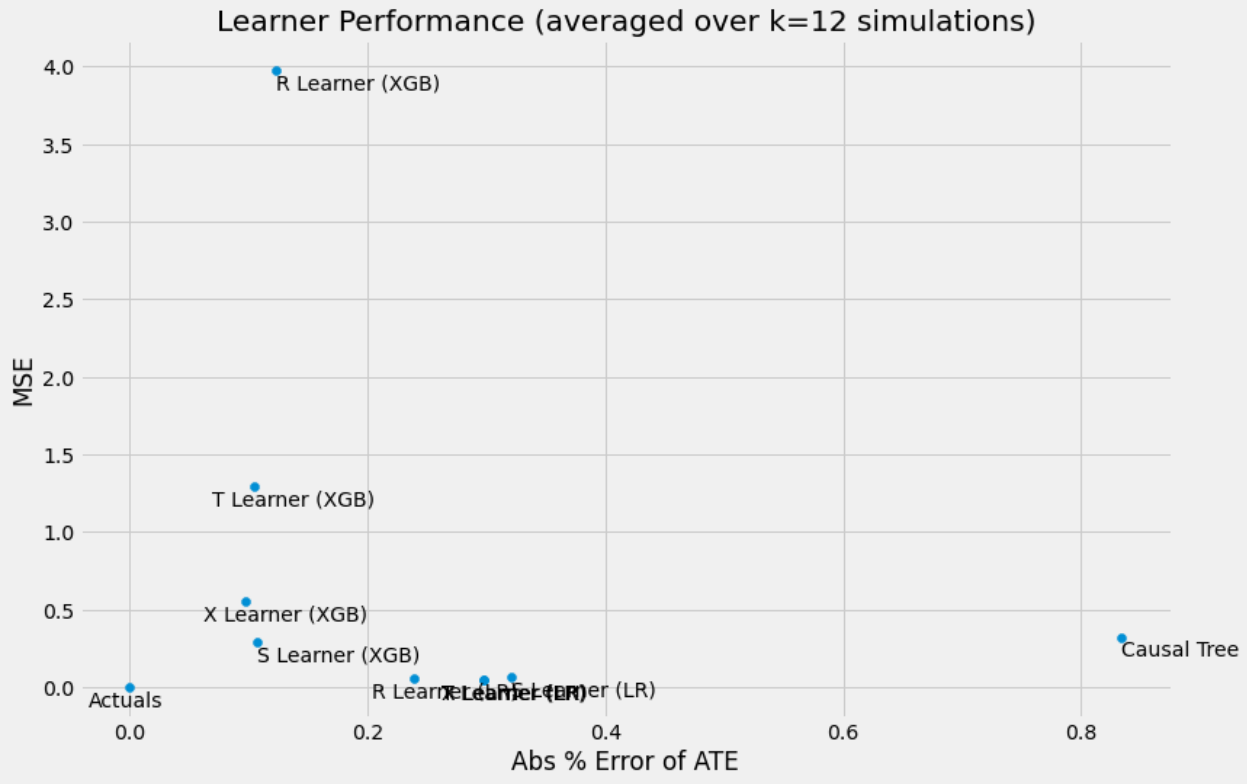
4.6.1 Single Simulation

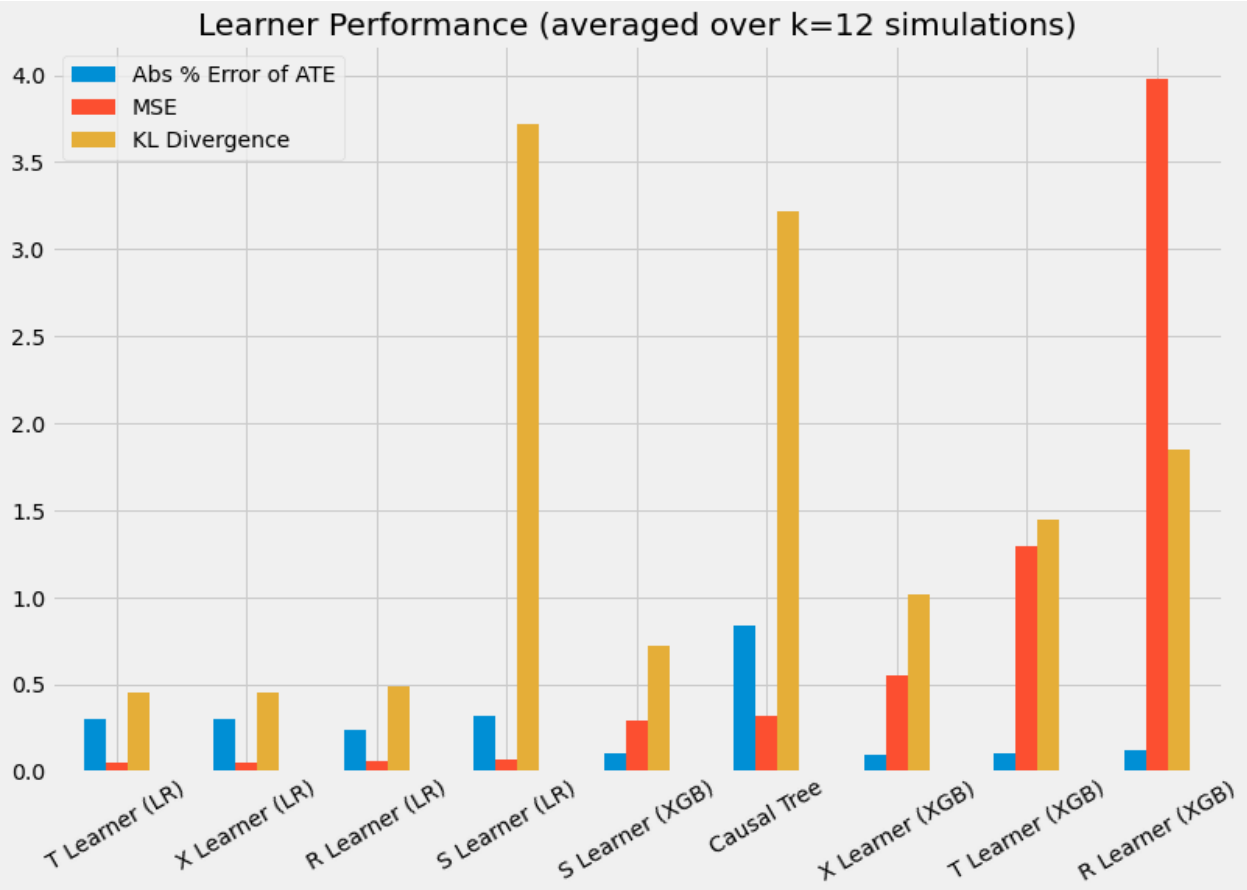
```
from causalml.dataset import *

# 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
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')
```



4.6.2 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_summary, k=num_simulations)

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

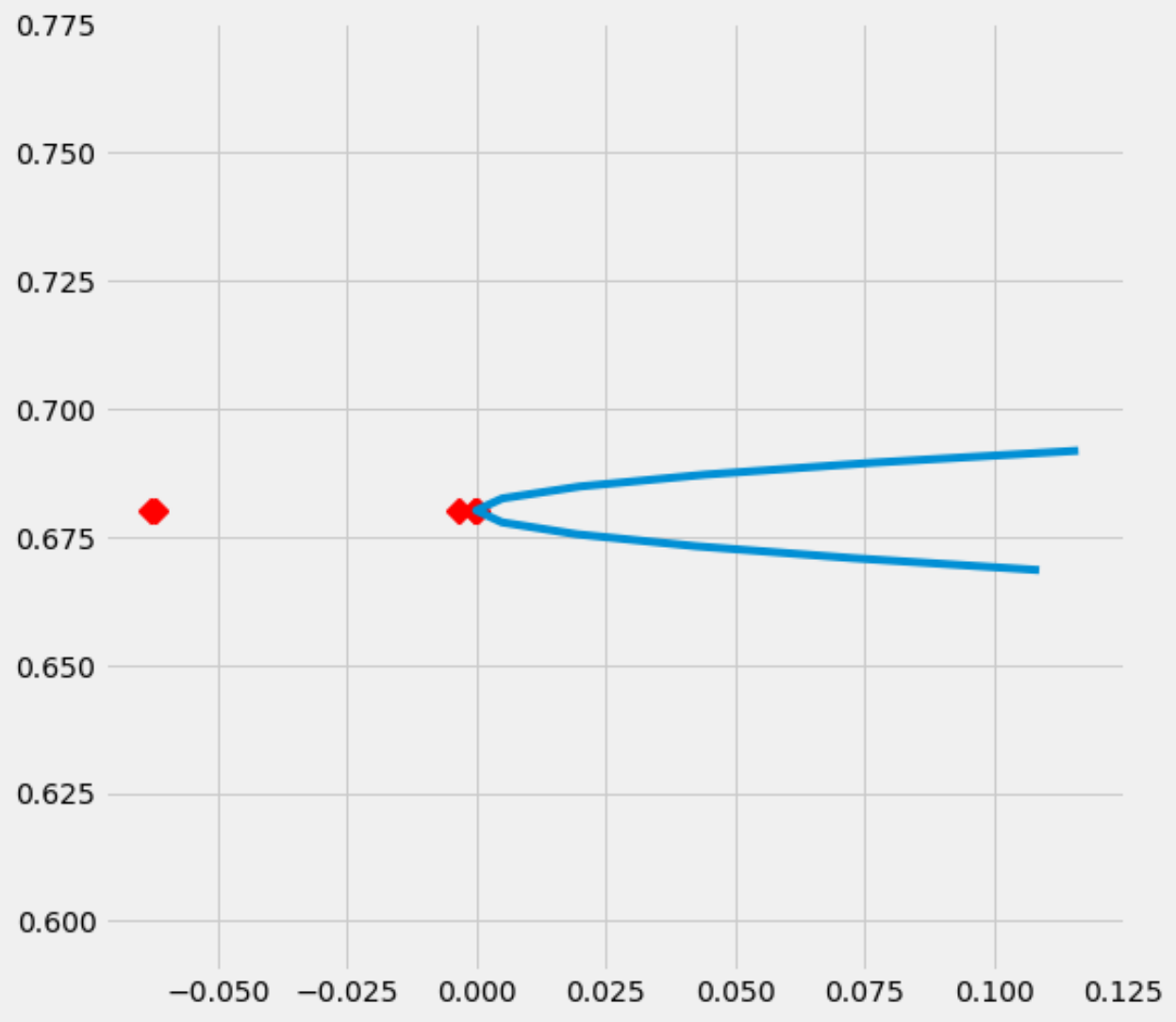

Learner Performance (averaged over k=12 simulations)
MSE
Abs % Error of ATE
R Learner (XGB)
T Learner (XGB)
X Learner (XGB)
S Learner (XGB)
Causal Tree
Actuals
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8


Learner Performance (averaged over k=12 simulations)
Abs % Error of ATE
MSE
KL Divergence
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
T Learner (LR)
X Learner (LR)
R Learner (LR)
S Learner (LR)
S Learner (XGB)
Causal Tree
X Learner (XGB)
T Learner (XGB)
R Learner (XGB)

4.7 Sensitivity Analysis

For more details, please refer to the sensitivity_example_with_synthetic_data.ipynb notebook.

```
from causalml.metrics.sensitivity import Sensitivity
from causalml.metrics.sensitivity import SensitivitySelectionBias
from causalml.inference.meta import BaseXLearner
from sklearn.linear_model import LinearRegression

# Calling the Base XLearner class and return the sensitivity analysis summary report
learner_x = BaseXLearner(LinearRegression())
sens_x = Sensitivity(df=df, inference_features=INFERENCE_FEATURES, p_col='pihat',
                     treatment_col=TREATMENT_COL, outcome_col=OUTCOME_COL,␣
↪learner=learner_x)
# Here for Selection Bias method will use default one-sided confounding function and␣
↪alpha (quantile range of outcome values) input
sens_sumary_x = sens_x.sensitivity_analysis(methods=['Placebo Treatment',
                                                     'Random Cause',
                                                     'Subset Data',
                                                     'Random Replace',
                                                     'Selection Bias'], sample_size=0.
↪5)

# Selection Bias: Alignment confounding Function
sens_x_bias_alignment = SensitivitySelectionBias(df, INFERENCE_FEATURES, p_col='pihat
↪', treatment_col=TREATMENT_COL,
                                                 outcome_col=OUTCOME_COL, learner=learner_
↪x, confound='alignment',
                                                 alpha_range=None)
# Plot the results by rsquare with partial r-square results by each individual␣
↪features
sens_x_bias_alignment.plot(lls_x_bias_alignment, partial_rsqs_x_bias_alignment, type=
↪'r.squared', partial_rsqs=True)
```



4.8 Feature Selection

For more details, please refer to the feature_selection.ipynb notebook and the associated paper reference by Zhao, Zhenyu, et al.

```
from causalml.feature_selection.filters import FilterSelect
from causalml.dataset import make_uplift_classification

# define parameters for simulation
y_name = 'conversion'
treatment_group_keys = ['control', 'treatment1']
n = 100000
n_classification_features = 50
n_classification_informative = 10
n_classification_repeated = 0
n_uplift_increase_dict = {'treatment1': 8}
n_uplift_decrease_dict = {'treatment1': 4}
```

(continues on next page)

(continued from previous page)

```
delta_uplift_increase_dict = {'treatment1': 0.1}
delta_uplift_decrease_dict = {'treatment1': -0.1}

# make a synthetic uplift data set
random_seed = 20200808
df, X_names = make_uplift_classification(
    treatment_name=treatment_group_keys,
    y_name=y_name,
    n_samples=n,
    n_classification_features=n_classification_features,
    n_classification_informative=n_classification_informative,
    n_classification_repeated=n_classification_repeated,
    n_uplift_increase_dict=n_uplift_increase_dict,
    n_uplift_decrease_dict=n_uplift_decrease_dict,
    delta_uplift_increase_dict = delta_uplift_increase_dict,
    delta_uplift_decrease_dict = delta_uplift_decrease_dict,
    random_seed=random_seed
)

# Feature selection with Filter method
filter_f = FilterSelect()
method = 'F'
f_imp = filter_f.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1')
print(f_imp)

# Use likelihood ratio test method
method = 'LR'
lr_imp = filter_f.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1')
print(lr_imp)

# Use KL divergence method
method = 'KL'
kl_imp = filter_f.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1',
                      n_bins=10)
print(kl_imp)
```

CHAPTER

FIVE

EXAMPLES

Working example notebooks are available in the example folder.

Follow the below links for an approximate ordering of example tutorials from introductory to advanced features.

5.1 Meta-Learners Examples - Training, Estimation, Validation, Visualization

5.1.1 Introduction

In this notebook, we will generate some synthetic data to demonstrate how to use the various Meta-Learner algorithms in order to estimate Individual Treatment Effects and Average Treatment Effects with confidence intervals.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
from causalml.inference.meta import LRSRegressor
```

```
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_
↪clustering.py:35: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _pt_shuffle_rec(i, indexes, index_mask, partition_tree, M, pos):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_
↪clustering.py:54: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def delta_minimization_order(all_masks, max_swap_size=100, num_passes=2):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_
↪clustering.py:63: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window(order, start, length):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_
↪clustering.py:69: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
```

(continues on next page)

(continued from previous page)

```
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window_score_gain(masks, order, start, length):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_
↪clustering.py:77: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _mask_delta_score(m1, m2):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/links.py:5:␣
↪NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the
↪'numba.jit' decorator. The implicit default value for this argument is currently␣
↪False, but it will be changed to True in Numba 0.59.0. See https://numba.
↪readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-
↪back-behaviour-when-using-jit for details.
  def identity(x):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/links.py:10:␣
↪NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the
↪'numba.jit' decorator. The implicit default value for this argument is currently␣
↪False, but it will be changed to True in Numba 0.59.0. See https://numba.
↪readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-
↪back-behaviour-when-using-jit for details.
  def _identity_inverse(x):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/links.py:15:␣
↪NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the
↪'numba.jit' decorator. The implicit default value for this argument is currently␣
↪False, but it will be changed to True in Numba 0.59.0. See https://numba.
↪readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-
↪back-behaviour-when-using-jit for details.
  def logit(x):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/links.py:20:␣
↪NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to the
↪'numba.jit' decorator. The implicit default value for this argument is currently␣
↪False, but it will be changed to True in Numba 0.59.0. See https://numba.
↪readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-
↪back-behaviour-when-using-jit for details.
  def _logit_inverse(x):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_masked_
↪model.py:363: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _build_fixed_single_output(averaged_outs, last_outs, outputs, batch_positions,␣
↪varying_rows, num_varying_rows, link, linearizing_weights):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_masked_
↪model.py:385: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _build_fixed_multi_output(averaged_outs, last_outs, outputs, batch_positions,␣
↪varying_rows, num_varying_rows, link, linearizing_weights):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_masked_
↪model.py:428: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
```

(continues on next page)

(continued from previous page)

```
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _init_masks(cluster_matrix, M, indices_row_pos, indptr):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/utils/_masked_
↪model.py:439: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _rec_fill_masks(cluster_matrix, indices_row_pos, indptr, indices, M, ind):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/maskers/_
↪tabular.py:186: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _single_delta_mask(dind, masked_inputs, last_mask, data, x, noop_code):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/maskers/_
↪tabular.py:197: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def _delta_masking(masks, x, curr_delta_inds, varying_rows_out,
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/tqdm/auto.py:22:␣
↪TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://
↪ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/maskers/_image.
↪py:175: NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied␣
↪to the 'numba.jit' decorator. The implicit default value for this argument is␣
↪currently False, but it will be changed to True in Numba 0.59.0. See https://numba.
↪readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode-fall-
↪back-behaviour-when-using-jit for details.
  def _jit_build_partition_tree(xmin, xmax, ymin, ymax, zmin, zmax, total_ywidth,␣
↪total_zwidth, M, clustering, q):
/Users/jeong/miniconda3/envs/causalml/lib/python3.8/site-packages/shap/explainers/_
↪partition.py:676: NumbaDeprecationWarning: The 'nopython' keyword argument was not␣
↪supplied to the 'numba.jit' decorator. The implicit default value for this argument␣
↪is currently False, but it will be changed to True in Numba 0.59.0. See https://
↪numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-
↪mode-fall-back-behaviour-when-using-jit for details.
  def lower_credit(i, value, M, values, clustering):
The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The␣
↪implicit default value for this argument is currently False, but it will be changed␣
↪to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/
↪deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for␣
↪details.
The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The␣
↪implicit default value for this argument is currently False, but it will be changed␣
↪to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/
↪deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for␣
↪details.
```

[3]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from xgboost import XGBRegressor
import warnings

from causalml.inference.meta import LRSRegressor
from causalml.inference.meta import XGBTRegressor, MLPTRegressor
from causalml.inference.meta import BaseXRegressor, BaseRRegressor, BaseSRegressor,␣
↪BaseTRegressor
from causalml.match import NearestNeighborMatch, MatchOptimizer, create_table_one
from causalml.propensity import ElasticNetPropensityModel
from causalml.dataset import *
from causalml.metrics import *

warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')

%matplotlib inline
```

```
Failed to import duecredit due to No module named 'duecredit'
```

[4]:
```
import importlib
print(importlib.metadata.version('causalml') )
```

```
0.15.1.dev0
```

5.1.2 Part A: Example Workflow using Synthetic Data

Generate synthetic data

- We have implemented 4 modes of generating synthetic data (specified by input parameter `mode`). Refer to the References section for more detail on these data generation processes.

[5]:
```
# Generate synthetic data using mode 1
y, X, treatment, tau, b, e = synthetic_data(mode=1, n=10000, p=8, sigma=1.0)
```

Calculate Average Treatment Effect (ATE)

A meta-learner can be instantiated by calling a base learner class and providing an sklearn/xgboost regressor class as input. Alternatively, we have provided some ready-to-use learners that have already inherited their respective base learner class capabilities. This is more abstracted and allows these tools to be quickly and readily usable.

[6]:
```
# Ready-to-use S-Learner using LinearRegression
learner_s = LRSRegressor()
ate_s = learner_s.estimate_ate(X=X, treatment=treatment, y=y)
print(ate_s)
print('ATE estimate: {:.03f}'.format(ate_s[0][0]))
print('ATE lower bound: {:.03f}'.format(ate_s[1][0]))
print('ATE upper bound: {:.03f}'.format(ate_s[2][0]))
```

(continues on next page)

(continued from previous page)

```
# After calling estimate_ate, add pretrain=True flag to skip training
# This flag is applicable for other meta learner
ate_s = learner_s.estimate_ate(X=X, treatment=treatment, y=y, pretrain=True)
print(ate_s)
print('ATE estimate: {:.03f}'.format(ate_s[0][0]))
print('ATE lower bound: {:.03f}'.format(ate_s[1][0]))
print('ATE upper bound: {:.03f}'.format(ate_s[2][0]))
```

```
(array([0.72721128]), array([0.67972656]), array([0.77469599]))
ATE estimate: 0.727
ATE lower bound: 0.680
ATE upper bound: 0.775
(array([0.72721128]), array([0.67972656]), array([0.77469599]))
ATE estimate: 0.727
ATE lower bound: 0.680
ATE upper bound: 0.775
```

[7]:
```
# Ready-to-use T-Learner using XGB
learner_t = XGBTRegressor()
ate_t = learner_t.estimate_ate(X=X, treatment=treatment, y=y)
print('Using the ready-to-use XGBTRegressor class')
print(ate_t)

# Calling the Base Learner class and feeding in XGB
learner_t = BaseTRegressor(learner=XGBRegressor())
ate_t = learner_t.estimate_ate(X=X, treatment=treatment, y=y)
print('\nUsing the BaseTRegressor class and using XGB (same result):')
print(ate_t)

# Calling the Base Learner class and feeding in LinearRegression
learner_t = BaseTRegressor(learner=LinearRegression())
ate_t = learner_t.estimate_ate(X=X, treatment=treatment, y=y)
print('\nUsing the BaseTRegressor class and using Linear Regression (different␣
↪result):')
print(ate_t)
```

```
Using the ready-to-use XGBTRegressor class
(array([0.55539207]), array([0.53185148]), array([0.57893267]))

Using the BaseTRegressor class and using XGB (same result):
(array([0.55539207]), array([0.53185148]), array([0.57893267]))

Using the BaseTRegressor class and using Linear Regression (different result):
(array([0.71740976]), array([0.67655445]), array([0.75826507]))
```

[8]:
```
# X Learner with propensity score input
# Calling the Base Learner class and feeding in XGB
learner_x = BaseXRegressor(learner=XGBRegressor())
ate_x = learner_x.estimate_ate(X=X, treatment=treatment, y=y, p=e)
print('Using the BaseXRegressor class and using XGB:')
print(ate_x)

# Calling the Base Learner class and feeding in LinearRegression
learner_x = BaseXRegressor(learner=LinearRegression())
ate_x = learner_x.estimate_ate(X=X, treatment=treatment, y=y, p=e)
print('\nUsing the BaseXRegressor class and using Linear Regression:')
print(ate_x)
```

```
Using the BaseXRegressor class and using XGB:
(array([0.52239345]), array([0.50279387]), array([0.54199302]))

Using the BaseXRegressor class and using Linear Regression:
(array([0.71740976]), array([0.67655445]), array([0.75826507]))
```

[9]:
```
# X Learner without propensity score input
# Calling the Base Learner class and feeding in XGB
learner_x = BaseXRegressor(XGBRegressor())
ate_x = learner_x.estimate_ate(X=X, treatment=treatment, y=y)
print('Using the BaseXRegressor class and using XGB without propensity score input:')
print(ate_x)

# Calling the Base Learner class and feeding in LinearRegression
learner_x = BaseXRegressor(learner=LinearRegression())
ate_x = learner_x.estimate_ate(X=X, treatment=treatment, y=y)
print('\nUsing the BaseXRegressor class and using Linear Regression without␣
↪propensity score input:')
print(ate_x)
```

```
Using the BaseXRegressor class and using XGB without propensity score input:
(array([0.52348025]), array([0.50385245]), array([0.54310804]))

Using the BaseXRegressor class and using Linear Regression without propensity score␣
↪input:
(array([0.71740976]), array([0.67655445]), array([0.75826507]))
```

[10]:
```
# R Learner with propensity score input
# Calling the Base Learner class and feeding in XGB
learner_r = BaseRRegressor(learner=XGBRegressor())
ate_r = learner_r.estimate_ate(X=X, treatment=treatment, y=y, p=e)
print('Using the BaseRRegressor class and using XGB:')
print(ate_r)

# Calling the Base Learner class and feeding in LinearRegression
learner_r = BaseRRegressor(learner=LinearRegression())
ate_r = learner_r.estimate_ate(X=X, treatment=treatment, y=y, p=e)
print('Using the BaseRRegressor class and using Linear Regression:')
print(ate_r)
```

```
Using the BaseRRegressor class and using XGB:
(array([0.51551318]), array([0.5150305]), array([0.51599587]))
Using the BaseRRegressor class and using Linear Regression:
(array([0.51503495]), array([0.51461987]), array([0.51545004]))
```

[11]:
```
# R Learner with propensity score input and random sample weight
# Calling the Base Learner class and feeding in XGB
learner_r = BaseRRegressor(learner=XGBRegressor())
sample_weight = np.random.randint(1, 3, len(y))
ate_r = learner_r.estimate_ate(X=X, treatment=treatment, y=y, p=e, sample_
↪weight=sample_weight)
print('Using the BaseRRegressor class and using XGB:')
print(ate_r)
```

```
Using the BaseRRegressor class and using XGB:
(array([0.48910448]), array([0.48861819]), array([0.48959077]))
```

[12]:
```
# R Learner without propensity score input
# Calling the Base Learner class and feeding in XGB
learner_r = BaseRRegressor(learner=XGBRegressor())
ate_r = learner_r.estimate_ate(X=X, treatment=treatment, y=y)
print('Using the BaseRRegressor class and using XGB without propensity score input:')
print(ate_r)

# Calling the Base Learner class and feeding in LinearRegression
learner_r = BaseRRegressor(learner=LinearRegression())
ate_r = learner_r.estimate_ate(X=X, treatment=treatment, y=y)
print('Using the BaseRRegressor class and using Linear Regression without propensity␣
↪score input:')
print(ate_r)
```

```
Using the BaseRRegressor class and using XGB without propensity score input:
(array([0.45400543]), array([0.45352042]), array([0.45449043]))
Using the BaseRRegressor class and using Linear Regression without propensity score␣
↪input:
(array([0.59802659]), array([0.59761147]), array([0.5984417]))
```

7. Calculate Individual Treatment Effect (ITE/CATE)

CATE stands for Conditional Average Treatment Effect.

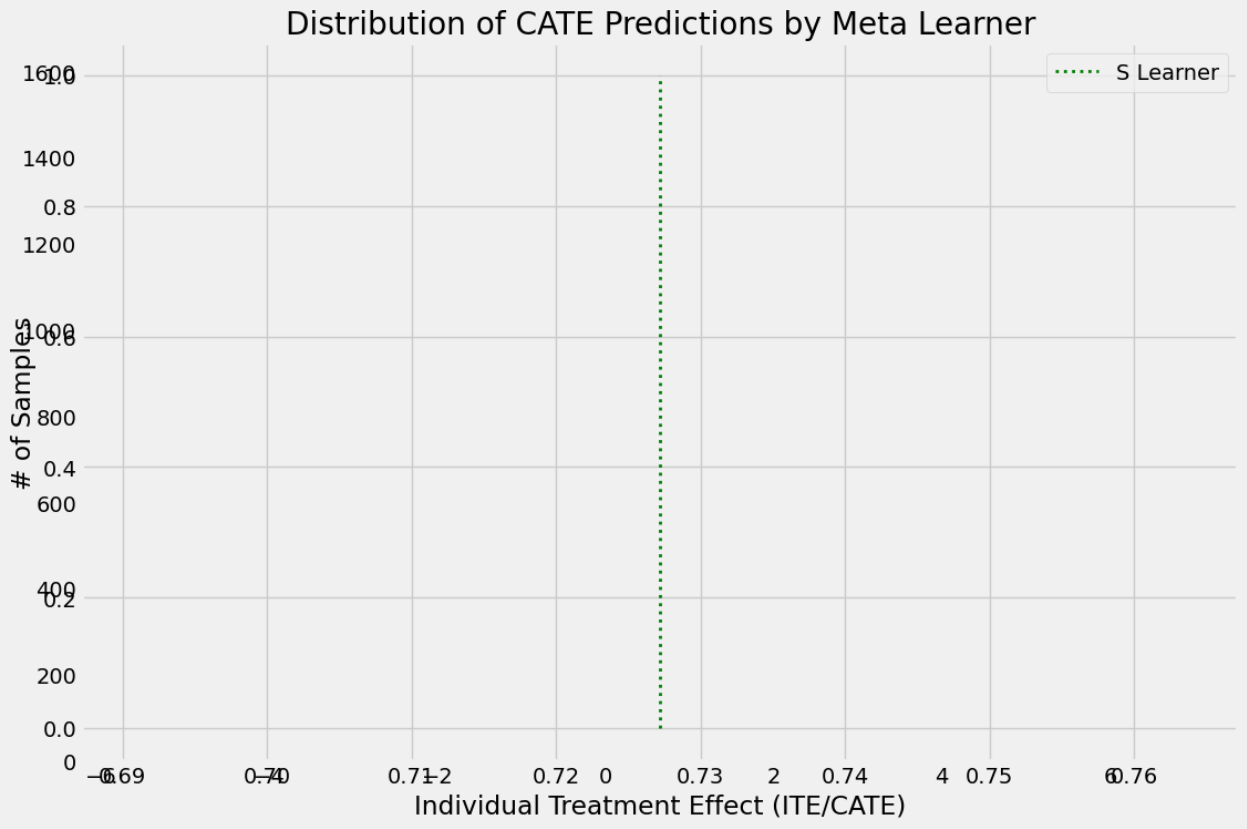
[13]:
```
# S Learner
learner_s = LRSRegressor()
cate_s = learner_s.fit_predict(X=X, treatment=treatment, y=y)

# T Learner
learner_t = BaseTRegressor(learner=XGBRegressor())
cate_t = learner_t.fit_predict(X=X, treatment=treatment, y=y)

# X Learner with propensity score input
learner_x = BaseXRegressor(learner=XGBRegressor())
cate_x = learner_x.fit_predict(X=X, treatment=treatment, y=y, p=e)

# X Learner without propensity score input
learner_x_no_p = BaseXRegressor(learner=XGBRegressor())
cate_x_no_p = learner_x_no_p.fit_predict(X=X, treatment=treatment, y=y)

# R Learner with propensity score input
learner_r = BaseRRegressor(learner=XGBRegressor())
cate_r = learner_r.fit_predict(X=X, treatment=treatment, y=y, p=e)

# R Learner without propensity score input
learner_r_no_p = BaseRRegressor(learner=XGBRegressor())
cate_r_no_p = learner_r_no_p.fit_predict(X=X, treatment=treatment, y=y)
```

[14]:
```
alpha=0.2
bins=30
plt.figure(figsize=(12,8))
plt.hist(cate_t, alpha=alpha, bins=bins, label='T Learner')
plt.hist(cate_x, alpha=alpha, bins=bins, label='X Learner')
plt.hist(cate_x_no_p, alpha=alpha, bins=bins, label='X Learner (no propensity score)')
plt.hist(cate_r, alpha=alpha, bins=bins, label='R Learner')
plt.hist(cate_r_no_p, alpha=alpha, bins=bins, label='R Learner (no propensity score)')
```

(continues on next page)

(continued from previous page)

```
plt.vlines(cate_s[0], 0, plt.axes().get_ylim()[1], label='S Learner',
           linestyles='dotted', colors='green', linewidth=2)
plt.title('Distribution of CATE Predictions by Meta Learner')
plt.xlabel('Individual Treatment Effect (ITE/CATE)')
plt.ylabel('# of Samples')
_=plt.legend()
```



5.1.3 Part B: Validating Meta-Learner Accuracy

We will validate the meta-learners' performance based on the same synthetic data generation method in Part A (`simulate_nuisance_and_easy_treatment`).

[15]:
```
train_summary, validation_summary = get_synthetic_summary_holdout(simulate_nuisance_
↪and_easy_treatment,
                                                                  n=10000,
                                                                  valid_size=0.2,
                                                                  k=10)
```

[16]:
```
train_summary
```

[16]:

	Abs % Error of ATE	MSE	KL Divergence
Actuals	0.000000	0.000000	0.000000
S Learner (LR)	0.349749	0.072543	3.703728
S Learner (XGB)	0.043545	0.103968	0.249110
T Learner (LR)	0.334790	0.031673	0.282270
T Learner (XGB)	0.039432	0.726171	1.099338

(continues on next page)

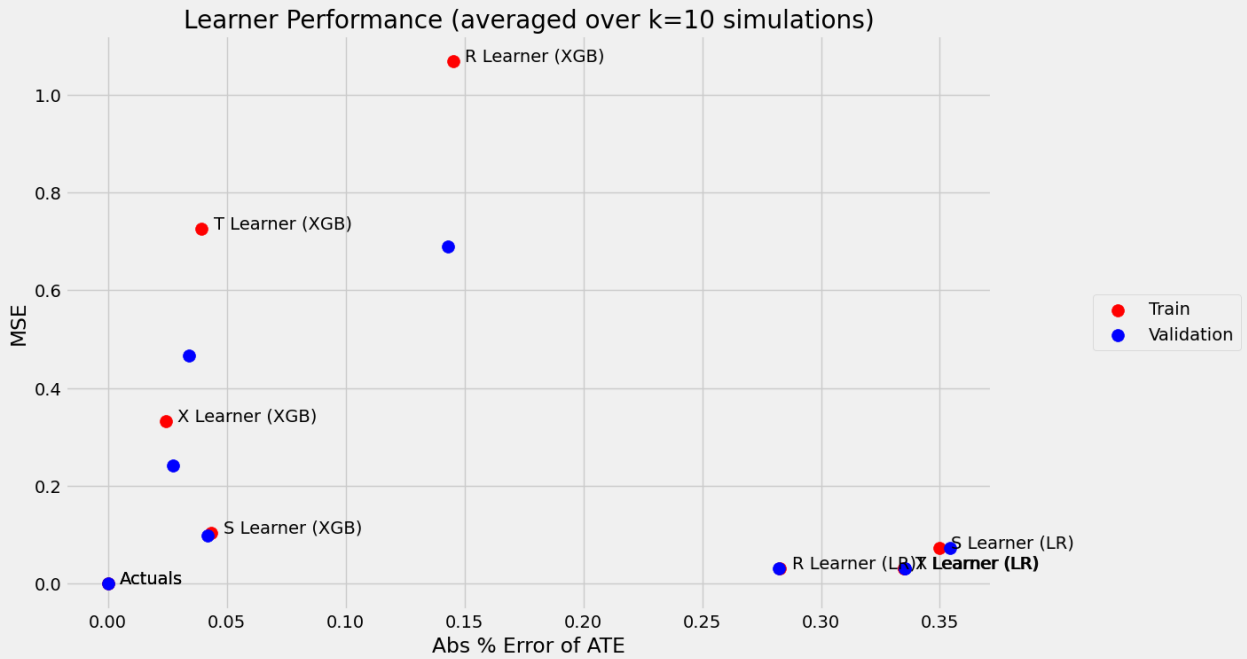
(continued from previous page)

```
X Learner (LR)              0.334790  0.031673       0.282270
X Learner (XGB)             0.024209  0.331760       0.682191
R Learner (LR)              0.282719  0.031079       0.267950
R Learner (XGB)             0.145131  1.068164       1.230508
```

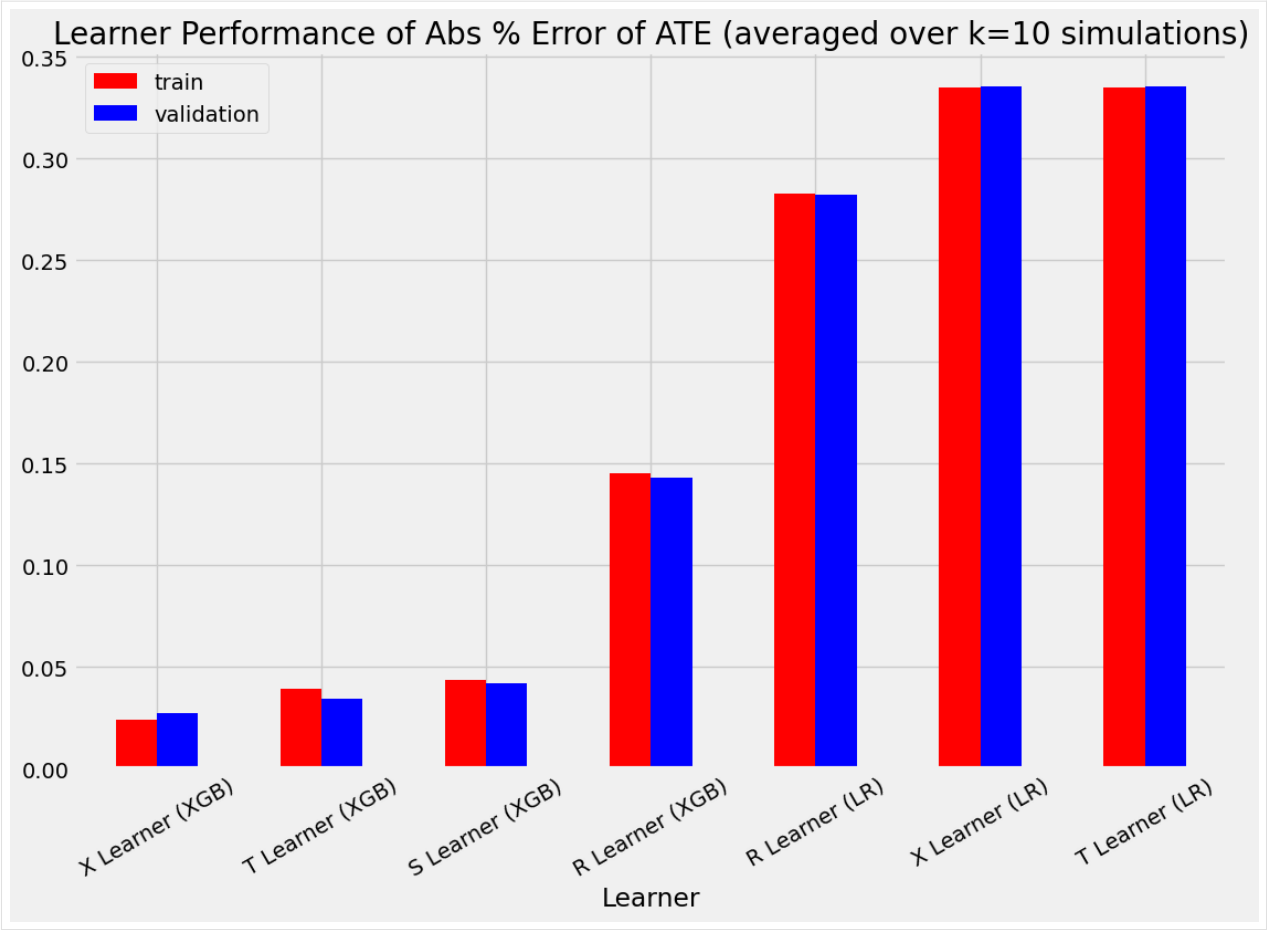
[17]:
```
validation_summary
```

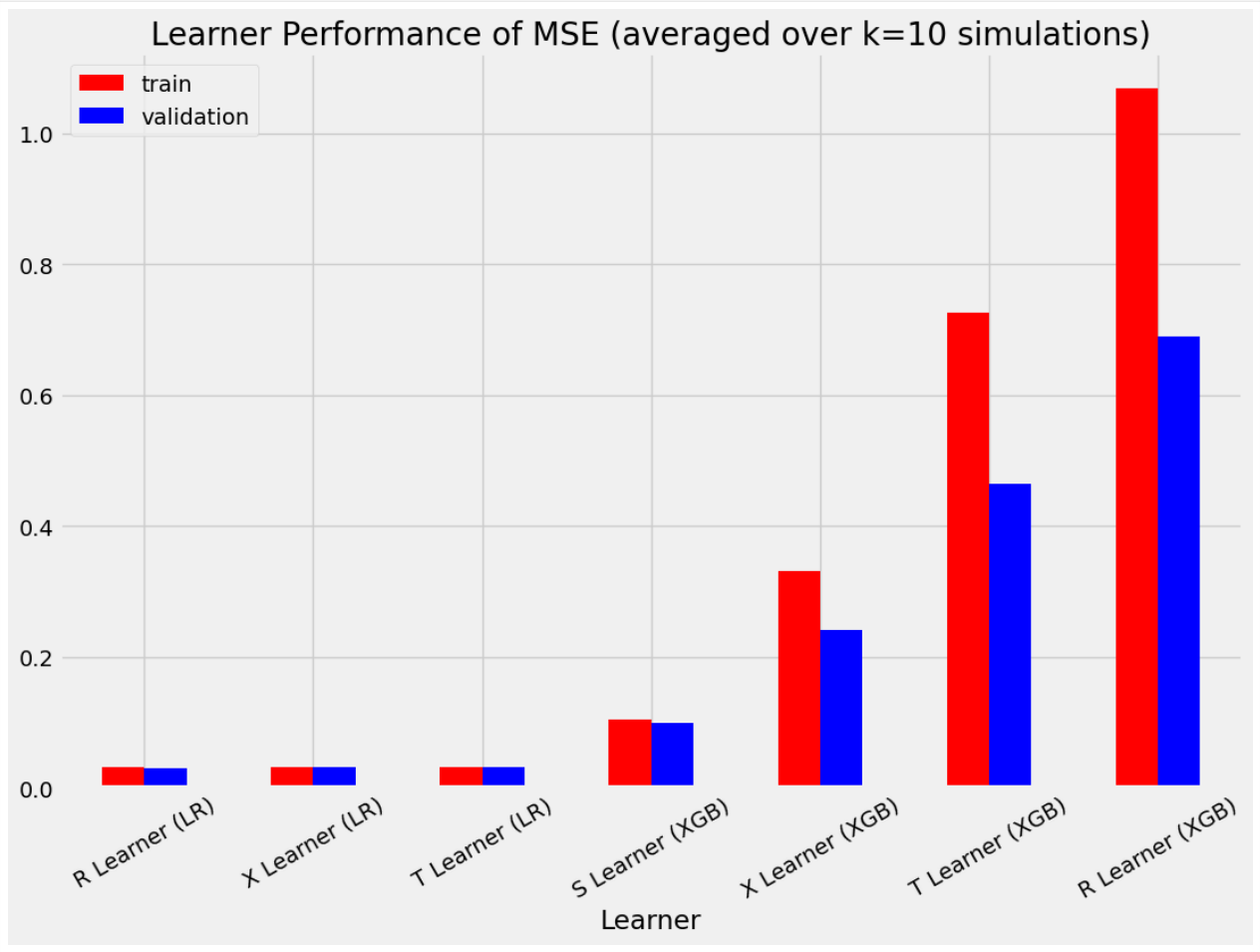
[17]:
```
                  Abs % Error of ATE       MSE  KL Divergence
Actuals                     0.000000  0.000000       0.000000
S Learner (LR)              0.354242  0.072989       3.797245
S Learner (XGB)             0.041916  0.098853       0.251344
T Learner (LR)              0.335444  0.031613       0.314690
T Learner (XGB)             0.034209  0.465331       0.904251
X Learner (LR)              0.335444  0.031613       0.314690
X Learner (XGB)             0.027473  0.241996       0.554173
R Learner (LR)              0.282196  0.030891       0.298666
R Learner (XGB)             0.143055  0.689088       1.061381
```

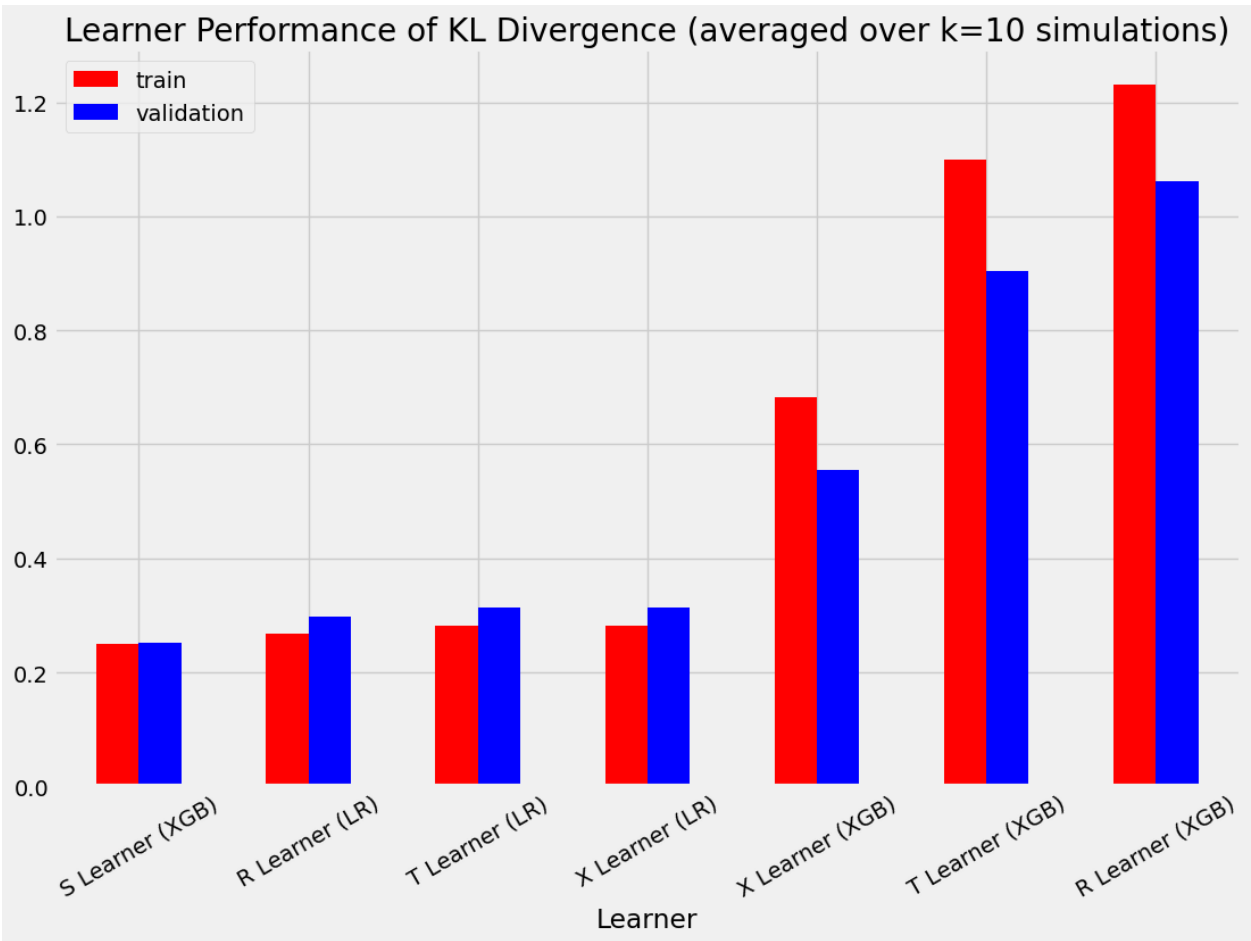
[18]:
```
scatter_plot_summary_holdout(train_summary,
                             validation_summary,
                             k=10,
                             label=['Train', 'Validation'],
                             drop_learners=[],
                             drop_cols=[])
```



[19]:
```
bar_plot_summary_holdout(train_summary,
                         validation_summary,
                         k=10,
                         drop_learners=['S Learner (LR)'],
                         drop_cols=[])
```

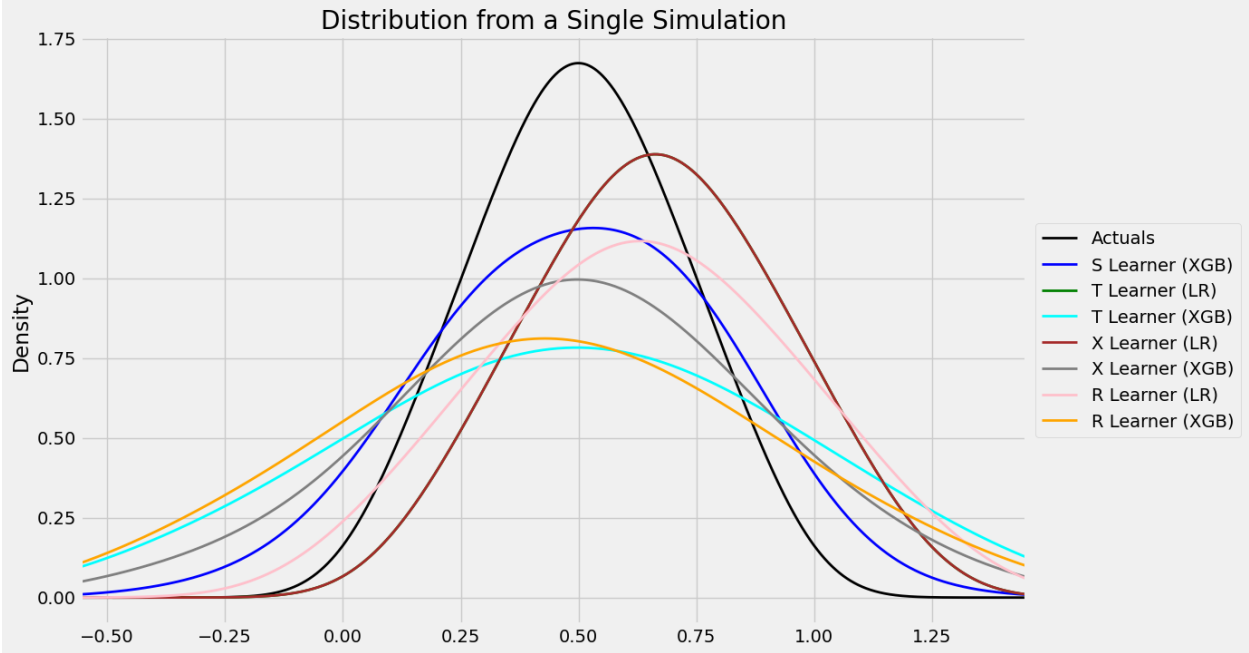

Learner Performance of Abs % Error of ATE (averaged over k=10 simulations)
train
validation
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
X Learner (XGB)
T Learner (XGB)
S Learner (XGB)
R Learner (XGB)
R Learner (LR)
X Learner (LR)
T Learner (LR)
Learner


Learner Performance of MSE (averaged over k=10 simulations)
train
validation
1.0
0.8
0.6
0.4
0.2
0.0
R Learner (LR)
X Learner (LR)
T Learner (LR)
S Learner (XGB)
X Learner (XGB)
T Learner (XGB)
R Learner (XGB)
Learner

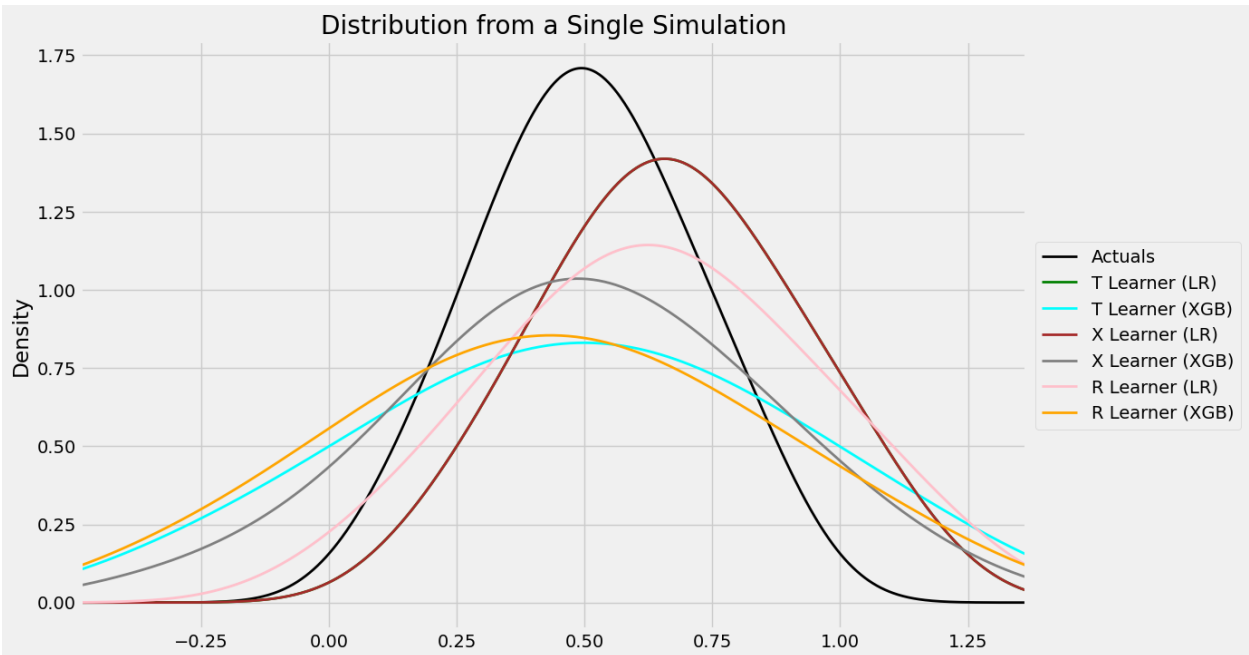


[20]:
```
# Single simulation
train_preds, valid_preds = get_synthetic_preds_holdout(simulate_nuisance_and_easy_
↪treatment,
                                                       n=50000,
                                                       valid_size=0.2)
```

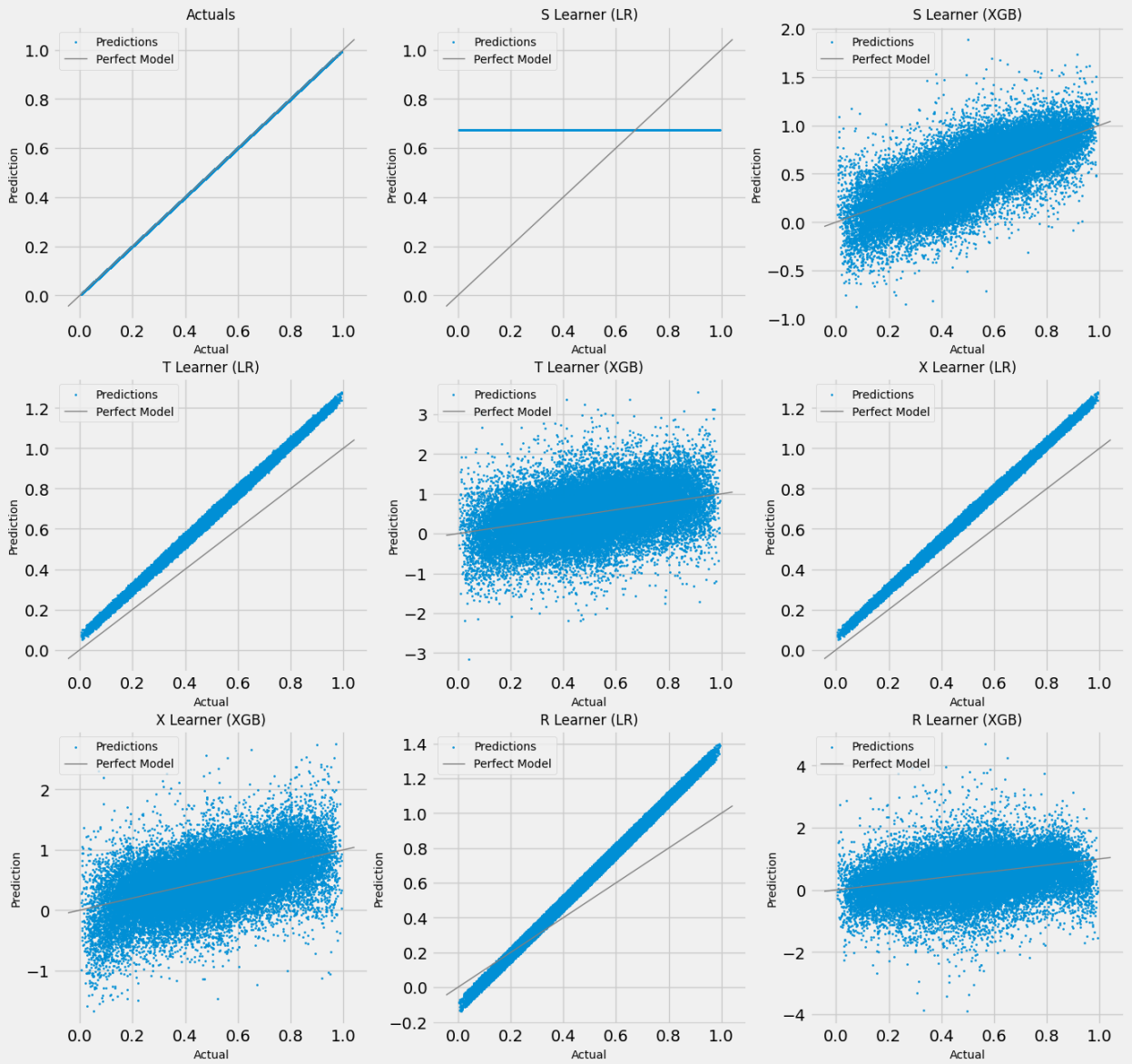
[21]:
```
#distribution plot for signle simulation of Training
distr_plot_single_sim(train_preds, kind='kde', linewidth=2, bw_method=0.5,
                      drop_learners=['S Learner (LR)',' S Learner (XGB)'])
```



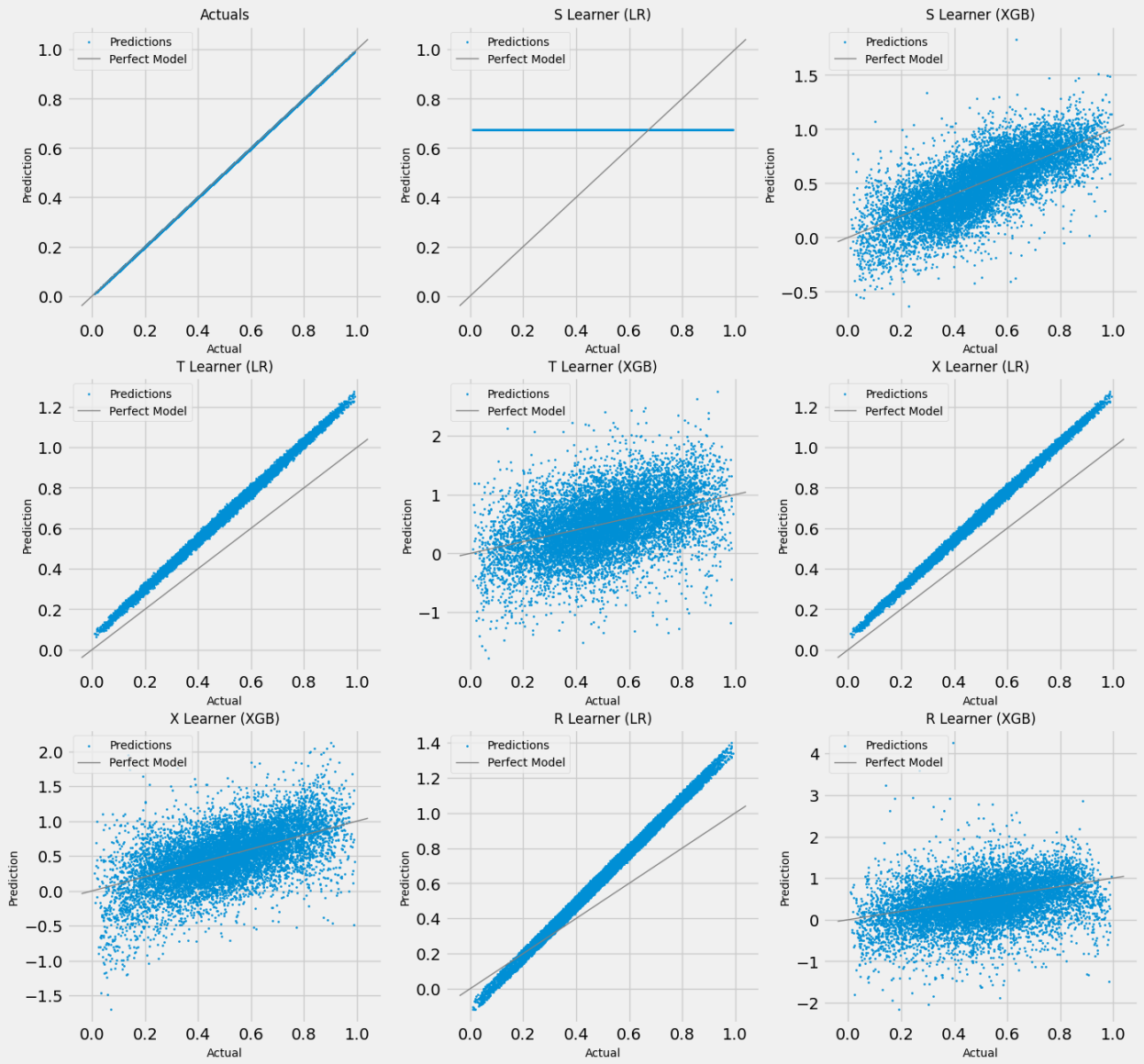
[22]:
```
#distribution plot for signle simulation of Validaiton
distr_plot_single_sim(valid_preds, kind='kde', linewidth=2, bw_method=0.5,
                      drop_learners=['S Learner (LR)', 'S Learner (XGB)'])
```



[23]:
```
# Scatter Plots for a Single Simulation of Training Data
scatter_plot_single_sim(train_preds)
```



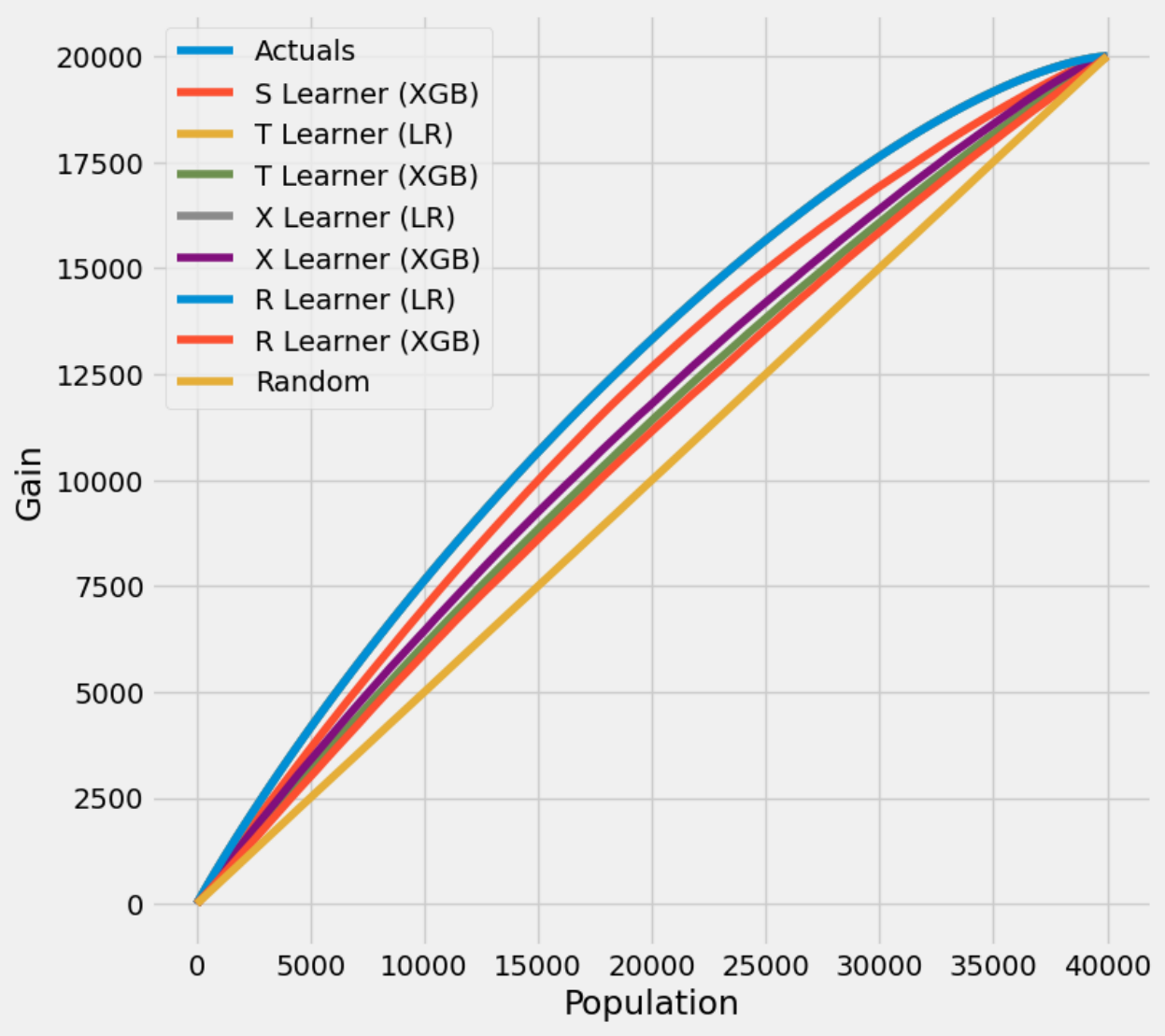
[24]:
```
# Scatter Plots for a Single Simulation of Validaiton Data
scatter_plot_single_sim(valid_preds)
```



[25]:
```
# Cumulative Gain AUUC values for a Single Simulation of Training Data
get_synthetic_auuc(train_preds, drop_learners=['S Learner (LR)'])
```

[25]:

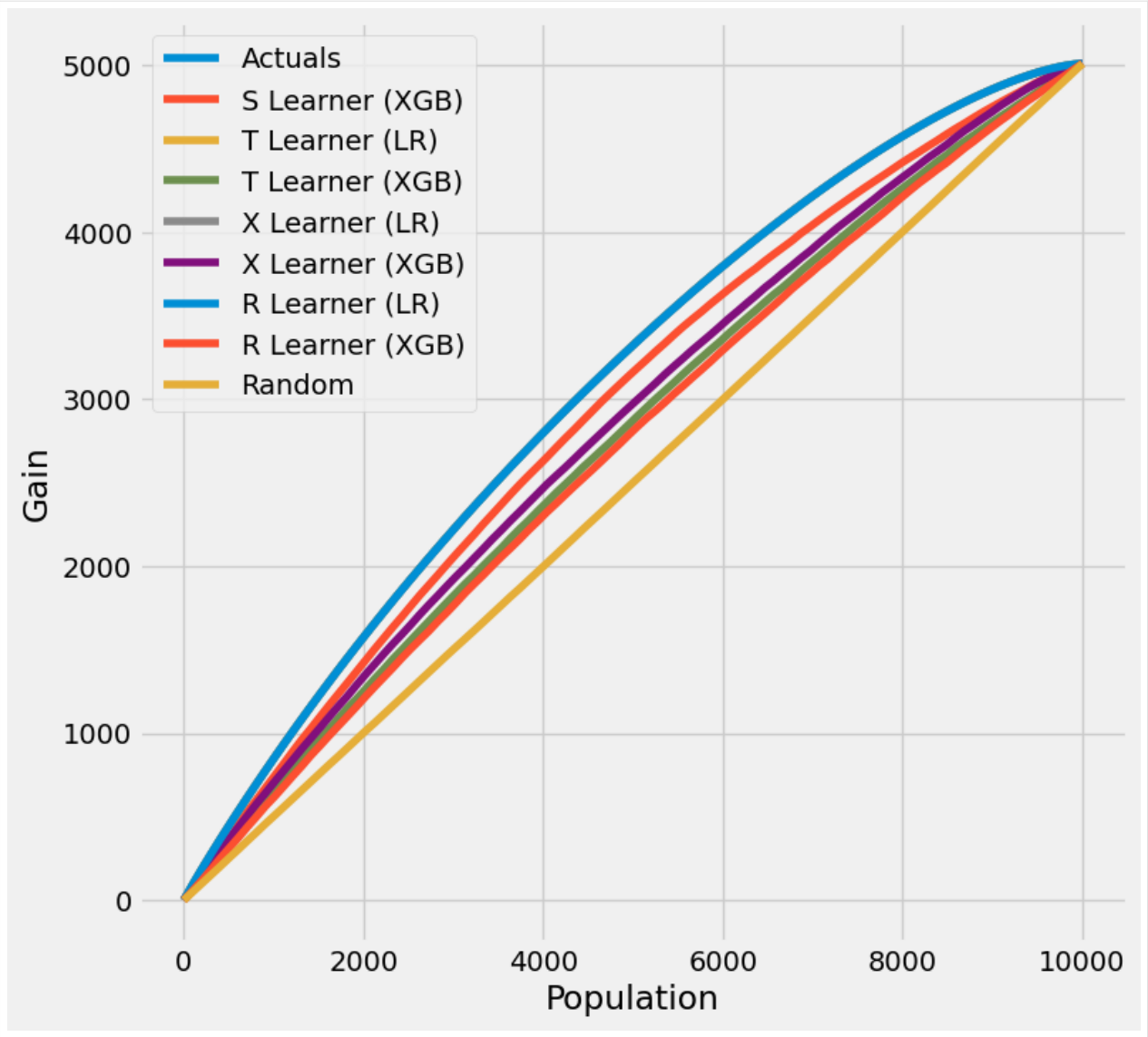
	Learner	cum_gain_auuc
0	Actuals	4.934321e+06
2	T Learner (LR)	4.932595e+06
4	X Learner (LR)	4.932595e+06
6	R Learner (LR)	4.931463e+06
1	S Learner (XGB)	4.707889e+06
5	X Learner (XGB)	4.507384e+06
3	T Learner (XGB)	4.389641e+06
7	R Learner (XGB)	4.309501e+06
8	Random	4.002357e+06



[26]:
```
# Cumulative Gain AUUC values for a Single Simulation of Validaiton Data
get_synthetic_auuc(valid_preds, drop_learners=['S Learner (LR)'])
```

[26]:

	Learner	cum_gain_auuc
0	Actuals	308122.561368
2	T Learner (LR)	308013.995722
4	X Learner (LR)	308013.995722
6	R Learner (LR)	307941.890461
1	S Learner (XGB)	294216.363545
5	X Learner (XGB)	283752.122952
3	T Learner (XGB)	276230.885568
7	R Learner (XGB)	271316.357530
8	Random	250262.193393



5.2 Uplift Trees Example with Synthetic Data

In this notebook, we use synthetic data to demonstrate the use of the tree-based algorithms.

[3]:
```
import numpy as np
import pandas as pd

from causalml.dataset import make_uplift_classification
from causalml.inference.tree import UpliftRandomForestClassifier
from causalml.metrics import plot_gain

from sklearn.model_selection import train_test_split
```

[4]:
```
import importlib
print(importlib.metadata.version('causalml') )
```

```
0.14.0
```

5.2.1 Generate synthetic dataset

The CausalML package contains various functions to generate synthetic datasets for uplift modeling. Here we generate a classification dataset using the make_uplift_classification() function.

[3]:
```
df, x_names = make_uplift_classification()
```

[4]:
```
df.head()
```

[4]:
```
  treatment_group_key  x1_informative  x2_informative  x3_informative  \
0             control       -0.542888        1.976361       -0.531359
1          treatment3        0.258654        0.552412        1.434239
2          treatment1        1.697012       -2.762600       -0.662874
3          treatment2       -1.441644        1.823648        0.789423
4             control       -0.625074        3.002388       -0.096288

   x4_informative  x5_informative  x6_irrelevant  x7_irrelevant  \
0       -2.354211       -0.380629      -2.614321      -0.128893
1       -1.422311        0.089131       0.790293       1.159513
2       -1.682340        1.217443       0.837982       1.042981
3       -0.295398        0.718509      -0.492993       0.947824
4        1.938235        3.392424      -0.465860      -0.919897

   x8_irrelevant  x9_irrelevant  …  x12_uplift_increase  x13_increase_mix  \
0       0.448689      -2.275192  …            -1.315304          0.742654
1       1.578868       0.166540  …            -1.391878         -0.623243
2       0.177398      -0.112409  …            -1.132497          1.050179
3      -1.307887       0.123340  …            -2.084619          0.058481
4      -1.072592      -1.331181  …            -1.403984          0.760430

   x14_uplift_increase  x15_uplift_increase  x16_increase_mix  \
0             1.891699            -2.428395          1.541875
1             2.443972            -2.889253          2.018585
2             1.573054            -1.788427          1.341609
3             1.369439             0.422538          1.087176
4             1.917635            -2.347675          1.560946

   x17_uplift_increase  x18_uplift_increase  x19_increase_mix  conversion  \
0            -0.817705            -0.610194         -0.591581           0
1            -1.109296            -0.380362         -1.667606           0
2            -0.749227            -2.091521         -0.471386           0
3            -0.966666            -1.785592         -1.268379           1
4            -0.833067            -1.407884         -0.781343           0

   treatment_effect
0                 0
1                 0
2                 0
3                 1
4                 0

[5 rows x 22 columns]
```

[5]:
```
# Look at the conversion rate and sample size in each group
df.pivot_table(values='conversion',
               index='treatment_group_key',
               aggfunc=[np.mean, np.size],
               margins=True)
```

[5]:

	mean	size
	conversion	conversion
treatment_group_key		
control	0.511	1000
treatment1	0.514	1000
treatment2	0.559	1000
treatment3	0.600	1000
All	0.546	4000

5.2.2 Run the uplift random forest classifier

In this section, we first fit the uplift random forest classifier using training data. We then use the fitted model to make a prediction using testing data. The prediction returns an ndarray in which each column contains the predicted uplift if the unit was in the corresponding treatment group.

[6]:
```
# Split data to training and testing samples for model validation (next section)
df_train, df_test = train_test_split(df, test_size=0.2, random_state=111)
```

[7]:
```
from causalml.inference.tree import UpliftTreeClassifier
```

[8]:
```
clf = UpliftTreeClassifier(control_name='control')
clf.fit(df_train[x_names].values,
        treatment=df_train['treatment_group_key'].values,
        y=df_train['conversion'].values)
p = clf.predict(df_test[x_names].values)
```

[9]:
```
df_res = pd.DataFrame(p, columns=clf.classes_)
df_res.head()
```

[9]:

	control	treatment1	treatment2	treatment3
0	0.506394	0.511811	0.573935	0.503778
1	0.506394	0.511811	0.573935	0.503778
2	0.580838	0.458824	0.508982	0.452381
3	0.482558	0.572327	0.556757	0.961538
4	0.482558	0.572327	0.556757	0.961538

[10]:
```
uplift_model = UpliftRandomForestClassifier(control_name='control')
```

[11]:
```
uplift_model.fit(df_train[x_names].values,
                 treatment=df_train['treatment_group_key'].values,
                 y=df_train['conversion'].values)
```

[12]:
```
df_res = uplift_model.predict(df_test[x_names].values, full_output=True)
print(df_res.shape)
df_res.head()
```

```
(800, 9)
```

[12]:
```
    control  treatment1  treatment2  treatment3  recommended_treatment  \
0  0.415263    0.401823    0.465554    0.391658                      2
1  0.412962    0.389346    0.476169    0.363343                      2
2  0.533442    0.548670    0.589756    0.588654                      2
3  0.344854    0.314433    0.370315    0.760676                      3
4  0.649657    0.602642    0.641364    0.851301                      3

   delta_treatment1  delta_treatment2  delta_treatment3  max_delta
0         -0.013440          0.050291         -0.023605   0.050291
1         -0.023616          0.063206         -0.049619   0.063206
2          0.015228          0.056313          0.055212   0.056313
3         -0.030420          0.025461          0.415822   0.415822
4         -0.047015         -0.008293          0.201644   0.201644
```

[13]:
```
y_pred = uplift_model.predict(df_test[x_names].values)
```

[14]:
```
y_pred.shape
```

[14]:
```
(800, 3)
```

[15]:
```
# Put the predictions to a DataFrame for a neater presentation
# The output of `predict()` is a numpy array with the shape of [n_sample, n_
↪treatment] excluding the
# predictions for the control group.
result = pd.DataFrame(y_pred,
                      columns=uplift_model.classes_[1:])
result.head()
```

[15]:
```
   treatment1  treatment2  treatment3
0   -0.013440    0.050291   -0.023605
1   -0.023616    0.063206   -0.049619
2    0.015228    0.056313    0.055212
3   -0.030420    0.025461    0.415822
4   -0.047015   -0.008293    0.201644
```

5.2.3 Create the uplift curve

The performance of the model can be evaluated with the help of the uplift curve.

5.2.4 Create a synthetic population

The uplift curve is calculated on a synthetic population that consists of those that were in the control group and those who happened to be in the treatment group recommended by the model. We use the synthetic population to calculate the *actual* treatment effect within *predicted* treatment effect quantiles. Because the data is randomized, we have a roughly equal number of treatment and control observations in the predicted quantiles and there is no self selection to treatment groups.

[16]:
```
# If all deltas are negative, assing to control; otherwise assign to the treatment
# with the highest delta
best_treatment = np.where((result < 0).all(axis=1),
                          'control',
                          result.idxmax(axis=1))
```

(continues on next page)

(continued from previous page)

```
# Create indicator variables for whether a unit happened to have the
# recommended treatment or was in the control group
actual_is_best = np.where(df_test['treatment_group_key'] == best_treatment, 1, 0)
actual_is_control = np.where(df_test['treatment_group_key'] == 'control', 1, 0)
```

[17]:
```
synthetic = (actual_is_best == 1) | (actual_is_control == 1)
synth = result[synthetic]
```

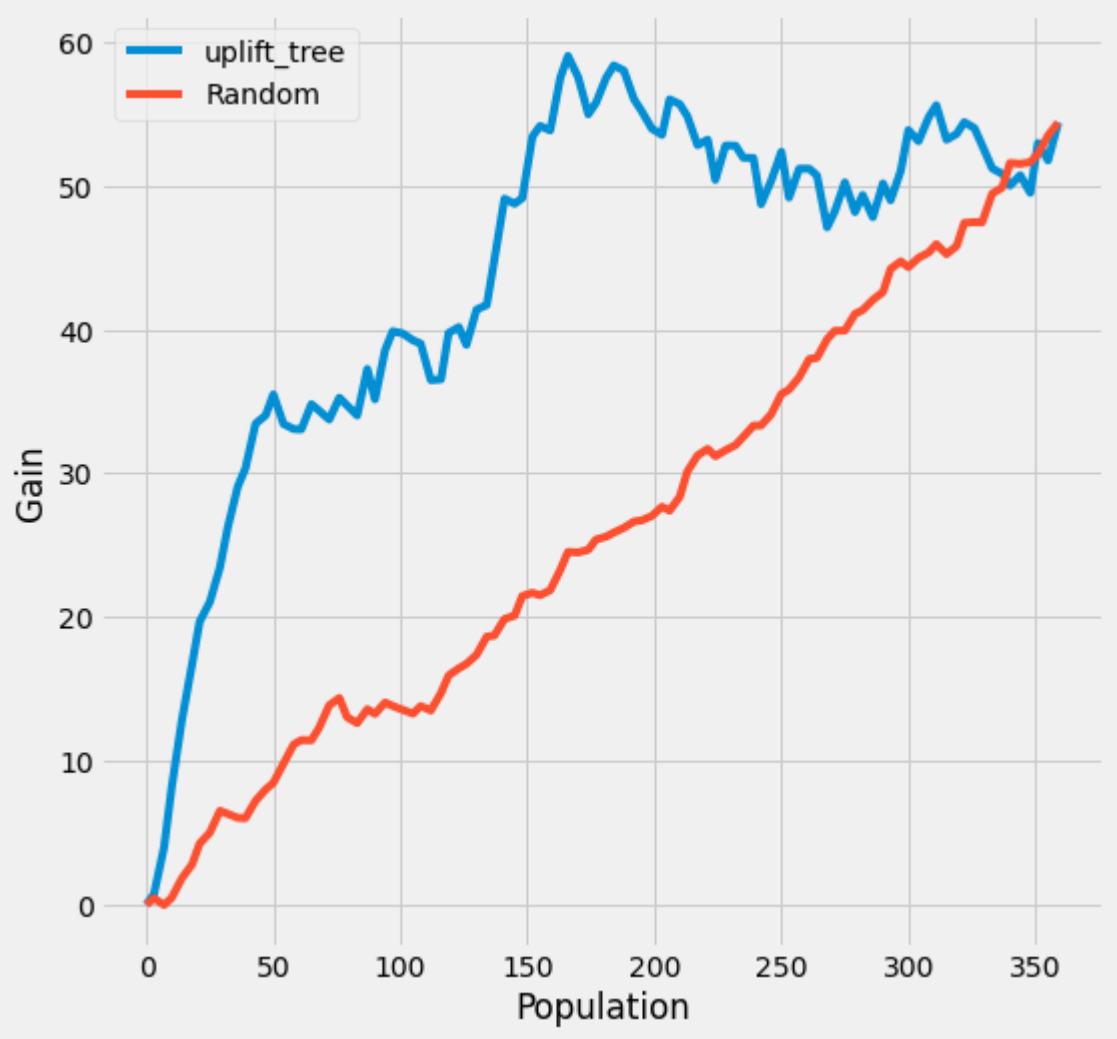
5.2.5 Calculate the observed treatment effect per predicted treatment effect quantile

We use the observed treatment effect to calculate the uplift curve, which answers the question: how much of the total cumulative uplift could we have captured by targeting a subset of the population sorted according to the predicted uplift, from highest to lowest?

CausalML has the plot_gain() function which calculates the uplift curve given a DataFrame containing the treatment assignment, observed outcome and the predicted treatment effect.

[18]:
```
auuc_metrics = (synth.assign(is_treated = 1 - actual_is_control[synthetic],
                             conversion = df_test.loc[synthetic, 'conversion'].values,
                             uplift_tree = synth.max(axis=1))
                     .drop(columns=list(uplift_model.classes_[1:])))
```

[19]:
```
plot_gain(auuc_metrics, outcome_col='conversion', treatment_col='is_treated')
```



[ ]:

5.3 Meta-Learners Examples - Single/Multiple Treatment Cases

This notebook only contains regression examples.

[1]:
```
%reload_ext autoreload
%autoreload 2
```

[2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from xgboost import XGBRegressor, XGBClassifier
import warnings
```

(continues on next page)

(continued from previous page)

```
# from causalml.inference.meta import XGBTLearner, MLPTLearner
from causalml.inference.meta import BaseSRegressor, BaseTRegressor, BaseXRegressor,␣
↪BaseRRegressor
from causalml.inference.meta import BaseSClassifier, BaseTClassifier, BaseXClassifier,
↪ BaseRClassifier
from causalml.inference.meta import LRSRegressor
from causalml.match import NearestNeighborMatch, MatchOptimizer, create_table_one
from causalml.propensity import ElasticNetPropensityModel
from causalml.dataset import *
from causalml.metrics import *

warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
pd.set_option('display.float_format', lambda x: '%.4f' % x)

# imports from package
import logging
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae
import statsmodels.api as sm
from copy import deepcopy

logger = logging.getLogger('causalml')
logging.basicConfig(level=logging.INFO)

%matplotlib inline
```

```
/Users/jeong/.conda/envs/py36/lib/python3.6/site-packages/sklearn/utils/deprecation.
↪py:144: FutureWarning: The sklearn.utils.testing module is  deprecated in version 0.
↪22 and will be removed in version 0.24. The corresponding classes / functions␣
↪should instead be imported from sklearn.utils. Anything that cannot be imported␣
↪from sklearn.utils is now part of the private API.
  warnings.warn(message, FutureWarning)
```

5.3.1 Single Treatment Case

Generate synthetic data

[3]:
```
# Generate synthetic data using mode 1
y, X, treatment, tau, b, e = synthetic_data(mode=1, n=10000, p=8, sigma=1.0)

treatment = np.array(['treatment_a' if val==1 else 'control' for val in treatment])
```

5.3.2 S-Learner

ATE

[4]:
```
learner_s = BaseSRegressor(XGBRegressor(), control_name='control')
ate_s = learner_s.estimate_ate(X=X, treatment=treatment, y=y, return_ci=False,␣
↪bootstrap_ci=False)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6622
INFO:causalml:    RMSE (Treatment):     0.6941
INFO:causalml:   sMAPE   (Control):     0.6536
INFO:causalml:   sMAPE (Treatment):     0.3721
INFO:causalml:    Gini   (Control):     0.8248
INFO:causalml:    Gini (Treatment):     0.8156
```

[5]:
```
ate_s
```

[5]:
```
array([0.57431368])
```

ATE w/ Confidence Intervals

[6]:
```
alpha = 0.05
learner_s = BaseSRegressor(XGBRegressor(), ate_alpha=alpha, control_name='control')
ate_s, ate_s_lb, ate_s_ub = learner_s.estimate_ate(X=X, treatment=treatment, y=y,␣
↪return_ci=True,
                                                   bootstrap_ci=False)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6622
INFO:causalml:    RMSE (Treatment):     0.6941
INFO:causalml:   sMAPE   (Control):     0.6536
INFO:causalml:   sMAPE (Treatment):     0.3721
INFO:causalml:    Gini   (Control):     0.8248
INFO:causalml:    Gini (Treatment):     0.8156
```

[7]:
```
np.vstack((ate_s_lb, ate_s, ate_s_ub))
```

[7]:
```
array([[0.54689052],
       [0.57431368],
       [0.60173684]])
```

ATE w/ Boostrap Confidence Intervals

[8]:
```
ate_s_b, ate_s_lb_b, ate_s_ub_b = learner_s.estimate_ate(X=X, treatment=treatment,␣
↪y=y, return_ci=True,
                                                         bootstrap_ci=True, n_
↪bootstraps=100, bootstrap_size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6622
INFO:causalml:    RMSE (Treatment):     0.6941
INFO:causalml:   sMAPE   (Control):     0.6536
INFO:causalml:   sMAPE (Treatment):     0.3721
```

(continues on next page)

(continued from previous page)

```
INFO:causalml:    Gini   (Control):     0.8248
INFO:causalml:    Gini (Treatment):     0.8156
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [01:14<00:00,  1.34it/s]
```

[9]:
```
np.vstack((ate_s_lb_b, ate_s_b, ate_s_ub_b))
```

[9]:
```
array([[0.51141982],
       [0.57431368],
       [0.64097547]])
```

CATE

[10]:
```
learner_s = BaseSRegressor(XGBRegressor(), control_name='control')
cate_s = learner_s.fit_predict(X=X, treatment=treatment, y=y, return_ci=False)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6622
INFO:causalml:    RMSE (Treatment):     0.6941
INFO:causalml:   sMAPE   (Control):     0.6536
INFO:causalml:   sMAPE (Treatment):     0.3721
INFO:causalml:    Gini   (Control):     0.8248
INFO:causalml:    Gini (Treatment):     0.8156
```

[11]:
```
cate_s
```

[11]:
```
array([[0.37674308],
       [0.42519259],
       [0.60864675],
       …,
       [0.19940662],
       [0.35013032],
       [0.78372002]])
```

CATE w/ Confidence Intervals

[12]:
```
alpha = 0.05
learner_s = BaseSRegressor(XGBRegressor(), ate_alpha=alpha, control_name='control')
cate_s, cate_s_lb, cate_s_ub = learner_s.fit_predict(X=X, treatment=treatment, y=y,␣
↪return_ci=True,
                                n_bootstraps=100, bootstrap_size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6622
INFO:causalml:    RMSE (Treatment):     0.6941
INFO:causalml:   sMAPE   (Control):     0.6536
INFO:causalml:   sMAPE (Treatment):     0.3721
INFO:causalml:    Gini   (Control):     0.8248
INFO:causalml:    Gini (Treatment):     0.8156
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [01:02<00:00,  1.59it/s]
```

[13]:
```
cate_s
```

[13]:
```
array([[0.37674308],
       [0.42519259],
       [0.60864675],
       …,
       [0.19940662],
       [0.35013032],
       [0.78372002]])
```

[14]:
```
cate_s_lb
```

[14]:
```
array([[-0.18972662],
       [ 0.20548496],
       [ 0.09983036],
       …,
       [-0.62837307],
       [-0.19766161],
       [-0.07736247]])
```

[15]:
```
cate_s_ub
```

[15]:
```
array([[0.8139405 ],
       [1.278447  ],
       [1.21720439],
       …,
       [0.90244564],
       [0.9450083 ],
       [1.1529291 ]])
```

5.3.3 T-Learner

ATE w/ Confidence Intervals

[16]:
```
learner_t = BaseTRegressor(XGBRegressor(), control_name='control')
ate_t, ate_t_lb, ate_t_ub = learner_t.estimate_ate(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
```

[17]:
```
np.vstack((ate_t_lb, ate_t, ate_t_ub))
```

[17]:
```
array([[0.55534845],
       [0.58090983],
       [0.60647121]])
```

ATE w/ Boostrap Confidence Intervals

[18]:
```
ate_t_b, ate_t_lb_b, ate_t_ub_b = learner_t.estimate_ate(X=X, treatment=treatment,␣
↪y=y, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [01:00<00:00,  1.66it/s]
```

[19]:
```
np.vstack((ate_t_lb_b, ate_t_b, ate_t_ub_b))
```

[19]:
```
array([[0.51343277],
       [0.58090983],
       [0.65843097]])
```

CATE

[20]:
```
learner_t = BaseTRegressor(XGBRegressor(), control_name='control')
cate_t = learner_t.fit_predict(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
```

[21]:
```
cate_t
```

[21]:
```
array([[ 0.23669004],
       [-0.0793891 ],
       [-0.10774326],
       …,
       [ 0.30539629],
       [ 0.50784194],
       [ 0.00356007]])
```

CATE w/ Confidence Intervals

[22]:
```
learner_t = BaseTRegressor(XGBRegressor(), control_name='control')
cate_t, cate_t_lb, cate_t_ub = learner_t.fit_predict(X=X, treatment=treatment, y=y,␣
↪return_ci=True, n_bootstraps=100,
                                                     bootstrap_size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [00:59<00:00,  1.68it/s]
```

[23]:
```
cate_t
```

[23]:
```
array([[ 0.23669004],
       [-0.0793891 ],
       [-0.10774326],
       …,
       [ 0.30539629],
       [ 0.50784194],
       [ 0.00356007]])
```

[24]:
```
cate_t_lb
```

[24]:
```
array([[-0.6752711 ],
       [-0.72038152],
       [-1.2330182 ],
       …,
       [-0.82131582],
       [-0.48846376],
       [-0.39046848]])
```

[25]:
```
cate_t_ub
```

[25]:
```
array([[1.66480025],
       [1.60697527],
       [2.06829221],
       …,
       [1.64941401],
       [1.59083122],
       [1.53139764]])
```

5.3.4 X-Learner

ATE w/ Confidence Intervals

With Propensity Score Input

[26]:
```
learner_x = BaseXRegressor(XGBRegressor(), control_name='control')
ate_x, ate_x_lb, ate_x_ub = learner_x.estimate_ate(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
```

[27]:
```
np.vstack((ate_x_lb, ate_x, ate_x_ub))
```

[27]:
```
array([[0.51454586],
       [0.53721713],
       [0.55988839]])
```

Without Propensity Score input

[28]:
```
ate_x_no_p, ate_x_lb_no_p, ate_x_ub_no_p = learner_x.estimate_ate(X=X,␣
↪treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
```

[29]:
```
np.vstack((ate_x_lb_no_p, ate_x_no_p, ate_x_ub_no_p))
```

[29]:
```
array([[0.51334384],
       [0.53600211],
       [0.55866038]])
```

[30]:
```
learner_x.propensity_model
```

[30]:
```
{'treatment_a': {'all training': LogisticRegressionCV(Cs=array([1.00230524, 2.
↪15608891, 4.63802765, 9.97700064]),
                       class_weight=None,
                       cv=StratifiedKFold(n_splits=3, random_state=None,␣
↪shuffle=True),
                       dual=False, fit_intercept=True, intercept_scaling=1.0,
                       l1_ratios=array([0.001     , 0.33366667, 0.66633333, 0.999    ␣
↪]),
                       max_iter=100, multi_class='auto', n_jobs=None,
                       penalty='elasticnet', random_state=None, refit=True,
                       scoring=None, solver='saga', tol=0.0001, verbose=0)}}
```

ATE w/ Boostrap Confidence Intervals

With Propensity Score Input

[31]:
```
ate_x_b, ate_x_lb_b, ate_x_ub_b = learner_x.estimate_ate(X=X, treatment=treatment,␣
↪y=y, p=e, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [01:55<00:00,  1.15s/it]
```

[32]:
```
np.vstack((ate_x_lb_b, ate_x_b, ate_x_ub_b))
```

[32]:
```
array([[0.46262759],
       [0.53721713],
       [0.59662513]])
```

Without Propensity Score Input

[33]:
```
ate_x_b_no_p, ate_x_lb_b_no_p, ate_x_ub_b_no_p = learner_x.estimate_ate(X=X,␣
↪treatment=treatment, y=y, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [01:56<00:00,  1.17s/it]
```

[34]:
```
np.vstack((ate_x_lb_b_no_p, ate_x_b_no_p, ate_x_ub_b_no_p))
```

[34]:
```
array([[0.44360865],
       [0.53598752],
       [0.59794413]])
```

CATE

With Propensity Score Input

[35]:
```
learner_x = BaseXRegressor(XGBRegressor(), control_name='control')
cate_x = learner_x.fit_predict(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
```

[36]:
```
cate_x
```

[36]:
```
array([[0.05178452],
       [0.01907274],
       [0.79584839],
       …,
       [0.18147876],
       [0.34742898],
       [0.23145415]])
```

Without Propensity Score Input

[37]:
```
cate_x_no_p = learner_x.fit_predict(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
```

[38]:
```
cate_x_no_p
```

[38]:
```
array([[0.06426511],
       [0.0189166 ],
       [0.78233515],
       …,
       [0.2237187 ],
       [0.29647103],
       [0.2359861 ]])
```

CATE w/ Confidence Intervals

With Propensity Score Input

[39]:
```
learner_x = BaseXRegressor(XGBRegressor(), control_name='control')
cate_x, cate_x_lb, cate_x_ub = learner_x.fit_predict(X=X, treatment=treatment, y=y,␣
↪p=e, return_ci=True,
                                                     n_bootstraps=100, bootstrap_
↪size=3000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [01:14<00:00,  1.34it/s]
```

[40]:
```
cate_x
```

[40]:
```
array([[0.05178452],
       [0.01907274],
       [0.79584839],
       …,
       [0.18147876],
       [0.34742898],
       [0.23145415]])
```

[41]:
```
cate_x_lb
```

[41]:
```
array([[-0.71763188],
       [-0.79487709],
       [-0.329782  ],
       …,
       [-0.57672694],
       [-0.48450804],
       [-0.43157597]])
```

[42]:
```
cate_x_ub
```

[42]:
```
array([[1.40320321],
       [1.59906792],
       [1.59324502],
       …,
       [1.07747513],
       [1.30836353],
       [1.18985624]])
```

Without Propensity Score Input

[43]:
```
cate_x_no_p, cate_x_lb_no_p, cate_x_ub_no_p = learner_x.fit_predict(X=X,␣
↪treatment=treatment, y=y, return_ci=True,
                                                    n_bootstraps=100, bootstrap_
↪size=3000)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4868
INFO:causalml:    RMSE (Treatment):     0.5434
INFO:causalml:   sMAPE   (Control):     0.5230
INFO:causalml:   sMAPE (Treatment):     0.3114
INFO:causalml:    Gini   (Control):     0.9216
INFO:causalml:    Gini (Treatment):     0.8988
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [01:16<00:00,  1.31it/s]
```

[44]:
```
cate_x_no_p
```

[44]:
```
array([[0.06430496],
       [0.01891659],
       [0.78209735],
       …,
       [0.22376976],
       [0.29645377],
       [0.23597794]])
```

[45]:
```
cate_x_lb_no_p
```

[45]:
```
array([[-0.62013372],
       [-0.90236405],
       [-0.31043938],
       …,
       [-0.54219561],
       [-0.2852425 ],
       [-0.37437315]])
```

[46]:
```
cate_x_ub_no_p
```

[46]:
```
array([[1.4199368 ],
       [1.45096372],
       [1.57656827],
       …,
       [1.34583137],
       [1.37899369],
       [1.25074382]])
```

5.3.5 R-Learner

ATE w/ Confidence Intervals

With Propensity Score Input

[47]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
ate_r, ate_r_lb, ate_r_ub = learner_r.estimate_ate(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
```

[48]:
```
np.vstack((ate_r_lb, ate_r, ate_r_ub))
```

[48]:
```
array([[0.55904178],
       [0.55951123],
       [0.55998069]])
```

Without Propensity Score Input

[49]:
```
ate_r_no_p, ate_r_lb_no_p, ate_r_ub_no_p = learner_r.estimate_ate(X=X,␣
↪treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
```

[50]:
```
np.vstack((ate_r_lb_no_p, ate_r_no_p, ate_r_ub_no_p))
```

[50]:
```
array([[0.49307912],
       [0.49354918],
       [0.49401924]])
```

[51]:
```
learner_r.propensity_model
```

[51]:
```
{'treatment_a': {'all training': LogisticRegressionCV(Cs=array([1.00230524, 2.
↪15608891, 4.63802765, 9.97700064]),
                        class_weight=None,
                        cv=KFold(n_splits=5, random_state=None, shuffle=True),
                        dual=False, fit_intercept=True, intercept_scaling=1.0,
                        l1_ratios=array([0.001     , 0.33366667, 0.66633333, 0.999    ␣
↪]),
                        max_iter=100, multi_class='auto', n_jobs=None,
                        penalty='elasticnet', random_state=None, refit=True,
                        scoring=None, solver='saga', tol=0.0001, verbose=0)}}
```

ATE w/ Boostrap Confidence Intervals

With Propensity Score Input

[52]:
```
ate_r_b, ate_r_lb_b, ate_r_ub_b = learner_r.estimate_ate(X=X, treatment=treatment,␣
↪y=y, p=e, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [01:56<00:00,  1.17s/it]
```

[53]:
```
np.vstack((ate_r_lb_b, ate_r_b, ate_r_ub_b))
```

[53]:
```
array([[0.37951505],
       [0.54612646],
       [0.53701368]])
```

Without Propensity Score Input

[54]:
```
ate_r_b_no_p, ate_r_lb_b_no_p, ate_r_ub_b_no_p = learner_r.estimate_ate(X=X,␣
↪treatment=treatment, y=y, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [02:42<00:00,  1.63s/it]
```

[55]:
```
np.vstack((ate_r_lb_b_no_p, ate_r_b_no_p, ate_r_ub_b_no_p))
```

[55]:
```
array([[0.37126915],
       [0.50635052],
       [0.51400059]])
```

CATE

With Propensity Score Input

[56]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
cate_r = learner_r.fit_predict(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
```

[57]:
```
cate_r
```

[57]:
```
array([[ 1.57365084],
       [-0.63619554],
       [-0.05320793],
       …,
       [ 0.56346375],
       [ 0.56288183],
       [ 0.87085617]])
```

Without Propensity Score Input

[58]:
```
cate_r_no_p = learner_r.fit_predict(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
```

[59]:
```
cate_r_no_p
```

[59]:
```
array([[-0.19582933],
       [-0.29006499],
       [ 0.46513131],
       …,
       [ 0.89712083],
       [ 0.81002617],
       [ 0.82598114]])
```

CATE w/ Confidence Intervals

With Propensity Score Input

[60]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
cate_r, cate_r_lb, cate_r_ub = learner_r.fit_predict(X=X, treatment=treatment, y=y,␣
↪p=e, return_ci=True,
                                                     n_bootstraps=100, bootstrap_
↪size=1000)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [00:46<00:00,  2.15it/s]
```

[61]:
```
cate_r
```

[61]:
```
array([[ 0.43967736],
       [-0.27467608],
       [-0.36704457],
       …,
       [ 1.70213294],
       [ 0.53581667],
       [ 0.67119908]])
```

[62]:
```
cate_r_lb
```

[62]:
```
array([[-2.36270347],
       [-2.10110987],
       [-3.33190218],
       …,
       [-2.25005704],
       [-2.08611215],
       [-1.89283199]])
```

[63]:
```
cate_r_ub
```

[63]:
```
array([[3.23361461],
       [4.39421365],
       [3.95620847],
       …,
       [3.15905744],
       [3.23586204],
       [2.31788745]])
```

Without Propensity Score Input

[64]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
cate_r_no_p, cate_r_lb_no_p, cate_r_ub_no_p = learner_r.fit_predict(X=X,␣
↪treatment=treatment, y=y, return_ci=True,
                                                    n_bootstraps=100, bootstrap_
↪size=1000)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [00:31<00:00,  3.14it/s]
```

[65]:
```
cate_r_no_p
```

[65]:
```
array([[-0.14972556],
       [ 0.18446118],
       [ 0.23380044],
       …,
       [ 0.55917108],
       [-0.16540062],
       [ 0.62050438]])
```

[66]:
```
cate_r_lb_no_p
```

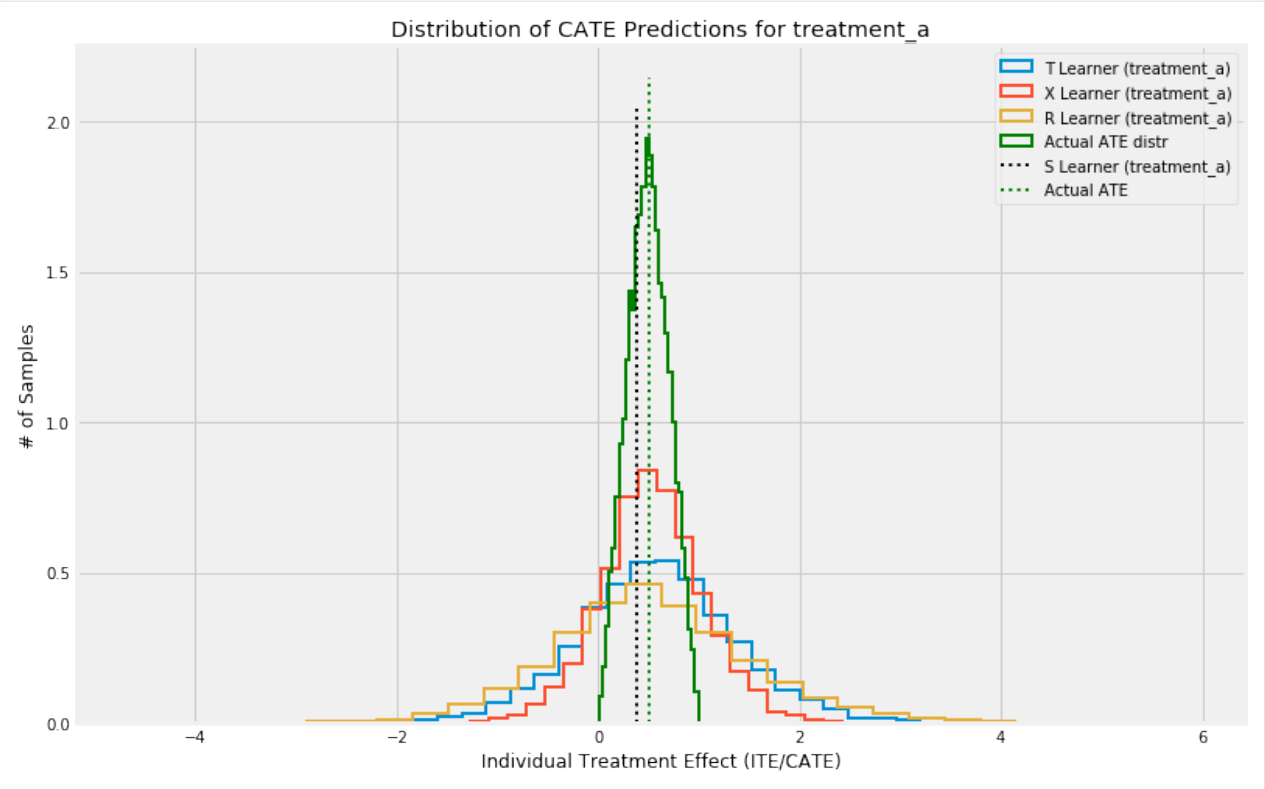
[66]:
```
array([[-2.37674593],
       [-1.66803797],
       [-3.47868801],
       …,
       [-1.95877534],
       [-2.32770172],
       [-1.68704787]])
```

[67]:
```
cate_r_ub_no_p
```

[67]:
```
array([[2.9130644 ],
       [3.99895564],
       [3.61212277],
       …,
       [3.174209  ],
       [3.38644627],
       [2.62858756]])
```

5.3.6 Visualize

[68]:
```
groups = learner_r._classes

alpha = 1
linewidth = 2
bins = 30
for group,idx in sorted(groups.items(), key=lambda x: x[1]):
    plt.figure(figsize=(12,8))
    plt.hist(cate_t[:,idx], alpha=alpha, bins=bins, label='T Learner ({})'.
↪format(group),
             histtype='step', linewidth=linewidth, density=True)
    plt.hist(cate_x[:,idx], alpha=alpha, bins=bins, label='X Learner ({})'.
↪format(group),
             histtype='step', linewidth=linewidth, density=True)
    plt.hist(cate_r[:,idx], alpha=alpha, bins=bins, label='R Learner ({})'.
↪format(group),
             histtype='step', linewidth=linewidth, density=True)
    plt.hist(tau, alpha=alpha, bins=bins, label='Actual ATE distr',
             histtype='step', linewidth=linewidth, color='green', density=True)
    plt.vlines(cate_s[0,idx], 0, plt.axes().get_ylim()[1], label='S Learner ({})'.
↪format(group),
               linestyles='dotted', linewidth=linewidth)
    plt.vlines(tau.mean(), 0, plt.axes().get_ylim()[1], label='Actual ATE',
               linestyles='dotted', linewidth=linewidth, color='green')

    plt.title('Distribution of CATE Predictions for {}'.format(group))
    plt.xlabel('Individual Treatment Effect (ITE/CATE)')
    plt.ylabel('# of Samples')
    _=plt.legend()
```



5.3.7 Multiple Treatment Case

Generate synthetic data

Note: we randomize the assignment of treatment flag AFTER the synthetic data generation process, so it doesn't make sense to measure accuracy metrics here. Next steps would be to include multi-treatment in the DGP itself.

[69]:
```
# Generate synthetic data using mode 1
y, X, treatment, tau, b, e = synthetic_data(mode=1, n=10000, p=8, sigma=1.0)

treatment = np.array([('treatment_a' if np.random.random() > 0.2 else 'treatment_b')
                      if val==1 else 'control' for val in treatment])

e = {group: e for group in np.unique(treatment)}
```

[70]:
```
pd.Series(treatment).value_counts()
```

[70]:
```
control        4768
treatment_a    4146
treatment_b    1086
dtype: int64
```

5.3.8 S-Learner

ATE

[71]:
```
learner_s = BaseSRegressor(XGBRegressor(), control_name='control')
ate_s = learner_s.estimate_ate(X=X, treatment=treatment, y=y, return_ci=False,␣
↪bootstrap_ci=False)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6339
INFO:causalml:    RMSE (Treatment):     0.6447
INFO:causalml:   sMAPE   (Control):     0.6148
INFO:causalml:   sMAPE (Treatment):     0.3498
INFO:causalml:    Gini   (Control):     0.8528
INFO:causalml:    Gini (Treatment):     0.8492
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.5584
INFO:causalml:    RMSE (Treatment):     0.4771
INFO:causalml:   sMAPE   (Control):     0.5699
INFO:causalml:   sMAPE (Treatment):     0.2768
INFO:causalml:    Gini   (Control):     0.8921
INFO:causalml:    Gini (Treatment):     0.9227
```

[72]:
```
ate_s
```

[72]:
```
array([0.58349553, 0.58778215])
```

[73]:
```
learner_s._classes
```

[73]:
```
{'treatment_a': 0, 'treatment_b': 1}
```

ATE w/ Confidence Intervals

[74]:
```
alpha = 0.05
learner_s = BaseSRegressor(XGBRegressor(), ate_alpha=alpha, control_name='control')
ate_s, ate_s_lb, ate_s_ub = learner_s.estimate_ate(X=X, treatment=treatment, y=y,␣
↪return_ci=True,
                                                   bootstrap_ci=False)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6339
INFO:causalml:    RMSE (Treatment):     0.6447
INFO:causalml:   sMAPE   (Control):     0.6148
INFO:causalml:   sMAPE (Treatment):     0.3498
INFO:causalml:    Gini   (Control):     0.8528
INFO:causalml:    Gini (Treatment):     0.8492
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.5584
INFO:causalml:    RMSE (Treatment):     0.4771
INFO:causalml:   sMAPE   (Control):     0.5699
INFO:causalml:   sMAPE (Treatment):     0.2768
INFO:causalml:    Gini   (Control):     0.8921
INFO:causalml:    Gini (Treatment):     0.9227
```

[75]:
```
np.vstack((ate_s_lb, ate_s, ate_s_ub))
```

[75]:
```
array([[0.5555693 , 0.55278018],
       [0.58349553, 0.58778215],
       [0.61142176, 0.62278413]])
```

ATE w/ Boostrap Confidence Intervals

[76]:
```
ate_s_b, ate_s_lb_b, ate_s_ub_b = learner_s.estimate_ate(X=X, treatment=treatment,␣
↪y=y, return_ci=True,
                                                         bootstrap_ci=True, n_
↪bootstraps=100, bootstrap_size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6339
INFO:causalml:    RMSE (Treatment):     0.6447
INFO:causalml:   sMAPE   (Control):     0.6148
INFO:causalml:   sMAPE (Treatment):     0.3498
INFO:causalml:    Gini   (Control):     0.8528
INFO:causalml:    Gini (Treatment):     0.8492
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.5584
INFO:causalml:    RMSE (Treatment):     0.4771
INFO:causalml:   sMAPE   (Control):     0.5699
INFO:causalml:   sMAPE (Treatment):     0.2768
INFO:causalml:    Gini   (Control):     0.8921
INFO:causalml:    Gini (Treatment):     0.9227
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [01:40<00:00,  1.00s/it]
```

[77]:
```
np.vstack((ate_s_lb_b, ate_s_b, ate_s_ub_b))
```

[77]:
```
array([[0.52550035, 0.52550035],
       [0.58349553, 0.58778215],
       [0.64944596, 0.64944596]])
```

CATE

[78]:
```
learner_s = BaseSRegressor(XGBRegressor(), control_name='control')
cate_s = learner_s.fit_predict(X=X, treatment=treatment, y=y, return_ci=False)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6339
INFO:causalml:    RMSE (Treatment):     0.6447
INFO:causalml:   sMAPE   (Control):     0.6148
INFO:causalml:   sMAPE (Treatment):     0.3498
INFO:causalml:    Gini   (Control):     0.8528
INFO:causalml:    Gini (Treatment):     0.8492
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.5584
INFO:causalml:    RMSE (Treatment):     0.4771
INFO:causalml:   sMAPE   (Control):     0.5699
INFO:causalml:   sMAPE (Treatment):     0.2768
INFO:causalml:    Gini   (Control):     0.8921
INFO:causalml:    Gini (Treatment):     0.9227
```

[79]:
```
cate_s
```

[79]:
```
array([[ 0.91381967,  0.82956386],
       [-0.17692167, -0.15709245],
       [ 0.90877771,  0.92332006],
       …,
       [ 0.86159408,  0.53687155],
       [ 0.66541922,  0.78590739],
       [ 1.05691028,  1.03345728]])
```

CATE w/ Confidence Intervals

[80]:
```
alpha = 0.05
learner_s = BaseSRegressor(XGBRegressor(), ate_alpha=alpha, control_name='control')
cate_s, cate_s_lb, cate_s_ub = learner_s.fit_predict(X=X, treatment=treatment, y=y,␣
↪return_ci=True,
                                n_bootstraps=100, bootstrap_size=3000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.6339
INFO:causalml:    RMSE (Treatment):     0.6447
INFO:causalml:   sMAPE   (Control):     0.6148
INFO:causalml:   sMAPE (Treatment):     0.3498
INFO:causalml:    Gini   (Control):     0.8528
INFO:causalml:    Gini (Treatment):     0.8492
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.5584
INFO:causalml:    RMSE (Treatment):     0.4771
INFO:causalml:   sMAPE   (Control):     0.5699
INFO:causalml:   sMAPE (Treatment):     0.2768
INFO:causalml:    Gini   (Control):     0.8921
INFO:causalml:    Gini (Treatment):     0.9227
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [01:03<00:00,  1.58it/s]
```

[81]:
```
cate_s
```

[81]:
```
array([[ 0.91381967,  0.82956386],
       [-0.17692167, -0.15709245],
       [ 0.90877771,  0.92332006],
       …,
       [ 0.86159408,  0.53687155],
       [ 0.66541922,  0.78590739],
       [ 1.05691028,  1.03345728]])
```

[82]:
```
cate_s_lb
```

[82]:
```
array([[ 0.23816384, -0.32713253],
       [-0.44141183, -0.42676411],
       [-0.00206863, -0.43860602],
       …,
       [ 0.29240462, -0.16563866],
       [-0.01797467, -0.10772878],
       [-0.51486325, -0.31691882]])
```

[83]:
```
cate_s_ub
```

[83]:
```
array([[1.40557503, 1.1807412 ],
       [1.06860972, 1.55298753],
       [1.38529261, 1.6596471 ],
       …,
       [1.56729684, 1.47052228],
       [1.16166003, 1.1144281 ],
       [1.68127107, 1.58984778]])
```

5.3.9 T-Learner

ATE w/ Confidence Intervals

[84]:
```
learner_t = BaseTRegressor(XGBRegressor(), control_name='control')
ate_t, ate_t_lb, ate_t_ub = learner_t.estimate_ate(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
```

[85]:
```
np.vstack((ate_t_lb, ate_t, ate_t_ub))
```

[85]:
```
array([[0.53107041, 0.5296616 ],
       [0.55739303, 0.55794811],
       [0.58371565, 0.58623463]])
```

ATE w/ Boostrap Confidence Intervals

[86]:
```
ate_t_b, ate_t_lb_b, ate_t_ub_b = learner_t.estimate_ate(X=X, treatment=treatment,␣
↪y=y, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
```

(continues on next page)

(continued from previous page)

```
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [01:32<00:00,  1.08it/s]
```

[87]:
```
np.vstack((ate_t_lb_b, ate_t_b, ate_t_ub_b))
```

[87]:
```
array([[0.51777538, 0.51777538],
       [0.55739303, 0.55794811],
       [0.67471492, 0.67471492]])
```

CATE

[88]:
```
learner_t = BaseTRegressor(XGBRegressor(), control_name='control')
cate_t = learner_t.fit_predict(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
```

[89]:
```
cate_t
```

[89]:
```
array([[ 1.47525787, -0.06651461],
       [ 1.26169336,  1.14718354],
       [ 1.68760026,  0.75878632],
       …,
       [ 0.37292147,  0.20537615],
       [ 0.84290075,  0.80045319],
       [ 1.64227223,  1.91352534]])
```

CATE w/ Confidence Intervals

[90]:
```
learner_t = BaseTRegressor(XGBRegressor(), control_name='control')
cate_t, cate_t_lb, cate_t_ub = learner_t.fit_predict(X=X, treatment=treatment, y=y,␣
↪return_ci=True, n_bootstraps=100,
                                                     bootstrap_size=3000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
```

(continues on next page)

(continued from previous page)

```
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [01:06<00:00,  1.51it/s]
```

[91]:
```
cate_t
```

[91]:
```
array([[ 1.47525787, -0.06651461],
       [ 1.26169336,  1.14718354],
       [ 1.68760026,  0.75878632],
       …,
       [ 0.37292147,  0.20537615],
       [ 0.84290075,  0.80045319],
       [ 1.64227223,  1.91352534]])
```

[92]:
```
cate_t_lb
```

[92]:
```
array([[-0.18706408, -0.84940575],
       [-1.01419897, -0.7311732 ],
       [-0.0427315 , -0.16378173],
       …,
       [-0.39076423, -0.16869925],
       [-0.17401927, -0.19503389],
       [-0.61903974, -1.15808628]])
```

[93]:
```
cate_t_ub
```

[93]:
```
array([[2.47563672, 1.69891493],
       [2.04089584, 1.76605188],
       [2.3567108 , 2.40833322],
       …,
       [2.17926003, 2.26919731],
       [2.15714553, 1.91076722],
       [2.27031788, 2.03901908]])
```

5.3.10 X-Learner

ATE w/ Confidence Intervals

With Propensity Score Input

[94]:
```
learner_x = BaseXRegressor(XGBRegressor(), control_name='control')
ate_x, ate_x_lb, ate_x_ub = learner_x.estimate_ate(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
```

[95]:
```
np.vstack((ate_x_lb, ate_x, ate_x_ub))
```

[95]:
```
array([[0.49573269, 0.54002602],
       [0.51860246, 0.56163457],
       [0.54147223, 0.58324311]])
```

Without Propensity Score Input

[96]:
```
ate_x_no_p, ate_x_lb_no_p, ate_x_ub_no_p = learner_x.estimate_ate(X=X,␣
↪treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
```

[97]:
```
np.vstack((ate_x_lb_no_p, ate_x_no_p, ate_x_ub_no_p))
```

[97]:
```
array([[0.50418298, 0.56976992],
       [0.52706595, 0.59243233],
       [0.54994892, 0.61509475]])
```

ATE w/ Boostrap Confidence Intervals

With Propensity Score Input

[98]:
```
ate_x_b, ate_x_lb_b, ate_x_ub_b = learner_x.estimate_ate(X=X, treatment=treatment,␣
↪y=y, p=e, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [02:55<00:00,  1.75s/it]
```

[99]:
```
np.vstack((ate_x_lb_b, ate_x_b, ate_x_ub_b))
```

[99]:
```
array([[0.49600789, 0.49600789],
       [0.51860246, 0.56163457],
       [0.63696386, 0.63696386]])
```

Without Propensity Score Input

[100]:
```
ate_x_b_no_p, ate_x_lb_b_no_p, ate_x_ub_b_no_p = learner_x.estimate_ate(X=X,␣
↪treatment=treatment, y=y, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
```

(continues on next page)

(continued from previous page)

```
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [02:54<00:00,  1.74s/it]
```

[101]:
```
np.vstack((ate_x_lb_b_no_p, ate_x_b_no_p, ate_x_ub_b_no_p))
```

[101]:
```
array([[0.50100288, 0.50100288],
       [0.52706414, 0.59242806],
       [0.66020792, 0.66020792]])
```

CATE

With Propensity Score Input

[102]:
```
learner_x = BaseXRegressor(XGBRegressor(), control_name='control')
cate_x = learner_x.fit_predict(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
```

[103]:
```
cate_x
```

[103]:
```
array([[ 0.57149441,  0.10240081],
       [-0.43192272,  1.48913118],
       [ 1.13622262,  0.65923928],
       …,
       [ 0.44651704, -0.23119723],
       [ 0.93875551,  0.77003003],
       [ 0.96697381,  0.99990004]])
```

Without Propensity Score Input

[104]:
```
cate_x_no_p = learner_x.fit_predict(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
```

(continues on next page)

(continued from previous page)

```
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
```

[105]:
```
cate_x_no_p
```

[105]:
```
array([[ 0.62959351, -0.00493521],
       [-0.48863166,  1.54109948],
       [ 1.17988308,  1.26200671],
       …,
       [ 0.41320951,  0.73251634],
       [ 0.91104634,  0.82359481],
       [ 1.08867931,  1.44193089]])
```

CATE w/ Confidence Intervals

With Propensity Score Input

[106]:
```
learner_x = BaseXRegressor(XGBRegressor(), control_name='control')
cate_x, cate_x_lb, cate_x_ub = learner_x.fit_predict(X=X, treatment=treatment, y=y,␣
↪p=e, return_ci=True,
                                                     n_bootstraps=100, bootstrap_
↪size=1000)
```

```
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [00:51<00:00,  1.94it/s]
```

[107]:
```
learner_x._classes
```

[107]:
```
{'treatment_a': 0, 'treatment_b': 1}
```

[108]:
```
cate_x
```

[108]:
```
array([[ 0.57149441,  0.10240081],
       [-0.43192272,  1.48913118],
```

(continues on next page)

(continued from previous page)

```
       [ 1.13622262,  0.65923928],
       …,
       [ 0.44651704, -0.23119723],
       [ 0.93875551,  0.77003003],
       [ 0.96697381,  0.99990004]])
```

[109]:
```
cate_x_lb
```

[109]:
```
array([[-0.23574115, -0.21029023],
       [-0.95699419, -1.05203708],
       [-0.49402807, -0.48280283],
       …,
       [-0.12162789, -0.26408791],
       [-0.52562958, -0.19338615],
       [-0.40858565, -0.88119588]])
```

[110]:
```
cate_x_ub
```

[110]:
```
array([[1.79950407, 2.11258332],
       [1.45309225, 1.48831446],
       [1.75564219, 2.03222137],
       …,
       [2.15191078, 2.30032378],
       [1.65228261, 1.40411322],
       [1.74815254, 1.68257617]])
```

Without Propensity Score Input

[111]:
```
cate_x_no_p, cate_x_lb_no_p, cate_x_ub_no_p = learner_x.fit_predict(X=X,␣
↪treatment=treatment, y=y, return_ci=True,
                                                    n_bootstraps=100, bootstrap_
↪size=1000)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Error metrics for group treatment_a
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.4669
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.2675
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9297
INFO:causalml:Error metrics for group treatment_b
INFO:causalml:    RMSE   (Control):     0.4743
INFO:causalml:    RMSE (Treatment):     0.0747
INFO:causalml:   sMAPE   (Control):     0.5062
INFO:causalml:   sMAPE (Treatment):     0.0568
INFO:causalml:    Gini   (Control):     0.9280
INFO:causalml:    Gini (Treatment):     0.9984
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [00:51<00:00,  1.94it/s]
```

[112]:
```
learner_x._classes
```

[112]:
```
{'treatment_a': 0, 'treatment_b': 1}
```

[113]:
```
cate_x_no_p
```

[113]:
```
array([[ 0.6294132 , -0.00492528],
       [-0.48876998,  1.54111376],
       [ 1.17989094,  1.2620318 ],
       …,
       [ 0.41319463,  0.73237091],
       [ 0.9108665 ,  0.82359564],
       [ 1.08868219,  1.441931  ]])
```

[114]:
```
cate_x_lb_no_p
```

[114]:
```
array([[-0.10073893, -0.38800051],
       [-0.81971717, -0.8298923 ],
       [-0.18606629, -0.32586878],
       …,
       [ 0.18372251, -0.12170252],
       [-0.21309623, -0.38600234],
       [-0.44863794, -0.39716903]])
```

[115]:
```
cate_x_ub_no_p
```

[115]:
```
array([[2.00312255, 2.10486085],
       [1.59355675, 1.76340695],
       [1.77980204, 2.35535097],
       …,
       [1.94828429, 1.94720835],
       [2.04021647, 1.71337955],
       [1.60121219, 1.82820234]])
```

5.3.11 R-Learner

ATE w/ Confidence Intervals

With Propensity Score Input

[116]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
ate_r, ate_r_lb, ate_r_ub = learner_r.estimate_ate(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
```

[117]:
```
np.vstack((ate_r_lb, ate_r, ate_r_ub))
```

[117]:
```
array([[0.52326968, 0.57744164],
       [0.52374892, 0.5781462 ],
       [0.52422816, 0.57885076]])
```

Without Propensity Score Input

[118]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
ate_r_no_p, ate_r_lb_no_p, ate_r_ub_no_p = learner_r.estimate_ate(X=X,␣
↪treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
```

[119]:
```
np.vstack((ate_r_lb_no_p, ate_r_no_p, ate_r_ub_no_p))
```

[119]:
```
array([[0.44161159, 0.71836119],
       [0.44209269, 0.71904979],
       [0.44257378, 0.71973838]])
```

[120]:
```
learner_r.propensity_model
```

[120]:
```
{'treatment_a': {'all training': LogisticRegressionCV(Cs=array([1.00230524, 2.
↪15608891, 4.63802765, 9.97700064]),
                      class_weight=None,
                      cv=KFold(n_splits=5, random_state=None, shuffle=True),
                      dual=False, fit_intercept=True, intercept_scaling=1.0,
                      l1_ratios=array([0.001     , 0.33366667, 0.66633333, 0.999    ␣
↪]),
                      max_iter=100, multi_class='auto', n_jobs=None,
                      penalty='elasticnet', random_state=None, refit=True,
                      scoring=None, solver='saga', tol=0.0001, verbose=0)},
 'treatment_b': {'all training': LogisticRegressionCV(Cs=array([1.00230524, 2.
↪15608891, 4.63802765, 9.97700064]),
                      class_weight=None,
                      cv=KFold(n_splits=5, random_state=None, shuffle=True),
                      dual=False, fit_intercept=True, intercept_scaling=1.0,
                      l1_ratios=array([0.001     , 0.33366667, 0.66633333, 0.999    ␣
↪]),
                      max_iter=100, multi_class='auto', n_jobs=None,
                      penalty='elasticnet', random_state=None, refit=True,
                      scoring=None, solver='saga', tol=0.0001, verbose=0)}}
```

ATE w/ Boostrap Confidence Intervals

With Propensity Score Input

[121]:
```
ate_r_b, ate_r_lb_b, ate_r_ub_b = learner_r.estimate_ate(X=X, treatment=treatment,␣
↪y=y, p=e, bootstrap_ci=True,
                                                         n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [02:19<00:00,  1.39s/it]
```

[122]:
```
np.vstack((ate_r_lb_b, ate_r_b, ate_r_ub_b))
```

[122]:
```
array([[0.40326436, 0.40326436],
       [0.50620059, 0.5478152 ],
       [0.5697328 , 0.5697328 ]])
```

Without Propensity Score Input

[123]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
ate_r_b_no_p, ate_r_lb_b_no_p, ate_r_ub_b_no_p = learner_r.estimate_ate(X=X,␣
↪treatment=treatment, y=y, bootstrap_ci=True,
                                                    n_bootstraps=100, bootstrap_
↪size=5000)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
INFO:causalml:Bootstrap Confidence Intervals for ATE
100%|██████████| 100/100 [02:19<00:00,  1.39s/it]
```

[124]:
```
np.vstack((ate_r_lb_b_no_p, ate_r_b_no_p, ate_r_ub_b_no_p))
```

[124]:
```
array([[0.45994051, 0.45994051],
       [0.44481491, 0.66323246],
       [0.68981572, 0.68981572]])
```

CATE

With Propensity Score Input

[125]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
cate_r = learner_r.fit_predict(X=X, treatment=treatment, y=y, p=e)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
```

[126]:
```
cate_r
```

[126]:
```
array([[ 5.57098567e-01,  1.77359581e-03],
       [ 1.08587885e+00,  2.48472750e-01],
       [ 3.34437251e-01,  1.69020355e+00],
       …,
       [-9.96065974e-01, -8.98482800e-02],
       [ 1.70625651e+00,  9.55640435e-01],
       [-1.88456130e+00,  6.50659442e-01]])
```

Without Propensity Score Input

[127]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
cate_r_no_p = learner_r.fit_predict(X=X, treatment=treatment, y=y)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
```

[128]:
```
cate_r_no_p
```

[128]:
```
array([[ 0.55478877,  0.87992519],
       [ 1.10120189,  1.29564619],
       [ 0.62448621,  0.41555083],
       …,
       [-0.53886592,  0.44593787],
       [ 1.25231111,  0.79904991],
       [-0.64419305, -0.23014426]])
```

CATE w/ Confidence Intervals

With Propensity Score Input

[129]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
cate_r, cate_r_lb, cate_r_ub = learner_r.fit_predict(X=X, treatment=treatment, y=y,␣
↪p=e, return_ci=True,
                                                     n_bootstraps=100, bootstrap_
↪size=1000)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
INFO:causalml:Bootstrap Confidence Intervals
100%|██████████| 100/100 [00:37<00:00,  2.65it/s]
```

[130]:
```
cate_r
```

[130]:
```
array([[ 1.75007784,  0.67752302],
       [ 0.77257723,  0.12910607],
       [ 1.08854032,  0.81679094],
       …,
       [-0.92310214,  0.645491  ],
       [ 0.92478108,  0.79903334],
       [-0.48311949,  1.00291944]])
```

[131]:
```
cate_r_lb
```

[131]:
```
array([[-0.801657  , -0.48754777],
       [-3.05317249, -5.37572038],
       [-1.50823961, -1.16822439],
       …,
       [-1.27909884, -1.2460175 ],
```

(continues on next page)

(continued from previous page)

```
       [-1.42656819, -1.59059022],
       [-1.90115855, -2.10247456]])
```

[132]:
```
cate_r_ub
```

[132]:
```
array([[4.06750882, 3.68516954],
       [4.21587243, 4.50271177],
       [4.33370841, 3.79358828],
       …,
       [3.53610538, 3.48638564],
       [3.71832166, 3.48292163],
       [5.01262635, 3.27047309]])
```

Without Propensity Score Input

[ ]:
```
learner_r = BaseRRegressor(XGBRegressor(), control_name='control')
cate_r_no_p, cate_r_lb_no_p, cate_r_ub_no_p = learner_r.fit_predict(X=X,␣
↪treatment=treatment, y=y, p=e, return_ci=True,
                                                    n_bootstraps=100, bootstrap_
↪size=1000)
```

```
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment_a with R-loss
INFO:causalml:training the treatment effect model for treatment_b with R-loss
INFO:causalml:Bootstrap Confidence Intervals
  2%|▏          | 2/100 [00:00<00:36,  2.69it/s]
```

[ ]:
```
cate_r_no_p
```

[ ]:
```
cate_r_lb_no_p
```

[ ]:
```
cate_r_ub_no_p
```

5.4 Uplift Trees/Forests Visualization

5.4.1 Introduction

This example notebooks illustrates how to visualize uplift trees for interpretation and diagnosis.

Supported Models

These visualization functions work only for tree-based 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

Supported Usage

This notebook will show how to use visualization for:

- Uplift Tree and Uplift Random Forest
  - Visualize a trained uplift classification tree model
  - Visualize an uplift tree in a trained uplift random forests
- Training and Validation Data
  - Visualize the validation tree: fill the trained uplift classification tree with validation (or testing) data, and show the statistics for both training data and validation data
- One Treatment Group and Multiple Treatment Groups
  - Visualize the case where there are one control group and one treatment group
  - Visualize the case where there are one control group and multiple treatment groups

5.4.2 Step 1 Load Modules

[1]:
```
from causalml.dataset import make_uplift_classification
from causalml.inference.tree import UpliftTreeClassifier, UpliftRandomForestClassifier
from causalml.inference.tree import uplift_tree_string, uplift_tree_plot
```

[2]:
```
import numpy as np
import pandas as pd
from IPython.display import Image
from sklearn.model_selection import train_test_split
```

5.4.3 One Control + One Treatment for Uplift Classification Tree

[3]:
```
# Data generation
df, x_names = make_uplift_classification()

# Rename features for easy interpretation of visualization
x_names_new = ['feature_%s'%(i) for i in range(len(x_names))]
rename_dict = {x_names[i]:x_names_new[i] for i in range(len(x_names))}
df = df.rename(columns=rename_dict)
x_names = x_names_new

df.head()

df = df[df['treatment_group_key'].isin(['control','treatment1'])]

# Look at the conversion rate and sample size in each group
df.pivot_table(values='conversion',
               index='treatment_group_key',
               aggfunc=[np.mean, np.size],
               margins=True)
```

[3]:
```
                          mean       size
                    conversion conversion
treatment_group_key
control                 0.5110       1000
treatment1              0.5140       1000
All                     0.5125       2000
```

[4]:
```
# Split data to training and testing samples for model validation (next section)
df_train, df_test = train_test_split(df, test_size=0.2, random_state=111)

# Train uplift tree
uplift_model = UpliftTreeClassifier(max_depth = 4, min_samples_leaf = 200, min_
↪samples_treatment = 50, n_reg = 100, evaluationFunction='KL', control_name='control
↪')

uplift_model.fit(df_train[x_names].values,
                 treatment=df_train['treatment_group_key'].values,
                 y=df_train['conversion'].values)
```

[5]:
```
# Print uplift tree as a string
result = uplift_tree_string(uplift_model.fitted_uplift_tree, x_names)
```

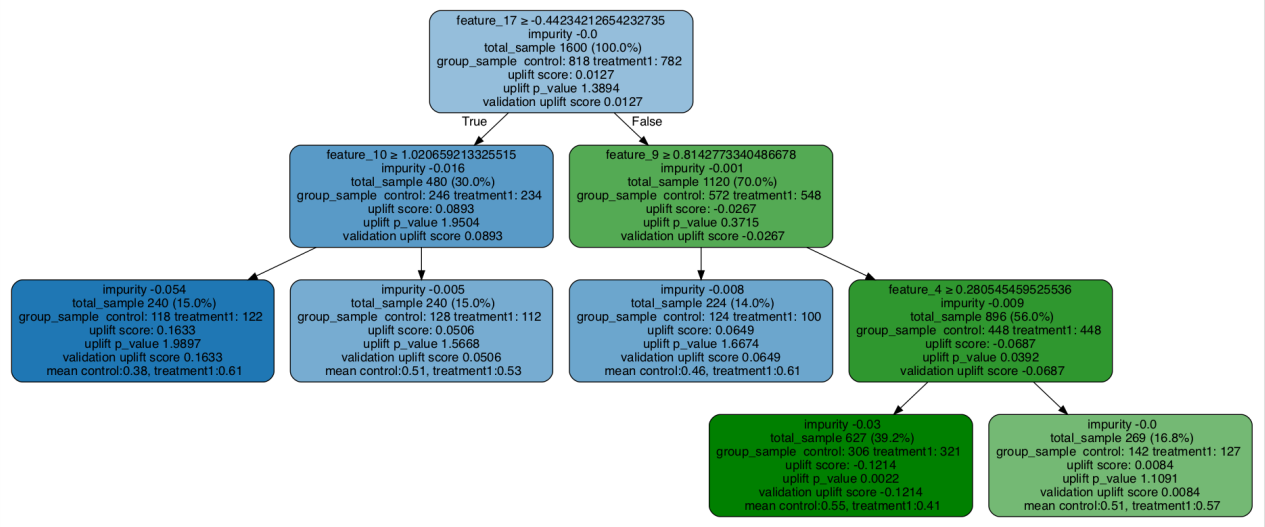
```
feature_17 >= -0.44234212654232735?
yes -> feature_10 >= 1.020659213325515?
                yes -> [0.3813559322033898, 0.6065573770491803]
                no  -> [0.5078125, 0.5267857142857143]
no  -> feature_9 >= 0.8142773340486678?
                yes -> [0.4596774193548387, 0.61]
                no  -> feature_4 >= 0.280545459525536?
                                yes -> [0.5522875816993464, 0.4143302180685358]
                                no  -> [0.5070422535211268, 0.5748031496062992]
```

Read the tree

- First line: node split condition
- impurity: the value for the loss function
- total_sample: total sample size in this node
- group_sample: sample size by treatment group
- uplift score: the treatment effect between treatment and control (when there are multiple treatment groups, this is the maximum of the treatment effects)
- uplift p_value: the p_value for the treatment effect
- validation uplift score: when validation data is filled in the tree, this reflects the uplift score based on the - validation data. It can be compared with the uplift score (for training data) to check if there are over-fitting issue.

[6]:
```
# Plot uplift tree
graph = uplift_tree_plot(uplift_model.fitted_uplift_tree,x_names)
Image(graph.create_png())
```

[6]:



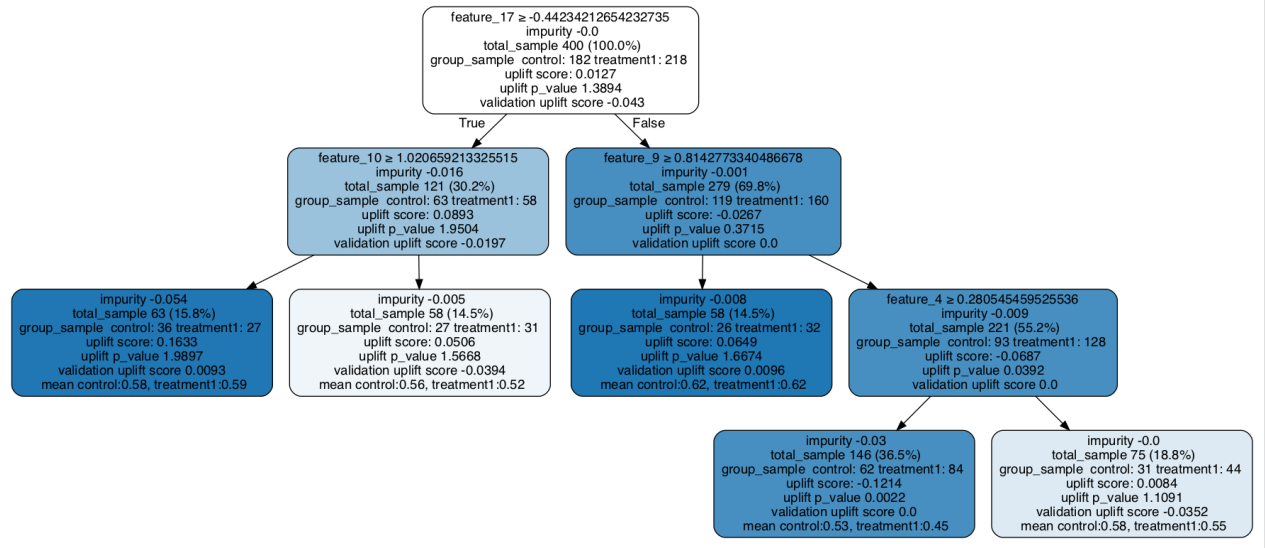
Visualize Validation Tree: One Control + One Treatment for Uplift Classification Tree

Note the validation uplift score will update.

[7]:

```
### Fill the trained tree with testing data set
# The uplift score based on testing dataset is shown as validation uplift score in␣
↪the tree nodes
uplift_model.fill(X=df_test[x_names].values, treatment=df_test['treatment_group_key'].
↪values, y=df_test['conversion'].values)

# Plot uplift tree
graph = uplift_tree_plot(uplift_model.fitted_uplift_tree,x_names)
Image(graph.create_png())
```

[7]:



Visualize a Tree in Random Forest

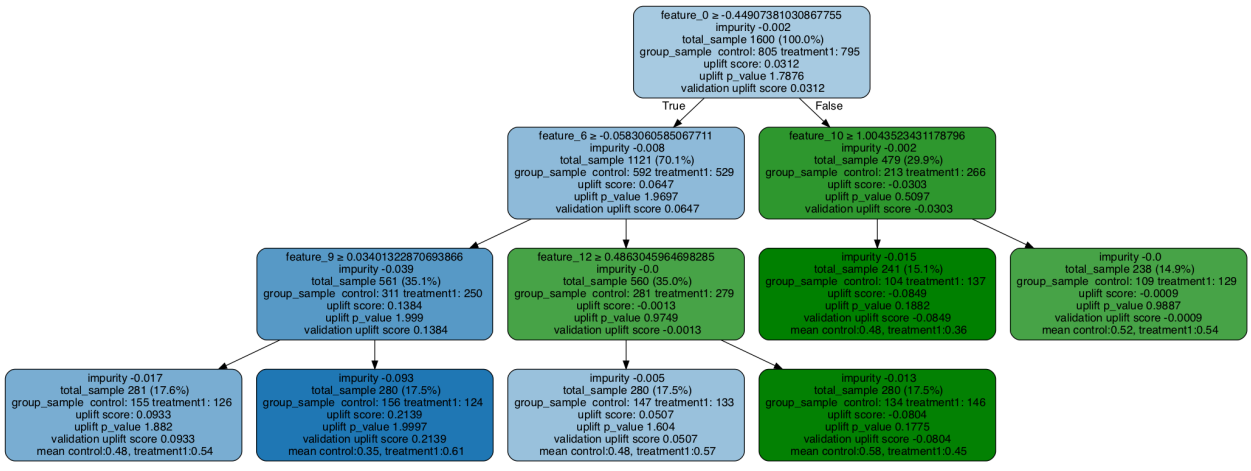
[8]:
```
# Split data to training and testing samples for model validation (next section)
df_train, df_test = train_test_split(df, test_size=0.2, random_state=111)

# Train uplift tree
uplift_model = UpliftRandomForestClassifier(n_estimators=5, max_depth = 5, min_
↪samples_leaf = 200, min_samples_treatment = 50, n_reg = 100, evaluationFunction='KL
↪', control_name='control')

uplift_model.fit(df_train[x_names].values,
                 treatment=df_train['treatment_group_key'].values,
                 y=df_train['conversion'].values)
```

[9]:
```
# Specify a tree in the random forest (the index can be any integer from 0 to n_
↪estimators-1)
uplift_tree = uplift_model.uplift_forest[0]
# Print uplift tree as a string
result = uplift_tree_string(uplift_tree.fitted_uplift_tree, x_names)
```

```
feature_0 >= -0.44907381030867755?
yes -> feature_6 >= -0.0583060585067711?
               yes -> feature_9 >= 0.03401322870693866?
                               yes -> [0.4774193548387097, 0.5396825396825397]
                               no  -> [0.34615384615384615, 0.6129032258064516]
               no  -> feature_12 >= 0.4863045964698285?
                               yes -> [0.48299319727891155, 0.5714285714285714]
                               no  -> [0.582089552238806, 0.4452054794520548]
no  -> feature_10 >= 1.0043523431178796?
               yes -> [0.4807692307692308, 0.35766423357664234]
               no  -> [0.5229357798165137, 0.5426356589147286]
```

[10]:
```
# Plot uplift tree
graph = uplift_tree_plot(uplift_tree.fitted_uplift_tree,x_names)
Image(graph.create_png())
```

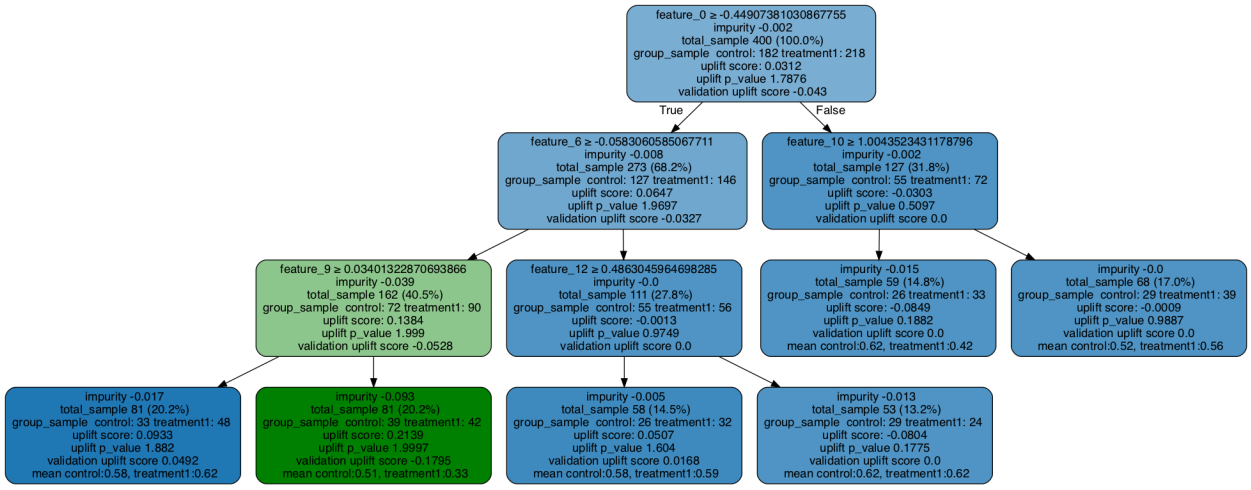
[10]:



Fill the tree with validation data

[11]:
```
### Fill the trained tree with testing data set
# The uplift score based on testing dataset is shown as validation uplift score in␣
↪the tree nodes
uplift_tree.fill(X=df_test[x_names].values, treatment=df_test['treatment_group_key'].
↪values, y=df_test['conversion'].values)

# Plot uplift tree
graph = uplift_tree_plot(uplift_tree.fitted_uplift_tree,x_names)
Image(graph.create_png())
```

[11]:



5.4.4 One Control + Multiple Treatments

[12]:
```
# Data generation
df, x_names = make_uplift_classification()
# Look at the conversion rate and sample size in each group
df.pivot_table(values='conversion',
               index='treatment_group_key',
               aggfunc=[np.mean, np.size],
               margins=True)
```

[12]:

	mean	size
	conversion	conversion
treatment_group_key		
control	0.511	1000
treatment1	0.514	1000
treatment2	0.559	1000
treatment3	0.600	1000
All	0.546	4000

[13]:
```
# Split data to training and testing samples for model validation (next section)
df_train, df_test = train_test_split(df, test_size=0.2, random_state=111)

# Train uplift tree
uplift_model = UpliftTreeClassifier(max_depth = 3, min_samples_leaf = 200, min_
↪samples_treatment = 50, n_reg = 100, evaluationFunction='KL', control_name='control
↪')
```

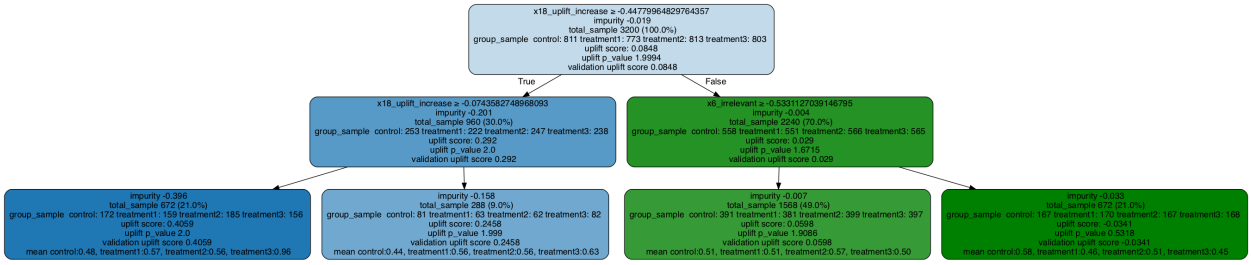
(continues on next page)

(continued from previous page)

```
uplift_model.fit(df_train[x_names].values,
                 treatment=df_train['treatment_group_key'].values,
                 y=df_train['conversion'].values)
```

[14]:
```
# Plot uplift tree
# The uplift score represents the best uplift score among all treatment effects
graph = uplift_tree_plot(uplift_model.fitted_uplift_tree,x_names)
Image(graph.create_png())
```

[14]:



[15]:
```
# Save the graph as pdf
graph.write_pdf("tbc.pdf")
# Save the graph as png
graph.write_png("tbc.png")
```

[15]:
```
True
```

5.5 Model Interpretation with Feature Importance and SHAP Values

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

from causalml.inference.meta import BaseSRegressor, BaseTRegressor, BaseXRegressor,␣
↪BaseRRegressor
from causalml.inference.tree import UpliftTreeClassifier, UpliftRandomForestClassifier
from causalml.dataset.regression import synthetic_data
from sklearn.linear_model import LinearRegression

import shap
import matplotlib.pyplot as plt

import time
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split

import os
import warnings
warnings.filterwarnings('ignore')
```

(continues on next page)

(continued from previous page)

```
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'  # for lightgbm to work

%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

```
The sklearn.utils.testing module is  deprecated in version 0.22 and will be removed␣
↪in version 0.24. The corresponding classes / functions should instead be imported␣
↪from sklearn.utils. Anything that cannot be imported from sklearn.utils is now part␣
↪of the private API.
```

[2]:
```
plt.style.use('fivethirtyeight')
```

[4]:
```
n_features = 25
n_samples = 10000
y, X, w, tau, b, e = synthetic_data(mode=1, n=n_samples, p=n_features, sigma=0.5)
```

[5]:
```
w_multi = np.array(['treatment_A' if x==1 else 'control' for x in w])
e_multi = {'treatment_A': e}
```

[6]:
```
feature_names = ['stars', 'tiger', 'merciful', 'quixotic', 'fireman', 'dependent',
                 'shelf', 'touch', 'barbarous', 'clammy', 'playground', 'rain', 'offer
↪',
                 'cute', 'future', 'damp', 'nonchalant', 'change', 'rigid',
↪'sweltering',
                 'eight', 'wrap', 'lethal', 'adhesive', 'lip']  # specify feature␣
↪names

model_tau = LGBMRegressor(importance_type='gain')  # specify model for model_tau
```

5.5.1 S Learner

[7]:
```
base_algo = LGBMRegressor()
# base_algo = XGBRegressor()
# base_algo = RandomForestRegressor()
# base_algo = LinearRegression()

slearner = BaseSRegressor(base_algo, control_name='control')
slearner.estimate_ate(X, w_multi, y)
```

[7]:
```
array([0.56829617])
```

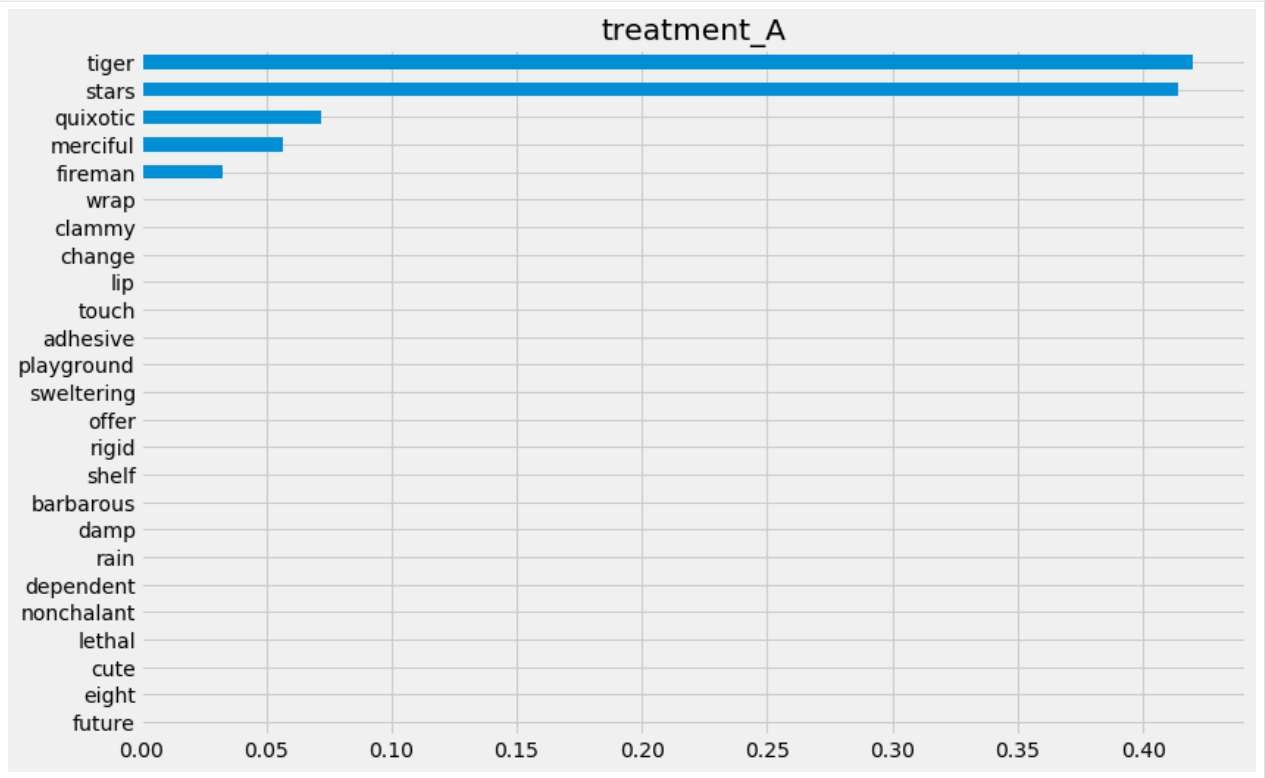
[8]:
```
slearner_tau = slearner.fit_predict(X, w_multi, y)
```

Feature Importance (method = `auto`)

[9]:
```
slearner.get_importance(X=X,
                        tau=slearner_tau,
                        normalize=True,
                        method='auto',
                        features=feature_names)
```

[9]:
```
{'treatment_A': tiger         0.419967
 stars          0.413894
 quixotic       0.072241
 merciful       0.056910
 fireman        0.032434
 wrap           0.000407
 clammy         0.000383
 change         0.000306
 lip            0.000299
 touch          0.000281
 adhesive       0.000253
 playground     0.000235
 sweltering     0.000233
 offer          0.000232
 rigid          0.000217
 shelf          0.000208
 barbarous      0.000192
 damp           0.000192
 rain           0.000184
 dependent      0.000180
 nonchalant     0.000171
 lethal         0.000159
 cute           0.000154
 eight          0.000138
 future         0.000131
 dtype: float64}
```

[10]:
```
slearner.plot_importance(X=X,
                         tau=slearner_tau,
                         normalize=True,
                         method='auto',
                         features=feature_names)
```



Feature Importance (method = `permutation`)

[11]:
```
slearner.get_importance(X=X,
                        tau=slearner_tau,
                        method='permutation',
                        features=feature_names,
                        random_state=42)
```

[11]:
```
{'treatment_A': tiger         0.963026
 stars          0.869475
 quixotic       0.163553
 merciful       0.101724
 fireman        0.065210
 touch          0.000389
 clammy         0.000370
 adhesive       0.000180
 wrap           0.000150
 sweltering     0.000144
 change         0.000104
 lethal         0.000095
 damp           0.000071
 shelf          0.000040
 rigid          0.000028
 barbarous      0.000026
 playground     0.000021
 nonchalant    -0.000014
 cute          -0.000020
 rain          -0.000034
 offer         -0.000046
```

(continues on next page)

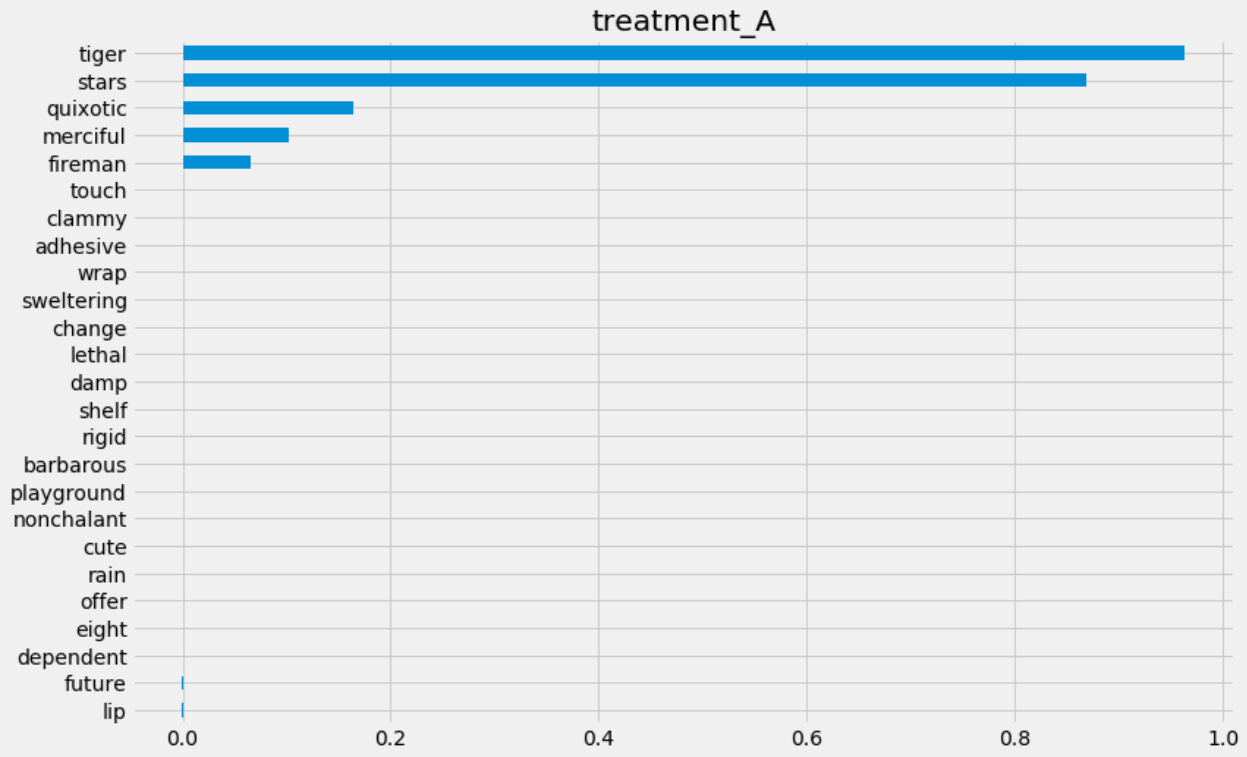
(continued from previous page)

```
 eight         -0.000054
 dependent     -0.000060
 future        -0.000091
 lip           -0.000097
 dtype: float64}
```

[12]:
```
start_time = time.time()

slearner.get_importance(X=X,
                        tau=slearner_tau,
                        method='permutation',
                        features=feature_names,
                        random_state=42)

print("Elapsed time: %s seconds" % (time.time() - start_time))
```

```
Elapsed time: 37.788124799728394 seconds
```

[13]:
```
slearner.plot_importance(X=X,
                         tau=slearner_tau,
                         method='permutation',
                         features=feature_names,
                         random_state=42)
```



Feature Importance (`sklearn.inspection.permutation_importance`)

[14]:
```
start_time = time.time()

X_train, X_test, y_train, y_test = train_test_split(X, slearner_tau, test_size=0.3,␣
↪random_state=42)
model_tau_fit = model_tau.fit(X_train, y_train)

perm_imp_test = permutation_importance(
    estimator=model_tau_fit,
    X=X_test,
    y=y_test,
    random_state=42).importances_mean
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)

print("Elapsed time: %s seconds" % (time.time() - start_time))
```

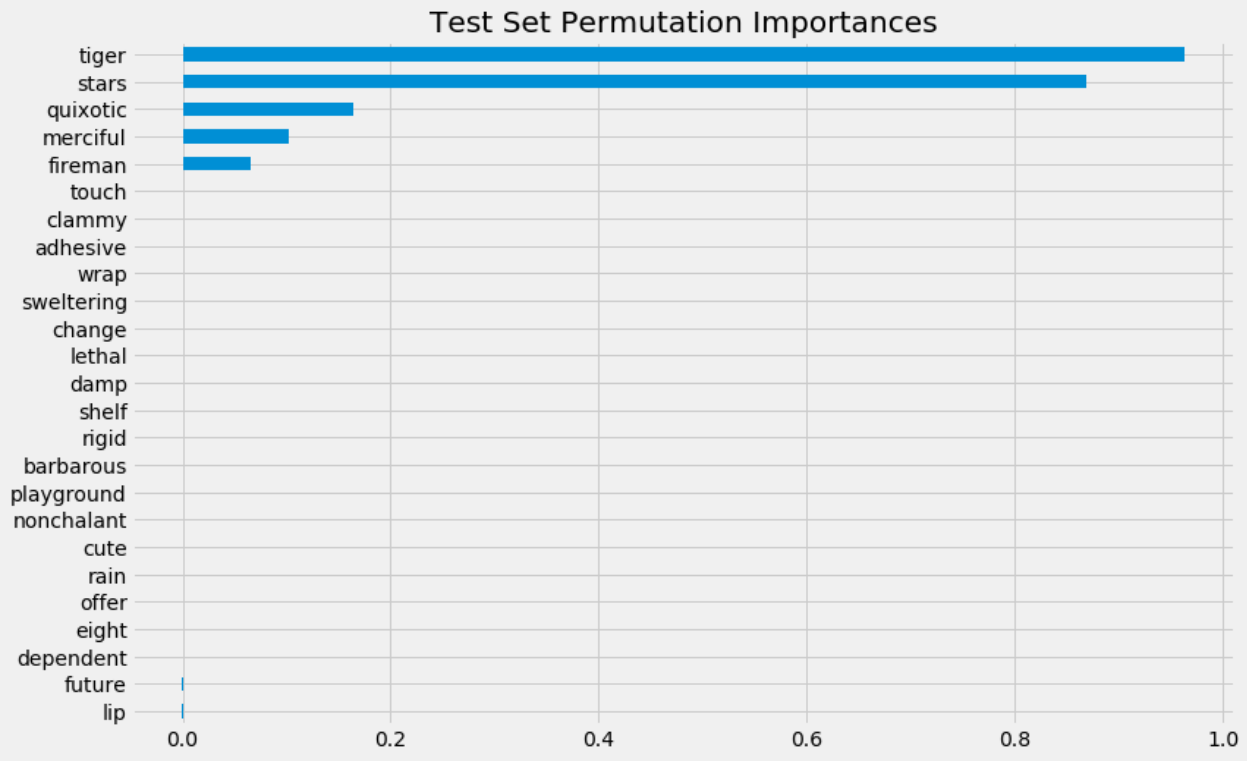
```
Elapsed time: 14.822510957717896 seconds
```

[15]:
```
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)
```

[15]:
```
tiger          0.963026
stars          0.869475
quixotic       0.163553
merciful       0.101724
fireman        0.065210
touch          0.000389
clammy         0.000370
adhesive       0.000180
wrap           0.000150
sweltering     0.000144
change         0.000104
lethal         0.000095
damp           0.000071
shelf          0.000040
rigid          0.000028
barbarous      0.000026
playground     0.000021
nonchalant    -0.000014
cute          -0.000020
rain          -0.000034
offer         -0.000046
eight         -0.000054
dependent     -0.000060
future        -0.000091
lip           -0.000097
dtype: float64
```

[16]:
```
pd.Series(perm_imp_test, feature_names).sort_values().plot(kind='barh', figsize=(12,␣
↪8))
plt.title('Test Set Permutation Importances')
```

[16]:
```
Text(0.5, 1.0, 'Test Set Permutation Importances')
```



[17]:
```
perm_imp_train = permutation_importance(
    estimator=model_tau_fit,
    X=X_train,
    y=y_train,
    random_state=42).importances_mean
pd.Series(perm_imp_train, feature_names).sort_values(ascending=False)
```

[17]:
```
tiger          0.912573
stars          0.871412
quixotic       0.164476
merciful       0.104541
fireman        0.064374
lip            0.001931
lethal         0.001112
future         0.001104
clammy         0.000977
touch          0.000935
damp           0.000868
wrap           0.000868
change         0.000824
sweltering     0.000806
adhesive       0.000732
offer          0.000690
rain           0.000652
barbarous      0.000525
rigid          0.000492
eight          0.000458
dependent      0.000438
cute           0.000419
nonchalant     0.000405
```

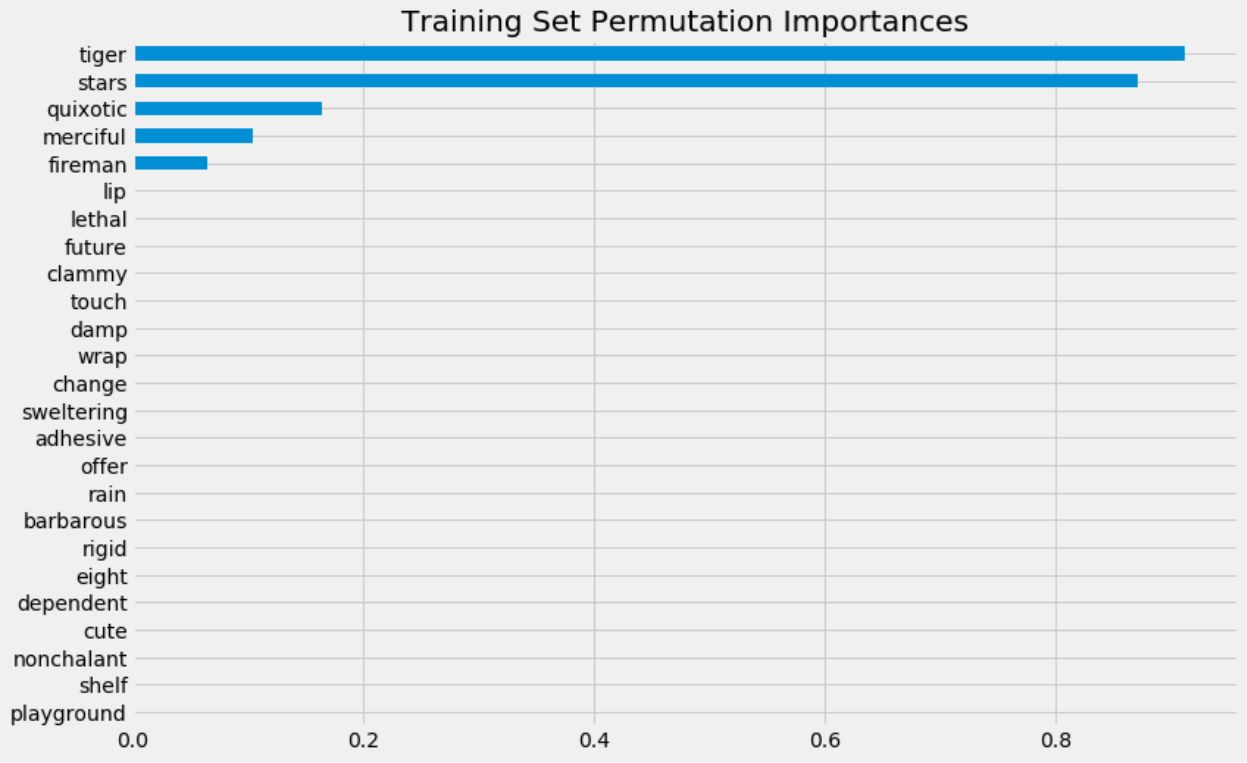
(continues on next page)

(continued from previous page)

```
shelf          0.000400
playground     0.000354
dtype: float64
```

[18]:
```
pd.Series(perm_imp_train, feature_names).sort_values().plot(kind='barh', figsize=(12,
↪8))
plt.title('Training Set Permutation Importances')
```

[18]:
```
Text(0.5, 1.0, 'Training Set Permutation Importances')
```



Shapley Values

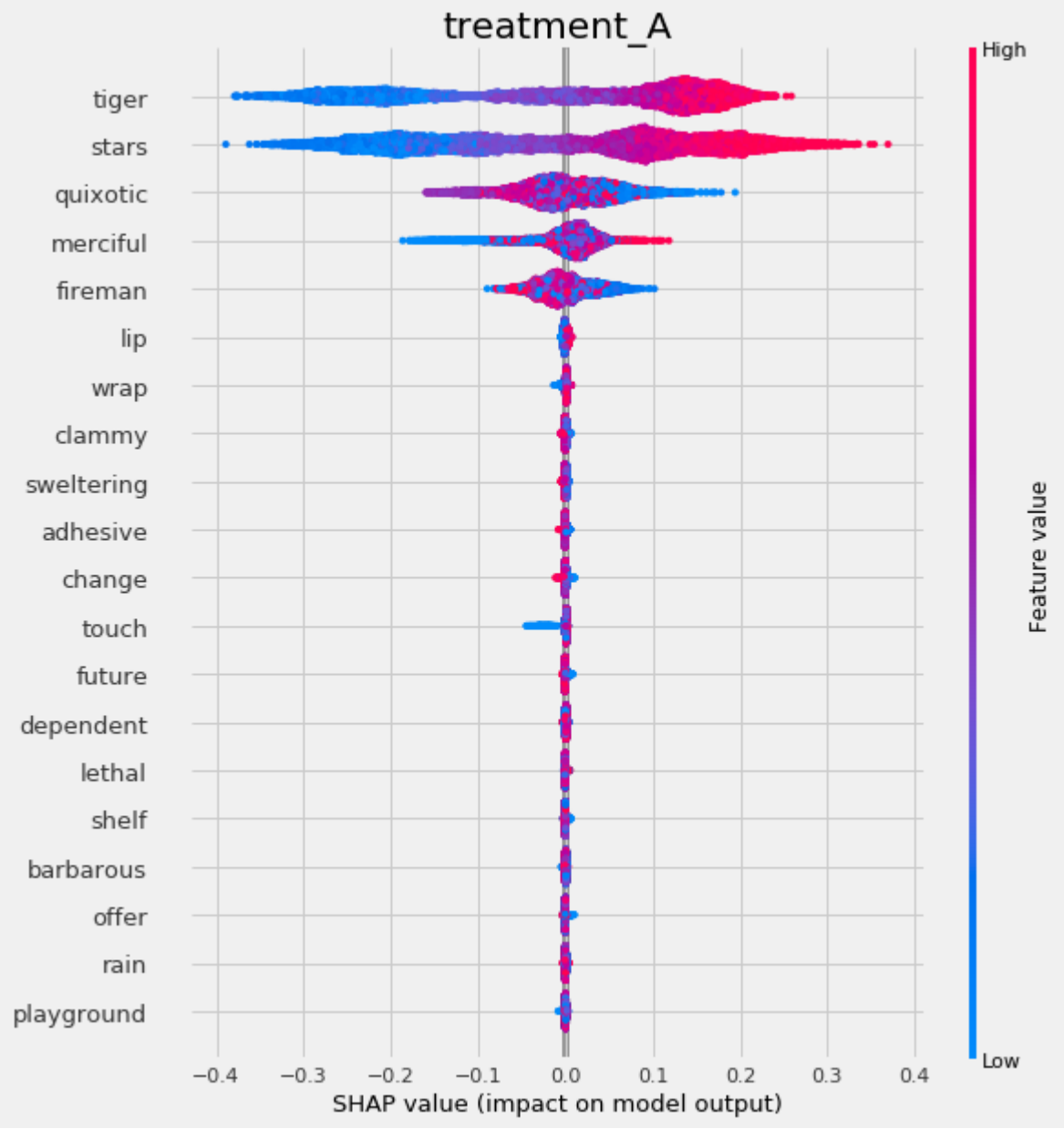
[19]:
```
shap_slearner = slearner.get_shap_values(X=X, tau=slearner_tau)
shap_slearner
```

[19]:
```
{'treatment_A': array([[ 4.10078017e-02, -3.44817262e-02, -5.43404776e-03, …,
         -4.74545331e-04, -1.51053586e-03,  3.90095411e-03],
        [-7.48726271e-02,  5.93780768e-02, -1.41883322e-02, …,
          7.46974369e-04, -4.48063259e-04, -1.89122689e-03],
        [ 8.76198804e-02, -1.16128067e-02,  4.81884470e-03, …,
         -4.35674464e-04,  1.93345867e-03,  3.70921426e-03],
        …,
        [ 1.97191229e-01,  1.04795472e-01,  6.66297704e-03, …,
         -4.94229406e-04,  1.23164980e-03, -1.94624556e-03],
        [-2.51788728e-01,  1.66874562e-02,  3.63517776e-02, …,
         -4.77522143e-04,  1.13078435e-03,  1.69601440e-03],
        [-3.20539506e-02,  2.13426166e-01, -7.80250031e-02, …,
         -1.84885894e-04,  1.69764654e-04, -3.78072076e-03]])}
```

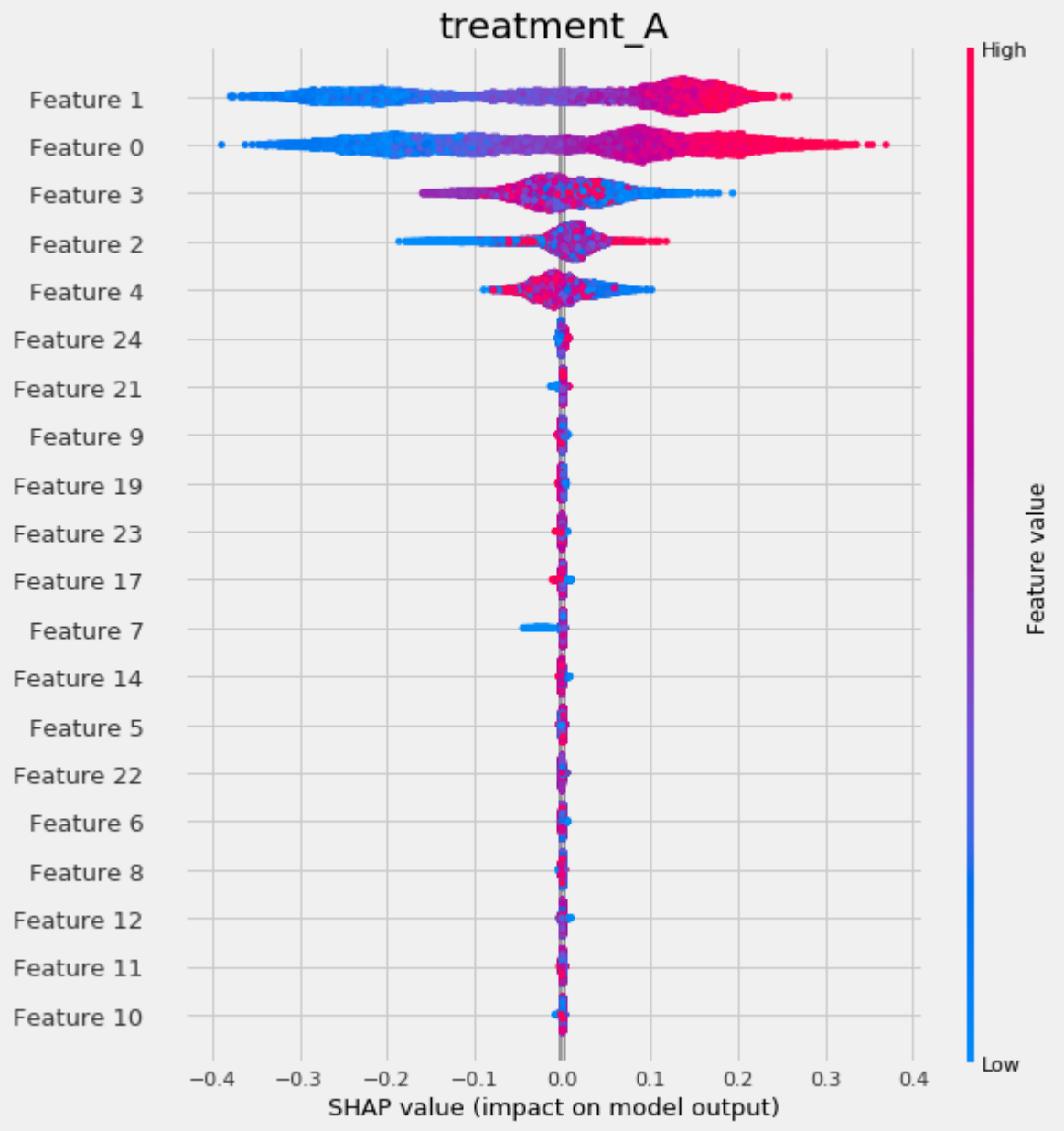
[20]:
```
np.mean(np.abs(shap_slearner['treatment_A']),axis=0)
```

[20]:
```
array([0.13950704, 0.14386761, 0.02545777, 0.04069884, 0.02323508,
       0.00065427, 0.00049449, 0.00085658, 0.00047613, 0.00106313,
       0.00039083, 0.00039238, 0.0004238 , 0.00033561, 0.00080356,
       0.00035307, 0.00024251, 0.0008808 , 0.00035521, 0.00104124,
       0.00022112, 0.00119311, 0.00060483, 0.00089334, 0.00178355])
```

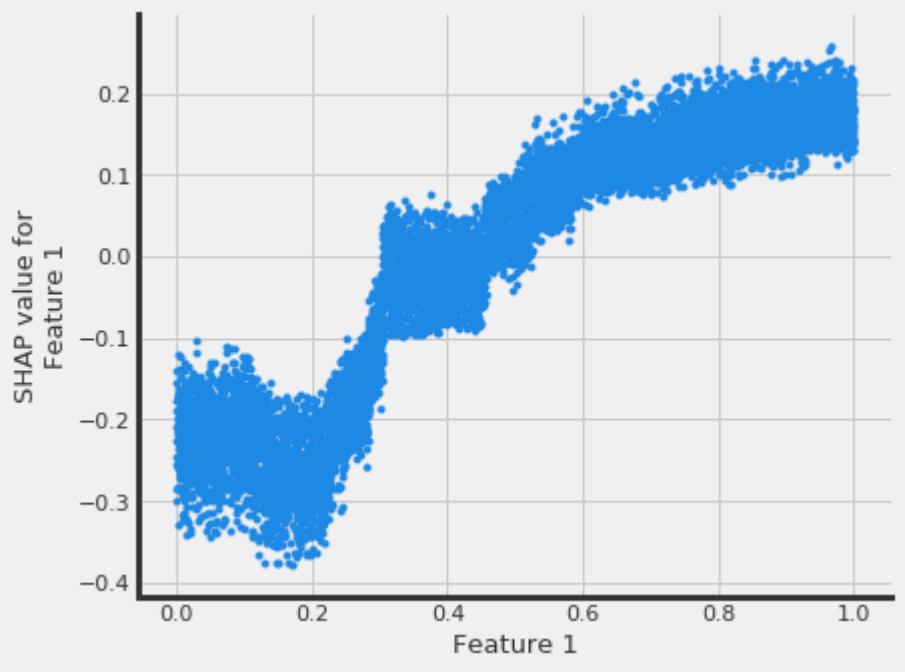
[21]:
```
# Plot shap values without specifying shap_dict
slearner.plot_shap_values(X=X, tau=slearner_tau, features=feature_names)
```



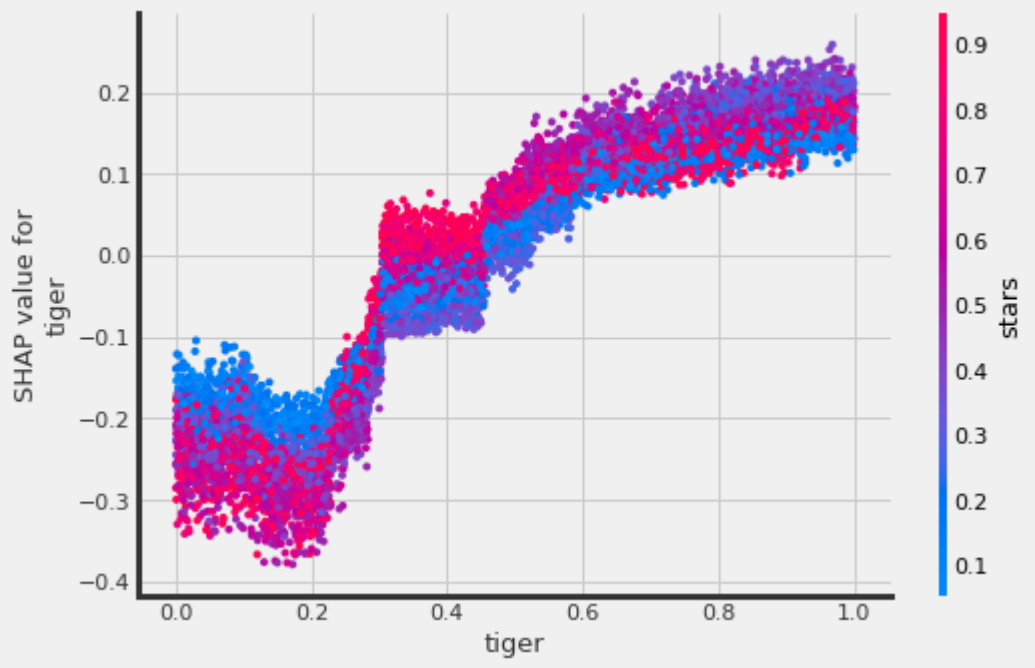
[22]:
```
# Plot shap values WITH specifying shap_dict
slearner.plot_shap_values(X=X, shap_dict=shap_slearner)
```



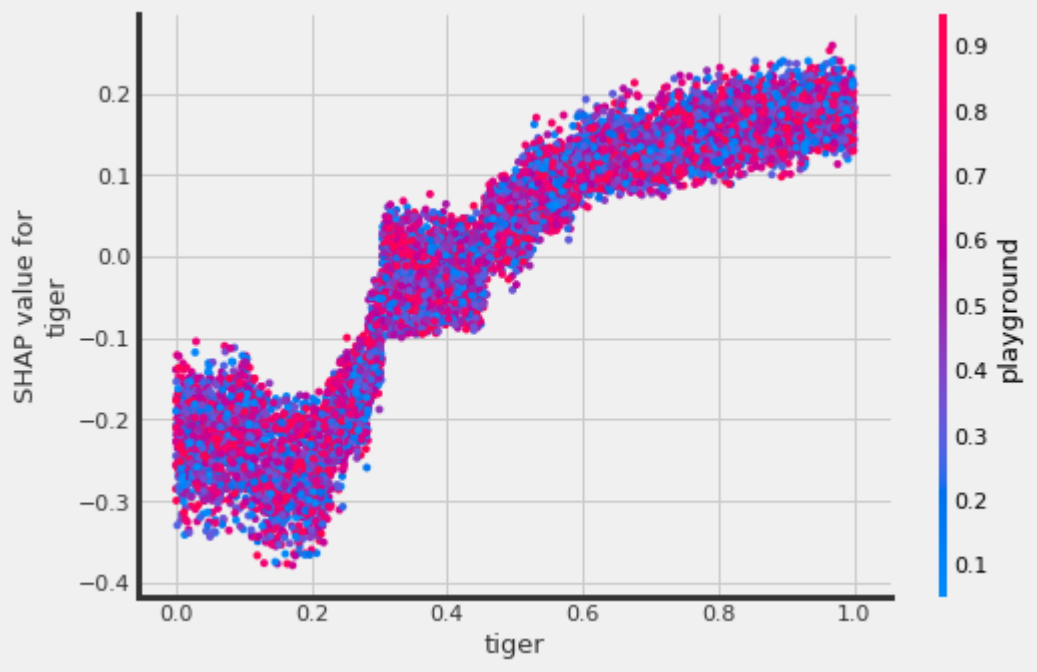
[23]:
```
# interaction_idx set to None (no color coding for interaction effects)
slearner.plot_shap_dependence(treatment_group='treatment_A',
                              feature_idx=1,
                              X=X,
                              tau=slearner_tau,
                              interaction_idx=None,
                              shap_dict=shap_slearner)
```



[24]:
```
# interaction_idx set to 'auto' (searches for feature with greatest approximate␣
↪interaction)
# specify feature names
slearner.plot_shap_dependence(treatment_group='treatment_A',
                              feature_idx='tiger',
                              X=X,
                              tau=slearner_tau,
                              interaction_idx='auto',
                              shap_dict=shap_slearner,
                              features=feature_names)
```



[25]:
```
# interaction_idx set to specific index
slearner.plot_shap_dependence(treatment_group='treatment_A',
                              feature_idx=1,
                              X=X,
                              tau=slearner_tau,
                              interaction_idx=10,
                              shap_dict=shap_slearner,
                              features=feature_names)
```



5.5.2 T Learner

[26]:
```
tlearner = BaseTRegressor(LGBMRegressor(), control_name='control')
tlearner.estimate_ate(X, w_multi, y)
```

[26]:
```
(array([0.5526554]), array([0.53763828]), array([0.56767251]))
```

[27]:
```
tlearner_tau = tlearner.fit_predict(X, w_multi, y)
```

Feature Importance (method = `auto`)

[28]:
```
tlearner.get_importance(X=X,
                        tau=tlearner_tau,
                        normalize=True,
                        method='auto',
                        features=feature_names)
```

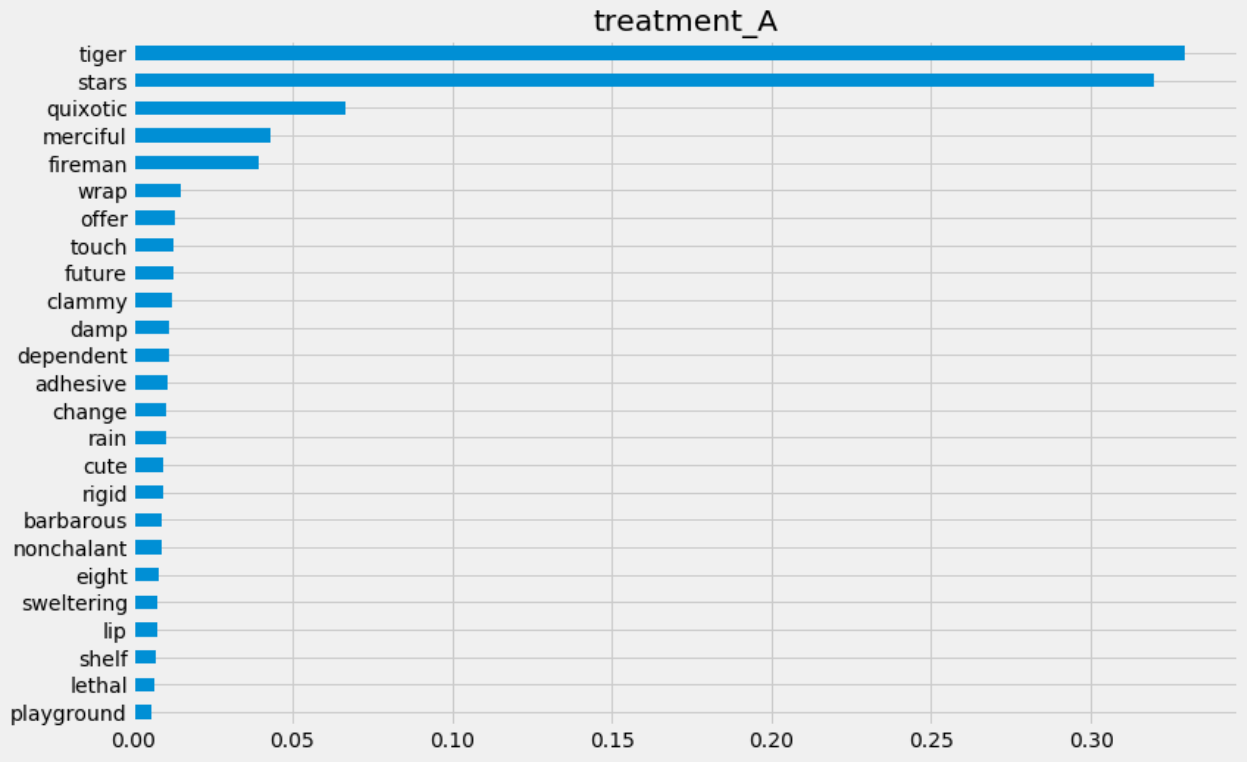
[28]:
```
{'treatment_A': tiger         0.329522
 stars         0.319934
 quixotic      0.066615
 merciful      0.043139
```

(continues on next page)

(continued from previous page)

```
fireman        0.039397
wrap           0.015105
offer          0.013031
touch          0.012786
future         0.012633
clammy         0.012428
damp           0.011408
dependent      0.011313
adhesive       0.010930
change         0.010475
rain           0.010393
cute           0.009622
rigid          0.009564
barbarous      0.009170
nonchalant     0.009108
eight          0.008167
sweltering     0.007606
lip            0.007596
shelf          0.007189
lethal         0.006894
playground     0.005973
dtype: float64}
```

[29]:
```
tlearner.plot_importance(X=X,
                         tau=tlearner_tau,
                         normalize=True,
                         method='auto',
                         features=feature_names)
```

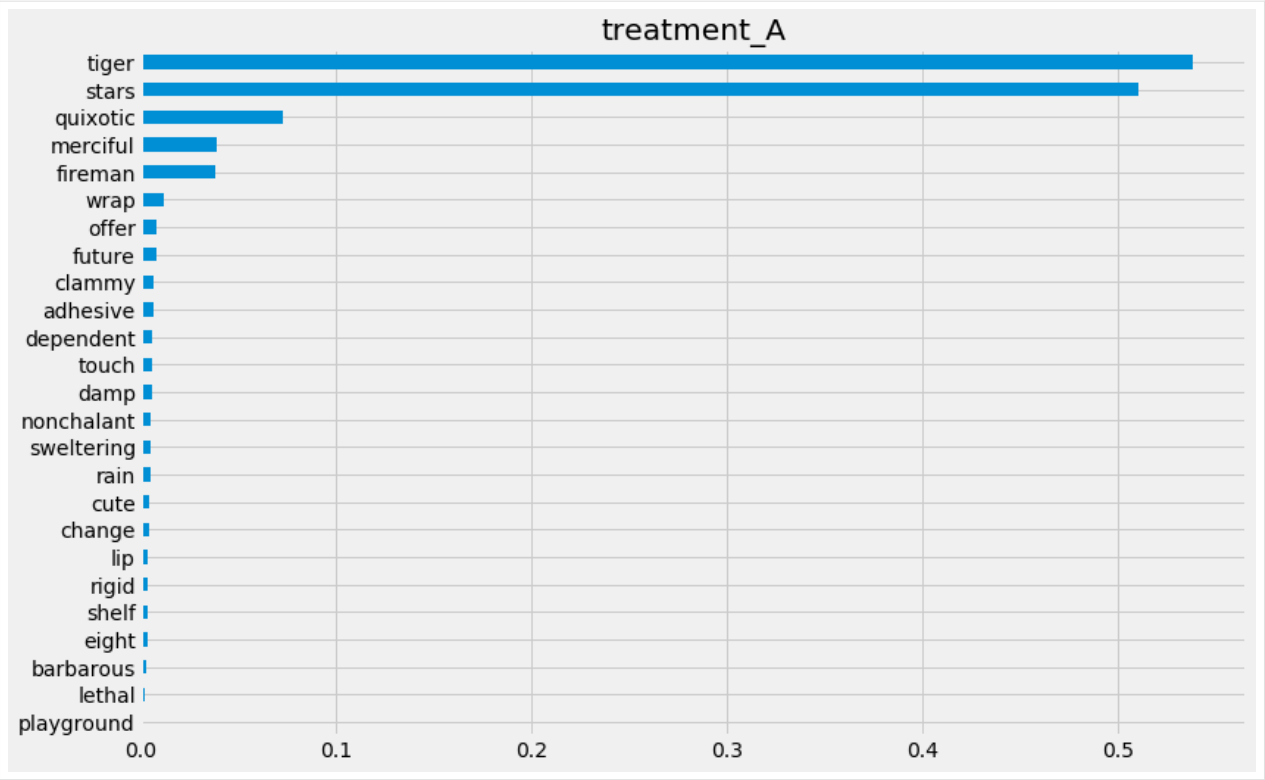


Feature Importance (method = `permutation`)

[30]:
```
tlearner.get_importance(X=X,
                        tau=tlearner_tau,
                        method='permutation',
                        features=feature_names,
                        random_state=42)
```

[30]:
```
{'treatment_A': tiger         0.538136
 stars          0.510393
 quixotic       0.072974
 merciful       0.038492
 fireman        0.037728
 wrap           0.012041
 offer          0.008361
 future         0.007785
 clammy         0.006456
 adhesive       0.006216
 dependent      0.006018
 touch          0.005865
 damp           0.005544
 nonchalant     0.005190
 sweltering     0.005030
 rain           0.004813
 cute           0.004293
 change         0.004053
 lip            0.003858
 rigid          0.003853
 shelf          0.003634
 eight          0.003334
 barbarous      0.002836
 lethal         0.002367
 playground     0.000314
 dtype: float64}
```

[31]:
```
tlearner.plot_importance(X=X,
                         tau=tlearner_tau,
                         method='permutation',
                         features=feature_names,
                         random_state=42)
```



Feature Importance (`sklearn.inspection.permutation_importance`)

[32]:
```
start_time = time.time()

X_train, X_test, y_train, y_test = train_test_split(X, tlearner_tau, test_size=0.3,␣
↪random_state=42)
model_tau_fit = model_tau.fit(X_train, y_train)

perm_imp_test = permutation_importance(
    estimator=model_tau_fit,
    X=X_test,
    y=y_test,
    random_state=42).importances_mean
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)

print("Elapsed time: %s seconds" % (time.time() - start_time))
```

```
Elapsed time: 16.60052752494812 seconds
```

[33]:
```
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)
```

[33]:
```
tiger          0.538136
stars          0.510393
quixotic       0.072974
merciful       0.038492
fireman        0.037728
wrap           0.012041
offer          0.008361
future         0.007785
```

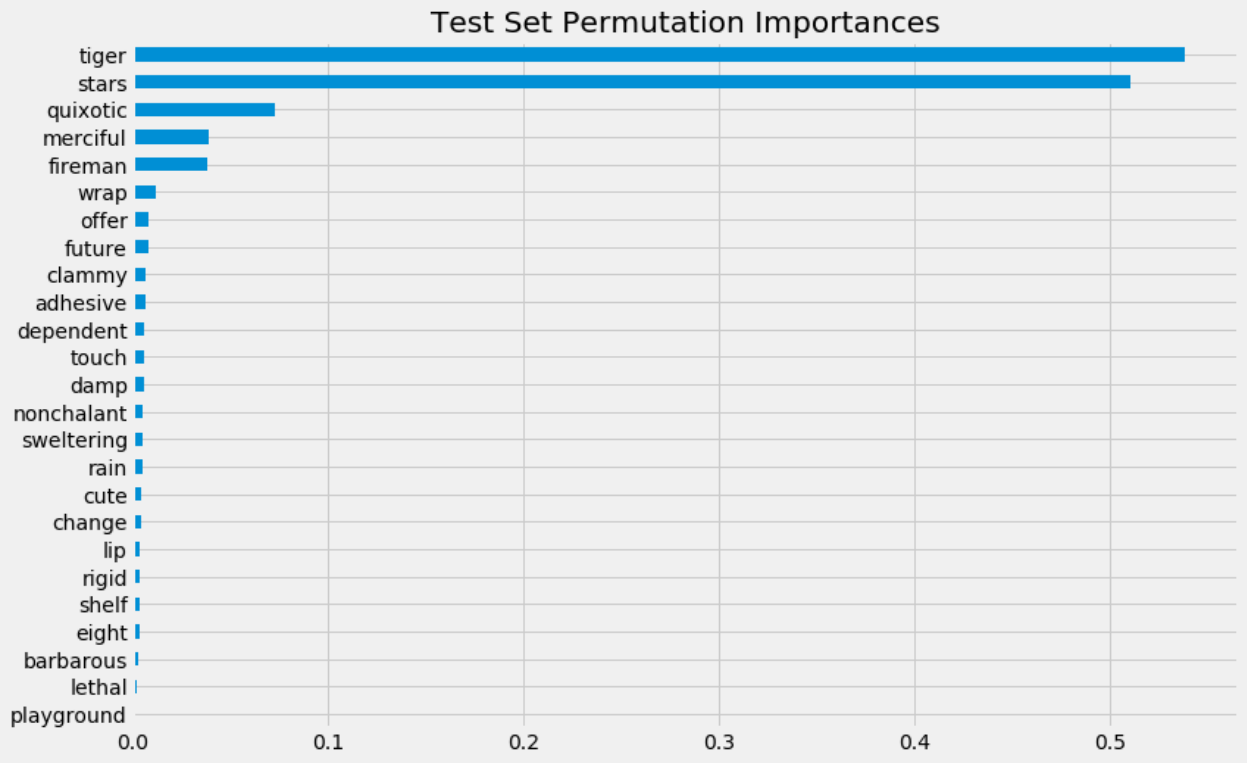
(continues on next page)

(continued from previous page)

```
clammy          0.006456
adhesive        0.006216
dependent       0.006018
touch           0.005865
damp            0.005544
nonchalant      0.005190
sweltering      0.005030
rain            0.004813
cute            0.004293
change          0.004053
lip             0.003858
rigid           0.003853
shelf           0.003634
eight           0.003334
barbarous       0.002836
lethal          0.002367
playground      0.000314
dtype: float64
```

[34]:
```
pd.Series(perm_imp_test, feature_names).sort_values().plot(kind='barh', figsize=(12,␣
↪8))
plt.title('Test Set Permutation Importances')
```

[34]:
```
Text(0.5, 1.0, 'Test Set Permutation Importances')
```

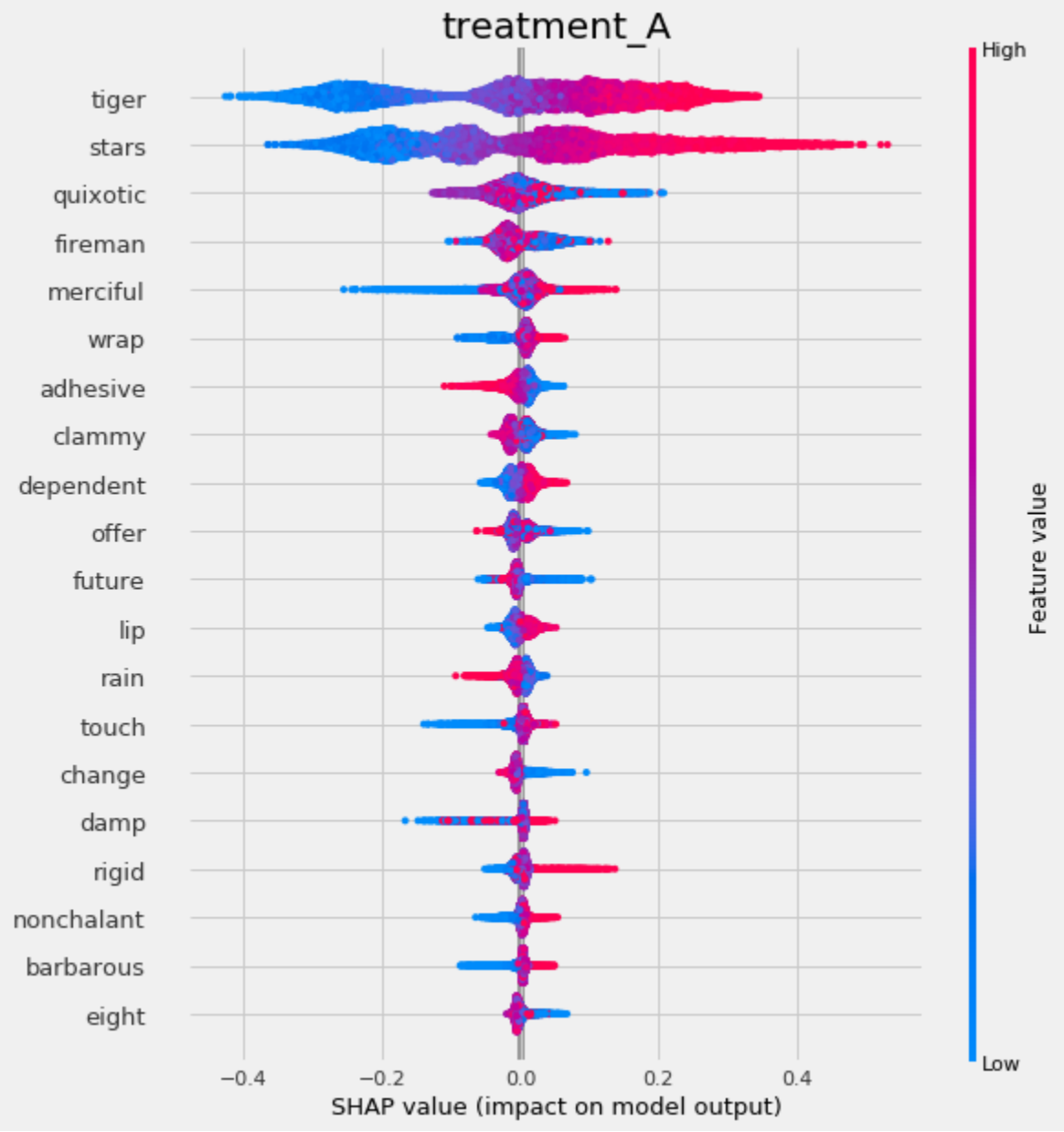


Shapley Values

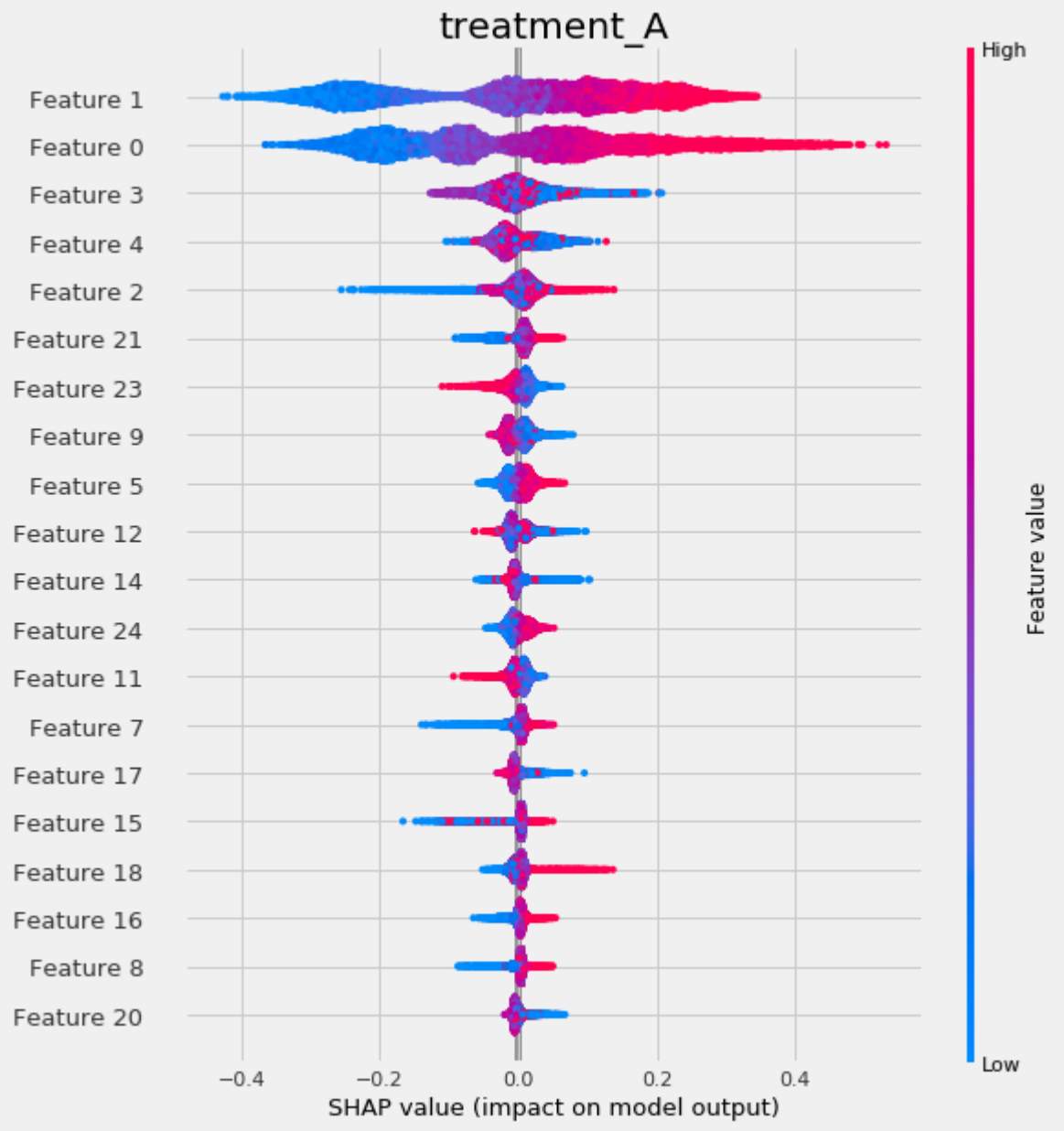
[35]:
```
shap_tlearner = tlearner.get_shap_values(X=X, tau=tlearner_tau)
shap_tlearner
```

[35]:
```
{'treatment_A': array([[ 0.03170431, -0.02653401, -0.04181033, …, -0.00420727,
         -0.00209201,  0.0116853 ],
        [-0.09827316,  0.02655629, -0.02626074, …, -0.00074733,
          0.00907333,  0.0007965 ],
        [ 0.05350246, -0.01205391,  0.00787274, …,  0.00092083,
          0.01316705,  0.01219494],
        …,
        [ 0.29451126,  0.07890184, -0.00674396, …, -0.003012  ,
          0.01859159, -0.0096335 ],
        [-0.2375042 , -0.00485028, -0.00101973, …,  0.00079727,
          0.01883852,  0.00980794],
        [-0.05199902,  0.1479534 , -0.09951596, …,  0.01449447,
          0.01699256, -0.01394553]])}
```

[36]:
```
# Plot shap values without specifying shap_dict
tlearner.plot_shap_values(X=X, tau=tlearner_tau, features=feature_names)
```



[37]:
```
# Plot shap values WITH specifying shap_dict
tlearner.plot_shap_values(X=X, shap_dict=shap_tlearner)
```



5.5.3 X Learner

[38]:
```
xlearner = BaseXRegressor(LGBMRegressor(), control_name='control')
xlearner.estimate_ate(X, w_multi, y, p=e_multi)
```

[38]:
```
(array([0.51497605]), array([0.50079629]), array([0.52915581]))
```

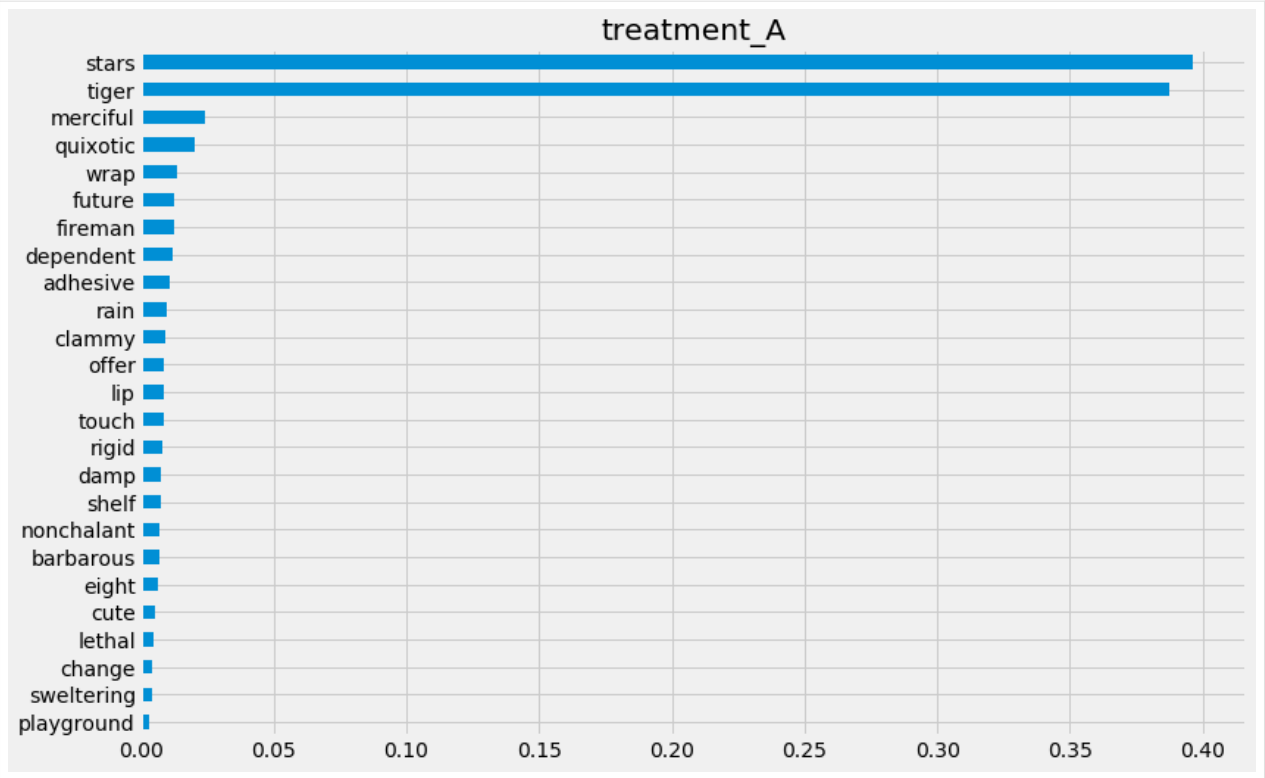
[39]:
```
xlearner_tau = xlearner.predict(X, w_multi, y, p=e_multi)
```

Feature Importance (method = `auto`)

[40]:
```
xlearner.get_importance(X=X,
                        tau=xlearner_tau,
                        normalize=True,
                        method='auto',
                        features=feature_names)
```

[40]:
```
{'treatment_A': stars         0.396410
 tiger          0.387525
 merciful       0.023992
 quixotic       0.020416
 wrap           0.013560
 future         0.012550
 fireman        0.012385
 dependent      0.012259
 adhesive       0.010841
 rain           0.009530
 clammy         0.009327
 offer          0.008513
 lip            0.008454
 touch          0.008432
 rigid          0.008281
 damp           0.007743
 shelf          0.007601
 nonchalant     0.007137
 barbarous      0.006748
 eight          0.006329
 cute           0.005616
 lethal         0.004837
 change         0.004130
 sweltering     0.004092
 playground     0.003290
 dtype: float64}
```

[41]:
```
xlearner.plot_importance(X=X,
                         tau=xlearner_tau,
                         normalize=True,
                         method='auto',
                         features=feature_names)
```



Feature Importance (method = `permutation`)

[42]:
```
xlearner.get_importance(X=X,
                        tau=xlearner_tau,
                        method='permutation',
                        features=feature_names,
                        random_state=42)
```

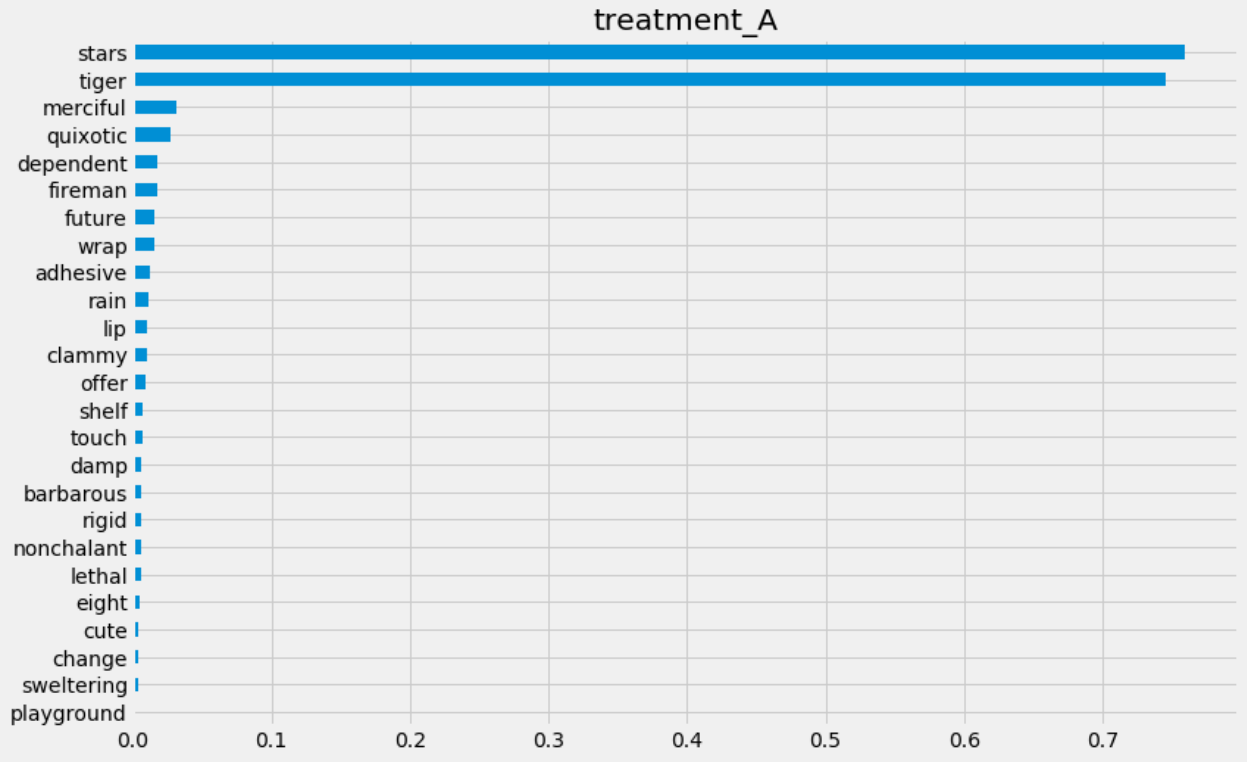
[42]:
```
{'treatment_A': stars         0.759553
 tiger          0.745122
 merciful       0.031355
 quixotic       0.027350
 dependent      0.018033
 fireman        0.017579
 future         0.015751
 wrap           0.015741
 adhesive       0.011913
 rain           0.011430
 lip            0.010565
 clammy         0.010158
 offer          0.008963
 shelf          0.007556
 touch          0.007548
 damp           0.006499
 barbarous      0.006480
 rigid          0.006472
 nonchalant     0.006457
 lethal         0.006313
 eight          0.004812
```

(continues on next page)

(continued from previous page)

```
cute          0.004193
change        0.003709
sweltering    0.003384
playground    0.001421
dtype: float64}
```

[43]:
```
xlearner.plot_importance(X=X,
                         tau=xlearner_tau,
                         method='permutation',
                         features=feature_names,
                         random_state=42)
```



Feature Importance (`sklearn.inspection.permutation_importance`)

[44]:
```
start_time = time.time()

X_train, X_test, y_train, y_test = train_test_split(X, xlearner_tau, test_size=0.3,␣
↪random_state=42)
model_tau_fit = model_tau.fit(X_train, y_train)

perm_imp_test = permutation_importance(
    estimator=model_tau_fit,
    X=X_test,
    y=y_test,
    random_state=42).importances_mean
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)

print("Elapsed time: %s seconds" % (time.time() - start_time))
```

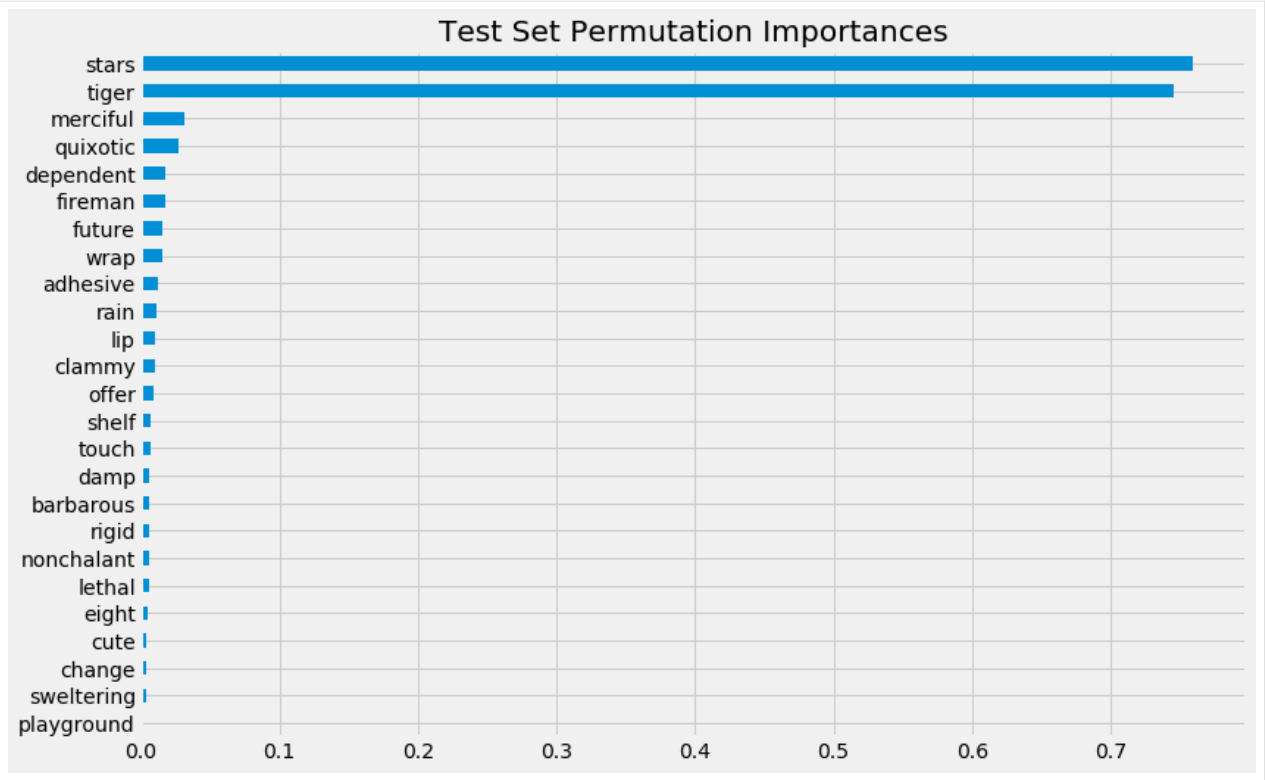
```
Elapsed time: 13.757911920547485 seconds
```

[45]:
```
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)
```

[45]:
```
stars          0.759553
tiger          0.745122
merciful       0.031355
quixotic       0.027350
dependent      0.018033
fireman        0.017579
future         0.015751
wrap           0.015741
adhesive       0.011913
rain           0.011430
lip            0.010565
clammy         0.010158
offer          0.008963
shelf          0.007556
touch          0.007548
damp           0.006499
barbarous      0.006480
rigid          0.006472
nonchalant     0.006457
lethal         0.006313
eight          0.004812
cute           0.004193
change         0.003709
sweltering     0.003384
playground     0.001421
dtype: float64
```

[46]:
```
pd.Series(perm_imp_test, feature_names).sort_values().plot(kind='barh', figsize=(12,␣
↪8))
plt.title('Test Set Permutation Importances')
```

[46]:
```
Text(0.5, 1.0, 'Test Set Permutation Importances')
```

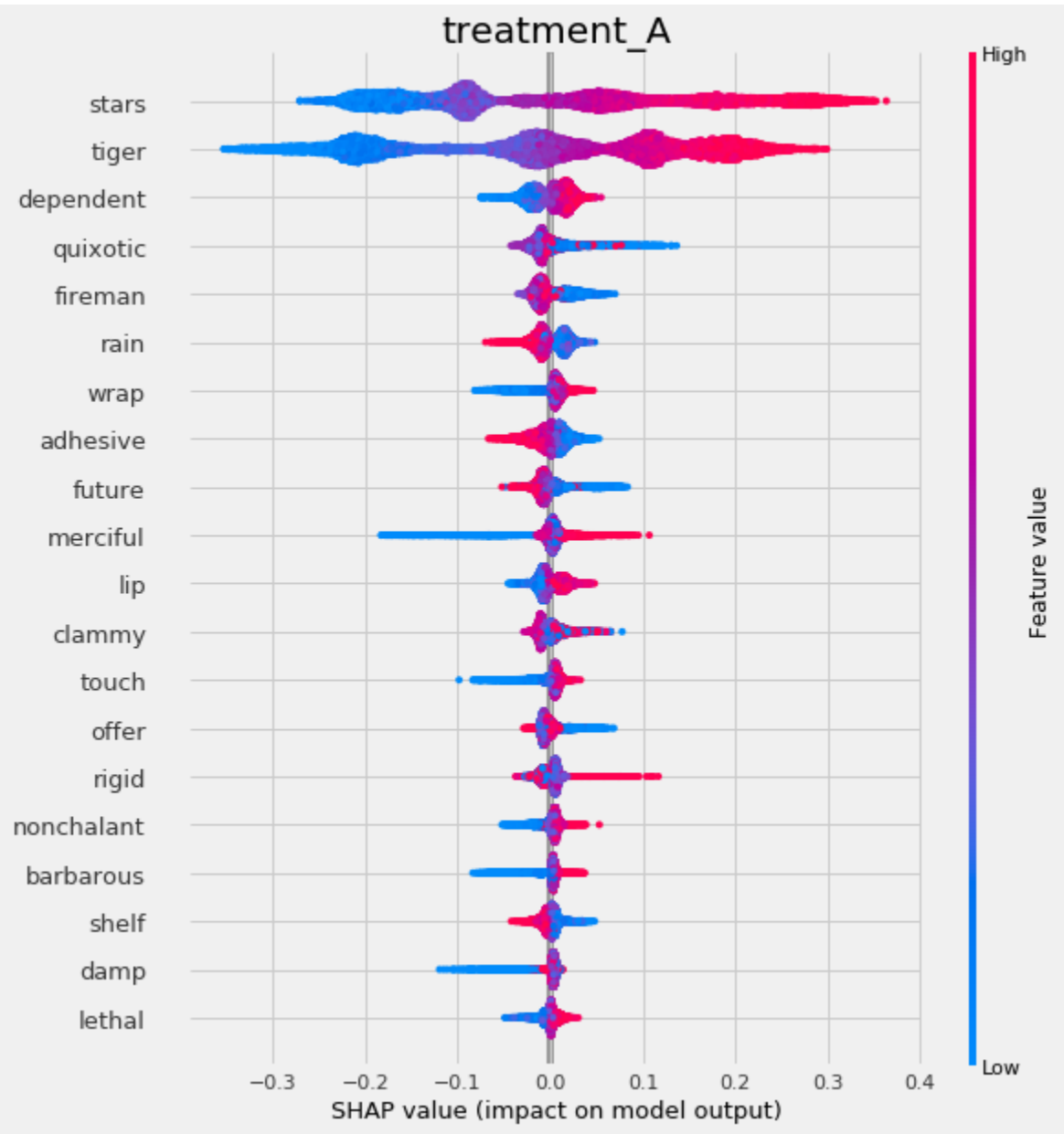


Shapley Values

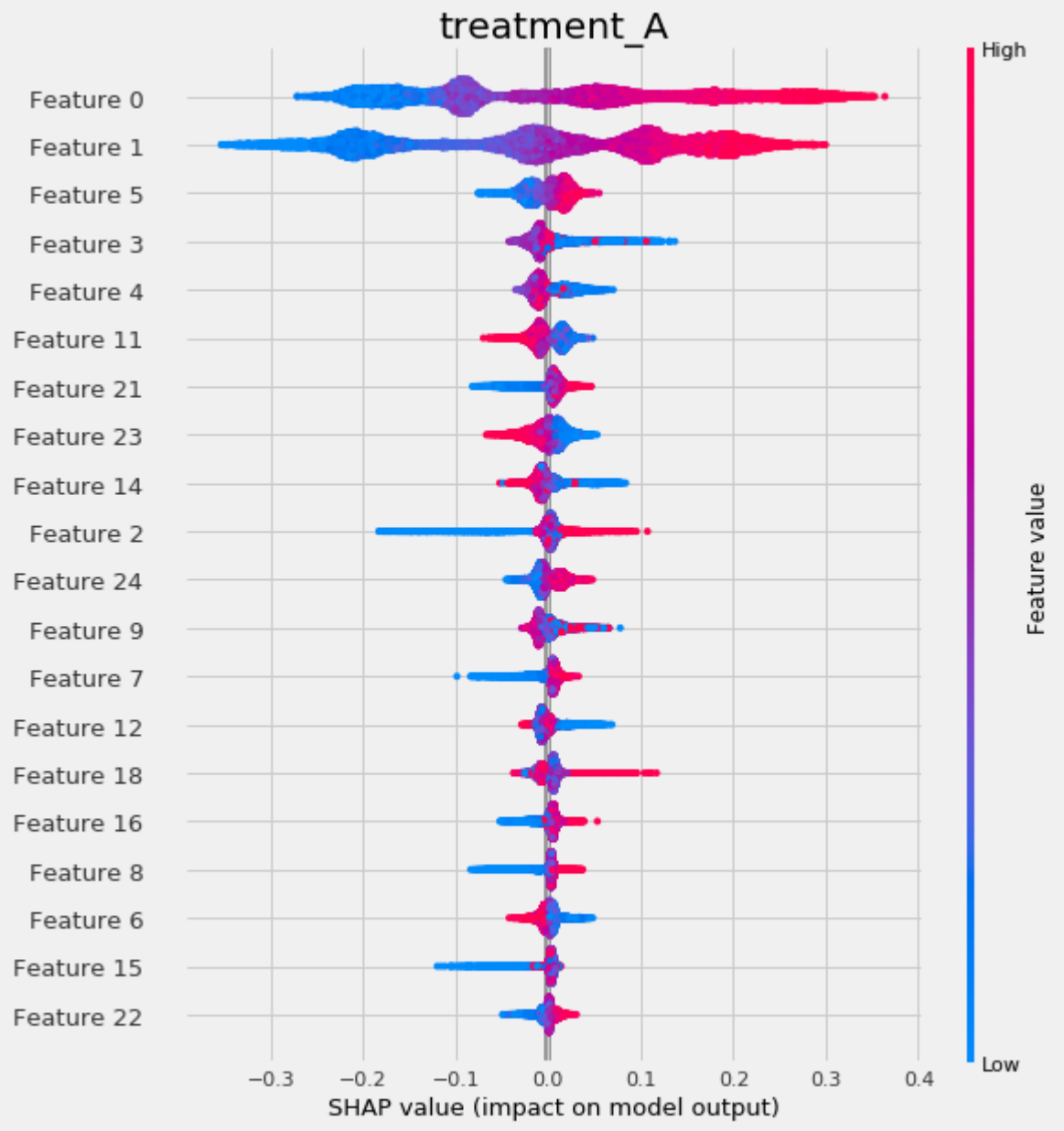
[47]:
```
shap_xlearner = xlearner.get_shap_values(X=X, tau=xlearner_tau)
shap_xlearner
```

[47]:
```
{'treatment_A': array([[ 0.05905145, -0.01813719, -0.00228681, …,  0.00163275,
          0.000808  ,  0.01982337],
        [-0.09223067,  0.03460351, -0.00243063, …, -0.00886324,
          0.00251886, -0.00680032],
        [ 0.07817859, -0.01975654,  0.00473035, …, -0.00076119,
          0.0218636 ,  0.01243895],
        …,
        [ 0.30115384,  0.09553369, -0.00154573, …, -0.00331466,
          0.00920979, -0.0128445 ],
        [-0.21004379, -0.03674163, -0.00241997, …,  0.00449733,
          0.01845317,  0.01552738],
        [-0.11479351,  0.06604962, -0.14693142, …,  0.00789741,
          0.00943036, -0.01086603]])}
```

[48]:
```
# shap_dict not specified
xlearner.plot_shap_values(X=X, tau=xlearner_tau, features=feature_names)
```



[49]:
```
# shap_dict specified
xlearner.plot_shap_values(X=X, shap_dict=shap_xlearner)
```



5.5.4 R Learner

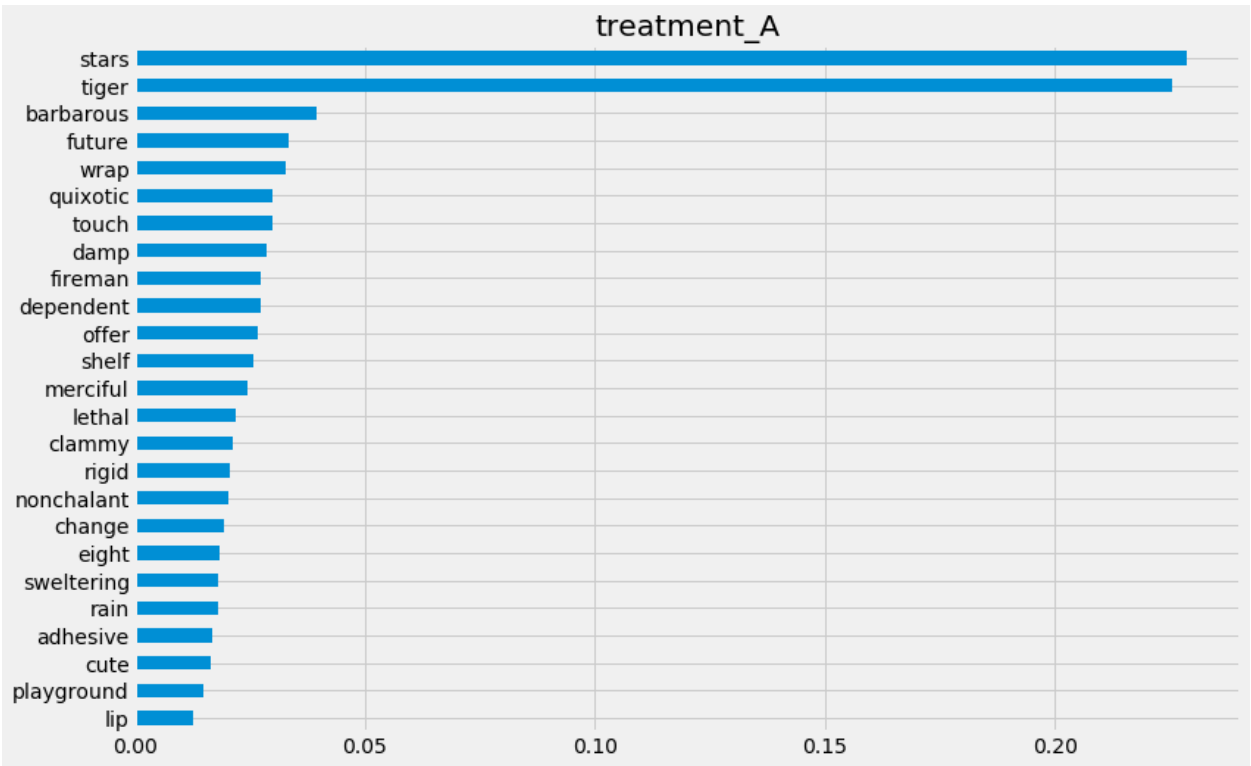
[50]:
```
rlearner = BaseRRegressor(LGBMRegressor(), control_name='control')
rlearner_tau = rlearner.fit_predict(X, w_multi, y, p=e_multi)
```

Feature Importance (method = `auto`)

[51]:
```
rlearner.get_importance(X=X,
                        tau=rlearner_tau,
                        normalize=True,
                        method='auto',
                        features=feature_names)
```

[51]:
```
{'treatment_A': stars         0.228704
 tiger          0.225389
 barbarous      0.039622
 future         0.033504
 wrap           0.032853
 quixotic       0.030002
 touch          0.029991
 damp           0.028726
 fireman        0.027299
 dependent      0.027245
 offer          0.026600
 shelf          0.025857
 merciful       0.024646
 lethal         0.022051
 clammy         0.021187
 rigid          0.020775
 nonchalant     0.020411
 change         0.019242
 eight          0.018544
 sweltering     0.018139
 rain           0.018029
 adhesive       0.016737
 cute           0.016656
 playground     0.014999
 lip            0.012792
 dtype: float64}
```

[63]:
```
rlearner.plot_importance(X=X,
                         tau=rlearner_tau,
                         method='auto',
                         features=feature_names)
```



Feature Importance (method = `permutation`)

[64]:
```
rlearner.get_importance(X=X,
                        tau=rlearner_tau,
                        method='permutation',
                        features=feature_names,
                        random_state=42)
```

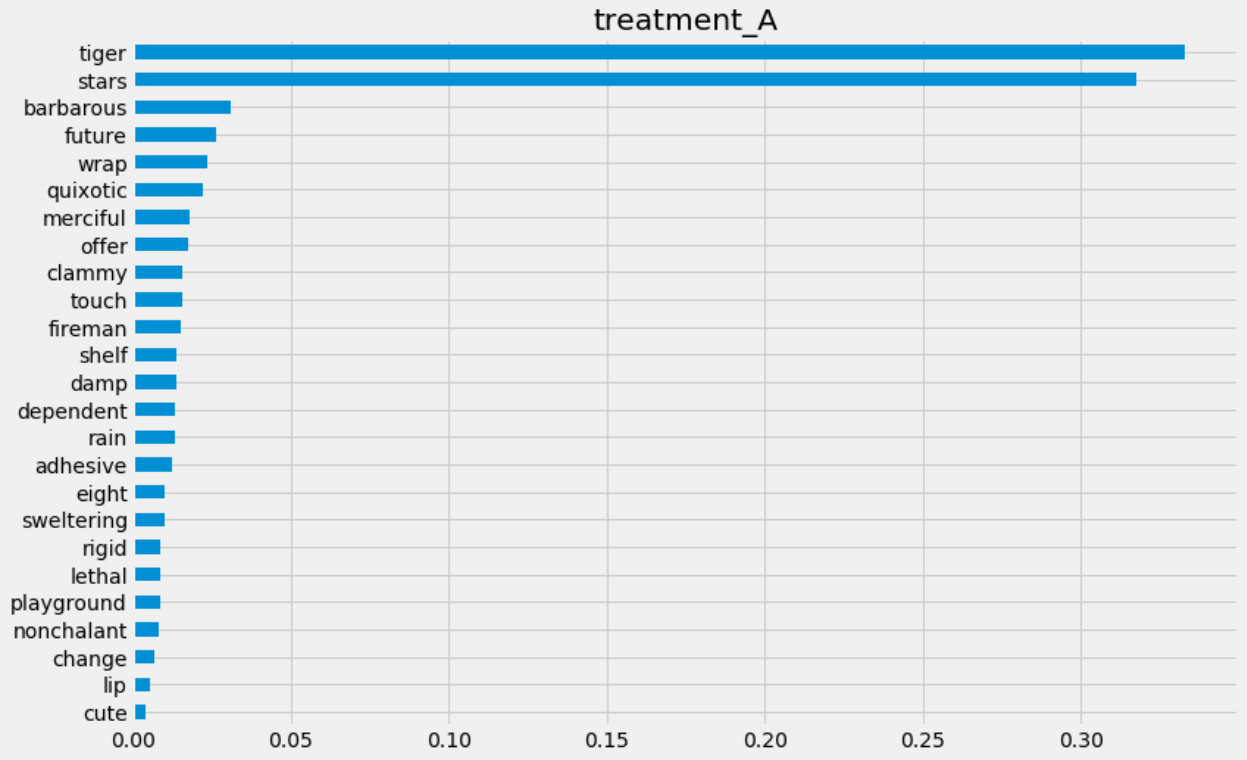
[64]:
```
{'treatment_A': tiger         0.333106
 stars          0.317470
 barbarous      0.030943
 future         0.026448
 wrap           0.023439
 quixotic       0.022111
 merciful       0.018122
 offer          0.017440
 clammy         0.015891
 touch          0.015746
 fireman        0.015017
 shelf          0.013932
 damp           0.013886
 dependent      0.013519
 rain           0.013181
 adhesive       0.012412
 eight          0.010187
 sweltering     0.010025
 rigid          0.008814
 lethal         0.008810
 playground     0.008513
```

(continues on next page)

(continued from previous page)

```
nonchalant    0.008323
change        0.006865
lip           0.005458
cute          0.004243
dtype: float64}
```

[65]:
```
rlearner.plot_importance(X=X,
                         tau=rlearner_tau,
                         method='permutation',
                         features=feature_names,
                         random_state=42)
```



Feature Importance (`sklearn.inspection.permutation_importance`)

[66]:
```
start_time = time.time()

X_train, X_test, y_train, y_test = train_test_split(X, rlearner_tau, test_size=0.3,␣
↪random_state=42)
model_tau_fit = model_tau.fit(X_train, y_train)

perm_imp_test = permutation_importance(
    estimator=model_tau_fit,
    X=X_test,
    y=y_test,
    random_state=42).importances_mean
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)

print("Elapsed time: %s seconds" % (time.time() - start_time))
```

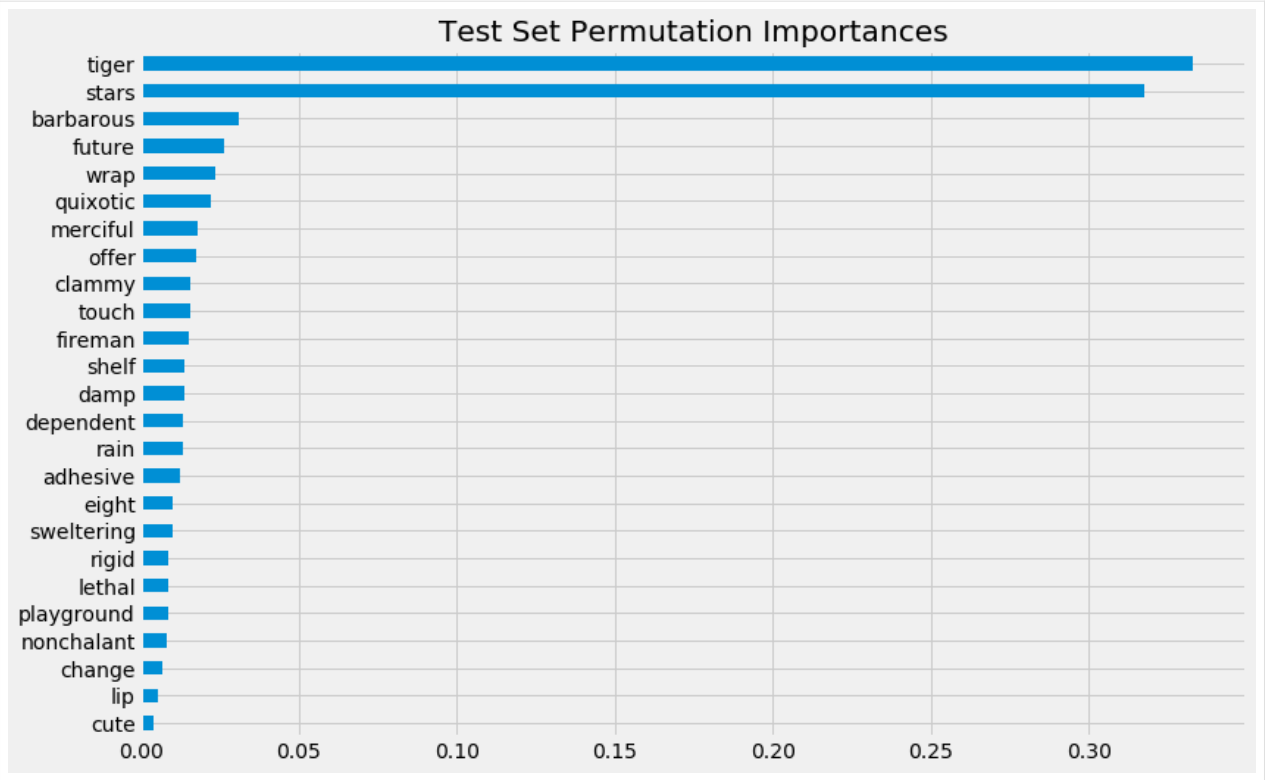
```
Elapsed time: 90.21177053451538 seconds
```

[67]:
```
pd.Series(perm_imp_test, feature_names).sort_values(ascending=False)
```

[67]:
```
tiger          0.333106
stars          0.317470
barbarous      0.030943
future         0.026448
wrap           0.023439
quixotic       0.022111
merciful       0.018122
offer          0.017440
clammy         0.015891
touch          0.015746
fireman        0.015017
shelf          0.013932
damp           0.013886
dependent      0.013519
rain           0.013181
adhesive       0.012412
eight          0.010187
sweltering     0.010025
rigid          0.008814
lethal         0.008810
playground     0.008513
nonchalant     0.008323
change         0.006865
lip            0.005458
cute           0.004243
dtype: float64
```

[68]:
```
pd.Series(perm_imp_test, feature_names).sort_values().plot(kind='barh', figsize=(12,␣
↪8))
plt.title('Test Set Permutation Importances')
```

[68]:
```
Text(0.5, 1.0, 'Test Set Permutation Importances')
```

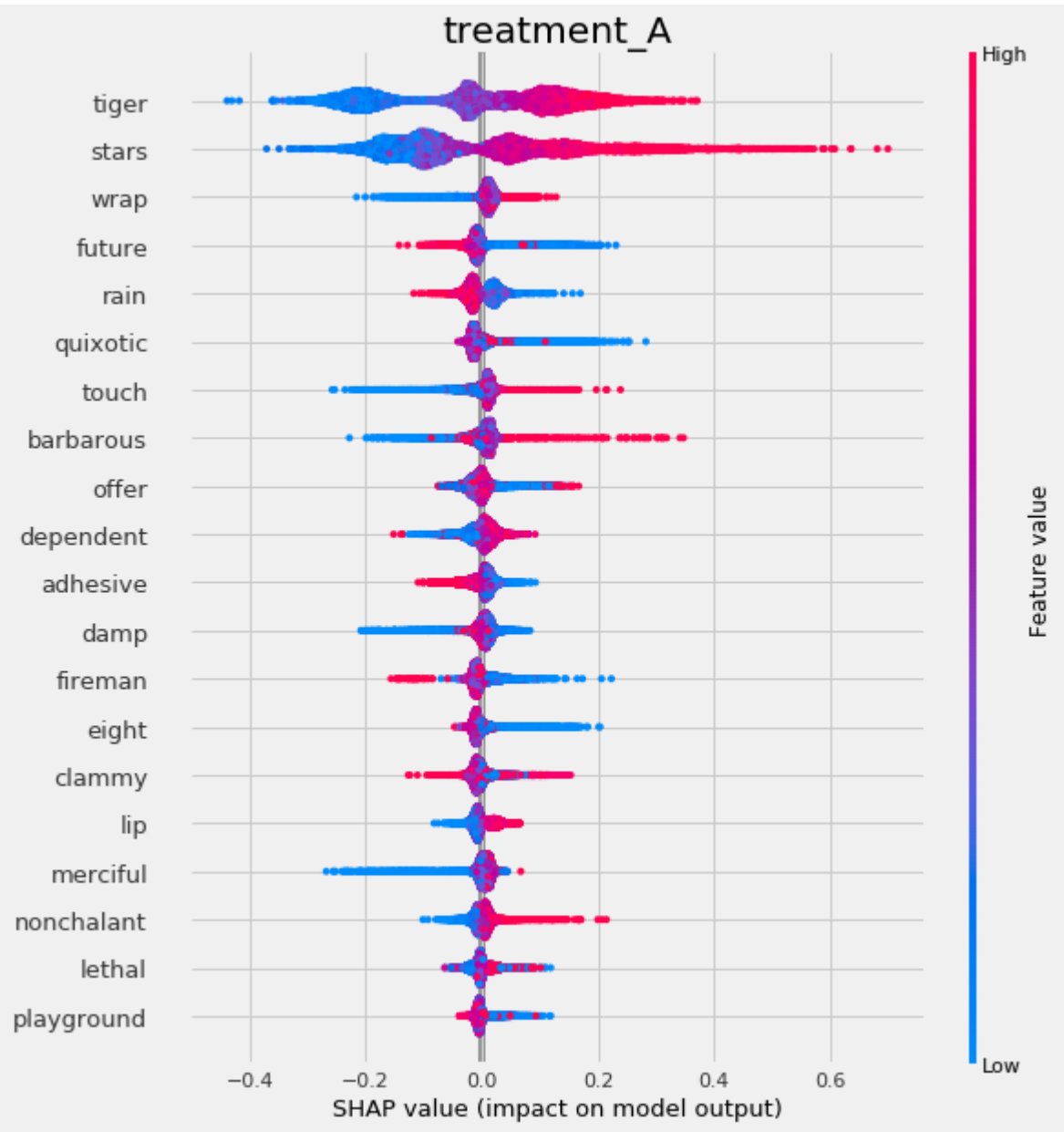


Shapley Values

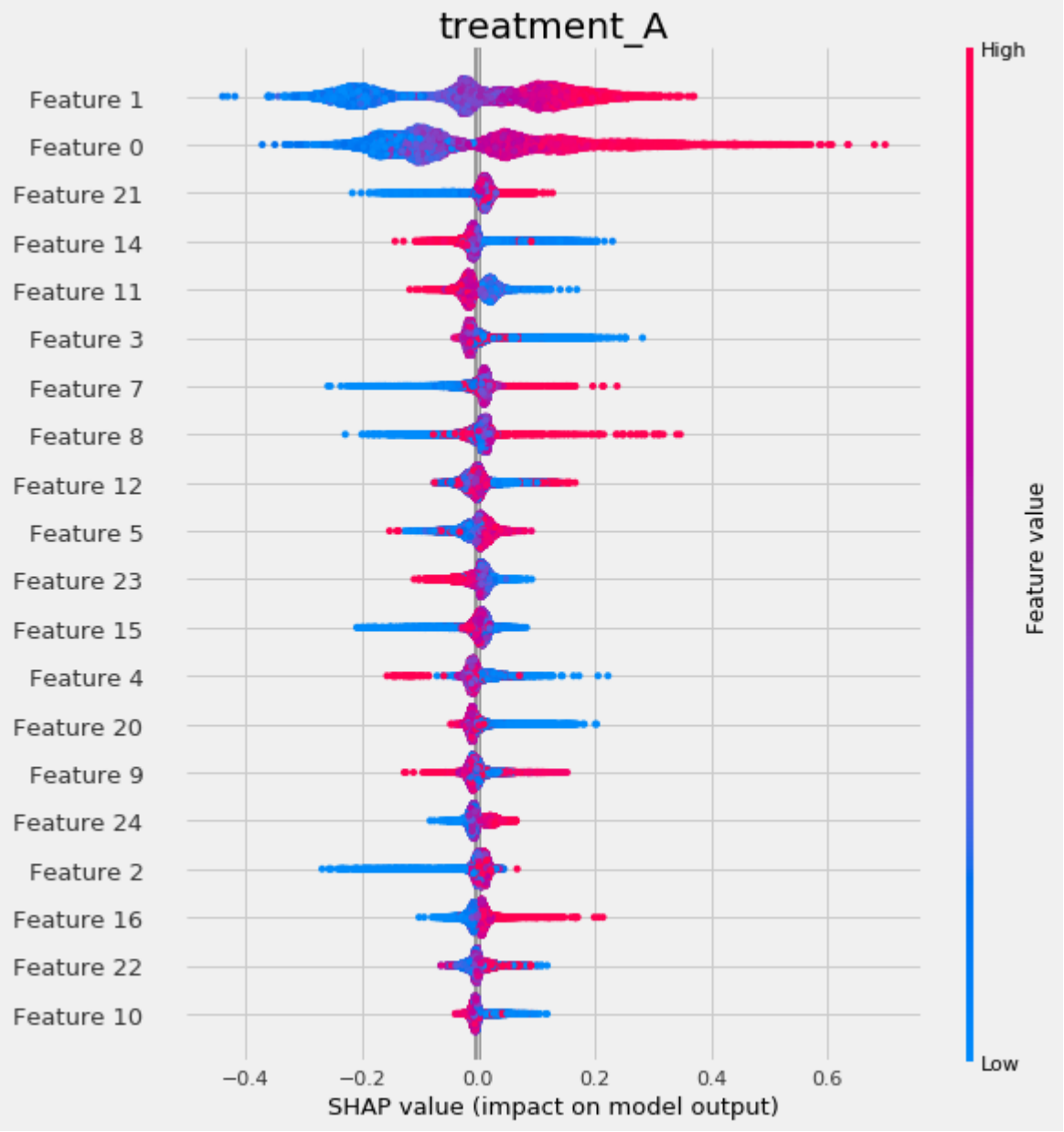
[69]:
```
shap_rlearner = rlearner.get_shap_values(X=X, tau=rlearner_tau)
shap_rlearner
```

[69]:
```
{'treatment_A': array([[ 0.03538328, -0.01669669, -0.00440836, …, -0.00239448,
          0.00593215,  0.01938478],
        [-0.10946828,  0.04119494, -0.00412831, …, -0.00789067,
          0.01280531, -0.00584103],
        [ 0.05171293, -0.00447188,  0.00395468, …, -0.00422879,
          0.00992719,  0.00150335],
        …,
        [ 0.31724012,  0.07934517,  0.00141576, …, -0.0094692 ,
          0.0169413 , -0.03495447],
        [-0.20257113, -0.03005302, -0.00690099, …, -0.00055628,
          0.02064072,  0.0141801 ],
        [-0.07420896,  0.10717246, -0.04564806, …,  0.01367809,
          0.01263303, -0.01483177]])}
```

[70]:
```
# without providing shap_dict
rlearner.plot_shap_values(X=X, tau=rlearner_tau, features=feature_names)
```



[71]:
```
# with providing shap_dict
rlearner.plot_shap_values(X=X, shap_dict=shap_rlearner)
```



5.5.5 Uplift Tree/Forest

Note that uplift trees/forests are only implemented for classification at the moment, hence the following section uses a different synthetic data generation process.

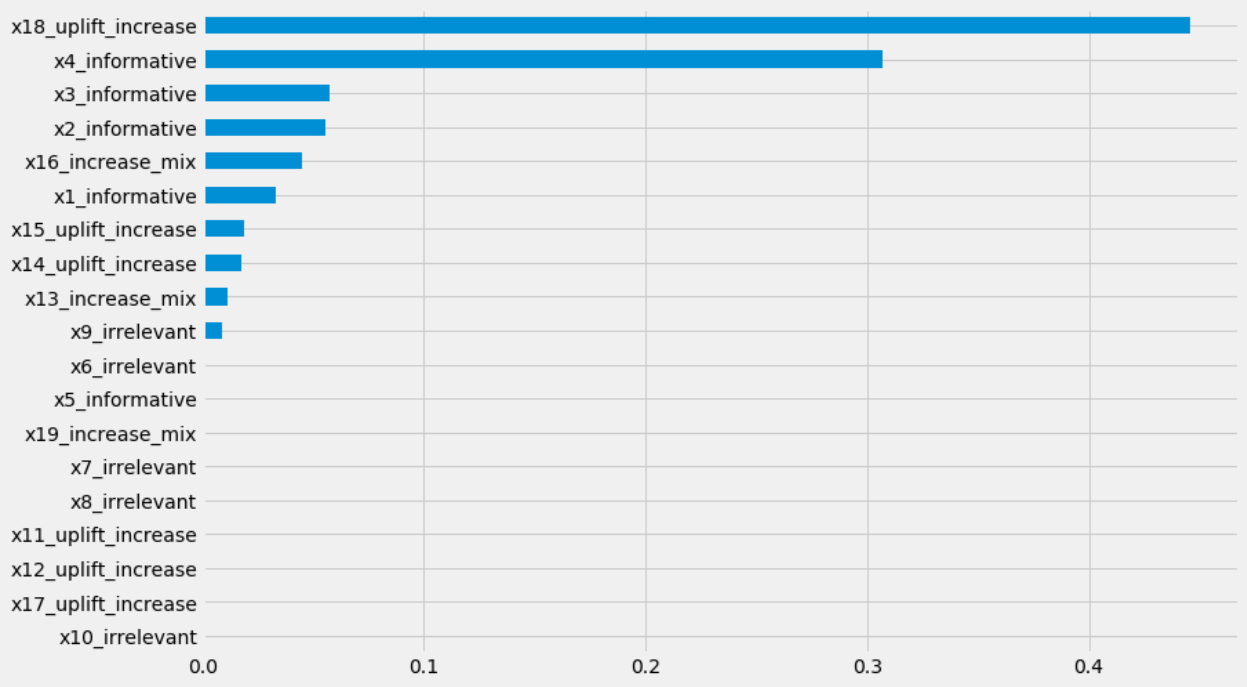
UpliftTreeClassifier

[61]:
```
from causalml.dataset import make_uplift_classification

df, x_names = make_uplift_classification()
```

[62]:
```
uplift_tree = UpliftTreeClassifier(control_name='control')

uplift_tree.fit(X=df[x_names].values,
                treatment=df['treatment_group_key'].values,
                y=df['conversion'].values)
```

[63]:
```
pd.Series(uplift_tree.feature_importances_, index=x_names).sort_values().plot(kind=
↪'barh', figsize=(12,8))
```

[63]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff8f0e13e48>
```

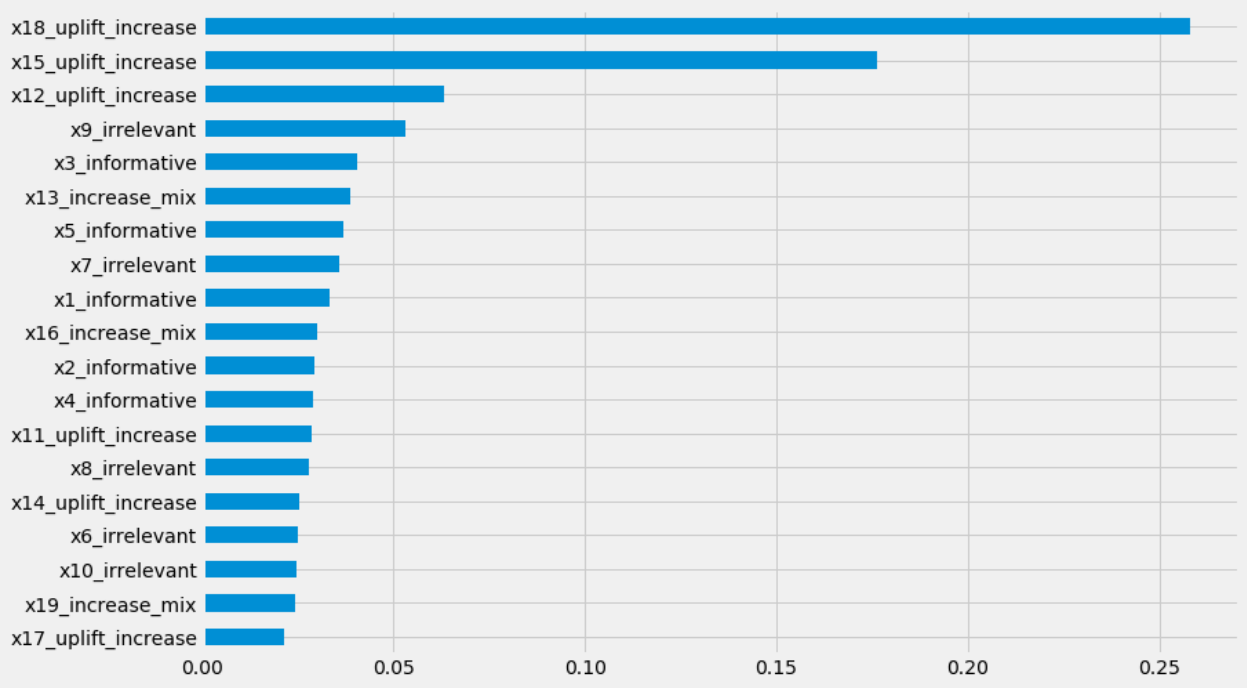


UpliftRandomForestClassifier

[64]:
```
uplift_rf = UpliftRandomForestClassifier(control_name='control')

uplift_rf.fit(X=df[x_names].values,
              treatment=df['treatment_group_key'].values,
              y=df['conversion'].values)
```

[65]:
```
pd.Series(uplift_rf.feature_importances_, index=x_names).sort_values().plot(kind='barh
↪', figsize=(12,8))
```

[65]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff90d027358>
```



[ ]:

5.6 Uplift Curves with TMLE Example

This notebook demonstrates the issue of using uplift curves without knowing true treatment effect and how to solve it by using TMLE as a proxy of the true treatment effect.

[2]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

[3]:
```
import os
base_path = os.path.abspath("../")
os.chdir(base_path)
```

[4]:
```
import logging
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
import sys
import warnings
warnings.simplefilter("ignore", UserWarning)

from lightgbm import LGBMRegressor
```

[5]:
```
import causalml
```

(continues on next page)

(continued from previous page)

```
from causalml.dataset import synthetic_data
from causalml.inference.meta import BaseXRegressor, TMLELearner
from causalml.metrics.visualize import *
from causalml.propensity import calibrate

import importlib
print(importlib.metadata.version('causalml') )
```

```
0.14.0
```

[8]:
```
logger = logging.getLogger('causalml')
logger.setLevel(logging.DEBUG)
plt.style.use('fivethirtyeight')
```

5.6.1 Generating Synthetic Data

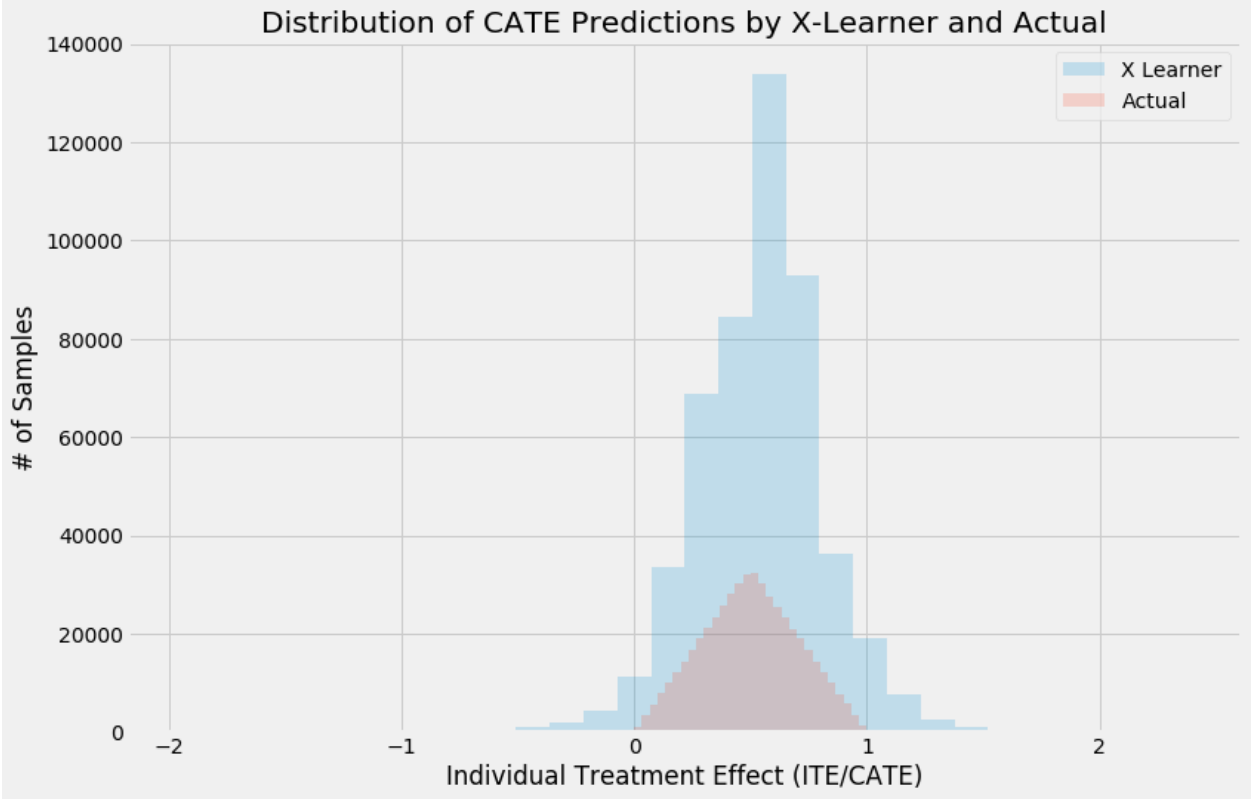
[9]:
```
# Generate synthetic data using mode 1
y, X, treatment, tau, b, e = synthetic_data(mode=1, n=1000000, p=10, sigma=5.)
```

[10]:
```
X_train, X_test, y_train, y_test, e_train, e_test, treatment_train, treatment_test,␣
↪tau_train, tau_test, b_train, b_test = train_test_split(X, y, e, treatment, tau, b,␣
↪test_size=0.5, random_state=42)
```

5.6.2 Calculating Individual Treatment Effect (ITE/CATE)

[12]:
```
# X Learner
learner_x = BaseXRegressor(learner=LGBMRegressor())
learner_x.fit(X=X_train, treatment=treatment_train, y=y_train)
cate_x_test = learner_x.predict(X=X_test, p=e_test, treatment=treatment_test).
↪flatten()
```

[13]:
```
alpha=0.2
bins=30
plt.figure(figsize=(12,8))
plt.hist(cate_x_test, alpha=alpha, bins=bins, label='X Learner')
plt.hist(tau_test, alpha=alpha, bins=bins, label='Actual')

plt.title('Distribution of CATE Predictions by X-Learner and Actual')
plt.xlabel('Individual Treatment Effect (ITE/CATE)')
plt.ylabel('# of Samples')
_=plt.legend()
```



5.6.3 Validating CATE without TMLE

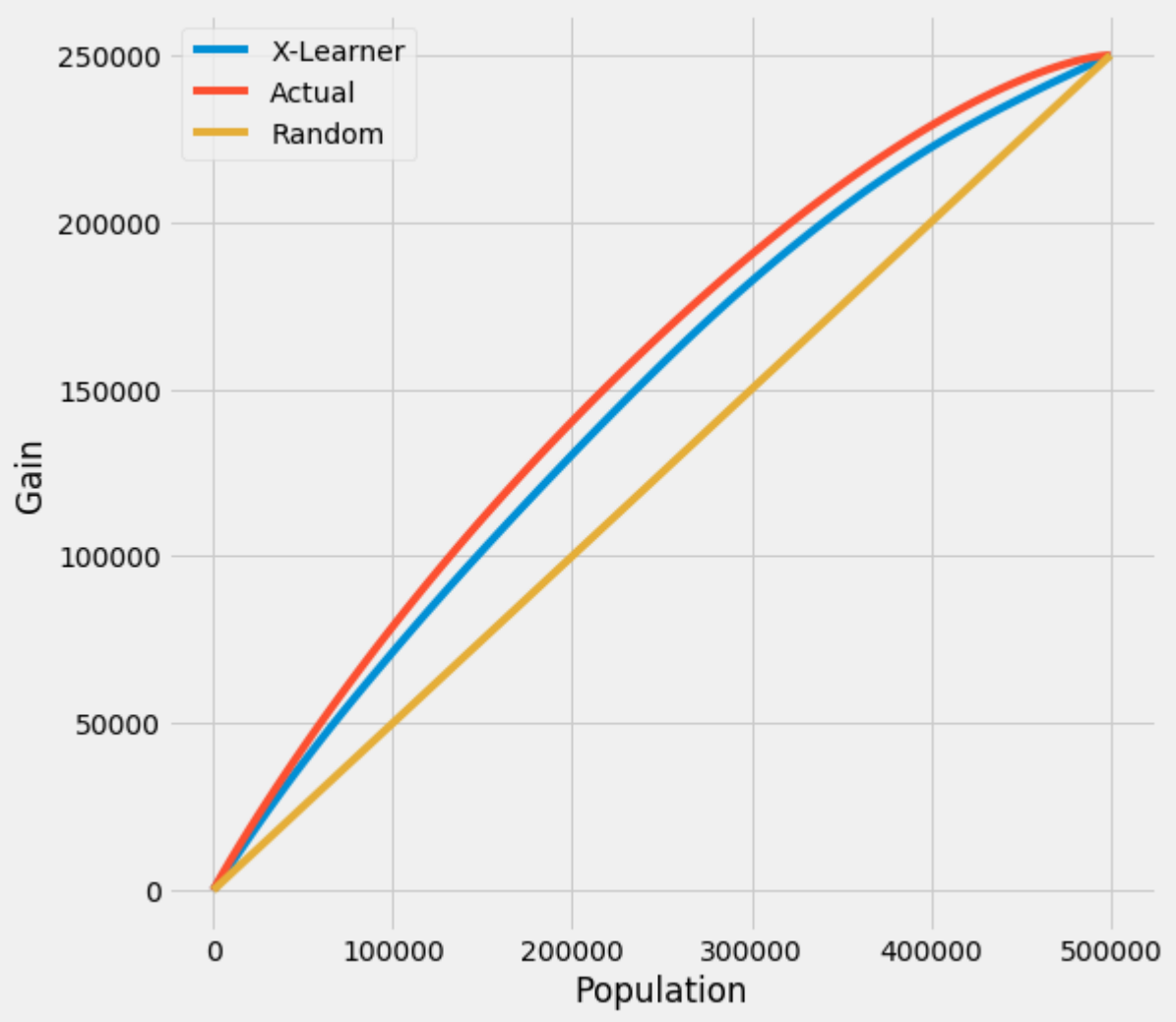
[14]:
```
df = pd.DataFrame({'y': y_test, 'w': treatment_test, 'tau': tau_test, 'X-Learner':␣
↪cate_x_test, 'Actual': tau_test})
```

Uplift Curve With Ground Truth

If true treatment effect is known as in simulations, the uplift curve of a model uses the cumulative sum of the treatment effect sorted by model's CATE estimate.

In the figure below, the uplift curve of X-learner shows positive lift close to the optimal lift by the ground truth.

[15]:
```
plot(df, outcome_col='y', treatment_col='w', treatment_effect_col='tau')
```

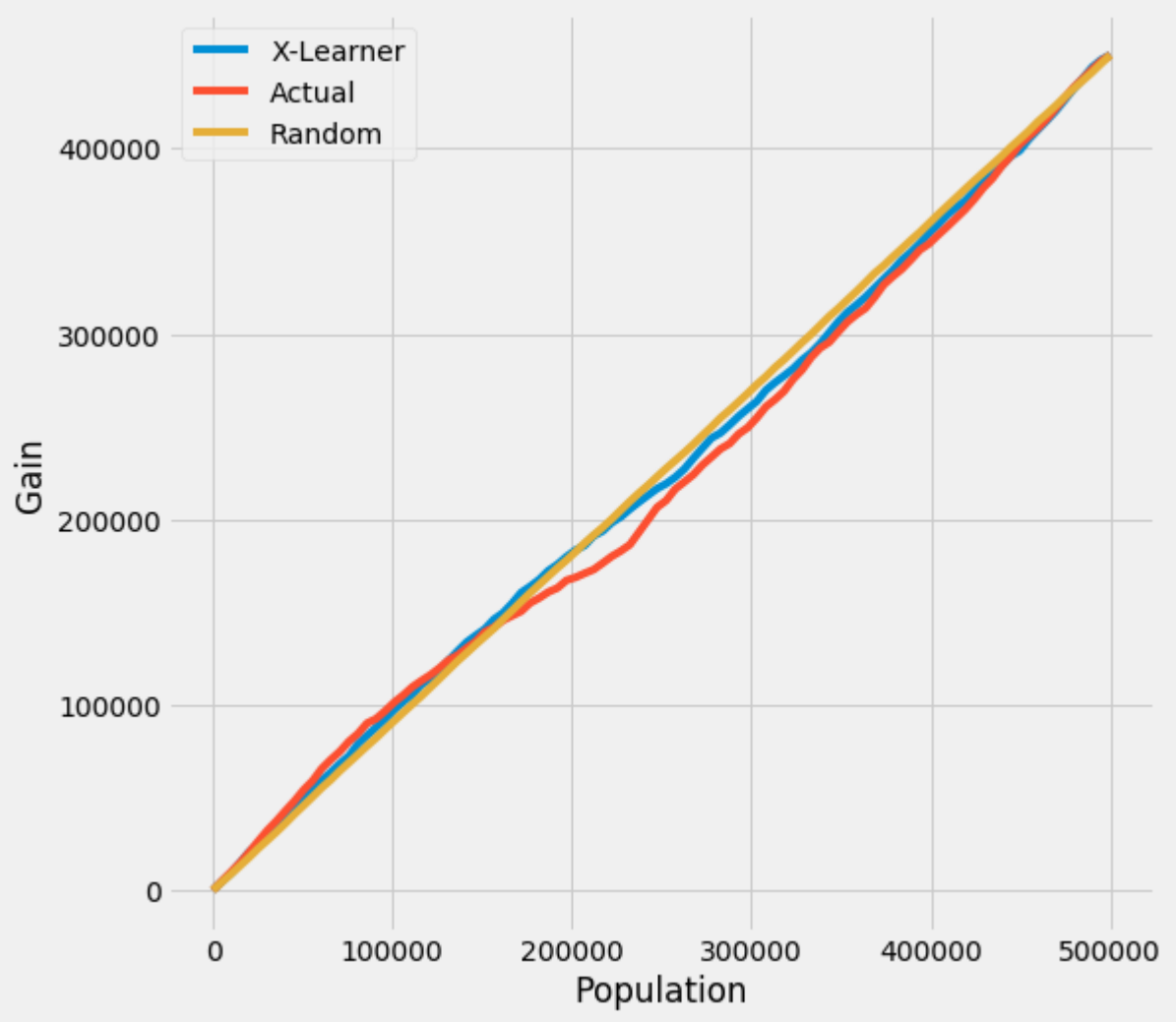


Uplift Curve Without Ground Truth

If true treatment effect is unknown as in practice, the uplift curve of a model uses the cumulative mean difference of outcome in the treatment and control group sorted by model's CATE estimate.

In the figure below, the uplift curves of X-learner as well as the ground truth show no lift incorrectly.

[16]:
```
plot(df.drop('tau', axis=1), outcome_col='y', treatment_col='w')
```



5.6.4 TMLE

Uplift Curve with TMLE as Ground Truth

By using TMLE as a proxy of the ground truth, the uplift curves of X-learner and the ground truth become close to the original using the ground truth.

[17]:
```
n_fold = 5
kf = KFold(n_splits=n_fold)
```

[18]:
```
df = pd.DataFrame({'y': y_test, 'w': treatment_test, 'p': e_test, 'X-Learner': cate_x_
↪test, 'Actual': tau_test})
```

[19]:
```
inference_cols = []
for i in range(X_test.shape[1]):
    col = 'col_' + str(i)
    df[col] = X_test[:,i]
    inference_cols.append(col)
```

[20]:
```
df.head()
```

[20]:
```
          y  w         p  X-Learner    Actual     col_0     col_1     col_2  \
0  6.808806  1  0.750090   0.909261  0.856218  0.911884  0.800551  0.637318
1  5.074509  1  0.828351   0.696708  0.613880  0.871032  0.356727  0.168573
2 -8.293643  0  0.230920   0.456776  0.335491  0.531401  0.139581  0.604482
3  4.511347  0  0.306119   0.189546  0.388202  0.615514  0.160891  0.825520
4  5.418169  0  0.293402   0.299151  0.476290  0.839696  0.112883  0.964546

      col_3     col_4     col_5     col_6     col_7     col_8     col_9
0  0.584033  0.041204  0.541312  0.183795  0.604942  0.802967  0.321925
1  0.291071  0.953692  0.838566  0.497353  0.777390  0.811558  0.076717
2  0.051055  0.651872  0.878593  0.592694  0.695946  0.972597  0.178291
3  0.544876  0.107617  0.746920  0.002706  0.963717  0.603323  0.506294
4  0.336093  0.548355  0.649487  0.905765  0.249261  0.070978  0.947820
```

[21]:
```
tmle_df = get_tmlegain(df, inference_col=inference_cols, outcome_col='y', treatment_
↪col='w', p_col='p',
                       n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```

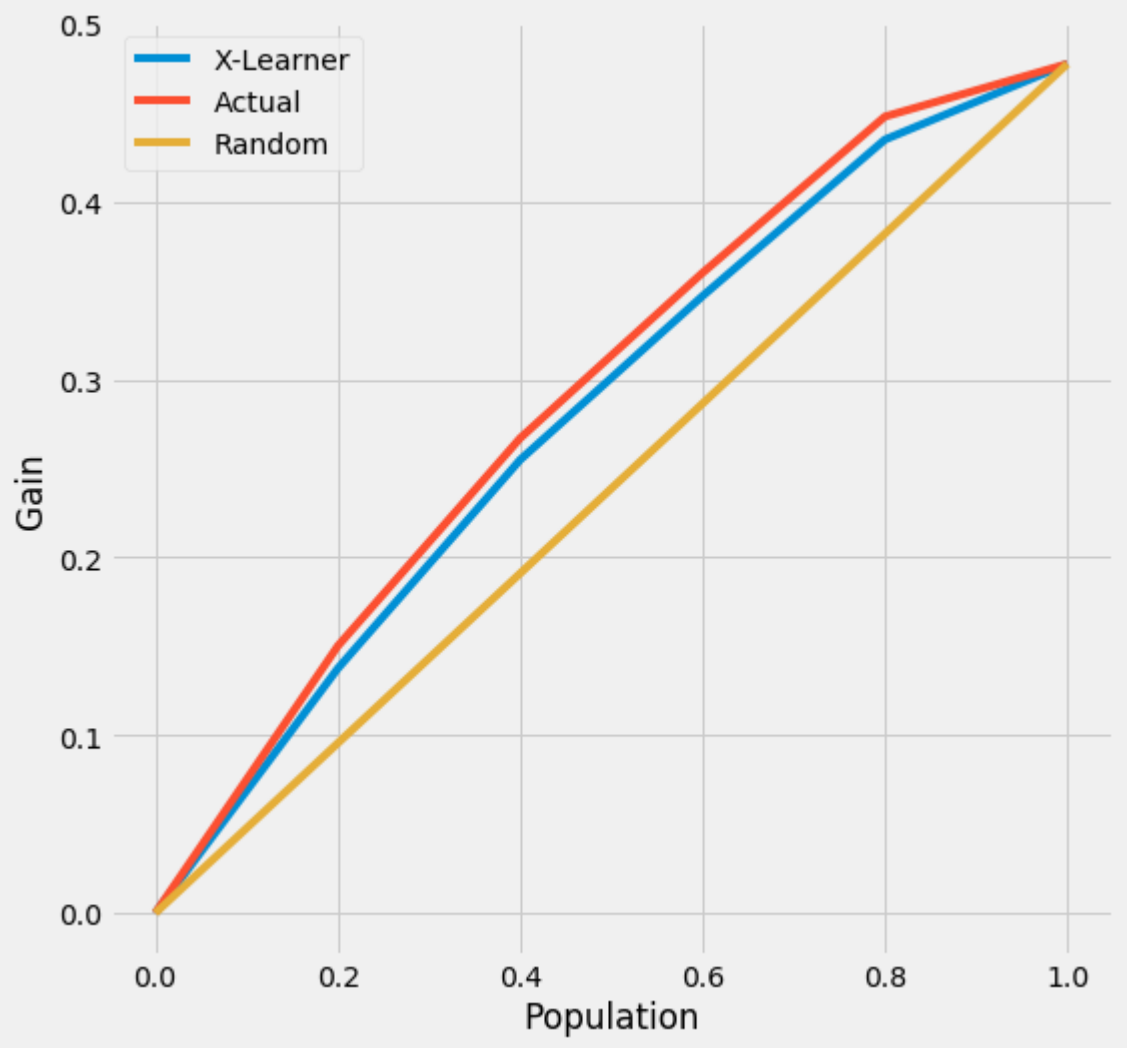
[22]:
```
tmle_df
```

[22]:
```
     X-Learner    Actual    Random
0.0   0.000000  0.000000  0.000000
0.2   0.137463  0.150106  0.095493
0.4   0.254839  0.267014  0.190986
0.6   0.346940  0.359990  0.286479
0.8   0.434913  0.447867  0.381972
1.0   0.477465  0.477465  0.477465
```

Uplift Curve wihtout CI

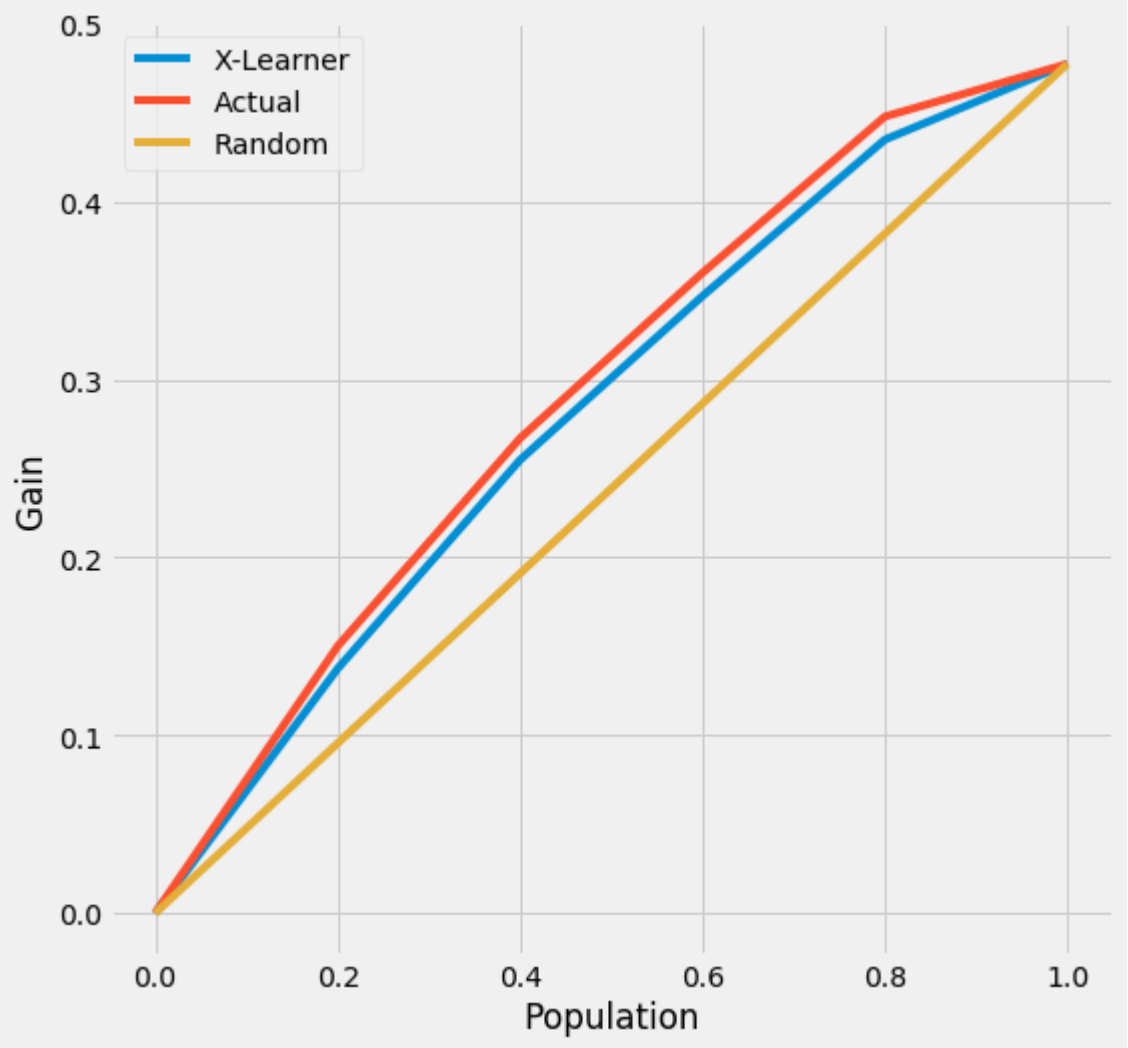
Here we can directly use plot$_tmle()functiontogeneratetheresultsandplotupliftcurve$

[23]:
```
plot_tmlegain(df, inference_col=inference_cols, outcome_col='y', treatment_col='w', p_
↪col='p',
              n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```



We also provide the api call directly with plot() by input the kind as 'tmle'

[24]:
```
plot(df, kind='gain', tmle=True, inference_col=inference_cols, outcome_col='y',␣
↪treatment_col='w', p_col='p',
     n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```



AUUC Score

[25]:
```
auuc_score(df, tmle=True, inference_col=inference_cols, outcome_col='y', treatment_
↪col='w', p_col='p',
           n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```

[25]:
```
X-Learner    0.275270
Actual       0.283740
Random       0.238733
dtype: float64
```

Uplift Curve with CI

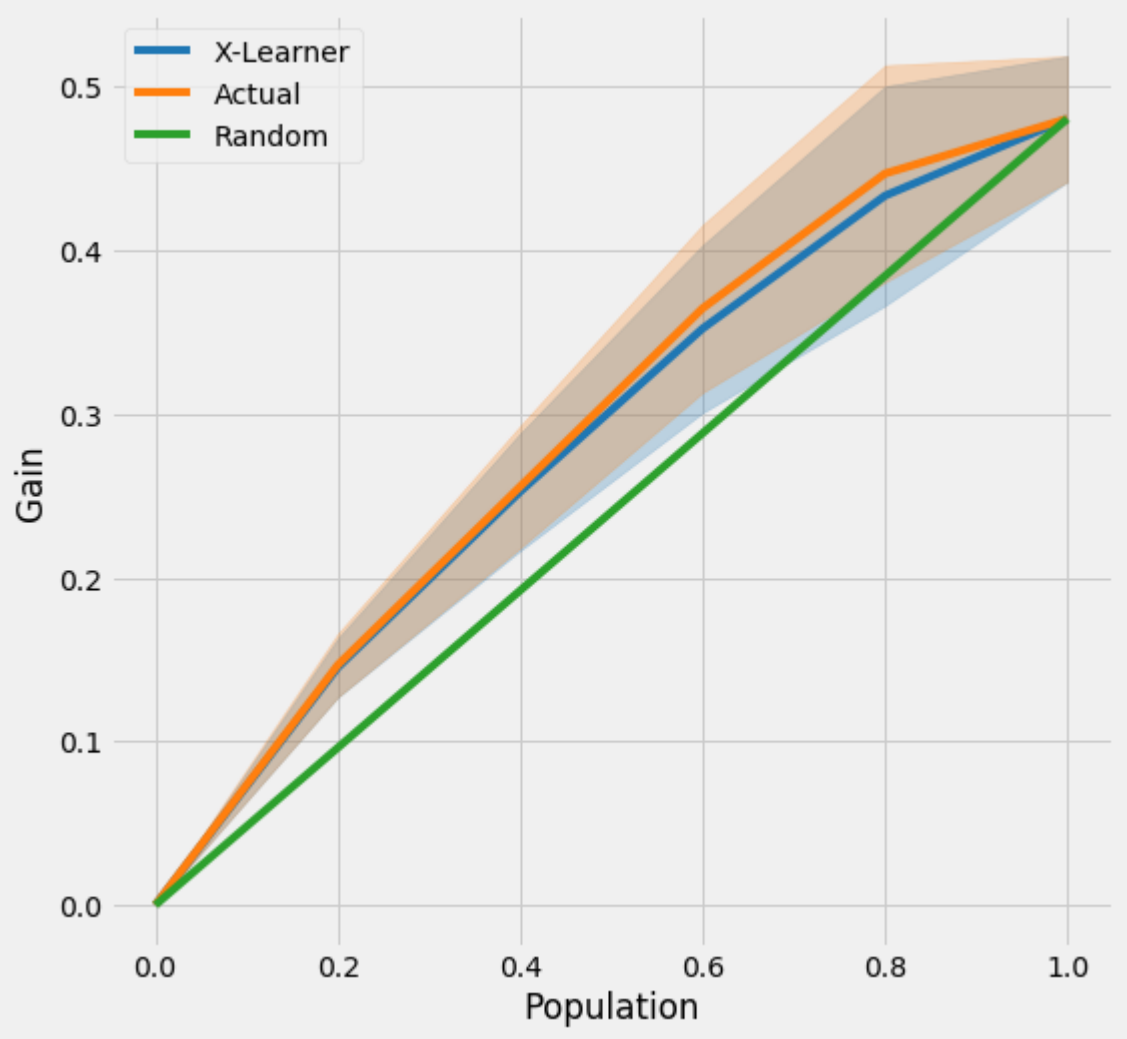
[25]:
```
tmle_df = get_tmlegain(df, inference_col=inference_cols, outcome_col='y', treatment_
↪col='w', p_col='p',
                       n_segment=5, cv=kf, calibrate_propensity=True, ci=True)
```

[26]:
```
tmle_df
```

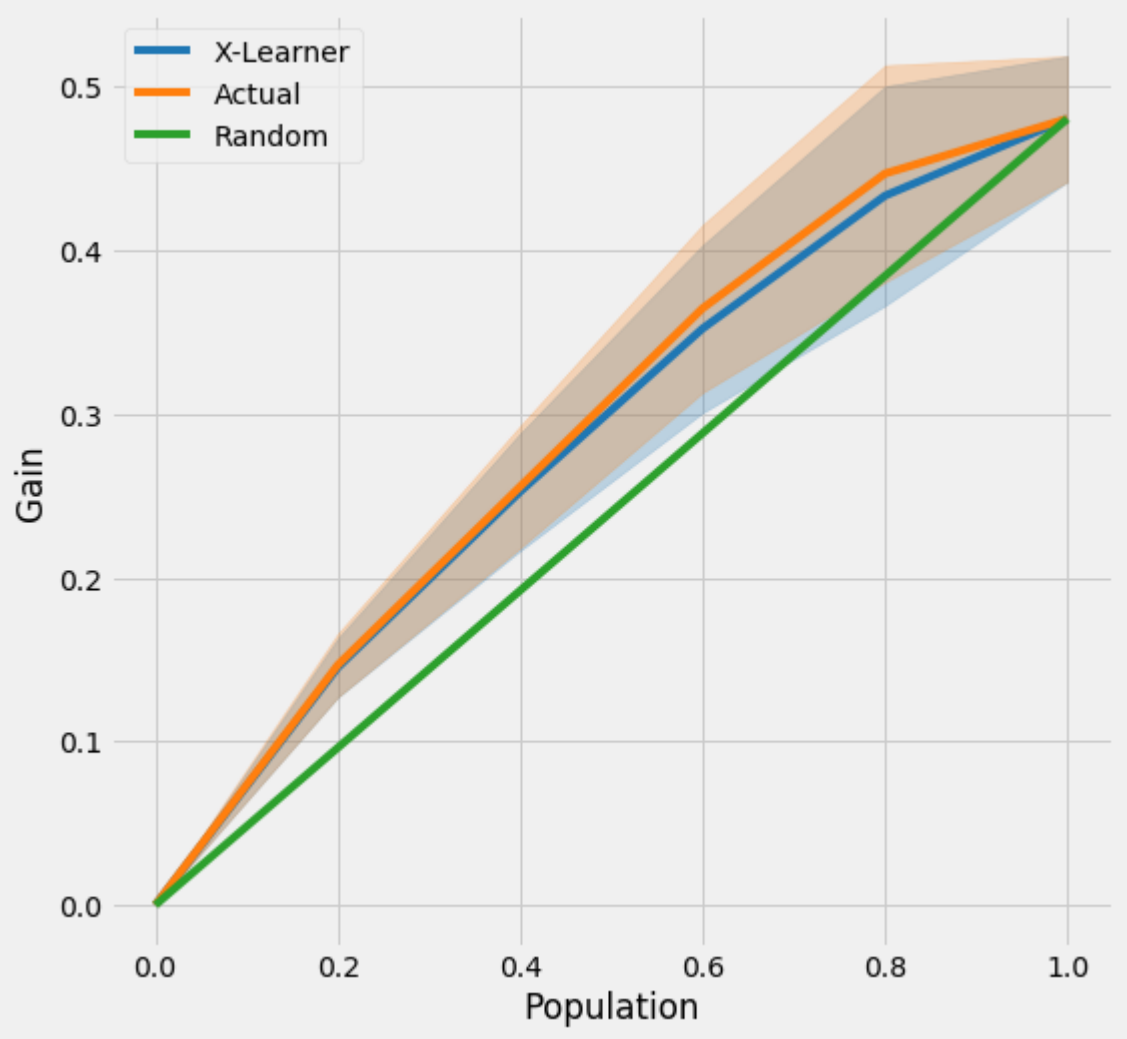
[26]:

	X-Learner	Actual	X-Learner LB	Actual LB	X-Learner UB	Actual UB	Random
0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.2	0.145151	0.146628	0.127016	0.127210	0.163285	0.166046	0.096077
0.4	0.252563	0.255667	0.216629	0.218323	0.288496	0.293011	0.192154
0.6	0.352174	0.364541	0.300866	0.313233	0.403483	0.415850	0.288231
0.8	0.433351	0.446890	0.366285	0.380624	0.500417	0.513157	0.384308
1.0	0.480384	0.480384	0.441999	0.441999	0.518770	0.518770	0.480384

[27]:
```
plot_tmlegain(df, inference_col=inference_cols, outcome_col='y', treatment_col='w', p_
↪col='p',
              n_segment=5, cv=kf, calibrate_propensity=True, ci=True)
```



[29]:
```
plot(df, kind='gain', tmle=True, inference_col=inference_cols, outcome_col='y',␣
↪treatment_col='w', p_col='p',
     n_segment=5, cv=kf, calibrate_propensity=True, ci=True)
```



Qini Curve with TMLE as Ground Truth

Qini Curve without CI

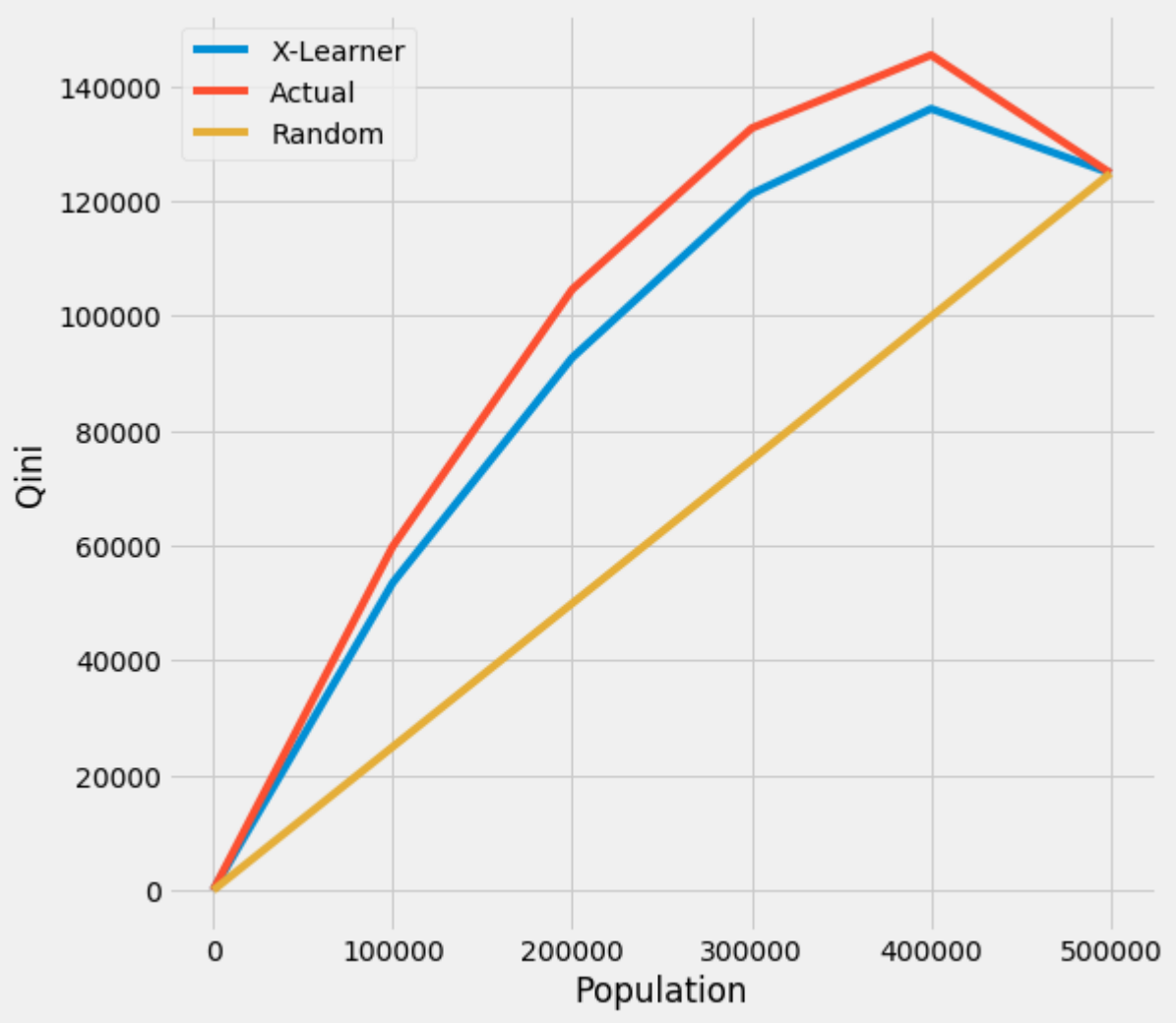
[30]:
```
qini = get_tmleqini(df, inference_col=inference_cols, outcome_col='y', treatment_col=
↪'w', p_col='p',
                    n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```

[31]:
```
qini
```

[31]:

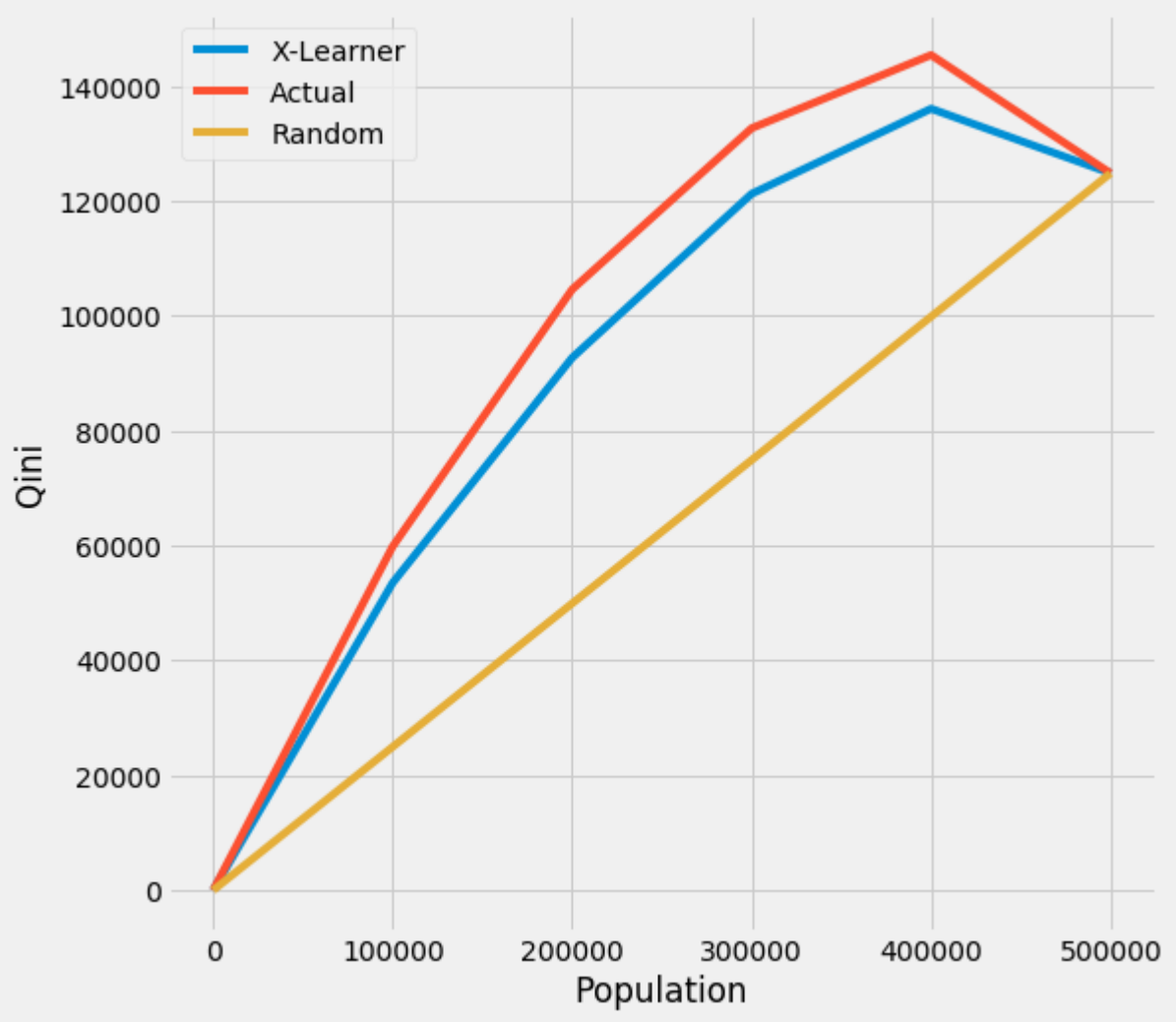
	X-Learner	Actual	Random
0.0	0.000000	0.000000	0.000000
100000.0	53513.373815	59840.329296	24964.329463
200000.0	92693.576894	104578.508000	49928.658925
300000.0	121232.782373	132653.427128	74892.988388
400000.0	136045.083604	145388.277994	99857.317851
500000.0	124821.647313	124821.647313	124821.647313

[32]:
```
plot_tmleqini(df, inference_col=inference_cols, outcome_col='y', treatment_col='w', p_
↪col='p',
              n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```



We also provide the api call directly with plot() by input the kind as 'tmleqini'

[34]:
```
plot(df, kind='qini', tmle=True, inference_col=inference_cols, outcome_col='y',␣
↪treatment_col='w', p_col='p',
     n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```



Qini Score

[26]:
```
qini_score(df, tmle=True, inference_col=inference_cols, outcome_col='y', treatment_
↪col='w', p_col='p',
           n_segment=5, cv=kf, calibrate_propensity=True, ci=False)
```

[26]:
```
X-Learner    23814.998608
Actual       33683.500462
Random           0.000000
dtype: float64
```

Qini Curve with CI

[36]:
```
qini = get_tmleqini(df, inference_col=inference_cols, outcome_col='y', treatment_col=
↪'w', p_col='p',
                    n_segment=5, cv=kf, calibrate_propensity=True, ci=True)
```

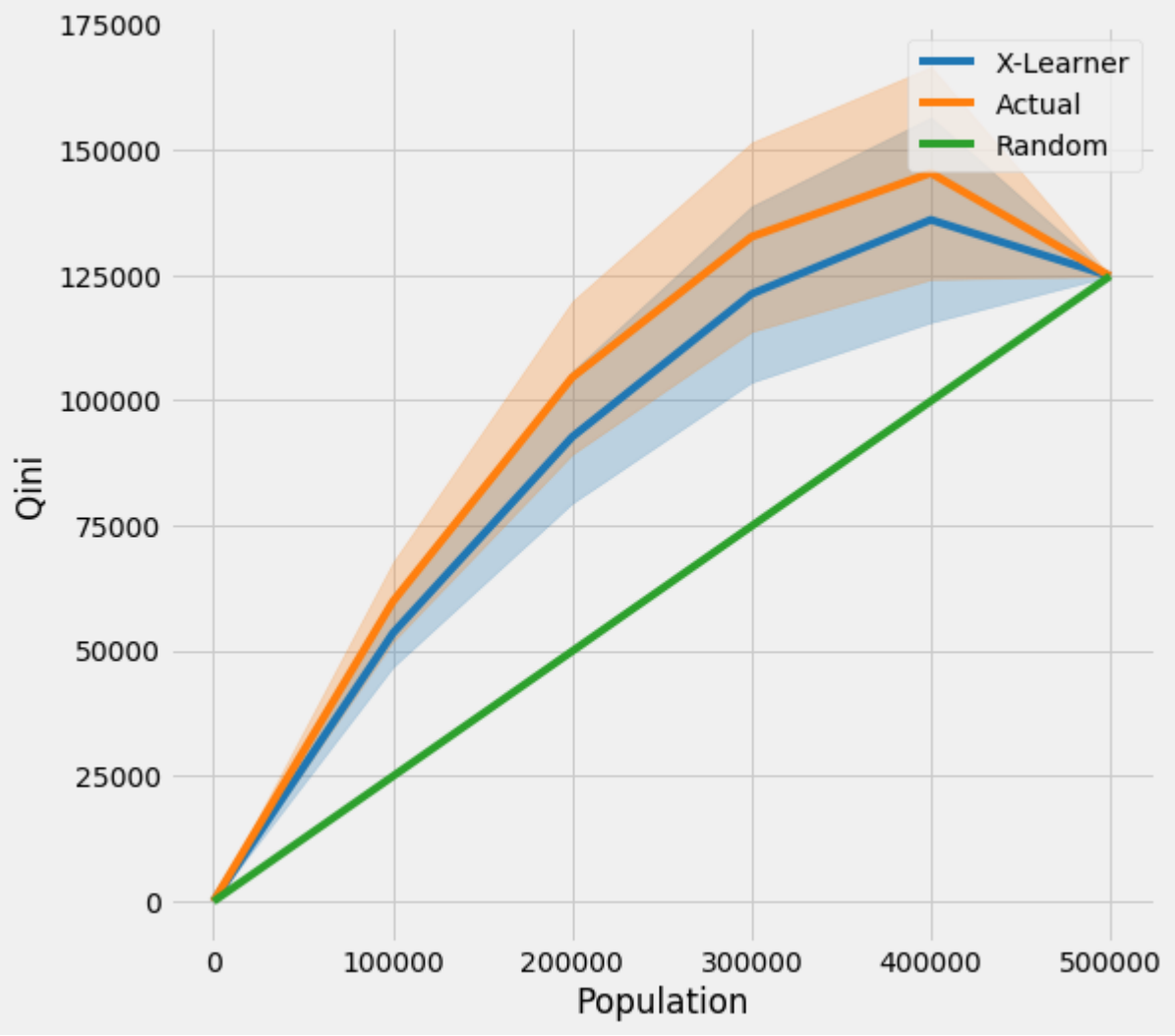
[37]:
```
qini
```

[37]:

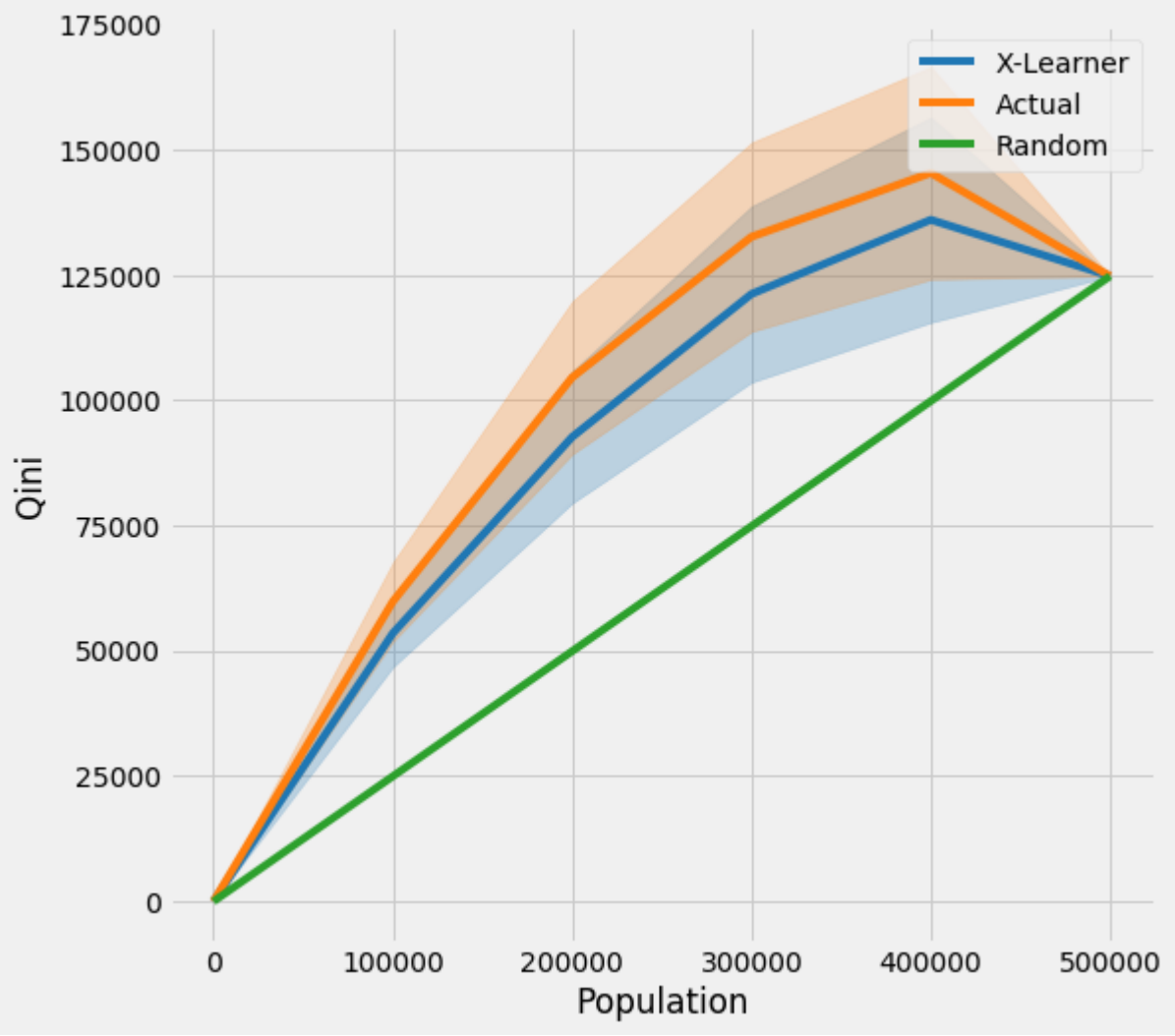
	X-Learner	Actual	X-Learner LB	Actual LB	\
0.0	0.000000	0.000000	0.000000	0.000000	
100000.0	53513.373815	59840.329296	46827.611036	51915.622165	
200000.0	92693.576894	104578.508000	79515.426034	89298.783725	
300000.0	121232.782373	132653.427128	103649.630931	113772.913012	
400000.0	136045.083604	145388.277994	115586.643138	124194.530581	
500000.0	124821.647313	124821.647313	124821.647313	124821.647313	

	X-Learner UB	Actual UB	Random
0.0	0.000000	0.000000	0.000000
100000.0	60199.136594	67765.036427	24964.329463
200000.0	105871.727753	119858.232275	49928.658925
300000.0	138815.933816	151533.941243	74892.988388
400000.0	156503.524070	166582.025407	99857.317851
500000.0	124821.647313	124821.647313	124821.647313

[38]:
```
plot_tmleqini(df, inference_col=inference_cols, outcome_col='y', treatment_col='w', p_
↪col='p',
              n_segment=5, cv=kf, calibrate_propensity=True, ci=True)
```



[39]:
```
plot(df, kind='qini', tmle=True, inference_col=inference_cols, outcome_col='y',␣
↪treatment_col='w', p_col='p',
     n_segment=5, cv=kf, calibrate_propensity=True, ci=True)
```



5.7 DragonNet vs Meta-Learners Benchmark with IHDP + Synthetic Datasets

`Dragonnet` requires `tensorflow`. If you haven't, please install `tensorflow` as follows:

```
pip install tensorflow
```

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression
```

(continues on next page)

(continued from previous page)

```
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error as mse
from scipy.stats import entropy
import warnings

from causalml.inference.meta import LRSRegressor
from causalml.inference.meta import XGBTRegressor, MLPTRegressor
from causalml.inference.meta import BaseXRegressor, BaseRRegressor, BaseSRegressor,␣
↪BaseTRegressor
from causalml.inference.tf import DragonNet
from causalml.match import NearestNeighborMatch, MatchOptimizer, create_table_one
from causalml.propensity import ElasticNetPropensityModel
from causalml.dataset.regression import *
from causalml.metrics import *

import os, sys

%matplotlib inline

warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
sns.set_palette('Paired')
plt.rcParams['figure.figsize'] = (12,8)
```

5.7.1 IHDP semi-synthetic dataset

Hill introduced a semi-synthetic dataset constructed from the Infant Health and Development Program (IHDP). This dataset is based on a randomized experiment investigating the effect of home visits by specialists on future cognitive scores. The data has 747 observations (rows). The IHDP simulation is considered the de-facto standard benchmark for neural network treatment effect estimation methods.

The original paper uses 1000 realizations from the NCPI package, but for illustration purposes, we use 1 dataset (realization) as an example below.

[3]:
```
df = pd.read_csv(f'data/ihdp_npci_3.csv', header=None)
cols =  ["treatment", "y_factual", "y_cfactual", "mu0", "mu1"] + [f'x{i}' for i in␣
↪range(1,26)]
df.columns = cols
```

[4]:
```
df.shape
```

[4]:
```
(747, 30)
```

[5]:
```
df.head()
```

[5]:

	treatment	y_factual	y_cfactual	mu0	mu1	x1	x2	\
0	1	5.931652	3.500591	2.253801	7.136441	-0.528603	-0.343455	
1	0	2.175966	5.952101	1.257592	6.553022	-1.736945	-1.802002	
2	0	2.180294	7.175734	2.384100	7.192645	-0.807451	-0.202946	
3	0	3.587662	7.787537	4.009365	7.712456	0.390083	0.596582	
4	0	2.372618	5.461871	2.481631	7.232739	-1.045229	-0.602710	

(continues on next page)

(continued from previous page)

```
         x3        x4        x5  …  x16  x17  x18  x19  x20  x21  x22  x23  \
0  1.128554  0.161703 -0.316603  …    1    1    1    1    0    0    0    0
1  0.383828  2.244320 -0.629189  …    1    1    1    1    0    0    0    0
2 -0.360898 -0.879606  0.808706  …    1    0    1    1    0    0    0    0
3 -1.850350 -0.879606 -0.004017  …    1    0    1    1    0    0    0    0
4  0.011465  0.161703  0.683672  …    1    1    1    1    0    0    0    0

   x24  x25
0    0    0
1    0    0
2    0    0
3    0    0
4    0    0

[5 rows x 30 columns]
```

[6]:
```
pd.Series(df['treatment']).value_counts(normalize=True)
```

[6]:
```
0    0.813922
1    0.186078
Name: treatment, dtype: float64
```

[7]:
```
X = df.loc[:,'x1':]
treatment = df['treatment']
y = df['y_factual']
tau = df.apply(lambda d: d['y_factual'] - d['y_cfactual'] if d['treatment']==1
               else d['y_cfactual'] - d['y_factual'],
               axis=1)
```

[9]:
```
p_model = ElasticNetPropensityModel()
p = p_model.fit_predict(X, treatment)
```

[10]:
```
s_learner = BaseSRegressor(LGBMRegressor())
s_ate = s_learner.estimate_ate(X, treatment, y)[0]
s_ite = s_learner.fit_predict(X, treatment, y)

t_learner = BaseTRegressor(LGBMRegressor())
t_ate = t_learner.estimate_ate(X, treatment, y)[0][0]
t_ite = t_learner.fit_predict(X, treatment, y)

x_learner = BaseXRegressor(LGBMRegressor())
x_ate = x_learner.estimate_ate(X, treatment, y, p)[0][0]
x_ite = x_learner.fit_predict(X, treatment, y, p)

r_learner = BaseRRegressor(LGBMRegressor())
r_ate = r_learner.estimate_ate(X, treatment, y, p)[0][0]
r_ite = r_learner.fit_predict(X, treatment, y, p)
```

[11]:
```
dragon = DragonNet(neurons_per_layer=200, targeted_reg=True)
dragon_ite = dragon.fit_predict(X, treatment, y, return_components=False)
dragon_ate = dragon_ite.mean()
```

```
Epoch 1/30
10/10 [==============================] - 5s 156ms/step - loss: 1790.3492 - regression_
↪loss: 864.6742 - binary_classification_loss: 41.3394 - treatment_accuracy: 0.7299 -␣
```

(continues on next page)

(continued from previous page)

```
↪track_epsilon: 0.0063 - val_loss: 242.1589 - val_regression_loss: 87.0011 - val_
↪binary_classification_loss: 32.6806 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0055
Epoch 2/30
10/10 [==============================] - 0s 7ms/step - loss: 311.9302 - regression_
↪loss: 135.2392 - binary_classification_loss: 32.8420 - treatment_accuracy: 0.8643 -␣
↪track_epsilon: 0.0059 - val_loss: 230.2209 - val_regression_loss: 79.8030 - val_
↪binary_classification_loss: 34.3533 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0047
Epoch 3/30
10/10 [==============================] - 0s 6ms/step - loss: 274.1216 - regression_
↪loss: 118.1561 - binary_classification_loss: 31.3200 - treatment_accuracy: 0.8169 -␣
↪track_epsilon: 0.0044 - val_loss: 238.4452 - val_regression_loss: 82.0530 - val_
↪binary_classification_loss: 36.2376 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0049
Epoch 4/30
10/10 [==============================] - 0s 6ms/step - loss: 205.4690 - regression_
↪loss: 85.9585 - binary_classification_loss: 27.2440 - treatment_accuracy: 0.8606 -␣
↪track_epsilon: 0.0053 - val_loss: 235.7122 - val_regression_loss: 78.5524 - val_
↪binary_classification_loss: 39.7929 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0057
Epoch 1/300
10/10 [==============================] - 1s 41ms/step - loss: 195.6840 - regression_
↪loss: 80.7820 - binary_classification_loss: 27.7316 - treatment_accuracy: 0.8497 -␣
↪track_epsilon: 0.0054 - val_loss: 207.3960 - val_regression_loss: 67.6306 - val_
↪binary_classification_loss: 38.1122 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0177
Epoch 2/300
10/10 [==============================] - 0s 6ms/step - loss: 183.9956 - regression_
↪loss: 75.3269 - binary_classification_loss: 26.4330 - treatment_accuracy: 0.8622 -␣
↪track_epsilon: 0.0182 - val_loss: 197.0267 - val_regression_loss: 64.0559 - val_
↪binary_classification_loss: 38.4298 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0117
Epoch 3/300
10/10 [==============================] - 0s 6ms/step - loss: 178.8321 - regression_
↪loss: 72.7892 - binary_classification_loss: 26.7587 - treatment_accuracy: 0.8555 -␣
↪track_epsilon: 0.0081 - val_loss: 195.6257 - val_regression_loss: 63.5609 - val_
↪binary_classification_loss: 38.2400 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0073
Epoch 4/300
10/10 [==============================] - 0s 6ms/step - loss: 177.0419 - regression_
↪loss: 71.8475 - binary_classification_loss: 27.1255 - treatment_accuracy: 0.8521 -␣
↪track_epsilon: 0.0091 - val_loss: 200.6521 - val_regression_loss: 65.3493 - val_
↪binary_classification_loss: 37.6216 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0082
Epoch 5/300
10/10 [==============================] - 0s 6ms/step - loss: 198.0597 - regression_
↪loss: 82.4320 - binary_classification_loss: 27.0536 - treatment_accuracy: 0.8497 -␣
↪track_epsilon: 0.0076 - val_loss: 194.4365 - val_regression_loss: 63.1230 - val_
↪binary_classification_loss: 37.8598 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0064
Epoch 6/300
10/10 [==============================] - 0s 5ms/step - loss: 174.1273 - regression_
↪loss: 70.2306 - binary_classification_loss: 27.7639 - treatment_accuracy: 0.8460 -␣
↪track_epsilon: 0.0075 - val_loss: 194.3751 - val_regression_loss: 63.1176 - val_
↪binary_classification_loss: 37.9318 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0100
```

(continues on next page)

(continued from previous page)

```
Epoch 7/300
10/10 [==============================] - 0s 6ms/step - loss: 187.2528 - regression_
↪loss: 77.2338 - binary_classification_loss: 26.6574 - treatment_accuracy: 0.8545 -␣
↪track_epsilon: 0.0094 - val_loss: 193.4222 - val_regression_loss: 62.8618 - val_
↪binary_classification_loss: 37.8932 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0100
Epoch 8/300
10/10 [==============================] - 0s 6ms/step - loss: 179.5122 - regression_
↪loss: 72.3961 - binary_classification_loss: 28.5867 - treatment_accuracy: 0.8357 -␣
↪track_epsilon: 0.0110 - val_loss: 196.3768 - val_regression_loss: 63.7690 - val_
↪binary_classification_loss: 37.5827 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0094
Epoch 9/300
10/10 [==============================] - 0s 6ms/step - loss: 180.4453 - regression_
↪loss: 74.0497 - binary_classification_loss: 26.0765 - treatment_accuracy: 0.8582 -␣
↪track_epsilon: 0.0077 - val_loss: 192.4576 - val_regression_loss: 62.1838 - val_
↪binary_classification_loss: 37.8295 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0104
Epoch 10/300
10/10 [==============================] - 0s 6ms/step - loss: 159.2664 - regression_
↪loss: 63.3608 - binary_classification_loss: 26.5497 - treatment_accuracy: 0.8544 -␣
↪track_epsilon: 0.0118 - val_loss: 192.8072 - val_regression_loss: 62.6150 - val_
↪binary_classification_loss: 38.0268 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0147
Epoch 11/300
10/10 [==============================] - 0s 6ms/step - loss: 157.3967 - regression_
↪loss: 62.1042 - binary_classification_loss: 27.1641 - treatment_accuracy: 0.8496 -␣
↪track_epsilon: 0.0139 - val_loss: 190.8307 - val_regression_loss: 61.5484 - val_
↪binary_classification_loss: 37.7637 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0105
Epoch 12/300
10/10 [==============================] - 0s 6ms/step - loss: 171.3408 - regression_
↪loss: 69.0045 - binary_classification_loss: 27.2531 - treatment_accuracy: 0.8468 -␣
↪track_epsilon: 0.0092 - val_loss: 190.0314 - val_regression_loss: 61.2129 - val_
↪binary_classification_loss: 37.7777 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0087
Epoch 13/300
10/10 [==============================] - 0s 6ms/step - loss: 162.0215 - regression_
↪loss: 62.9260 - binary_classification_loss: 30.3296 - treatment_accuracy: 0.8189 -␣
↪track_epsilon: 0.0084 - val_loss: 189.2025 - val_regression_loss: 60.8970 - val_
↪binary_classification_loss: 37.7255 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0126
Epoch 14/300
10/10 [==============================] - 0s 6ms/step - loss: 173.8554 - regression_
↪loss: 70.6637 - binary_classification_loss: 26.2371 - treatment_accuracy: 0.8540 -␣
↪track_epsilon: 0.0126 - val_loss: 189.1865 - val_regression_loss: 60.8833 - val_
↪binary_classification_loss: 37.6686 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0099
Epoch 15/300
10/10 [==============================] - 0s 6ms/step - loss: 157.4061 - regression_
↪loss: 63.5324 - binary_classification_loss: 24.4340 - treatment_accuracy: 0.8718 -␣
↪track_epsilon: 0.0093 - val_loss: 186.8761 - val_regression_loss: 60.0529 - val_
↪binary_classification_loss: 37.9590 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0089
Epoch 16/300
10/10 [==============================] - 0s 6ms/step - loss: 172.2053 - regression_
↪loss: 69.6182 - binary_classification_loss: 26.8678 - treatment_accuracy: 0.8502 -␣
```

(continues on next page)

(continued from previous page)

```
↪track_epsilon: 0.0080 - val_loss: 188.4488 - val_regression_loss: 60.4575 - val_
↪binary_classification_loss: 37.6228 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0113
Epoch 17/300
10/10 [==============================] - 0s 6ms/step - loss: 162.3663 - regression_
↪loss: 65.2318 - binary_classification_loss: 26.0364 - treatment_accuracy: 0.8562 -␣
↪track_epsilon: 0.0105 - val_loss: 186.5810 - val_regression_loss: 59.7591 - val_
↪binary_classification_loss: 37.6693 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0114
Epoch 18/300
10/10 [==============================] - 0s 6ms/step - loss: 163.8466 - regression_
↪loss: 65.5130 - binary_classification_loss: 26.7643 - treatment_accuracy: 0.8503 -␣
↪track_epsilon: 0.0119 - val_loss: 190.8825 - val_regression_loss: 62.1830 - val_
↪binary_classification_loss: 37.9718 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0087
Epoch 19/300
10/10 [==============================] - 0s 6ms/step - loss: 167.6180 - regression_
↪loss: 68.2214 - binary_classification_loss: 25.3027 - treatment_accuracy: 0.8620 -␣
↪track_epsilon: 0.0066 - val_loss: 185.6225 - val_regression_loss: 59.4746 - val_
↪binary_classification_loss: 37.6837 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0060
Epoch 20/300
10/10 [==============================] - 0s 6ms/step - loss: 168.5476 - regression_
↪loss: 68.3371 - binary_classification_loss: 25.8652 - treatment_accuracy: 0.8578 -␣
↪track_epsilon: 0.0079 - val_loss: 184.5200 - val_regression_loss: 59.2031 - val_
↪binary_classification_loss: 37.5099 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0098
Epoch 21/300
10/10 [==============================] - 0s 6ms/step - loss: 157.9161 - regression_
↪loss: 63.3173 - binary_classification_loss: 25.2899 - treatment_accuracy: 0.8634 -␣
↪track_epsilon: 0.0083 - val_loss: 185.0839 - val_regression_loss: 59.1452 - val_
↪binary_classification_loss: 37.5366 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0079
Epoch 22/300
10/10 [==============================] - 0s 6ms/step - loss: 160.4739 - regression_
↪loss: 63.1629 - binary_classification_loss: 28.0595 - treatment_accuracy: 0.8358 -␣
↪track_epsilon: 0.0086 - val_loss: 183.9351 - val_regression_loss: 59.1525 - val_
↪binary_classification_loss: 37.5067 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0066
Epoch 23/300
10/10 [==============================] - 0s 6ms/step - loss: 155.0890 - regression_
↪loss: 60.5116 - binary_classification_loss: 28.0962 - treatment_accuracy: 0.8349 -␣
↪track_epsilon: 0.0046 - val_loss: 183.6170 - val_regression_loss: 58.7653 - val_
↪binary_classification_loss: 37.2800 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0051
Epoch 24/300
10/10 [==============================] - 0s 6ms/step - loss: 149.4288 - regression_
↪loss: 58.8520 - binary_classification_loss: 25.9568 - treatment_accuracy: 0.8546 -␣
↪track_epsilon: 0.0052 - val_loss: 188.7191 - val_regression_loss: 60.7916 - val_
↪binary_classification_loss: 37.1127 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0069
Epoch 25/300
10/10 [==============================] - 0s 6ms/step - loss: 156.3095 - regression_
↪loss: 61.0641 - binary_classification_loss: 28.1708 - treatment_accuracy: 0.8315 -␣
↪track_epsilon: 0.0080 - val_loss: 182.5451 - val_regression_loss: 58.6875 - val_
↪binary_classification_loss: 37.3179 - val_treatment_accuracy: 0.7244 - val_track_
```

(continues on next page)

(continued from previous page)

```
↪epsilon: 0.0075
Epoch 26/300
10/10 [==============================] - 0s 6ms/step - loss: 154.8900 - regression_
↪loss: 61.1673 - binary_classification_loss: 26.2975 - treatment_accuracy: 0.8542 -␣
↪track_epsilon: 0.0059 - val_loss: 184.6580 - val_regression_loss: 59.6472 - val_
↪binary_classification_loss: 37.4789 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0040
Epoch 27/300
10/10 [==============================] - 0s 6ms/step - loss: 153.7275 - regression_
↪loss: 60.6342 - binary_classification_loss: 26.5628 - treatment_accuracy: 0.8494 -␣
↪track_epsilon: 0.0054 - val_loss: 187.6736 - val_regression_loss: 60.4940 - val_
↪binary_classification_loss: 37.1448 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0046
Epoch 28/300
10/10 [==============================] - 0s 6ms/step - loss: 159.4707 - regression_
↪loss: 63.5524 - binary_classification_loss: 26.3749 - treatment_accuracy: 0.8515 -␣
↪track_epsilon: 0.0043 - val_loss: 185.6613 - val_regression_loss: 60.3003 - val_
↪binary_classification_loss: 37.4100 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0041
Epoch 29/300
10/10 [==============================] - 0s 6ms/step - loss: 144.6116 - regression_
↪loss: 57.2770 - binary_classification_loss: 24.2153 - treatment_accuracy: 0.8699 -␣
↪track_epsilon: 0.0037 - val_loss: 183.7683 - val_regression_loss: 58.8389 - val_
↪binary_classification_loss: 37.2161 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0046
Epoch 30/300
10/10 [==============================] - 0s 6ms/step - loss: 156.1744 - regression_
↪loss: 61.9692 - binary_classification_loss: 26.0622 - treatment_accuracy: 0.8516 -␣
↪track_epsilon: 0.0058 - val_loss: 181.6741 - val_regression_loss: 58.3027 - val_
↪binary_classification_loss: 37.2483 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0056
Epoch 31/300
10/10 [==============================] - 0s 6ms/step - loss: 149.5090 - regression_
↪loss: 59.6090 - binary_classification_loss: 24.2077 - treatment_accuracy: 0.8685 -␣
↪track_epsilon: 0.0052 - val_loss: 183.1471 - val_regression_loss: 58.8850 - val_
↪binary_classification_loss: 37.3616 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0048
Epoch 32/300
10/10 [==============================] - 0s 6ms/step - loss: 158.8317 - regression_
↪loss: 63.1364 - binary_classification_loss: 26.3766 - treatment_accuracy: 0.8524 -␣
↪track_epsilon: 0.0040 - val_loss: 182.8958 - val_regression_loss: 58.5878 - val_
↪binary_classification_loss: 37.0665 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0047
Epoch 33/300
10/10 [==============================] - 0s 6ms/step - loss: 153.5294 - regression_
↪loss: 59.8976 - binary_classification_loss: 27.7524 - treatment_accuracy: 0.8380 -␣
↪track_epsilon: 0.0065 - val_loss: 184.9241 - val_regression_loss: 59.4232 - val_
↪binary_classification_loss: 36.8858 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0053
Epoch 34/300
10/10 [==============================] - 0s 6ms/step - loss: 149.8718 - regression_
↪loss: 59.2581 - binary_classification_loss: 25.4000 - treatment_accuracy: 0.8586 -␣
↪track_epsilon: 0.0050 - val_loss: 181.1425 - val_regression_loss: 58.2419 - val_
↪binary_classification_loss: 37.1219 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0023
Epoch 35/300
10/10 [==============================] - 0s 6ms/step - loss: 147.3198 - regression_
```

(continues on next page)

(continued from previous page)

```
↪loss: 58.8689 - binary_classification_loss: 23.9842 - treatment_accuracy: 0.8717 -␣
↪track_epsilon: 0.0020 - val_loss: 182.3788 - val_regression_loss: 58.5675 - val_
↪binary_classification_loss: 37.1934 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0054
Epoch 36/300
10/10 [==============================] - 0s 6ms/step - loss: 143.2286 - regression_
↪loss: 55.5299 - binary_classification_loss: 26.2675 - treatment_accuracy: 0.8521 -␣
↪track_epsilon: 0.0059 - val_loss: 183.0977 - val_regression_loss: 59.1656 - val_
↪binary_classification_loss: 37.0963 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0033
Epoch 37/300
10/10 [==============================] - 0s 6ms/step - loss: 149.7070 - regression_
↪loss: 59.5447 - binary_classification_loss: 24.7418 - treatment_accuracy: 0.8625 -␣
↪track_epsilon: 0.0023 - val_loss: 183.3780 - val_regression_loss: 58.6777 - val_
↪binary_classification_loss: 37.0991 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0018
Epoch 38/300
10/10 [==============================] - 0s 6ms/step - loss: 156.6596 - regression_
↪loss: 62.0057 - binary_classification_loss: 26.7843 - treatment_accuracy: 0.8461 -␣
↪track_epsilon: 0.0034 - val_loss: 183.2619 - val_regression_loss: 58.6840 - val_
↪binary_classification_loss: 36.8925 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0030
Epoch 39/300
10/10 [==============================] - 0s 6ms/step - loss: 155.1980 - regression_
↪loss: 60.9731 - binary_classification_loss: 27.2335 - treatment_accuracy: 0.8443 -␣
↪track_epsilon: 0.0027 - val_loss: 181.5153 - val_regression_loss: 58.3543 - val_
↪binary_classification_loss: 37.0235 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0026
Epoch 40/300
10/10 [==============================] - 0s 6ms/step - loss: 148.6864 - regression_
↪loss: 58.6236 - binary_classification_loss: 25.4435 - treatment_accuracy: 0.8563 -␣
↪track_epsilon: 0.0023 - val_loss: 181.4140 - val_regression_loss: 58.1816 - val_
↪binary_classification_loss: 36.9654 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0040

Epoch 00040: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.
Epoch 41/300
10/10 [==============================] - 0s 6ms/step - loss: 144.7339 - regression_
↪loss: 56.2116 - binary_classification_loss: 26.5446 - treatment_accuracy: 0.8501 -␣
↪track_epsilon: 0.0048 - val_loss: 180.4425 - val_regression_loss: 57.9971 - val_
↪binary_classification_loss: 36.8370 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0046
Epoch 42/300
10/10 [==============================] - 0s 6ms/step - loss: 152.6819 - regression_
↪loss: 59.9780 - binary_classification_loss: 26.7807 - treatment_accuracy: 0.8460 -␣
↪track_epsilon: 0.0035 - val_loss: 181.0754 - val_regression_loss: 58.0677 - val_
↪binary_classification_loss: 36.8611 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0022
Epoch 43/300
10/10 [==============================] - 0s 6ms/step - loss: 141.4455 - regression_
↪loss: 55.1271 - binary_classification_loss: 25.5353 - treatment_accuracy: 0.8533 -␣
↪track_epsilon: 0.0021 - val_loss: 181.6914 - val_regression_loss: 58.2652 - val_
↪binary_classification_loss: 36.9244 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0029
Epoch 44/300
10/10 [==============================] - 0s 6ms/step - loss: 143.5718 - regression_
```

(continues on next page)

(continued from previous page)

```
↪loss: 54.9356 - binary_classification_loss: 27.7278 - treatment_accuracy: 0.8368 -␣
↪track_epsilon: 0.0035 - val_loss: 182.9616 - val_regression_loss: 58.7127 - val_
↪binary_classification_loss: 36.7605 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0030
Epoch 45/300
10/10 [==============================] - 0s 6ms/step - loss: 148.0318 - regression_
↪loss: 58.2297 - binary_classification_loss: 25.4171 - treatment_accuracy: 0.8623 -␣
↪track_epsilon: 0.0028 - val_loss: 182.2657 - val_regression_loss: 58.7829 - val_
↪binary_classification_loss: 36.9492 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0031
Epoch 46/300
10/10 [==============================] - 0s 6ms/step - loss: 148.2912 - regression_
↪loss: 58.4527 - binary_classification_loss: 25.4111 - treatment_accuracy: 0.8550 -␣
↪track_epsilon: 0.0034 - val_loss: 181.4299 - val_regression_loss: 58.1758 - val_
↪binary_classification_loss: 36.8661 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0024
Epoch 47/300
10/10 [==============================] - 0s 6ms/step - loss: 149.8918 - regression_
↪loss: 57.9189 - binary_classification_loss: 28.0705 - treatment_accuracy: 0.8318 -␣
↪track_epsilon: 0.0019 - val_loss: 181.7516 - val_regression_loss: 58.2234 - val_
↪binary_classification_loss: 36.8086 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0017
Epoch 48/300
10/10 [==============================] - 0s 6ms/step - loss: 143.9826 - regression_
↪loss: 56.0217 - binary_classification_loss: 25.9636 - treatment_accuracy: 0.8518 -␣
↪track_epsilon: 0.0018 - val_loss: 183.2366 - val_regression_loss: 58.7730 - val_
↪binary_classification_loss: 36.7622 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0020

Epoch 00048: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-06.
Epoch 49/300
10/10 [==============================] - 0s 6ms/step - loss: 141.9178 - regression_
↪loss: 54.9617 - binary_classification_loss: 26.0167 - treatment_accuracy: 0.8551 -␣
↪track_epsilon: 0.0025 - val_loss: 181.6692 - val_regression_loss: 58.2019 - val_
↪binary_classification_loss: 36.8163 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0035
Epoch 50/300
10/10 [==============================] - 0s 6ms/step - loss: 154.0470 - regression_
↪loss: 60.6821 - binary_classification_loss: 26.7084 - treatment_accuracy: 0.8442 -␣
↪track_epsilon: 0.0037 - val_loss: 181.5136 - val_regression_loss: 58.1967 - val_
↪binary_classification_loss: 36.7926 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0030
Epoch 51/300
10/10 [==============================] - 0s 6ms/step - loss: 154.2879 - regression_
↪loss: 61.1156 - binary_classification_loss: 25.9554 - treatment_accuracy: 0.8530 -␣
↪track_epsilon: 0.0026 - val_loss: 181.1187 - val_regression_loss: 58.0944 - val_
↪binary_classification_loss: 36.7854 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0019
Epoch 52/300
10/10 [==============================] - 0s 6ms/step - loss: 147.1910 - regression_
↪loss: 57.9212 - binary_classification_loss: 25.5444 - treatment_accuracy: 0.8585 -␣
↪track_epsilon: 0.0019 - val_loss: 180.9492 - val_regression_loss: 58.0477 - val_
↪binary_classification_loss: 36.8212 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0023
Epoch 53/300
10/10 [==============================] - 0s 6ms/step - loss: 144.5095 - regression_
↪loss: 57.1991 - binary_classification_loss: 24.5623 - treatment_accuracy: 0.8633 -␣
```

(continues on next page)

(continued from previous page)

```
↪track_epsilon: 0.0025 - val_loss: 181.2697 - val_regression_loss: 58.0844 - val_
↪binary_classification_loss: 36.8072 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0025
Epoch 54/300
10/10 [==============================] - 0s 6ms/step - loss: 149.1749 - regression_
↪loss: 58.6545 - binary_classification_loss: 26.4255 - treatment_accuracy: 0.8508 -␣
↪track_epsilon: 0.0027 - val_loss: 181.6855 - val_regression_loss: 58.2050 - val_
↪binary_classification_loss: 36.8246 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0024
Epoch 55/300
10/10 [==============================] - 0s 6ms/step - loss: 143.3488 - regression_
↪loss: 57.4108 - binary_classification_loss: 22.5247 - treatment_accuracy: 0.8810 -␣
↪track_epsilon: 0.0018 - val_loss: 181.5240 - val_regression_loss: 58.1889 - val_
↪binary_classification_loss: 36.8670 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0017
Epoch 56/300
10/10 [==============================] - 0s 6ms/step - loss: 145.3050 - regression_
↪loss: 57.0797 - binary_classification_loss: 25.3895 - treatment_accuracy: 0.8567 -␣
↪track_epsilon: 0.0020 - val_loss: 181.2868 - val_regression_loss: 58.1061 - val_
↪binary_classification_loss: 36.7694 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0020
Epoch 57/300
10/10 [==============================] - 0s 6ms/step - loss: 150.1354 - regression_
↪loss: 58.8804 - binary_classification_loss: 26.4138 - treatment_accuracy: 0.8482 -␣
↪track_epsilon: 0.0023 - val_loss: 181.1185 - val_regression_loss: 58.1035 - val_
↪binary_classification_loss: 36.8033 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0027
Epoch 58/300
10/10 [==============================] - 0s 6ms/step - loss: 145.2017 - regression_
↪loss: 57.0173 - binary_classification_loss: 25.5667 - treatment_accuracy: 0.8535 -␣
↪track_epsilon: 0.0028 - val_loss: 181.2872 - val_regression_loss: 58.0891 - val_
↪binary_classification_loss: 36.7888 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0023
Epoch 59/300
10/10 [==============================] - 0s 6ms/step - loss: 147.7799 - regression_
↪loss: 57.4570 - binary_classification_loss: 26.7010 - treatment_accuracy: 0.8456 -␣
↪track_epsilon: 0.0022 - val_loss: 181.2169 - val_regression_loss: 58.0942 - val_
↪binary_classification_loss: 36.7608 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0024

Epoch 00059: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-06.
Epoch 60/300
10/10 [==============================] - 0s 6ms/step - loss: 143.9625 - regression_
↪loss: 56.2486 - binary_classification_loss: 25.4459 - treatment_accuracy: 0.8584 -␣
↪track_epsilon: 0.0021 - val_loss: 180.9821 - val_regression_loss: 58.0493 - val_
↪binary_classification_loss: 36.7809 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0018
Epoch 61/300
10/10 [==============================] - 0s 6ms/step - loss: 145.4550 - regression_
↪loss: 55.9254 - binary_classification_loss: 27.6142 - treatment_accuracy: 0.8356 -␣
↪track_epsilon: 0.0018 - val_loss: 181.0581 - val_regression_loss: 58.0568 - val_
↪binary_classification_loss: 36.7708 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0018
Epoch 62/300
10/10 [==============================] - 0s 6ms/step - loss: 146.6129 - regression_
↪loss: 57.8608 - binary_classification_loss: 24.9736 - treatment_accuracy: 0.8610 -␣
↪track_epsilon: 0.0019 - val_loss: 181.3456 - val_regression_loss: 58.1192 - val_
```

(continues on next page)

(continued from previous page)

```
↪binary_classification_loss: 36.7725 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0019
Epoch 63/300
10/10 [==============================] - 0s 6ms/step - loss: 148.2220 - regression_
↪loss: 58.4927 - binary_classification_loss: 25.3438 - treatment_accuracy: 0.8560 -␣
↪track_epsilon: 0.0019 - val_loss: 181.4519 - val_regression_loss: 58.1659 - val_
↪binary_classification_loss: 36.7662 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0019
Epoch 64/300
10/10 [==============================] - 0s 6ms/step - loss: 140.0457 - regression_
↪loss: 53.7192 - binary_classification_loss: 26.7202 - treatment_accuracy: 0.8449 -␣
↪track_epsilon: 0.0018 - val_loss: 181.4884 - val_regression_loss: 58.1761 - val_
↪binary_classification_loss: 36.7904 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0018

Epoch 00064: ReduceLROnPlateau reducing learning rate to 6.24999984211172e-07.
Epoch 65/300
10/10 [==============================] - 0s 6ms/step - loss: 138.7515 - regression_
↪loss: 54.9897 - binary_classification_loss: 22.9222 - treatment_accuracy: 0.8764 -␣
↪track_epsilon: 0.0019 - val_loss: 181.4227 - val_regression_loss: 58.1461 - val_
↪binary_classification_loss: 36.7599 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021
Epoch 66/300
10/10 [==============================] - 0s 6ms/step - loss: 150.2262 - regression_
↪loss: 59.0825 - binary_classification_loss: 26.3098 - treatment_accuracy: 0.8488 -␣
↪track_epsilon: 0.0021 - val_loss: 181.3573 - val_regression_loss: 58.1316 - val_
↪binary_classification_loss: 36.7465 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0022
Epoch 67/300
10/10 [==============================] - 0s 5ms/step - loss: 149.6757 - regression_
↪loss: 59.4637 - binary_classification_loss: 24.7313 - treatment_accuracy: 0.8603 -␣
↪track_epsilon: 0.0021 - val_loss: 181.3223 - val_regression_loss: 58.1212 - val_
↪binary_classification_loss: 36.7481 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021
Epoch 68/300
10/10 [==============================] - 0s 5ms/step - loss: 151.1728 - regression_
↪loss: 60.0713 - binary_classification_loss: 25.0154 - treatment_accuracy: 0.8615 -␣
↪track_epsilon: 0.0021 - val_loss: 181.1846 - val_regression_loss: 58.0904 - val_
↪binary_classification_loss: 36.7660 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0020
Epoch 69/300
10/10 [==============================] - 0s 6ms/step - loss: 148.2535 - regression_
↪loss: 59.0331 - binary_classification_loss: 24.1994 - treatment_accuracy: 0.8658 -␣
↪track_epsilon: 0.0021 - val_loss: 181.2948 - val_regression_loss: 58.1039 - val_
↪binary_classification_loss: 36.7439 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0023

Epoch 00069: ReduceLROnPlateau reducing learning rate to 3.12499992105586e-07.
Epoch 70/300
10/10 [==============================] - 0s 6ms/step - loss: 145.3151 - regression_
↪loss: 57.0691 - binary_classification_loss: 25.4983 - treatment_accuracy: 0.8584 -␣
↪track_epsilon: 0.0023 - val_loss: 181.3544 - val_regression_loss: 58.1245 - val_
↪binary_classification_loss: 36.7449 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0023
Epoch 71/300
10/10 [==============================] - 0s 6ms/step - loss: 150.1866 - regression_
```

(continues on next page)

(continued from previous page)

```
↪loss: 57.9752 - binary_classification_loss: 28.1966 - treatment_accuracy: 0.8297 -␣
↪track_epsilon: 0.0023 - val_loss: 181.2958 - val_regression_loss: 58.1128 - val_
↪binary_classification_loss: 36.7442 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0022
Epoch 72/300
10/10 [==============================] - 0s 6ms/step - loss: 144.9820 - regression_
↪loss: 57.4340 - binary_classification_loss: 24.4047 - treatment_accuracy: 0.8619 -␣
↪track_epsilon: 0.0022 - val_loss: 181.3424 - val_regression_loss: 58.1227 - val_
↪binary_classification_loss: 36.7440 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0023
Epoch 73/300
10/10 [==============================] - 0s 6ms/step - loss: 148.8112 - regression_
↪loss: 58.0125 - binary_classification_loss: 26.8692 - treatment_accuracy: 0.8447 -␣
↪track_epsilon: 0.0022 - val_loss: 181.3199 - val_regression_loss: 58.1187 - val_
↪binary_classification_loss: 36.7438 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021
Epoch 74/300
10/10 [==============================] - 0s 6ms/step - loss: 144.3984 - regression_
↪loss: 56.9031 - binary_classification_loss: 24.6382 - treatment_accuracy: 0.8624 -␣
↪track_epsilon: 0.0021 - val_loss: 181.3810 - val_regression_loss: 58.1361 - val_
↪binary_classification_loss: 36.7440 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021

Epoch 00074: ReduceLROnPlateau reducing learning rate to 1.56249996052793e-07.
Epoch 75/300
10/10 [==============================] - 0s 6ms/step - loss: 147.5547 - regression_
↪loss: 57.7667 - binary_classification_loss: 26.1622 - treatment_accuracy: 0.8478 -␣
↪track_epsilon: 0.0021 - val_loss: 181.3161 - val_regression_loss: 58.1183 - val_
↪binary_classification_loss: 36.7473 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021
Epoch 76/300
10/10 [==============================] - 0s 6ms/step - loss: 140.5001 - regression_
↪loss: 53.5784 - binary_classification_loss: 27.3214 - treatment_accuracy: 0.8388 -␣
↪track_epsilon: 0.0021 - val_loss: 181.2723 - val_regression_loss: 58.1086 - val_
↪binary_classification_loss: 36.7488 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0022
Epoch 77/300
10/10 [==============================] - 0s 6ms/step - loss: 143.8736 - regression_
↪loss: 55.9839 - binary_classification_loss: 26.1250 - treatment_accuracy: 0.8466 -␣
↪track_epsilon: 0.0022 - val_loss: 181.2639 - val_regression_loss: 58.1073 - val_
↪binary_classification_loss: 36.7513 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021
Epoch 78/300
10/10 [==============================] - 0s 6ms/step - loss: 146.6917 - regression_
↪loss: 58.5758 - binary_classification_loss: 23.5315 - treatment_accuracy: 0.8700 -␣
↪track_epsilon: 0.0022 - val_loss: 181.2961 - val_regression_loss: 58.1147 - val_
↪binary_classification_loss: 36.7518 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0022
Epoch 79/300
10/10 [==============================] - 0s 6ms/step - loss: 143.4007 - regression_
↪loss: 54.8006 - binary_classification_loss: 27.7054 - treatment_accuracy: 0.8383 -␣
↪track_epsilon: 0.0021 - val_loss: 181.3115 - val_regression_loss: 58.1188 - val_
↪binary_classification_loss: 36.7477 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021

Epoch 00079: ReduceLROnPlateau reducing learning rate to 7.81249980263965e-08.
Epoch 80/300
```

(continues on next page)

(continued from previous page)

```
10/10 [==============================] - 0s 6ms/step - loss: 145.6183 - regression_
↪loss: 57.3271 - binary_classification_loss: 25.1687 - treatment_accuracy: 0.8574 -␣
↪track_epsilon: 0.0021 - val_loss: 181.2945 - val_regression_loss: 58.1152 - val_
↪binary_classification_loss: 36.7475 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0021
Epoch 81/300
10/10 [==============================] - 0s 6ms/step - loss: 142.3395 - regression_
↪loss: 54.9129 - binary_classification_loss: 26.6164 - treatment_accuracy: 0.8461 -␣
↪track_epsilon: 0.0021 - val_loss: 181.3037 - val_regression_loss: 58.1177 - val_
↪binary_classification_loss: 36.7449 - val_treatment_accuracy: 0.7244 - val_track_
↪epsilon: 0.0022
```

[12]:
```
df_preds = pd.DataFrame([s_ite.ravel(),
                         t_ite.ravel(),
                         x_ite.ravel(),
                         r_ite.ravel(),
                         dragon_ite.ravel(),
                         tau.ravel(),
                         treatment.ravel(),
                         y.ravel()],
                        index=['S','T','X','R','dragonnet','tau','w','y']).T

df_cumgain = get_cumgain(df_preds)
```

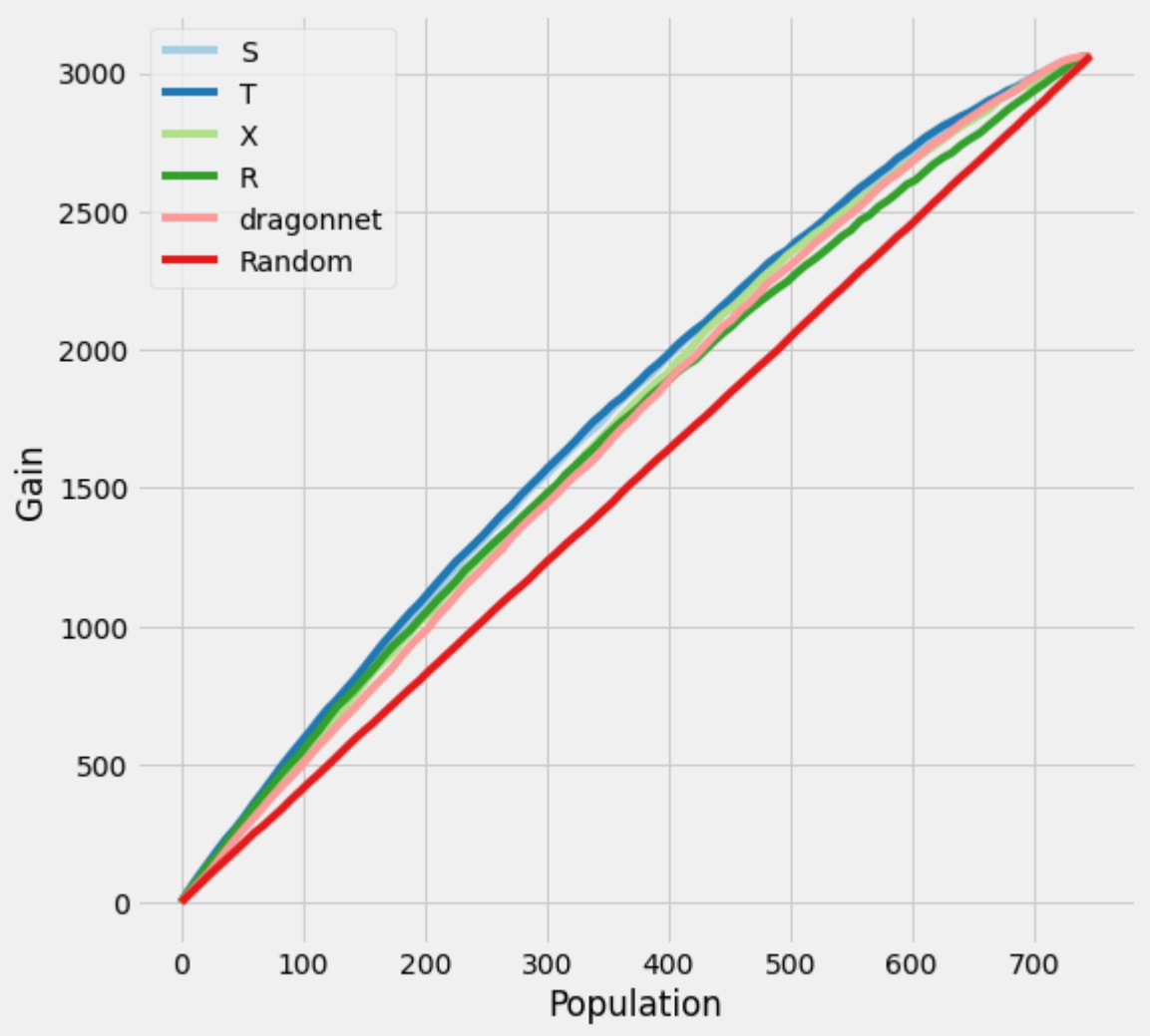
[13]:
```
df_result = pd.DataFrame([s_ate, t_ate, x_ate, r_ate, dragon_ate, tau.mean()],
                      index=['S','T','X','R','dragonnet','actual'], columns=['ATE'])
df_result['MAE'] = [mean_absolute_error(t,p) for t,p in zip([s_ite, t_ite, x_ite, r_
↪ite, dragon_ite],
                                                            [tau.values.reshape(-1,
↪1)]*5 )
                ] + [None]
df_result['AUUC'] = auuc_score(df_preds)
```

[14]:
```
df_result
```

[14]:

	ATE	MAE	AUUC
S	4.054511	1.027666	0.575822
T	4.100199	0.980788	0.580929
X	4.020918	1.116303	0.564651
R	4.257976	1.665557	0.556855
dragonnet	4.006536	1.165061	0.556426
actual	4.098887	NaN	NaN

[15]:
```
plot_gain(df_preds)
```



5.7.2 Synthetic Data Generation

[16]:
```
y, X, w, tau, b, e = simulate_nuisance_and_easy_treatment(n=1000)

X_train, X_val, y_train, y_val, w_train, w_val, tau_train, tau_val, b_train, b_val, e_
↪train, e_val = \
    train_test_split(X, y, w, tau, b, e, test_size=0.2, random_state=123,␣
↪shuffle=True)

preds_dict_train = {}
preds_dict_valid = {}

preds_dict_train['Actuals'] = tau_train
preds_dict_valid['Actuals'] = tau_val

preds_dict_train['generated_data'] = {
    'y': y_train,
    'X': X_train,
    'w': w_train,
```

(continues on next page)

(continued from previous page)

```
    'tau': tau_train,
    'b': b_train,
    'e': e_train}
preds_dict_valid['generated_data'] = {
    'y': y_val,
    'X': X_val,
    'w': w_val,
    'tau': tau_val,
    'b': b_val,
    'e': e_val}

# Predict p_hat because e would not be directly observed in real-life
p_model = ElasticNetPropensityModel()
p_hat_train = p_model.fit_predict(X_train, w_train)
p_hat_val = p_model.fit_predict(X_val, w_val)

for base_learner, label_l in zip([BaseSRegressor, BaseTRegressor, BaseXRegressor,␣
↪BaseRRegressor],
                                 ['S', 'T', 'X', 'R']):
    for model, label_m in zip([LinearRegression, XGBRegressor], ['LR', 'XGB']):
        # RLearner will need to fit on the p_hat
        if label_l != 'R':
            learner = base_learner(model())
            # fit the model on training data only
            learner.fit(X=X_train, treatment=w_train, y=y_train)
            try:
                preds_dict_train['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_train, p=p_hat_train).
↪flatten()
                preds_dict_valid['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_val, p=p_hat_val).
↪flatten()
            except TypeError:
                preds_dict_train['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_train, treatment=w_train,
↪ y=y_train).flatten()
                preds_dict_valid['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_val, treatment=w_val,␣
↪y=y_val).flatten()
        else:
            learner = base_learner(model())
            learner.fit(X=X_train, p=p_hat_train, treatment=w_train, y=y_train)
            preds_dict_train['{} Learner ({})'.format(
                label_l, label_m)] = learner.predict(X=X_train).flatten()
            preds_dict_valid['{} Learner ({})'.format(
                label_l, label_m)] = learner.predict(X=X_val).flatten()

learner = DragonNet(verbose=False)
learner.fit(X_train, treatment=w_train, y=y_train)
preds_dict_train['DragonNet'] = learner.predict_tau(X=X_train).flatten()
preds_dict_valid['DragonNet'] = learner.predict_tau(X=X_val).flatten()
```

[17]:
```
actuals_train = preds_dict_train['Actuals']
actuals_validation = preds_dict_valid['Actuals']

synthetic_summary_train = pd.DataFrame({label: [preds.mean(), mse(preds, actuals_
```

(continues on next page)

(continued from previous page)

```
↪train)] for label, preds
                                          in preds_dict_train.items() if 'generated'␣
↪not in label.lower()},
                                         index=['ATE', 'MSE']).T
synthetic_summary_train['Abs % Error of ATE'] = np.abs(
    (synthetic_summary_train['ATE']/synthetic_summary_train.loc['Actuals', 'ATE']) -␣
↪1)

synthetic_summary_validation = pd.DataFrame({label: [preds.mean(), mse(preds, actuals_
↪validation)]
                                             for label, preds in preds_dict_valid.
↪items()
                                             if 'generated' not in label.lower()},
                                            index=['ATE', 'MSE']).T
synthetic_summary_validation['Abs % Error of ATE'] = np.abs(
    (synthetic_summary_validation['ATE']/synthetic_summary_validation.loc['Actuals',
↪'ATE']) - 1)

# calculate kl divergence for training
for label in synthetic_summary_train.index:
    stacked_values = np.hstack((preds_dict_train[label], actuals_train))
    stacked_low = np.percentile(stacked_values, 0.1)
    stacked_high = np.percentile(stacked_values, 99.9)
    bins = np.linspace(stacked_low, stacked_high, 100)

    distr = np.histogram(preds_dict_train[label], bins=bins)[0]
    distr = np.clip(distr/distr.sum(), 0.001, 0.999)
    true_distr = np.histogram(actuals_train, bins=bins)[0]
    true_distr = np.clip(true_distr/true_distr.sum(), 0.001, 0.999)

    kl = entropy(distr, true_distr)
    synthetic_summary_train.loc[label, 'KL Divergence'] = kl

# calculate kl divergence for validation
for label in synthetic_summary_validation.index:
    stacked_values = np.hstack((preds_dict_valid[label], actuals_validation))
    stacked_low = np.percentile(stacked_values, 0.1)
    stacked_high = np.percentile(stacked_values, 99.9)
    bins = np.linspace(stacked_low, stacked_high, 100)

    distr = np.histogram(preds_dict_valid[label], bins=bins)[0]
    distr = np.clip(distr/distr.sum(), 0.001, 0.999)
    true_distr = np.histogram(actuals_validation, bins=bins)[0]
    true_distr = np.clip(true_distr/true_distr.sum(), 0.001, 0.999)

    kl = entropy(distr, true_distr)
    synthetic_summary_validation.loc[label, 'KL Divergence'] = kl
```

[18]:
```
df_preds_train = pd.DataFrame([preds_dict_train['S Learner (LR)'].ravel(),
                               preds_dict_train['S Learner (XGB)'].ravel(),
                               preds_dict_train['T Learner (LR)'].ravel(),
                               preds_dict_train['T Learner (XGB)'].ravel(),
                               preds_dict_train['X Learner (LR)'].ravel(),
                               preds_dict_train['X Learner (XGB)'].ravel(),
                               preds_dict_train['R Learner (LR)'].ravel(),
                               preds_dict_train['R Learner (XGB)'].ravel(),
```

(continues on next page)

(continued from previous page)

```
                                preds_dict_train['DragonNet'].ravel(),
                                preds_dict_train['generated_data']['tau'].ravel(),
                                preds_dict_train['generated_data']['w'].ravel(),
                                preds_dict_train['generated_data']['y'].ravel()],
                               index=['S Learner (LR)','S Learner (XGB)',
                                      'T Learner (LR)','T Learner (XGB)',
                                      'X Learner (LR)','X Learner (XGB)',
                                      'R Learner (LR)','R Learner (XGB)',
                                      'DragonNet','tau','w','y']).T

synthetic_summary_train['AUUC'] = auuc_score(df_preds_train).iloc[:-1]
```

[19]:
```
df_preds_validation = pd.DataFrame([preds_dict_valid['S Learner (LR)'].ravel(),
                                preds_dict_valid['S Learner (XGB)'].ravel(),
                                preds_dict_valid['T Learner (LR)'].ravel(),
                                preds_dict_valid['T Learner (XGB)'].ravel(),
                                preds_dict_valid['X Learner (LR)'].ravel(),
                                preds_dict_valid['X Learner (XGB)'].ravel(),
                                preds_dict_valid['R Learner (LR)'].ravel(),
                                preds_dict_valid['R Learner (XGB)'].ravel(),
                                preds_dict_valid['DragonNet'].ravel(),
                                preds_dict_valid['generated_data']['tau'].ravel(),
                                preds_dict_valid['generated_data']['w'].ravel(),
                                preds_dict_valid['generated_data']['y'].ravel()],
                               index=['S Learner (LR)','S Learner (XGB)',
                                      'T Learner (LR)','T Learner (XGB)',
                                      'X Learner (LR)','X Learner (XGB)',
                                      'R Learner (LR)','R Learner (XGB)',
                                      'DragonNet','tau','w','y']).T

synthetic_summary_validation['AUUC'] = auuc_score(df_preds_validation).iloc[:-1]
```

[20]:
```
synthetic_summary_train
```

[20]:

	ATE	MSE	Abs % Error of ATE	KL Divergence	AUUC
Actuals	0.484486	0.000000	0.000000	0.000000	NaN
S Learner (LR)	0.528743	0.044194	0.091349	3.473087	0.508067
S Learner (XGB)	0.358208	0.310652	0.260643	0.817620	0.544115
T Learner (LR)	0.493815	0.022688	0.019255	0.289978	0.610855
T Learner (XGB)	0.397053	1.350928	0.180466	1.452143	0.521719
X Learner (LR)	0.493815	0.022688	0.019255	0.289978	0.610855
X Learner (XGB)	0.341352	0.620992	0.295435	1.086086	0.534827
R Learner (LR)	0.457692	0.028116	0.055304	0.335083	
R Learner (XGB)	0.434709	4.575591	0.102741	1.907325	
DragonNet	0.410899	0.044120	0.151888	0.467829	

(continues on next page)

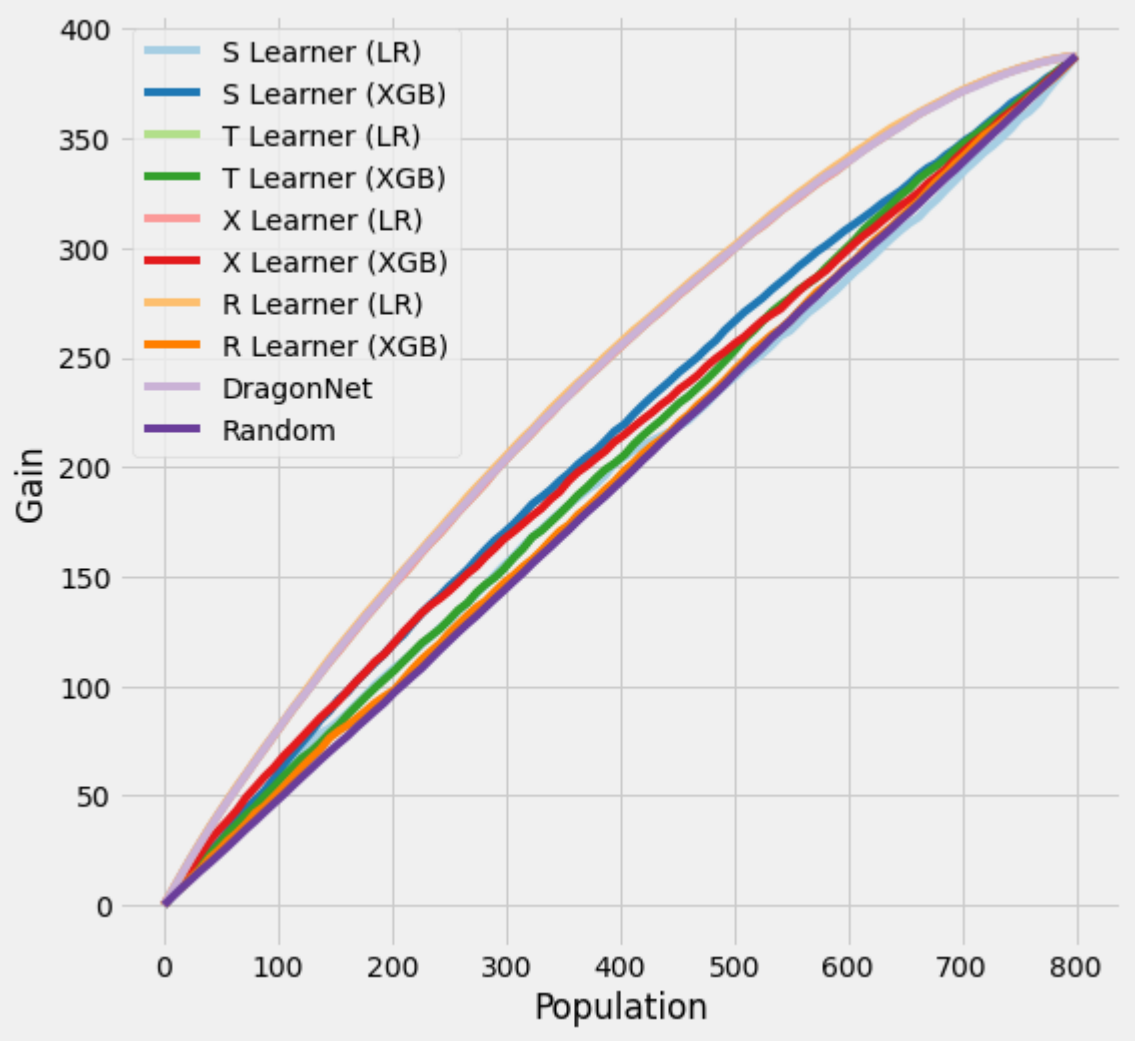
(continued from previous page)

```
R Learner (LR)    0.614414
R Learner (XGB)   0.505088
DragonNet         0.611620
```

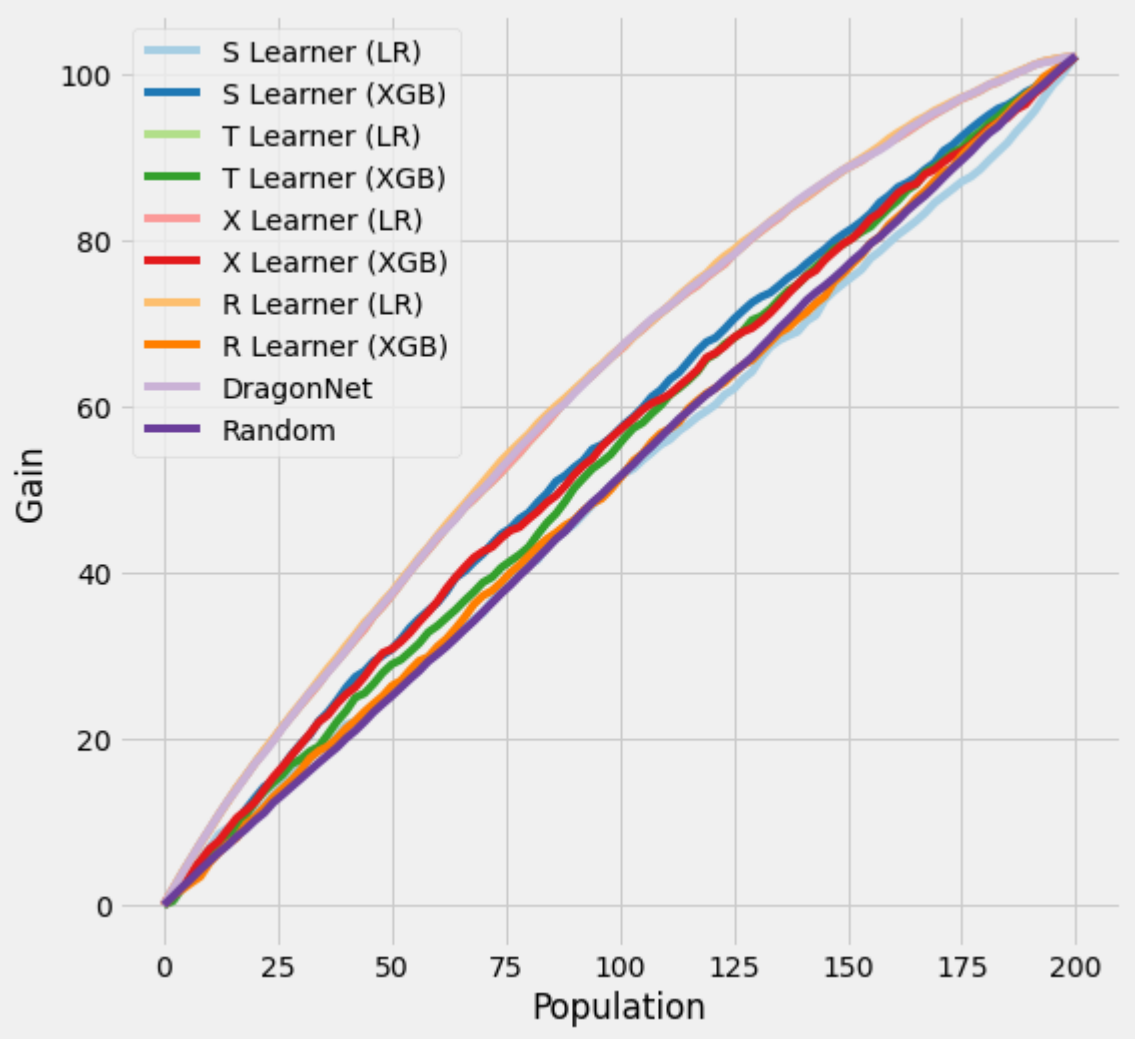
[21]:
```
synthetic_summary_validation
```

[21]:
```
                       ATE       MSE  Abs % Error of ATE  KL Divergence  \
Actuals           0.511242  0.000000            0.000000       0.000000
S Learner (LR)    0.528743  0.042236            0.034233       4.574498
S Learner (XGB)   0.434208  0.260496            0.150680       0.854890
T Learner (LR)    0.541503  0.025840            0.059191       0.686602
T Learner (XGB)   0.483404  0.679398            0.054452       1.215394
X Learner (LR)    0.541503  0.025840            0.059191       0.686602
X Learner (XGB)   0.330427  0.344865            0.353678       1.227041
R Learner (LR)    0.510236  0.030801            0.001967       0.654228
R Learner (XGB)   0.417823  1.990451            0.182730       1.650560
DragonNet         0.462146  0.043679            0.096032       0.825673

                     AUUC
Actuals               NaN
S Learner (LR)    0.495423
S Learner (XGB)   0.544206
T Learner (LR)    0.604712
T Learner (XGB)   0.526918
X Learner (LR)    0.604712
X Learner (XGB)   0.536599
R Learner (LR)    0.608133
R Learner (XGB)   0.504991
DragonNet         0.605744
```

[22]:
```
plot_gain(df_preds_train)
```



[23]:
```
plot_gain(df_preds_validation)
```



[ ]:

5.8 2SLS Benchmarks with NLSYM + Synthetic Datasets

We demonstrate the use of 2SLS from the package to estimate the average treatment effect by semi-synthetic data and full synthetic data.

[1]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

[2]:
```
import os
base_path = os.path.abspath("../")
os.chdir(base_path)
```

[52]:
```
import logging
from matplotlib import pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import numpy as np
import pandas as pd
import sys
from scipy import stats
```

[39]:
```
import causalml
from causalml.inference.iv import IVRegressor
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
```

5.8.1 Semi-Synthetic Data from NLSYM

The data generation mechanism is described in Syrgkanis et al "*Machine Learning Estimation of Heterogeneous Treatment Effects with Instruments*" (2019).

Data Loading

[5]:
```
df = pd.read_csv("examples/data/card.csv")
```

[6]:
```
df.head()
```

[6]:
```
   id  nearc2  nearc4  educ  age  fatheduc  motheduc  weight  momdad14  \
0   2       0       0     7   29       NaN       NaN  158413         1
1   3       0       0    12   27       8.0       8.0  380166         1
2   4       0       0    12   34      14.0      12.0  367470         1
3   5       1       1    11   27      11.0      12.0  380166         1
4   6       1       1    12   34       8.0       7.0  367470         1

   sinmom14  …  smsa66  wage  enroll   kww     iq  married  libcrd14  exper  \
0         0  …       1   548       0  15.0    NaN      1.0       0.0     16
1         0  …       1   481       0  35.0   93.0      1.0       1.0      9
2         0  …       1   721       0  42.0  103.0      1.0       1.0     16
3         0  …       1   250       0  25.0   88.0      1.0       1.0     10
4         0  …       1   729       0  34.0  108.0      1.0       0.0     16

      lwage  expersq
0  6.306275      256
1  6.175867       81
2  6.580639      256
3  5.521461      100
4  6.591674      256

[5 rows x 34 columns]
```

[7]:
```
df.columns.values
```

[7]:
```
array(['id', 'nearc2', 'nearc4', 'educ', 'age', 'fatheduc', 'motheduc',
       'weight', 'momdad14', 'sinmom14', 'step14', 'reg661', 'reg662',
       'reg663', 'reg664', 'reg665', 'reg666', 'reg667', 'reg668',
       'reg669', 'south66', 'black', 'smsa', 'south', 'smsa66', 'wage',
       'enroll', 'kww', 'iq', 'married', 'libcrd14', 'exper', 'lwage',
       'expersq'], dtype=object)
```

[30]:
```
data_filter = df['educ'] >= 6
# outcome variable
y=df[data_filter]['lwage'].values
# treatment variable
treatment=df[data_filter]['educ'].values
# instrumental variable
w=df[data_filter]['nearc4'].values

Xdf=df[data_filter][['fatheduc', 'motheduc', 'momdad14', 'sinmom14', 'reg661', 'reg662
↪',
      'reg663', 'reg664', 'reg665', 'reg666', 'reg667', 'reg668',
      'reg669', 'south66', 'black', 'smsa', 'south', 'smsa66',
      'exper', 'expersq']]
Xdf['fatheduc']=Xdf['fatheduc'].fillna(value=Xdf['fatheduc'].mean())
Xdf['motheduc']=Xdf['motheduc'].fillna(value=Xdf['motheduc'].mean())
Xscale=Xdf.copy()
Xscale[['fatheduc', 'motheduc', 'exper', 'expersq']]=StandardScaler().fit_
↪transform(Xscale[['fatheduc', 'motheduc', 'exper', 'expersq']])
X=Xscale.values
```

[32]:
```
Xscale.describe()
```

[32]:

	fatheduc	motheduc	momdad14	sinmom14	reg661
count	2.991000e+03	2.991000e+03	2991.000000	2991.000000	2991.000000
mean	-3.529069e-16	-1.704346e-15	0.790371	0.100301	0.046807
std	1.000167e+00	1.000167e+00	0.407112	0.300451	0.211261
min	-3.101056e+00	-3.502453e+00	0.000000	0.000000	0.000000
25%	-6.303764e-01	-4.656485e-01	1.000000	0.000000	0.000000
50%	0.000000e+00	2.091970e-01	1.000000	0.000000	0.000000
75%	6.049634e-01	5.466197e-01	1.000000	0.000000	0.000000
max	2.457973e+00	2.571156e+00	1.000000	1.000000	1.000000

	reg662	reg663	reg664	reg665	reg666
count	2991.000000	2991.000000	2991.000000	2991.000000	2991.000000
mean	0.161484	0.196924	0.064527	0.205951	0.094952
std	0.368039	0.397741	0.245730	0.404463	0.293197
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

	reg667	reg668	reg669	south66	black
count	2991.000000	2991.000000	2991.000000	2991.000000	2991.000000
mean	0.109997	0.028419	0.090939	0.410899	0.231361
std	0.312938	0.166193	0.287571	0.492079	0.421773
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

	smsa	south	smsa66	exper	expersq
count	2991.000000	2991.000000	2991.000000	2.991000e+03	2.991000e+03
mean	0.715145	0.400201	0.651622	4.285921e-16	3.040029e-17
std	0.451421	0.490021	0.476536	1.000167e+00	1.000167e+00

(continues on next page)

(continued from previous page)

```
min       0.000000     0.000000     0.000000 -2.159127e+00 -1.147691e+00
25%       0.000000     0.000000     0.000000 -6.858865e-01 -7.077287e-01
50%       1.000000     0.000000     1.000000 -1.948066e-01 -3.655360e-01
75%       1.000000     1.000000     1.000000  5.418134e-01  3.310707e-01
max       1.000000     1.000000     1.000000  3.242753e+00  4.767355e+00
```

Semi-Synthetic Data Generation

[29]:
```
def semi_synth_nlsym(X, w, random_seed=None):
    np.random.seed(random_seed)
    nobs = X.shape[0]
    nv = np.random.uniform(0, 1, size=nobs)
    c0 = np.random.uniform(0.2, 0.3)
    C = c0 * X[:,1]
    # Treatment compliance depends on mother education
    treatment = C * w + X[:,1] + nv
    # Treatment effect depends no mother education and single-mom family at age 14
    theta = 0.1 + 0.05 * X[:,1] - 0.1*X[:,3]
    # Additional effect on the outcome from mother education
    f = 0.05 * X[:,1]
    y = theta * (treatment + nv) + f + np.random.normal(0, 0.1, size=nobs)

    return y, treatment, theta
```

[33]:
```
y_sim, treatment_sim, theta = semi_synth_nlsym(Xdf.values, w)
```

Estimation

[36]:
```
# True value
theta.mean()
```

[36]:
```
0.6089706667314586
```

[38]:
```
# 2SLS estimate
iv_fit = IVRegressor()
iv_fit.fit(X, treatment_sim, y_sim, w)
ate, ate_sd = iv_fit.predict()
(ate, ate_sd)
```

[38]:
```
(0.6611532131769402, 0.013922622951893662)
```

[51]:
```
# OLS estimate
ols_fit=sm.OLS(y_sim, sm.add_constant(np.c_[treatment_sim, X], prepend=False)).fit()
(ols_fit.params[0], ols_fit.bse[0])
```

[51]:
```
(0.7501211540497275, 0.012800163754977008)
```

5.8.2 Pure Synthetic Data

The data generation mechanism is described in Hong et al "*Semiparametric Efficiency in Nonlinear LATE Models*" (2010).

Data Generation

[54]:
```
def synthetic_data(n=10000, random_seed=None):
    np.random.seed(random_seed)
    gamma0 = -0.5
    gamma1 = 1.0
    delta = 1.0
    x = np.random.uniform(size=n)
    v = np.random.normal(size=n)
    d1 = (gamma0 + x*gamma1 + delta + v>=0).astype(float)
    d0 = (gamma0 + x*gamma1 + v>=0).astype(float)

    alpha = 1.0
    beta = 0.5
    lambda11 = 2.0
    lambda00 = 1.0
    xi1 = np.random.poisson(np.exp(alpha+x*beta))
    xi2 = np.random.poisson(np.exp(x*beta))
    xi3 = np.random.poisson(np.exp(lambda11), size=n)
    xi4 = np.random.poisson(np.exp(lambda00), size=n)

    y1 = xi1 + xi3 * ((d1==1) & (d0==1)) + xi4 * ((d1==0) & (d0==0))
    y0 = xi2 + xi3 * ((d1==1) & (d0==1)) + xi4 * ((d1==0) & (d0==0))

    z = np.random.binomial(1, stats.norm.cdf(x))
    d = d1*z + d0*(1-z)
    y = y1*d + y0*(1-d)

    return y, x, d, z, y1[(d1>d0)].mean()-y0[(d1>d0)].mean()
```

[55]:
```
y, x, d, z, late = synthetic_data()
```

Estimation

[56]:
```
# True value
late
```

[56]:
```
2.1789099526066353
```

[57]:
```
# 2SLS estimate
iv_fit = IVRegressor()
iv_fit.fit(x, d, y, z)
ate, ate_sd = iv_fit.predict()
(ate, ate_sd)
```

[57]:
```
(2.1900472390231775, 0.2623695460540134)
```

[59]:
```
# OLS estimate
ols_fit=sm.OLS(y, sm.add_constant(np.c_[d, x], prepend=False)).fit()
(ols_fit.params[0], ols_fit.bse[0])
```

[59]:
```
(5.3482879532439975, 0.09201397327077365)
```

[ ]:
```

```

5.9 Sensitivity Analysis Examples

5.9.1 Methods

We provided five methods for sensitivity analysis including (Placebo Treatment, Random Cause, Subset Data, Random Replace and Selection Bias). This notebook will walkthrough how to use the combined function sensitivity_analysis() to compare different method and also how to use each individual method separately:

1. Placebo Treatment: Replacing treatment with a random variable
2. Irrelevant Additional Confounder: Adding a random common cause variable
3. Subset validation: Removing a random subset of the data
4. Selection Bias method with One Sided confounding function and Alignment confounding function
5. Random Replace: Random replace a covariate with an irrelevant variable

[2]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
```

[3]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import warnings
import matplotlib
from causalml.inference.meta import BaseXLearner
from causalml.dataset import synthetic_data

from causalml.metrics.sensitivity import Sensitivity
from causalml.metrics.sensitivity import SensitivityRandomReplace,␣
↪SensitivitySelectionBias

plt.style.use('fivethirtyeight')
matplotlib.rcParams['figure.figsize'] = [8, 8]
warnings.filterwarnings('ignore')

# logging.basicConfig(level=logging.INFO)

pd.options.display.float_format = '{:.4f}'.format
```

```
/Users/jing.pan/anaconda3/envs/causalml_3_6/lib/python3.6/site-packages/sklearn/utils/
↪deprecation.py:144: FutureWarning: The sklearn.utils.testing module is  deprecated␣
↪in version 0.22 and will be removed in version 0.24. The corresponding classes /␣
↪functions should instead be imported from sklearn.utils. Anything that cannot be␣
↪imported from sklearn.utils is now part of the private API.
  warnings.warn(message, FutureWarning)
```

5.9.2 Generate Synthetic data

[4]:
```
# Generate synthetic data using mode 1
num_features = 6
y, X, treatment, tau, b, e = synthetic_data(mode=1, n=100000, p=num_features, sigma=1.
↪0)
```

[5]:
```
tau.mean()
```

[5]:
```
0.5001096146567363
```

5.9.3 Define Features

[6]:
```
# Generate features names
INFERENCE_FEATURES = ['feature_' + str(i) for i in range(num_features)]
TREATMENT_COL = 'target'
OUTCOME_COL = 'outcome'
SCORE_COL = 'pihat'
```

[7]:
```
df = pd.DataFrame(X, columns=INFERENCE_FEATURES)
df[TREATMENT_COL] = treatment
df[OUTCOME_COL] = y
df[SCORE_COL] = e
```

[8]:
```
df.head()
```

[8]:
```
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  target  \
0     0.9536     0.2911     0.0432     0.8720     0.5190     0.0822       1
1     0.2390     0.3096     0.5115     0.2048     0.8914     0.5015       0
2     0.1091     0.0765     0.7428     0.6951     0.4580     0.7800       0
3     0.2055     0.3967     0.6278     0.2086     0.3865     0.8860       0
4     0.4501     0.0578     0.3972     0.4100     0.5760     0.4764       0

   outcome  pihat
0   2.0220 0.7657
1  -0.0732 0.2304
2  -1.4947 0.1000
3   0.6458 0.2533
4  -0.0018 0.1000
```

5.9.4 With all Covariates

Sensitivity Analysis Summary Report (with One-sided confounding function and default alpha)

[9]:
```
# Calling the Base XLearner class and return the sensitivity analysis summary report
learner_x = BaseXLearner(LinearRegression())
sens_x = Sensitivity(df=df, inference_features=INFERENCE_FEATURES, p_col='pihat',
                     treatment_col=TREATMENT_COL, outcome_col=OUTCOME_COL,␣
↪learner=learner_x)
# Here for Selection Bias method will use default one-sided confounding function and␣
↪alpha (quantile range of outcome values) input
sens_sumary_x = sens_x.sensitivity_analysis(methods=['Placebo Treatment',
```

(continues on next page)

(continued from previous page)

```
                                                    'Random Cause',
                                                    'Subset Data',
                                                    'Random Replace',
                                                    'Selection Bias'], sample_size=0.
↪5)
```

[10]:
```
# From the following results, refutation methods show our model is pretty robust;
# When alpah > 0, the treated group always has higher mean potential outcomes than␣
↪the control; when  < 0, the control group is better off.
sens_sumary_x
```

[10]:
```
                                          Method    ATE New ATE  \
0                                Placebo Treatment 0.6801 -0.0025
0                                     Random Cause 0.6801  0.6801
0                         Subset Data(sample size @0.5) 0.6801  0.6874
0                                   Random Replace 0.6801  0.6799
0  Selection Bias (alpha@-0.80111, with r-sqaure:… 0.6801  1.3473
0  Selection Bias (alpha@-0.64088, with r-sqaure:… 0.6801  1.2139
0  Selection Bias (alpha@-0.48066, with r-sqaure:… 0.6801  1.0804
0  Selection Bias (alpha@-0.32044, with r-sqaure:… 0.6801  0.9470
0  Selection Bias (alpha@-0.16022, with r-sqaure:… 0.6801  0.8135
0       Selection Bias (alpha@0.0, with r-sqaure:0.0 0.6801  0.6801
0  Selection Bias (alpha@0.16022, with r-sqaure:0… 0.6801  0.5467
0  Selection Bias (alpha@0.32044, with r-sqaure:0… 0.6801  0.4132
0  Selection Bias (alpha@0.48066, with r-sqaure:0… 0.6801  0.2798
0  Selection Bias (alpha@0.64088, with r-sqaure:0… 0.6801  0.1463
0  Selection Bias (alpha@0.80111, with r-sqaure:0… 0.6801  0.0129

  New ATE LB New ATE UB
0    -0.0158     0.0107
0     0.6673     0.6929
0     0.6693     0.7055
0     0.6670     0.6929
0     1.3347     1.3599
0     1.2013     1.2265
0     1.0678     1.0931
0     0.9343     0.9597
0     0.8008     0.8263
0     0.6673     0.6929
0     0.5338     0.5595
0     0.4003     0.4261
0     0.2668     0.2928
0     0.1332     0.1594
0    -0.0003     0.0261
```

Random Replace

[11]:
```
# Replace feature_0 with an irrelevent variable
sens_x_replace = SensitivityRandomReplace(df=df, inference_features=INFERENCE_
↪FEATURES, p_col='pihat',
                                          treatment_col=TREATMENT_COL, outcome_
↪col=OUTCOME_COL, learner=learner_x,
                                          sample_size=0.9, replaced_feature='feature_0
↪')
```

(continues on next page)

(continued from previous page)

```
s_check_replace = sens_x_replace.summary(method='Random Replace')
s_check_replace
```

[11]:

```
           Method    ATE New ATE New ATE LB New ATE UB
0  Random Replace 0.6801  0.8072     0.7943     0.8200
```

Selection Bias: Alignment confounding Function

[12]:

```
sens_x_bias_alignment = SensitivitySelectionBias(df, INFERENCE_FEATURES, p_col='pihat
↪', treatment_col=TREATMENT_COL,
                                                 outcome_col=OUTCOME_COL,␣
↪learner=learner_x, confound='alignment',
                                                 alpha_range=None)
```

[13]:

```
lls_x_bias_alignment, partial_rsqs_x_bias_alignment = sens_x_bias_alignment.
↪causalsens()
```

[14]:

```
lls_x_bias_alignment
```

[14]:

```
    alpha   rsqs  New ATE  New ATE LB  New ATE UB
0 -0.8011 0.1088   0.6685      0.6556      0.6813
0 -0.6409 0.0728   0.6708      0.6580      0.6836
0 -0.4807 0.0425   0.6731      0.6604      0.6859
0 -0.3204 0.0194   0.6754      0.6627      0.6882
0 -0.1602 0.0050   0.6778      0.6650      0.6905
0  0.0000 0.0000   0.6801      0.6673      0.6929
0  0.1602 0.0050   0.6824      0.6696      0.6953
0  0.3204 0.0200   0.6848      0.6718      0.6977
0  0.4807 0.0443   0.6871      0.6741      0.7001
0  0.6409 0.0769   0.6894      0.6763      0.7026
0  0.8011 0.1164   0.6918      0.6785      0.7050
```

[15]:

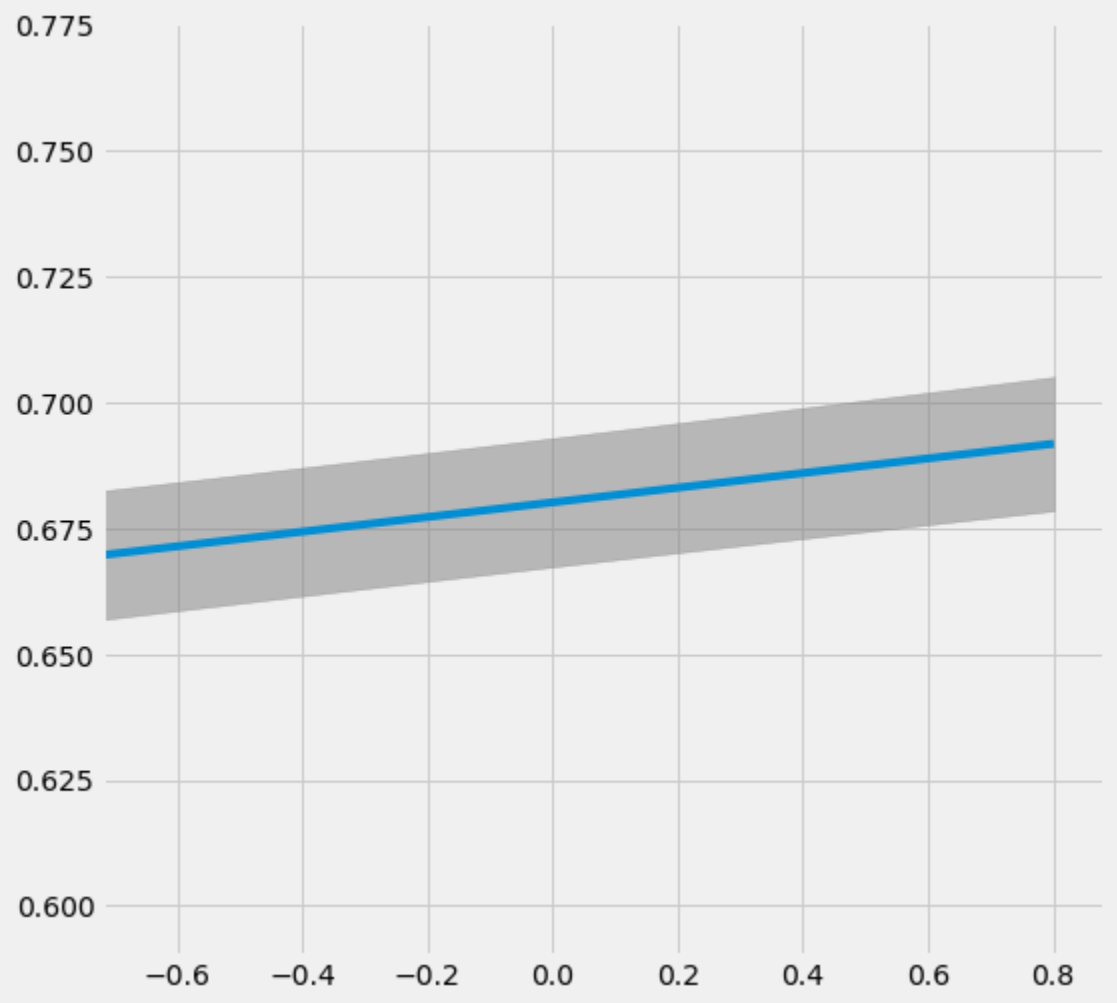
```
partial_rsqs_x_bias_alignment
```

[15]:

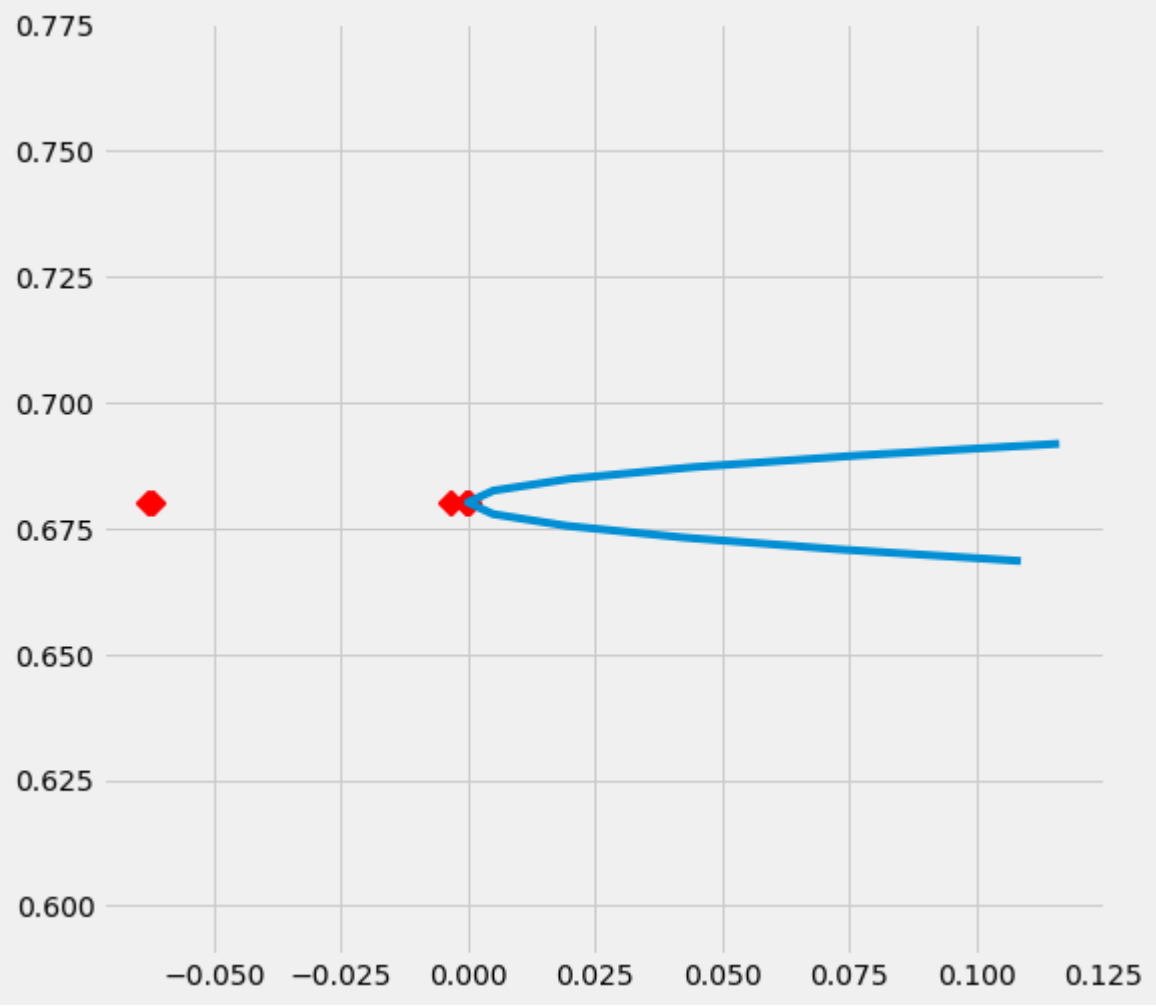
```
     feature partial_rsqs
0  feature_0      -0.0631
1  feature_1      -0.0619
2  feature_2      -0.0001
3  feature_3      -0.0033
4  feature_4      -0.0001
5  feature_5       0.0000
```

[16]:

```
# Plot the results by confounding vector and plot Confidence Intervals for ATE
sens_x_bias_alignment.plot(lls_x_bias_alignment, ci=True)
```



[17]:
```
# Plot the results by rsquare with partial r-square results by each individual␣
↪features
sens_x_bias_alignment.plot(lls_x_bias_alignment, partial_rsqs_x_bias_alignment, type=
↪'r.squared', partial_rsqs=True)
```



5.9.5 Drop One Confounder

[18]:
```
df_new = df.drop('feature_0', axis=1).copy()
INFERENCE_FEATURES_new = INFERENCE_FEATURES.copy()
INFERENCE_FEATURES_new.remove('feature_0')
df_new.head()
```

[18]:

	feature_1	feature_2	feature_3	feature_4	feature_5	target	outcome	\
0	0.2911	0.0432	0.8720	0.5190	0.0822	1	2.0220	
1	0.3096	0.5115	0.2048	0.8914	0.5015	0	-0.0732	
2	0.0765	0.7428	0.6951	0.4580	0.7800	0	-1.4947	
3	0.3967	0.6278	0.2086	0.3865	0.8860	0	0.6458	
4	0.0578	0.3972	0.4100	0.5760	0.4764	0	-0.0018	

	pihat
0	0.7657
1	0.2304
2	0.1000
3	0.2533
4	0.1000

[19]:
```
INFERENCE_FEATURES_new
```

[19]:
```
['feature_1', 'feature_2', 'feature_3', 'feature_4', 'feature_5']
```

Sensitivity Analysis Summary Report (with One-sided confounding function and default alpha)

[20]:
```
sens_x_new = Sensitivity(df=df_new, inference_features=INFERENCE_FEATURES_new, p_col=
↪'pihat',
                         treatment_col=TREATMENT_COL, outcome_col=OUTCOME_COL,␣
↪learner=learner_x)
# Here for Selection Bias method will use default one-sided confounding function and␣
↪alpha (quantile range of outcome values) input
sens_sumary_x_new = sens_x_new.sensitivity_analysis(methods=['Placebo Treatment',
                                                             'Random Cause',
                                                             'Subset Data',
                                                             'Random Replace',
                                                             'Selection Bias'], sample_size=0.
↪5)
```

[21]:
```
# Here we can see the New ATE restul from Random Replace method actually changed ~ 12.
↪5%
sens_sumary_x_new
```

[21]:

	Method	ATE	New ATE	New ATE LB	New ATE UB
0	Placebo Treatment	0.8072	0.0104	-0.0033	0.0242
0	Random Cause	0.8072	0.8072	0.7943	0.8201
0	Subset Data(sample size @0.5)	0.8072	0.8180	0.7998	0.8361
0	Random Replace	0.8072	0.8068	0.7938	0.8198
0	Selection Bias (alpha@-0.80111, with r-sqaure:…	0.8072	1.3799	1.3673	1.3925
0	Selection Bias (alpha@-0.64088, with r-sqaure:…	0.8072	1.2654	1.2527	1.2780
0	Selection Bias (alpha@-0.48066, with r-sqaure:…	0.8072	1.1508	1.1381	1.1635
0	Selection Bias (alpha@-0.32044, with r-sqaure:…	0.8072	1.0363	1.0235	1.0490
0	Selection Bias (alpha@-0.16022, with r-sqaure:…	0.8072	0.9217	0.9089	0.9345
0	Selection Bias (alpha@0.0, with r-sqaure:0.0	0.8072	0.8072	0.7943	0.8200
0	Selection Bias (alpha@0.16022, with r-sqaure:0…	0.8072	0.6926	0.6796	0.7056
0	Selection Bias (alpha@0.32044, with r-sqaure:0…	0.8072	0.5780	0.5650	0.5911
0	Selection Bias (alpha@0.48066, with r-sqaure:0…	0.8072	0.4635	0.4503	0.4767
0	Selection Bias (alpha@0.64088, with r-sqaure:0…	0.8072	0.3489	0.3356	0.3623
0	Selection Bias (alpha@0.80111, with r-sqaure:0…	0.8072	0.2344	0.2209	0.2479

Random Replace

[22]:
```
# Replace feature_0 with an irrelevent variable
sens_x_replace_new = SensitivityRandomReplace(df=df_new, inference_features=INFERENCE_
↪FEATURES_new, p_col='pihat',
                                              treatment_col=TREATMENT_COL, outcome_
↪col=OUTCOME_COL, learner=learner_x,
                                              sample_size=0.9, replaced_feature='feature_1
↪')
s_check_replace_new = sens_x_replace_new.summary(method='Random Replace')
s_check_replace_new
```

[22]:

	Method	ATE New	ATE New LB	New ATE UB
0	Random Replace	0.8072	0.9022	0.8893	0.9152

Selection Bias: Alignment confounding Function

[23]:
```
sens_x_bias_alignment_new = SensitivitySelectionBias(df_new, INFERENCE_FEATURES_new,␣
↪p_col='pihat', treatment_col=TREATMENT_COL,
                                                     outcome_col=OUTCOME_COL,␣
↪learner=learner_x, confound='alignment',
                                                     alpha_range=None)
```

[24]:
```
lls_x_bias_alignment_new, partial_rsqs_x_bias_alignment_new = sens_x_bias_alignment_
↪new.causalsens()
```

[25]:
```
lls_x_bias_alignment_new
```

[25]:

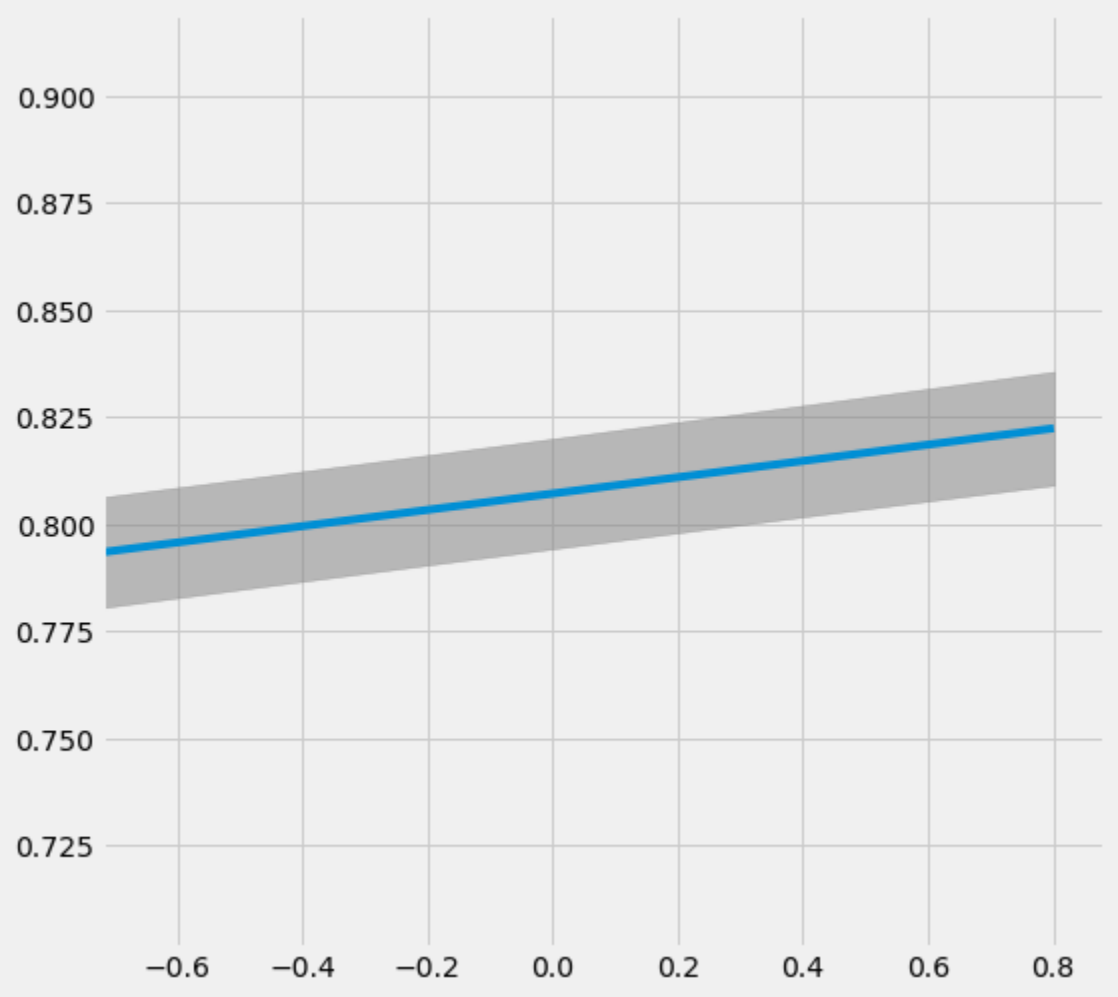
	alpha	rsqs	New ATE	New ATE LB	New ATE UB
0	-0.8011	0.1121	0.7919	0.7789	0.8049
0	-0.6409	0.0732	0.7950	0.7820	0.8079
0	-0.4807	0.0419	0.7980	0.7851	0.8109
0	-0.3204	0.0188	0.8011	0.7882	0.8139
0	-0.1602	0.0047	0.8041	0.7912	0.8170
0	0.0000	0.0000	0.8072	0.7943	0.8200
0	0.1602	0.0048	0.8102	0.7973	0.8231
0	0.3204	0.0189	0.8133	0.8003	0.8262
0	0.4807	0.0420	0.8163	0.8032	0.8294
0	0.6409	0.0736	0.8194	0.8062	0.8325
0	0.8011	0.1127	0.8224	0.8091	0.8357

[26]:
```
partial_rsqs_x_bias_alignment_new
```

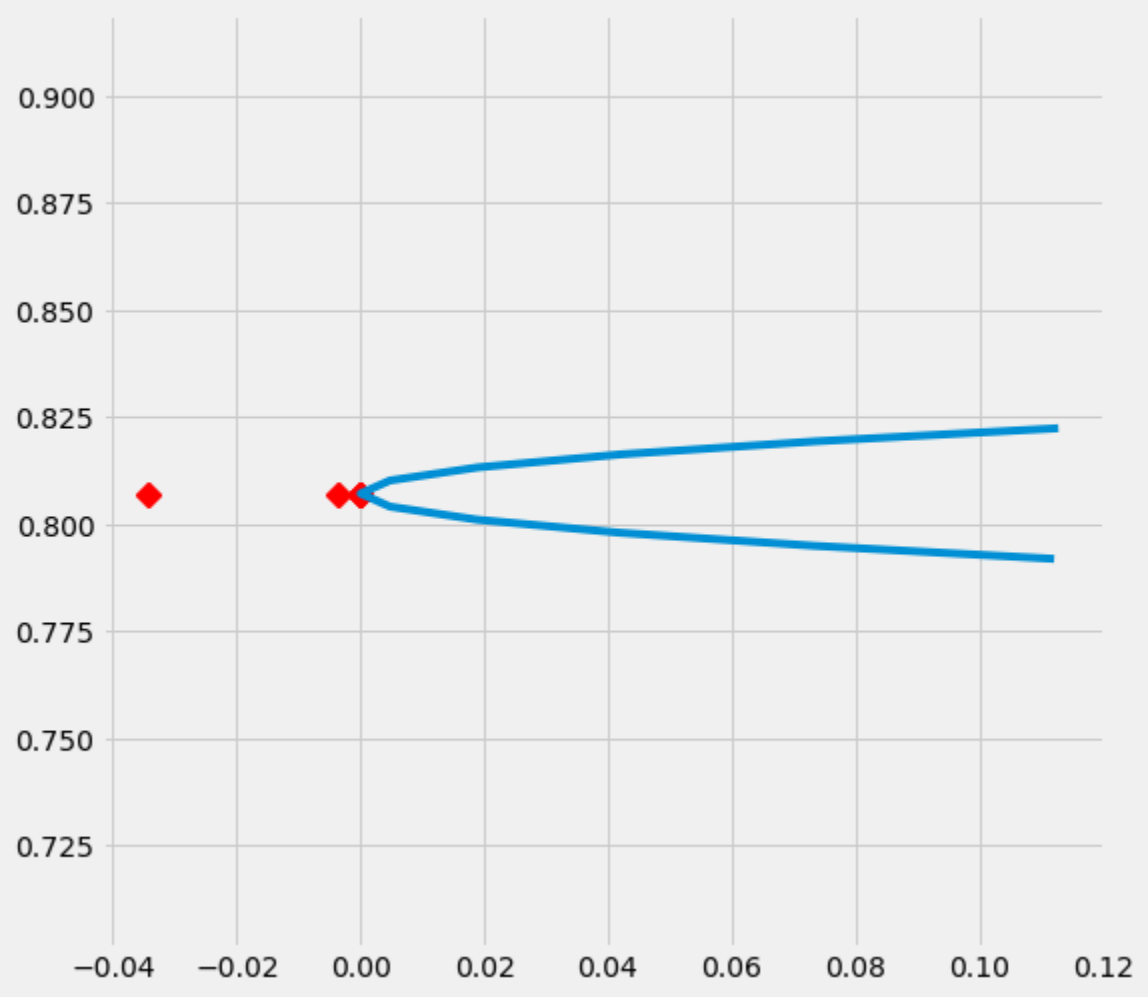
[26]:

	feature	partial_rsqs
0	feature_1	-0.0345
1	feature_2	-0.0001
2	feature_3	-0.0038
3	feature_4	-0.0001
4	feature_5	0.0000

[27]:
```
# Plot the results by confounding vector and plot Confidence Intervals for ATE
sens_x_bias_alignment_new.plot(lls_x_bias_alignment_new, ci=True)
```



[28]:
```
# Plot the results by rsquare with partial r-square results by each individual␣
↪features
sens_x_bias_alignment_new.plot(lls_x_bias_alignment_new, partial_rsqs_x_bias_
↪alignment_new, type='r.squared', partial_rsqs=True)
```



5.9.6 Generate a Selection Bias Set

[29]:
```
df_new_2 = df.copy()
df_new_2['treated_new'] = df['feature_0'].rank()
df_new_2['treated_new'] = [1 if i > df_new_2.shape[0]/2 else 0 for i in df_new_2[
↪'treated_new']]
```

[30]:
```
df_new_2.head()
```

[30]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	target	\
0	0.9536	0.2911	0.0432	0.8720	0.5190	0.0822	1	
1	0.2390	0.3096	0.5115	0.2048	0.8914	0.5015	0	
2	0.1091	0.0765	0.7428	0.6951	0.4580	0.7800	0	
3	0.2055	0.3967	0.6278	0.2086	0.3865	0.8860	0	
4	0.4501	0.0578	0.3972	0.4100	0.5760	0.4764	0	

	outcome	pihat	treated_new
0	2.0220	0.7657	1
1	-0.0732	0.2304	0
2	-1.4947	0.1000	0

(continues on next page)

(continued from previous page)

```
3   0.6458 0.2533            0
4  -0.0018 0.1000            0
```

Sensitivity Analysis Summary Report (with One-sided confounding function and default alpha)

[31]:
```
sens_x_new_2 = Sensitivity(df=df_new_2, inference_features=INFERENCE_FEATURES, p_col=
↪'pihat',
                      treatment_col='treated_new', outcome_col=OUTCOME_COL,␣
↪learner=learner_x)
# Here for Selection Bias method will use default one-sided confounding function and␣
↪alpha (quantile range of outcome values) input
sens_sumary_x_new_2 = sens_x_new_2.sensitivity_analysis(methods=['Placebo Treatment',
                                                         'Random Cause',
                                                         'Subset Data',
                                                         'Random Replace',
                                                         'Selection Bias'], sample_size=0.
↪5)
```

[32]:
```
sens_sumary_x_new_2
```

[32]:
```
                                          Method    ATE New ATE  \
0                                Placebo Treatment 0.0432  0.0081
0                                    Random Cause 0.0432  0.0432
0                      Subset Data(sample size @0.5) 0.0432  0.0976
0                                  Random Replace 0.0432  0.0433
0  Selection Bias (alpha@-0.80111, with r-sqaure:… 0.0432  0.8369
0  Selection Bias (alpha@-0.64088, with r-sqaure:… 0.0432  0.6782
0  Selection Bias (alpha@-0.48066, with r-sqaure:… 0.0432  0.5194
0  Selection Bias (alpha@-0.32044, with r-sqaure:… 0.0432  0.3607
0  Selection Bias (alpha@-0.16022, with r-sqaure:… 0.0432  0.2020
0       Selection Bias (alpha@0.0, with r-sqaure:0.0 0.0432  0.0432
0  Selection Bias (alpha@0.16022, with r-sqaure:0… 0.0432 -0.1155
0  Selection Bias (alpha@0.32044, with r-sqaure:0… 0.0432 -0.2743
0  Selection Bias (alpha@0.48066, with r-sqaure:0… 0.0432 -0.4330
0  Selection Bias (alpha@0.64088, with r-sqaure:0… 0.0432 -0.5918
0  Selection Bias (alpha@0.80111, with r-sqaure:0… 0.0432 -0.7505

  New ATE LB New ATE UB
0    -0.0052     0.0213
0     0.0296     0.0568
0     0.0784     0.1167
0     0.0297     0.0568
0     0.8239     0.8499
0     0.6651     0.6913
0     0.5063     0.5326
0     0.3474     0.3740
0     0.1885     0.2154
0     0.0296     0.0568
0    -0.1293    -0.1018
0    -0.2882    -0.2604
0    -0.4471    -0.4189
0    -0.6060    -0.5775
0    -0.7650    -0.7360
```

Random Replace

[33]:
```
# Replace feature_0 with an irrelevent variable
sens_x_replace_new_2 = SensitivityRandomReplace(df=df_new_2, inference_
↪features=INFERENCE_FEATURES, p_col='pihat',
                                                treatment_col='treated_new', outcome_
↪col=OUTCOME_COL, learner=learner_x,
                                                sample_size=0.9, replaced_feature='feature_0
↪')
s_check_replace_new_2 = sens_x_replace_new_2.summary(method='Random Replace')
s_check_replace_new_2
```

[33]:

	Method	ATE	New ATE	New ATE LB	New ATE UB
0	Random Replace	0.0432	0.4847	0.4713	0.4981

Selection Bias: Alignment confounding Function

[34]:
```
sens_x_bias_alignment_new_2 = SensitivitySelectionBias(df_new_2, INFERENCE_FEATURES,␣
↪p_col='pihat', treatment_col='treated_new',
                                                       outcome_col=OUTCOME_COL,␣
↪learner=learner_x, confound='alignment',
                                                       alpha_range=None)
```

[35]:
```
lls_x_bias_alignment_new_2, partial_rsqs_x_bias_alignment_new_2 = sens_x_bias_
↪alignment_new_2.causalsens()
```

[36]:
```
lls_x_bias_alignment_new_2
```

[36]:

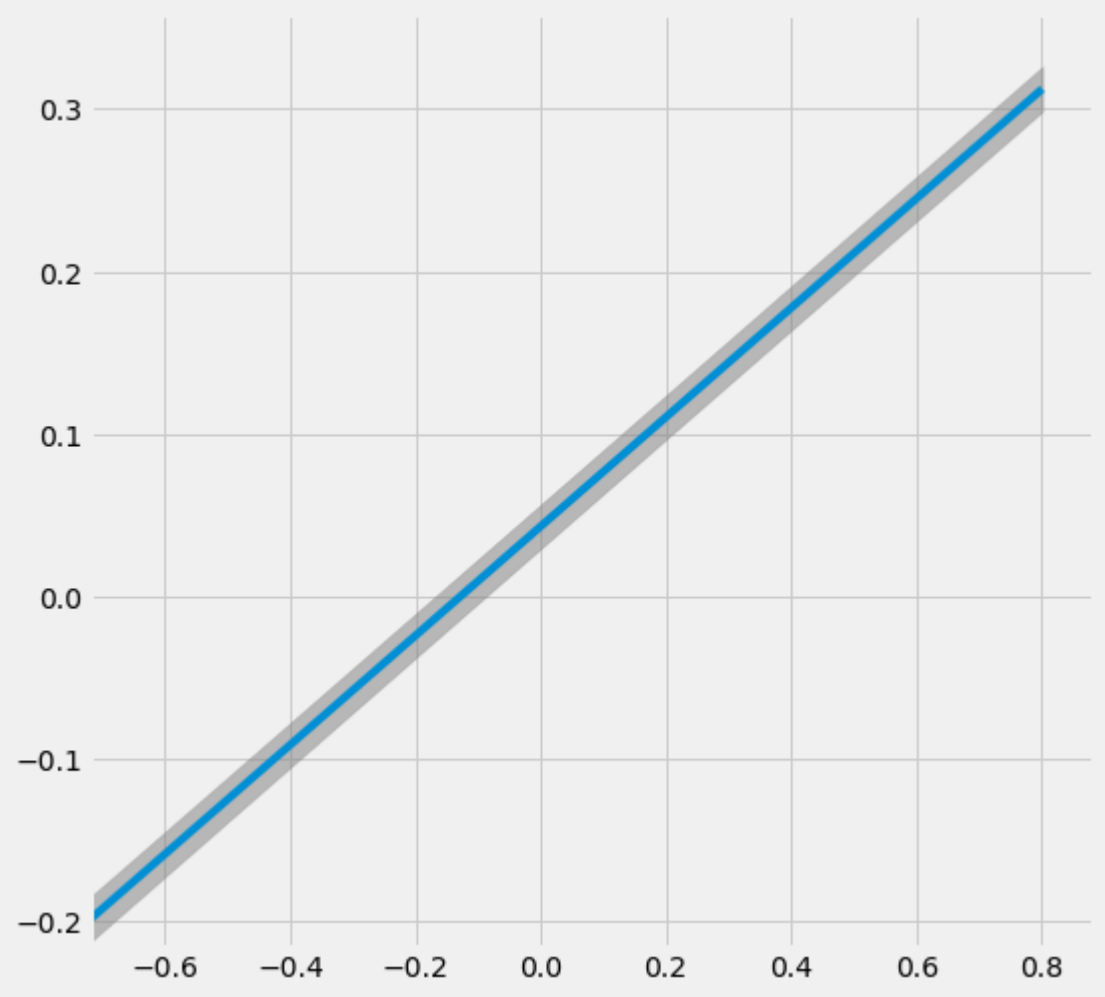
	alpha	rsqs	New ATE	New ATE LB	New ATE UB
0	-0.8011	0.0604	-0.2260	-0.2399	-0.2120
0	-0.6409	0.0415	-0.1721	-0.1860	-0.1583
0	-0.4807	0.0250	-0.1183	-0.1320	-0.1045
0	-0.3204	0.0119	-0.0645	-0.0781	-0.0508
0	-0.1602	0.0032	-0.0106	-0.0242	0.0030
0	0.0000	0.0000	0.0432	0.0296	0.0568
0	0.1602	0.0035	0.0971	0.0835	0.1106
0	0.3204	0.0148	0.1509	0.1373	0.1645
0	0.4807	0.0347	0.2047	0.1911	0.2183
0	0.6409	0.0635	0.2586	0.2449	0.2722
0	0.8011	0.1013	0.3124	0.2986	0.3262

[37]:
```
partial_rsqs_x_bias_alignment_new_2
```

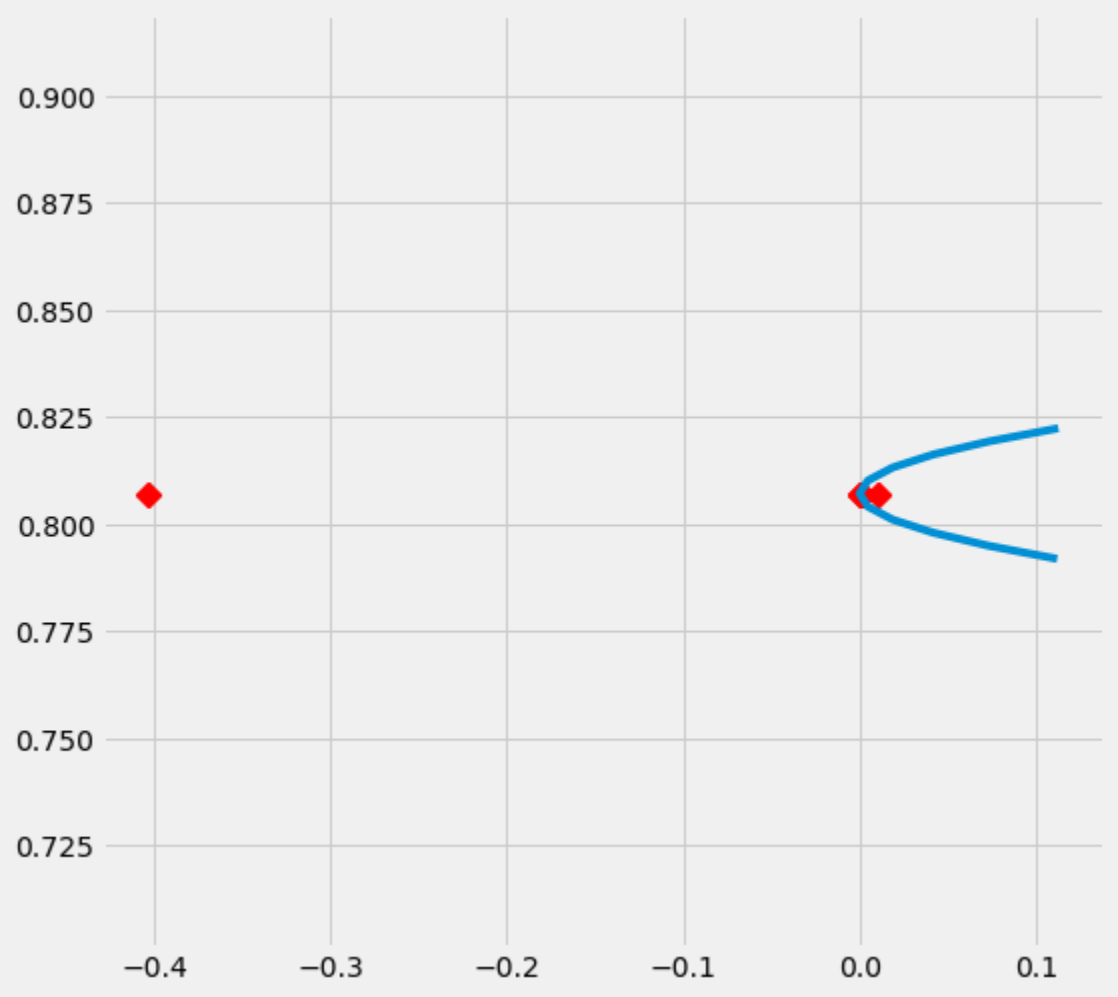
[37]:

	feature	partial_rsqs
0	feature_0	-0.4041
1	feature_1	0.0101
2	feature_2	0.0000
3	feature_3	0.0016
4	feature_4	0.0011
5	feature_5	0.0000

[38]:
```
# Plot the results by confounding vector and plot Confidence Intervals for ATE
sens_x_bias_alignment_new_2.plot(lls_x_bias_alignment_new_2, ci=True)
```



[39]:
```
# Plot the results by rsquare with partial r-square results by each individual␣
↪features
sens_x_bias_alignment_new_2.plot(lls_x_bias_alignment_new, partial_rsqs_x_bias_
↪alignment_new_2, type='r.squared', partial_rsqs=True)
```



5.10 Unit Selection Based on Counterfactual Logic by Li and Pearl (2019)

Causal ML contains an experimental version of the counterfactual unit selection method proposed by Li and Pearl (2019). The method has not been extensively tested or optimised so the user should proceed with caution. This notebook demonstrates the basic use of the counterfactual unit selector.

[2]:
```
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import train_test_split

from causalml.dataset import make_uplift_classification
from causalml.optimize import CounterfactualUnitSelector
from causalml.optimize import get_treatment_costs
from causalml.optimize import get_actual_value

import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import seaborn as sns
sns.set_style('white')

%matplotlib inline
```

5.10.1 Generate data

We first generate some synthetic data using the built-in function.

[3]:
```
df, X_names = make_uplift_classification(n_samples=5000,
                                         treatment_name=['control', 'treatment'])
```

[4]:
```
# Lump all treatments together for this demo
df['treatment_numeric'] = df['treatment_group_key'].replace({'control': 0, 'treatment
↪': 1})
```

[5]:
```
df['treatment_group_key'].value_counts()
```

[5]:
```
treatment    5000
control      5000
Name: treatment_group_key, dtype: int64
```

5.10.2 Specify payoffs

In the context of a simple two-armed experiment, the counterfactual unit selection approach considers the following four segments of individuals:

- Never-takers: those who will not convert whether or not they are in the treatment
- Always-takers: those who will convert whether or not they are in the treatment
- Compliers: those who will convert if they are in the treatment and will not convert if they are in the control
- Defiers: those who will convert if they are in the control and will not convert if they are in the treatment

If we assume that the payoff from conversion is $20 and the conversion cost of a treatment is $2.5, then we can calculate the payoffs for targeting each type of individual as follows. For nevertakers, the payoff is always $0 because they will not convert or use a promotion. For alwaystakers, the payoff is -$2.5 because they would convert anyway but now we additionally give them a treatment worth $2.5. For compliers, the payoff is the benefit from conversion minus the cost of the treatment, and for defiers the payoff is -$20 because they would convert if we didn't treat them.

[6]:
```
nevertaker_payoff = 0
alwaystaker_payoff = -2.5
complier_payoff = 17.5
defier_payoff = -20
```

5.10.3 Run counterfactual unit selector

In this section we run the CounterfactualUnitSelector model and compare its performance against random assignment and a scheme in which all units are assigned to the treatment that has the best conversion in the training set. We measure the performance by looking at the average actual value payoff from those units in the testing set who happen to be in the treatment group recommended by each approach.

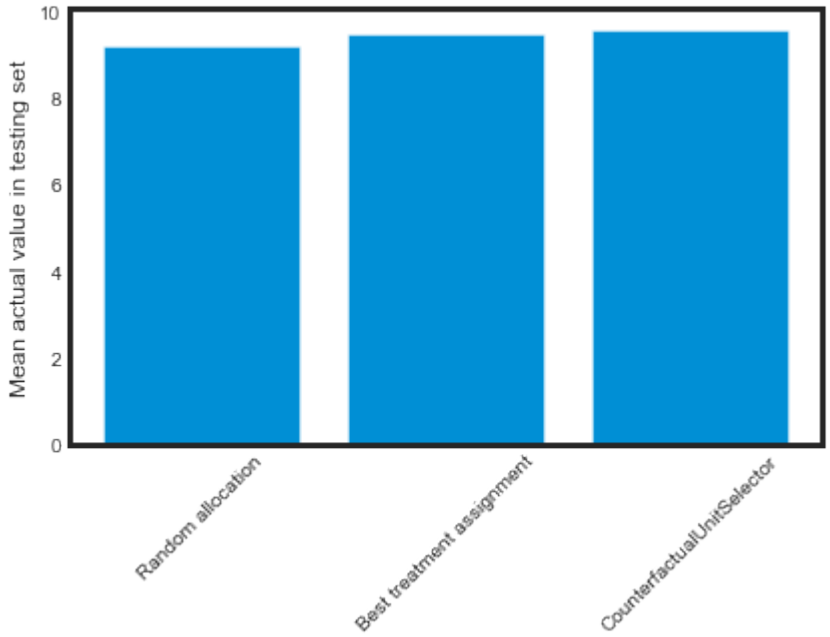
[7]:
```
# Specify the same costs as above but in a different form
tc_dict = {'control': 0, 'treatment': 2.5}
ic_dict = {'control': 0, 'treatment': 0}
conversion_value = np.full(df.shape[0], 20)

# Use the above information to get the cost of each treatment
cc_array, ic_array, conditions = get_treatment_costs(
    treatment=df['treatment_group_key'], control_name='control',
    cc_dict=tc_dict, ic_dict=ic_dict)

# Get the actual value of having a unit in their actual treatment
actual_value = get_actual_value(treatment=df['treatment_group_key'],
                                observed_outcome=df['conversion'],
                                conversion_value=conversion_value,
                                conditions=conditions,
                                conversion_cost=cc_array,
                                impression_cost=ic_array)
```

[8]:
```
df_train, df_test = train_test_split(df)
train_idx = df_train.index
test_idx = df_test.index
```

[9]:
```
# Get the outcome if treatments were allocated randomly
random_allocation_value = actual_value.loc[test_idx].mean()

# Get the actual value of those individuals who are in the best
# treatment group
best_ate = df_train.groupby(
    'treatment_group_key')['conversion'].mean().idxmax()
actual_is_best_ate = df_test['treatment_group_key'] == best_ate
best_ate_value = actual_value.loc[test_idx][actual_is_best_ate].mean()
```

[10]:
```
cus = CounterfactualUnitSelector(learner=LogisticRegressionCV(),
                                 nevertaker_payoff=nevertaker_payoff,
                                 alwaystaker_payoff=alwaystaker_payoff,
                                 complier_payoff=complier_payoff,
                                 defier_payoff=defier_payoff)

cus.fit(data=df_train.drop('treatment_group_key', 1),
        treatment='treatment_numeric',
        outcome='conversion')

cus_pred = cus.predict(data=df_test.drop('treatment_group_key', 1),
                       treatment='treatment_numeric',
                       outcome='conversion')

best_cus = np.where(cus_pred > 0, 1, 0)
actual_is_cus = df_test['treatment_numeric'] == best_cus.ravel()
cus_value = actual_value.loc[test_idx][actual_is_cus].mean()
```

[11]:
```
labels = ['Random allocation', 'Best treatment assignment',
↪'CounterfactualUnitSelector']
values = [random_allocation_value, best_ate_value, cus_value]

plt.bar(labels, values)
plt.ylabel('Mean actual value in testing set')
plt.xticks(rotation=45)
plt.show()
```



5.11 Counterfactual Value Estimation Using Outcome Imputation by Li and Pearl (2019)

5.11.1 Introduction

The goal in uplift modeling is usually to predict the best treatment condition for an individual. Most of the time, the best treatment condition is assumed to be the one that has the highest probability of some "conversion event" such as the individual's purchasing a product. This is the traditional approach in which the goal is to maximize conversion.

However, if the goal of uplift modeling is to maximize value, then it is not safe to assume that the best treatment group is the one with the highest expected conversion. For example, it might be that the payoff from conversion is not sufficient to offset the cost of the treatment, or it might be that the treatment targets individuals who would convert anyway (Li and Pearl 2019). Therefore, it is often important to conduct some kind of value optimization together with uplift modeling, in order to determine the treatment group with the best value, not just the best lift.

The Causal ML package includes the CounterfactualValueEstimator class to conduct simple imputation-based value optimization. This notebook demonstrates the use of CounterfactualValueEstimator to determine the best treatment group when the costs of treatments are taken into account. We consider two kinds of costs:

- **Conversion costs** are those that we must endure if an individual who is in the treatment group converts. A typical example would be the cost of a promotional voucher.

- **Impression costs** are those that we need to pay for each individual in the treatment group irrespective of whether they convert. A typical example would be the cost associated with sending an SMS or email.

The proposed method takes two inputs: the CATE estimate $\hat{\tau}$ learned by any suitable method, and the predicted outcome for an individual learned by what we call the conversion probability model that estimates the conditional probability of conversion $P(Y = 1 \mid X = x, W = x)$ where W is the treatment group indicator. That is, the model estimates the probability of conversion for each individual using their observed pre-treatment features X. The output of this model is then combined with the predicted CATE in order to impute the expected conversion probability for each individual under *each treatment condition* as follows:

:nbsphinx-math:`begin{equation} hat{Y}_i^0 =

begin{cases} hat{m}(X_i, W_i) & text{for } W_i = 0 \ hat{m}(X_i, W_i) - hat{tau}_t(X_i) & text{for } W_i = t \ end{cases}

end{equation}`

:nbsphinx-math:`begin{equation} hat{Y}_i^t =

begin{cases} hat{m}(X_i, W_i) + hat{tau}_t(X_i) & text{for } W_i = 0 \ hat{m}(X_i, W_i) & text{for } W_i = t \ end{cases}

end{equation}`

The fact that we impute the conversion probability under each experimental condition–the actual as well as the counterfactual–gives our method its name. Using the estimated conversion probabilities, we then compute the expected payoff under each treatment condition while taking into account the value of conversion and the conversion and impression costs associated with each treatment, as follows (see Zhao and Harinen (2019) for more details):

:nbsphinx-math:`begin{equation}
mathbb{E}[(v - cc_t)Y_t - ic_t]

end{equation}`

where cc_t and ic_t are the conversion costs and impression costs, respectively.

[2]:
```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split

import xgboost as xgb

from causalml.dataset import make_uplift_classification
from causalml.inference.meta import BaseTClassifier
from causalml.optimize import CounterfactualValueEstimator
from causalml.optimize import get_treatment_costs
from causalml.optimize import get_actual_value

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')

%matplotlib inline
```

```
The sklearn.utils.testing module is  deprecated in version 0.22 and will be removed␣
↪in version 0.24. The corresponding classes / functions should instead be imported␣
↪from sklearn.utils. Anything that cannot be imported from sklearn.utils is now part␣
↪of the private API.
sklearn.tree._criterion.RegressionCriterion size changed, may indicate binary␣
↪incompatibility. Expected 168 from C header, got 360 from PyObject
```

(continues on next page)

(continued from previous page)

```
sklearn.tree._criterion.Criterion size changed, may indicate binary incompatibility.␣
↪Expected 160 from C header, got 352 from PyObject
sklearn.tree._criterion.ClassificationCriterion size changed, may indicate binary␣
↪incompatibility. Expected 176 from C header, got 368 from PyObject
```

5.11.2 Data generation

First, we simulate some heterogeneous treatment data using the built-in function.

[3]:
```
df, X_names = make_uplift_classification(
    n_samples=5000, treatment_name=['control', 'treatment1', 'treatment2'])
```

In this example, we assume there are no costs associated with assigning units into the control group, and that for the two treatment groups the conversion cost are \$2.5 and \$5, respectively. We assume the impression costs to be zero for one of the treatments and \$0.02 for the other. We also specify the payoff, which we here assume to be the same for everyone, \$20. However, these values could vary from individual to individual.

[4]:
```
# Put costs into dicts
conversion_cost_dict = {'control': 0, 'treatment1': 2.5, 'treatment2': 5}
impression_cost_dict = {'control': 0, 'treatment1': 0, 'treatment2': 0.02}

# Use a helper function to put treatment costs to array
cc_array, ic_array, conditions = get_treatment_costs(treatment=df['treatment_group_key
↪'],
                                                     control_name='control',
                                                     cc_dict=conversion_cost_dict,
                                                     ic_dict=impression_cost_dict)

# Put the conversion value into an array
conversion_value_array = np.full(df.shape[0], 20)
```

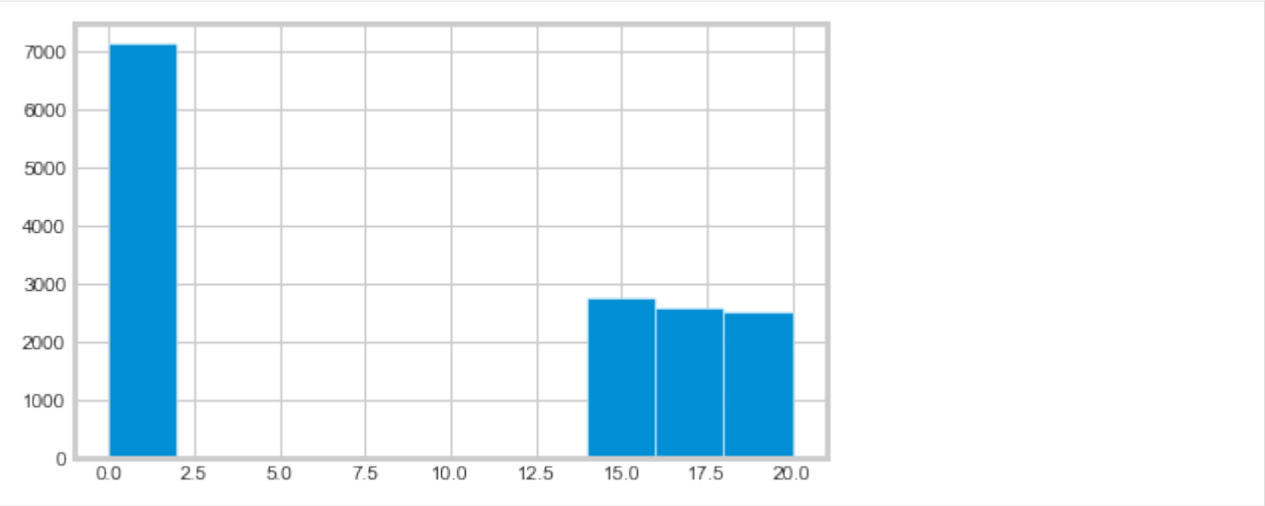
Next we calculate the value of actually having an individual in their actual treatment group using the equation for expected value under a treatment, ie:

:nbsphinx-math:`begin{equation}
mathbb{E}[(v - cc_t)Y_t - ic_t]

end{equation}`

[5]:
```
# Use a helper function to obtain the value of actual treatment
actual_value = get_actual_value(treatment=df['treatment_group_key'],
                                observed_outcome=df['conversion'],
                                conversion_value=conversion_value_array,
                                conditions=conditions,
                                conversion_cost=cc_array,
                                impression_cost=ic_array)
```

[6]:
```
plt.hist(actual_value)
plt.show()
```



5.11.3 Model evaluation

A common problem in the uplift modeling literature is that of evaluating the quality of the treatment recommendations produced by a model. The evaluation of uplift models is tricky because we do not observe treatment effects at an individual level directly in non-simulated data, so it is not possible to use standard model evaluation metrics such as mean squared error. Consequently, various authors have proposed various ways to work around this issue. For example, Schuler et al (2018) identify seven different evaluation strategies used in the literature.

Below, we use the approach of model evaluation put forward by Kaepelner et al (2014). The idea in this method is to evaluate the improvement we would gain if we targeted some as-yet untreated future population by using the recommendations produced by a particular model. To do so, we split the data into disjoint training and testing sets, and train our model on the training data. We then use the model to predict the best treatment group for units in the testing data, which in a simple two-arm trial is either treatment or control. In order to estimate the outcome for the future population if the model were to be used, we then select a subset of the testing data based on whether their observed treatment allocation happens to be the same as the one recommended by the model. This population is called "lucky".

Predicted best treatment	Actual treatment	Lucky
Control	Control	Yes
Control	Treatment	No
Treatment	Treatment	Yes
Treatment	Control	No

The average outcome for the "lucky" population can be taken to represent what the outcome would be for a future untreated population if we were to use the uplift model in question to allocate treatments. Recall that in all of the experiments the treatments are assumed to have been allocated randomly across the total population, so there should be no selection bias. The average outcome under a given model can then be compared with alternative treatment allocation strategies. As Kaepelner et al (2014) point out, two common strategies are random allocation and "best treatment" allocation. To estimate what the outcome for a future population would be under random allocation, we can simply look at the sample mean across the total test population. To estimate the same for the "best treatment" assignment, we can look at those units in the test set whose observed treatment assignment corresponds to the treatment group with the best average treatment effect. These alternative targeting strategies are interesting because they are a common practice in industry applications and elsewhere.

Performance against benchmarks

In this section, we compare four different targeting strategies:

- Random treatment allocation under which all units in the testing set are randomly assigned to treatments
- The "best treatment" allocation under which all units in the testing set are assigned to the treatment with the best conversion in the training set
- Allocation under an uplift model in which all units in the testing set are assigned to the treatment which is predicted to have the highest conversion rate according to an uplift model trained on the training set
- Allocation under the counterfactual value estimator model in which all units are assigned to the treatment group with the best predicted payoff

[7]:
```
df_train, df_test = train_test_split(df)
train_idx = df_train.index
test_idx = df_test.index
```

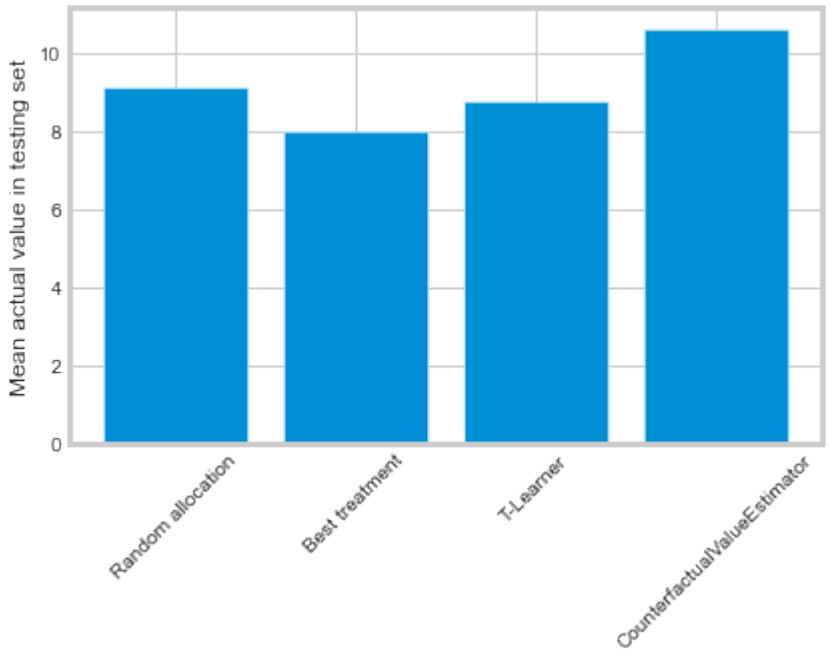
[8]:
```
# Calculate the benchmark value according to the random allocation
# and best treatment schemes
random_allocation_value = actual_value.loc[test_idx].mean()

best_ate = df_train.groupby(
    'treatment_group_key')['conversion'].mean().idxmax()

actual_is_best_ate = df_test['treatment_group_key'] == best_ate

best_ate_value = actual_value.loc[test_idx][actual_is_best_ate].mean()
```

[9]:
```
# Calculate the value under an uplift model
tm = BaseTClassifier(control_learner=xgb.XGBClassifier(),
                     treatment_learner=xgb.XGBClassifier(),
                     control_name='control')

tm.fit(df_train[X_names].values,
       df_train['treatment_group_key'],
       df_train['conversion'])

tm_pred = tm.predict(df_test[X_names].values)

pred_df = pd.DataFrame(tm_pred, columns=tm._classes)
tm_best = pred_df.idxmax(axis=1)
actual_is_tm_best = df_test['treatment_group_key'] == tm_best.ravel()
tm_value = actual_value.loc[test_idx][actual_is_tm_best].mean()
```

[10]:
```
# Estimate the conditional mean model; this is a pure curve
# fitting exercise
proba_model = xgb.XGBClassifier()

W_dummies = pd.get_dummies(df['treatment_group_key'])
XW = np.c_[df[X_names], W_dummies]

proba_model.fit(XW[train_idx], df_train['conversion'])
y_proba = proba_model.predict_proba(XW[test_idx])[:, 1]
```

[11]:
```
# Run the counterfactual calculation with TwoModel prediction
cve = CounterfactualValueEstimator(treatment=df_test['treatment_group_key'],
                                   control_name='control',
                                   treatment_names=conditions[1:],
                                   y_proba=y_proba,
                                   cate=tm_pred,
                                   value=conversion_value_array[test_idx],
                                   conversion_cost=cc_array[test_idx],
                                   impression_cost=ic_array[test_idx])

cve_best_idx = cve.predict_best()
cve_best = [conditions[idx] for idx in cve_best_idx]
actual_is_cve_best = df.loc[test_idx, 'treatment_group_key'] == cve_best
cve_value = actual_value.loc[test_idx][actual_is_cve_best].mean()
```

[12]:
```
labels = [
    'Random allocation',
    'Best treatment',
    'T-Learner',
    'CounterfactualValueEstimator'
]

values = [
    random_allocation_value,
    best_ate_value,
    tm_value,
    cve_value
]

plt.bar(labels, values)
plt.ylabel('Mean actual value in testing set')
plt.xticks(rotation=45)
plt.show()
```



Here, only CounterfactualValueEstimator improves upon random targeting. The "best treatment" and T-Learner approaches likely perform worse because they recommend costly treatments to individuals who would convert anyway.

5.12 Feature Selection for Uplift Trees by Zhao et al. (2020)

This notebook includes two sections:
- **Feature selection**: demonstrate how to use Filter methods to select the most important numeric features -
Performance evaluation: evaluate the AUUC performance with top features dataset

(Paper reference: Zhao, Zhenyu, et al. "Feature Selection Methods for Uplift Modeling." arXiv preprint arXiv:2005.03447 (2020).*)*

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
from causalml.dataset import make_uplift_classification
```

5.12.1 Import FilterSelect class for Filter methods

[3]:
```
from causalml.feature_selection.filters import FilterSelect
```

[4]:
```
from causalml.inference.tree import UpliftRandomForestClassifier
from causalml.inference.meta import BaseXRegressor, BaseRRegressor, BaseSRegressor,␣
↪BaseTRegressor
from causalml.metrics import plot_gain, auuc_score
```

[5]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

[6]:
```
import logging

logger = logging.getLogger('causalml')
logging.basicConfig(level=logging.INFO)
```

Generate dataset

Generate synthetic data using the built-in function.

[7]:
```
# define parameters for simulation

y_name = 'conversion'
treatment_group_keys = ['control', 'treatment1']
n = 10000
n_classification_features = 50
n_classification_informative = 10
n_classification_repeated = 0
n_uplift_increase_dict = {'treatment1': 8}
n_uplift_decrease_dict = {'treatment1': 4}
```

(continues on next page)

(continued from previous page)

```
delta_uplift_increase_dict = {'treatment1': 0.1}
delta_uplift_decrease_dict = {'treatment1': -0.1}

random_seed = 20200808
```

[8]:
```
df, X_names = make_uplift_classification(
    treatment_name=treatment_group_keys,
    y_name=y_name,
    n_samples=n,
    n_classification_features=n_classification_features,
    n_classification_informative=n_classification_informative,
    n_classification_repeated=n_classification_repeated,
    n_uplift_increase_dict=n_uplift_increase_dict,
    n_uplift_decrease_dict=n_uplift_decrease_dict,
    delta_uplift_increase_dict = delta_uplift_increase_dict,
    delta_uplift_decrease_dict = delta_uplift_decrease_dict,
    random_seed=random_seed
)
```

```
INFO:numexpr.utils:Note: NumExpr detected 12 cores but "NUMEXPR_MAX_THREADS" not set,␣
↪so enforcing safe limit of 8.
INFO:numexpr.utils:NumExpr defaulting to 8 threads.
```

[9]:
```
df.head()
```

[9]:

	treatment_group_key	x1_informative	x2_informative	x3_informative	x4_informative	x5_informative	x6_informative	x7_informative	x8_informative	x9_informative	…	x56_uplift_increase	x57_uplift_increase	x58_uplift_increase	x59_increase_mix	x60_uplift_decrease	x61_uplift_decrease	x62_uplift_decrease
0	treatment1	-4.004496	-1.250351	-2.800557	-0.368288	-0.115549	-2.492826	0.369516	0.290526	0.465153	…	0.496144	1.847680	-0.337894	-0.672058	1.180352	0.778013	0.931000
1	treatment1	-3.170028	-0.135293	1.484246	-2.131584	-0.760103	1.764765	0.972124	1.407131	-1.027603	…	0.574955	3.578138	0.678118	-0.545227	-0.143942	-0.015188	1.189643
2	treatment1	-0.763386	-0.785612	1.218781	-0.725835	1.044489	-1.521071	-2.266684	-1.614818	-0.113647	…	0.985076	1.079181	0.578092	0.574370	-0.477429	0.679070	1.650897
3	control	0.887727	0.049095	-2.242776	1.530966	0.392623	-0.203071	-0.549329	0.107296	-0.542277	…	-0.175352	0.683330	0.567545	0.349622			
4	control	-1.672922	-1.156145	3.871476	-1.883713	-0.220122	-4.615669	0.141980	-0.933756	-0.765592	…	0.485798	-0.355315	0.982488	-0.007260			

(continues on next page)

(continued from previous page)

```
3            -0.789203             2.315184             0.658607
4             2.895155             0.261848            -1.337001

   x63_uplift_decrease  conversion  treatment_effect
0             2.947160           0                 0
1             1.943692           1                 0
2             2.768897           1                 0
3             1.734836           0                 0
4            -0.639983           1                 0

[5 rows x 66 columns]
```

[10]:
```
# Look at the conversion rate and sample size in each group
df.pivot_table(values='conversion',
               index='treatment_group_key',
               aggfunc=[np.mean, np.size],
               margins=True)
```

[10]:
```
                          mean       size
                    conversion conversion
treatment_group_key
control                0.50180      10000
treatment1             0.59750      10000
All                    0.54965      20000
```

[11]:
```
X_names
```

[11]:
```
['x1_informative',
 'x2_informative',
 'x3_informative',
 'x4_informative',
 'x5_informative',
 'x6_informative',
 'x7_informative',
 'x8_informative',
 'x9_informative',
 'x10_informative',
 'x11_irrelevant',
 'x12_irrelevant',
 'x13_irrelevant',
 'x14_irrelevant',
 'x15_irrelevant',
 'x16_irrelevant',
 'x17_irrelevant',
 'x18_irrelevant',
 'x19_irrelevant',
 'x20_irrelevant',
 'x21_irrelevant',
 'x22_irrelevant',
 'x23_irrelevant',
 'x24_irrelevant',
 'x25_irrelevant',
 'x26_irrelevant',
 'x27_irrelevant',
 'x28_irrelevant',
 'x29_irrelevant',
 'x30_irrelevant',
```

(continues on next page)

(continued from previous page)

```
 'x31_irrelevant',
 'x32_irrelevant',
 'x33_irrelevant',
 'x34_irrelevant',
 'x35_irrelevant',
 'x36_irrelevant',
 'x37_irrelevant',
 'x38_irrelevant',
 'x39_irrelevant',
 'x40_irrelevant',
 'x41_irrelevant',
 'x42_irrelevant',
 'x43_irrelevant',
 'x44_irrelevant',
 'x45_irrelevant',
 'x46_irrelevant',
 'x47_irrelevant',
 'x48_irrelevant',
 'x49_irrelevant',
 'x50_irrelevant',
 'x51_uplift_increase',
 'x52_uplift_increase',
 'x53_uplift_increase',
 'x54_uplift_increase',
 'x55_uplift_increase',
 'x56_uplift_increase',
 'x57_uplift_increase',
 'x58_uplift_increase',
 'x59_increase_mix',
 'x60_uplift_decrease',
 'x61_uplift_decrease',
 'x62_uplift_decrease',
 'x63_uplift_decrease']
```

Feature selection with Filter methods

method = F (F Filter)

[12]:
```
filter_method = FilterSelect()
```

[13]:
```
# F Filter with order 1
method = 'F'
f_imp = filter_method.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1')
f_imp.head()
```

[13]:

	method	feature	rank	score	p_value	\
0	F filter	x53_uplift_increase	1.0	190.321410	4.262512e-43	
0	F filter	x57_uplift_increase	2.0	127.136380	2.127676e-29	
0	F filter	x3_informative	3.0	66.273458	4.152970e-16	
0	F filter	x4_informative	4.0	59.407590	1.341417e-14	
0	F filter	x62_uplift_decrease	5.0	3.957507	4.667636e-02	

misc

(continues on next page)

(continued from previous page)

```
0  df_num: 1.0, df_denom: 19996.0, order:1
0  df_num: 1.0, df_denom: 19996.0, order:1
0  df_num: 1.0, df_denom: 19996.0, order:1
0  df_num: 1.0, df_denom: 19996.0, order:1
0  df_num: 1.0, df_denom: 19996.0, order:1
```

[14]:
```
# F Filter with order 2
method = 'F'
f_imp = filter_method.get_importance(df, X_names, y_name, method,
                     treatment_group = 'treatment1', order=2)
f_imp.head()
```

[14]:
```
     method              feature  rank       score       p_value  \
0  F filter  x53_uplift_increase   1.0  107.368286  4.160720e-47
0  F filter  x57_uplift_increase   2.0   70.138050  4.423736e-31
0  F filter       x3_informative   3.0   36.499465  1.504356e-16
0  F filter       x4_informative   4.0   31.780547  1.658731e-14
0  F filter  x55_uplift_increase   5.0   27.494904  1.189886e-12

                                    misc
0  df_num: 2.0, df_denom: 19994.0, order:2
0  df_num: 2.0, df_denom: 19994.0, order:2
0  df_num: 2.0, df_denom: 19994.0, order:2
0  df_num: 2.0, df_denom: 19994.0, order:2
0  df_num: 2.0, df_denom: 19994.0, order:2
```

[15]:
```
# F Filter with order 3
method = 'F'
f_imp = filter_method.get_importance(df, X_names, y_name, method,
                     treatment_group = 'treatment1', order=3)
f_imp.head()
```

[15]:
```
     method              feature  rank      score       p_value  \
0  F filter  x53_uplift_increase   1.0  72.064224  2.373628e-46
0  F filter  x57_uplift_increase   2.0  46.841718  3.710784e-30
0  F filter       x3_informative   3.0  24.089980  1.484634e-15
0  F filter       x4_informative   4.0  23.097310  6.414267e-15
0  F filter  x55_uplift_increase   5.0  18.072880  1.044117e-11

                                    misc
0  df_num: 3.0, df_denom: 19992.0, order:3
0  df_num: 3.0, df_denom: 19992.0, order:3
0  df_num: 3.0, df_denom: 19992.0, order:3
0  df_num: 3.0, df_denom: 19992.0, order:3
0  df_num: 3.0, df_denom: 19992.0, order:3
```

method = LR (likelihood ratio test)

[16]:
```
# LR Filter with order 1
method = 'LR'
lr_imp = filter_method.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1')
lr_imp.head()
```

[16]:
```
      method              feature  rank       score       p_value  \
0  LR filter  x53_uplift_increase   1.0  203.811674  0.000000e+00
0  LR filter  x57_uplift_increase   2.0  133.175328  0.000000e+00
0  LR filter       x3_informative   3.0   64.366711  9.992007e-16
0  LR filter       x4_informative   4.0   52.389798  4.550804e-13
0  LR filter  x62_uplift_decrease   5.0    4.064347  4.379760e-02

              misc
0  df: 1, order: 1
0  df: 1, order: 1
0  df: 1, order: 1
0  df: 1, order: 1
0  df: 1, order: 1
```

[17]:
```
# LR Filter with order 2
method = 'LR'
lr_imp = filter_method.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1',order=2)
lr_imp.head()
```

[17]:
```
      method              feature  rank       score       p_value  \
0  LR filter  x53_uplift_increase   1.0  277.639095  0.000000e+00
0  LR filter  x57_uplift_increase   2.0  156.134112  0.000000e+00
0  LR filter  x55_uplift_increase   3.0   71.478979  3.330669e-16
0  LR filter       x3_informative   4.0   44.938973  1.744319e-10
0  LR filter       x4_informative   5.0   29.179971  4.609458e-07

              misc
0  df: 2, order: 2
0  df: 2, order: 2
0  df: 2, order: 2
0  df: 2, order: 2
0  df: 2, order: 2
```

[18]:
```
# LR Filter with order 3
method = 'LR'
lr_imp = filter_method.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1',order=3)
lr_imp.head()
```

[18]:
```
      method              feature  rank       score       p_value  \
0  LR filter  x53_uplift_increase   1.0  290.389201  0.000000e+00
0  LR filter  x57_uplift_increase   2.0  153.942614  0.000000e+00
0  LR filter  x55_uplift_increase   3.0   70.626667  3.108624e-15
0  LR filter       x3_informative   4.0   45.477851  7.323235e-10
0  LR filter       x4_informative   5.0   30.466528  1.100881e-06

              misc
0  df: 3, order: 3
0  df: 3, order: 3
```

(continues on next page)

(continued from previous page)

```
0  df: 3, order: 3
0  df: 3, order: 3
0  df: 3, order: 3
```

method = KL (KL divergence)

[19]:
```
method = 'KL'
kl_imp = filter_method.get_importance(df, X_names, y_name, method,
                      treatment_group = 'treatment1',
                      n_bins=10)
kl_imp.head()
```

[19]:

	method	feature	rank	score	p_value	misc
0	KL filter	x53_uplift_increase	1.0	0.022997	None	number_of_bins: 10
0	KL filter	x57_uplift_increase	2.0	0.014884	None	number_of_bins: 10
0	KL filter	x4_informative	3.0	0.012103	None	number_of_bins: 10
0	KL filter	x3_informative	4.0	0.010179	None	number_of_bins: 10
0	KL filter	x55_uplift_increase	5.0	0.003836	None	number_of_bins: 10

We found all these 3 filter methods were able to rank most of the **informative** and **uplift increase** features on the top.

Performance evaluation

Evaluate the AUUC (Area Under the Uplift Curve) score with several uplift models when using top features dataset

[20]:
```
# train test split
df_train, df_test = train_test_split(df, test_size=0.2, random_state=111)
```

[21]:
```
# convert treatment column to 1 (treatment1) and 0 (control)
treatments = np.where((df_test['treatment_group_key']=='treatment1'), 1, 0)
print(treatments[:10])
print(df_test['treatment_group_key'][:10])
```

```
[0 0 1 1 0 1 1 0 0 0]
18998       control
11536       control
8552     treatment1
2652     treatment1
19671       control
13244    treatment1
3075     treatment1
8746        control
18530       control
5066        control
Name: treatment_group_key, dtype: object
```

Uplift RandomForest Classfier

[22]:
```
uplift_model = UpliftRandomForestClassifier(control_name='control', max_depth=8)
```

[23]:
```
# using all features
features = X_names
uplift_model.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds = uplift_model.predict(df_test[features].values)
```

Select top N features based on KL filter

[24]:
```
top_n = 10
top_10_features = kl_imp['feature'][:top_n]
print(top_10_features)
```

```
0    x53_uplift_increase
0    x57_uplift_increase
0         x4_informative
0         x3_informative
0    x55_uplift_increase
0         x1_informative
0    x56_uplift_increase
0    x51_uplift_increase
0         x38_irrelevant
0    x58_uplift_increase
Name: feature, dtype: object
```

[25]:
```
top_n = 15
top_15_features = kl_imp['feature'][:top_n]
print(top_15_features)
```

```
0    x53_uplift_increase
0    x57_uplift_increase
0         x4_informative
0         x3_informative
0    x55_uplift_increase
0         x1_informative
0    x56_uplift_increase
0    x51_uplift_increase
0         x38_irrelevant
0    x58_uplift_increase
0         x48_irrelevant
0         x15_irrelevant
0         x27_irrelevant
0    x62_uplift_decrease
0         x23_irrelevant
Name: feature, dtype: object
```

[26]:
```
top_n = 20
top_20_features = kl_imp['feature'][:top_n]
print(top_20_features)
```

```
0    x53_uplift_increase
0    x57_uplift_increase
```

(continues on next page)

(continued from previous page)

```
0          x4_informative
0          x3_informative
0     x55_uplift_increase
0          x1_informative
0     x56_uplift_increase
0     x51_uplift_increase
0          x38_irrelevant
0     x58_uplift_increase
0          x48_irrelevant
0          x15_irrelevant
0          x27_irrelevant
0     x62_uplift_decrease
0          x23_irrelevant
0          x29_irrelevant
0          x6_informative
0          x45_irrelevant
0          x40_irrelevant
0          x25_irrelevant
Name: feature, dtype: object
```

5.12.2 Train the Uplift model again with top N features

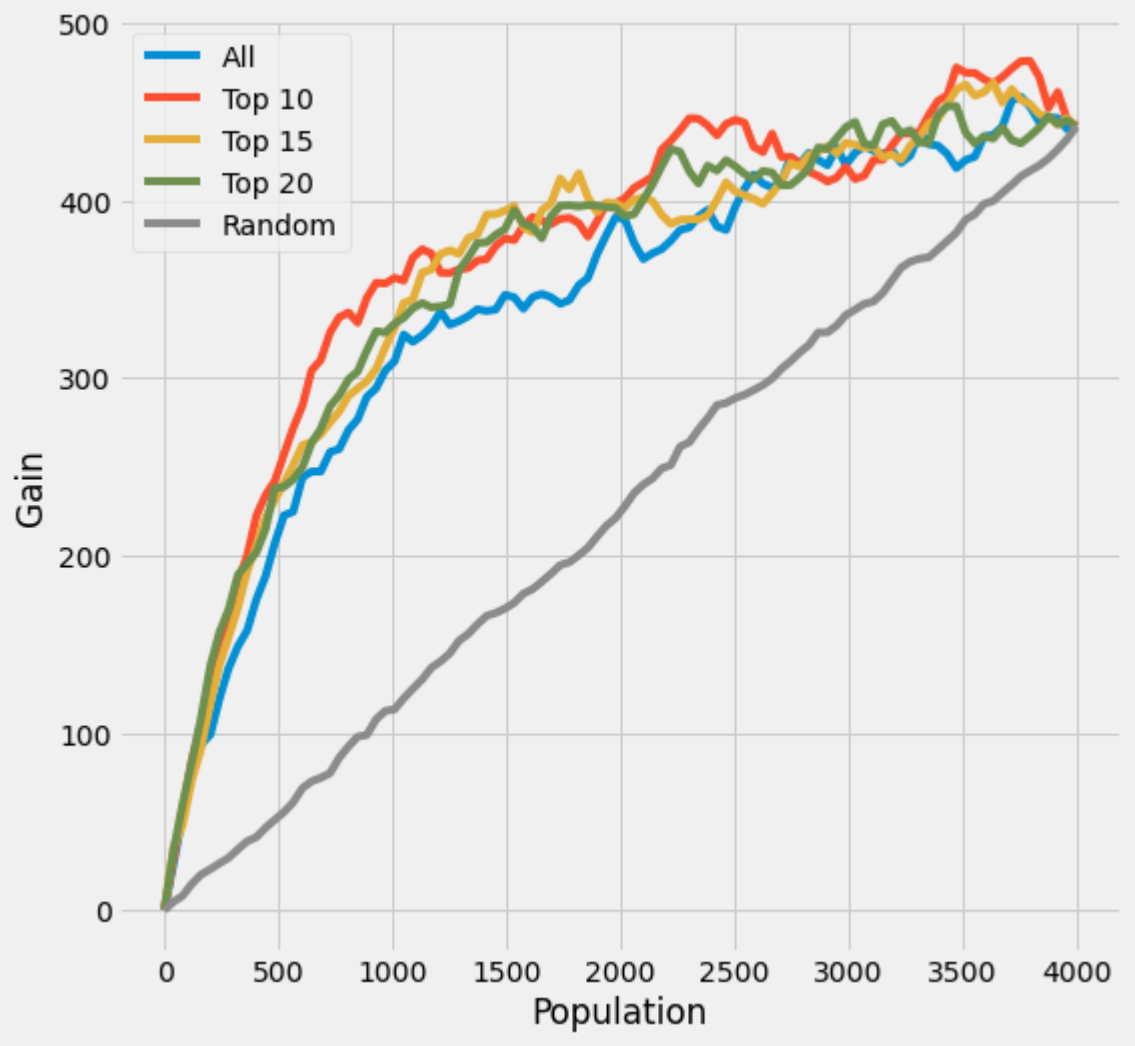
[27]:
```
# using top 10 features
features = top_10_features

uplift_model.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds_t10 = uplift_model.predict(df_test[features].values)
```

[28]:
```
# using top 15 features
features = top_15_features

uplift_model.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds_t15 = uplift_model.predict(df_test[features].values)
```

[29]:
```
# using top 20 features
features = top_20_features

uplift_model.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds_t20 = uplift_model.predict(df_test[features].values)
```

Print results for Uplift model

[30]:
```
df_preds = pd.DataFrame([y_preds.ravel(),
                         y_preds_t10.ravel(),
                         y_preds_t15.ravel(),
                         y_preds_t20.ravel(),
                         treatments,
                         df_test[y_name].ravel()],
                        index=['All', 'Top 10', 'Top 15', 'Top 20', 'is_treated', y_
↪name]).T

plot_gain(df_preds, outcome_col=y_name, treatment_col='is_treated')
```



[31]:
```
auuc_score(df_preds, outcome_col=y_name, treatment_col='is_treated')
```

[31]:
```
All         0.773405
Top 10      0.841204
Top 15      0.816100
Top 20      0.816252
Random      0.506801
```

(continues on next page)

(continued from previous page)

```
dtype: float64
```

R Learner as base and feed in Random Forest Regressor

[32]:
```
r_rf_learner = BaseRRegressor(
    RandomForestRegressor(
        n_estimators = 100,
        max_depth = 8,
        min_samples_leaf = 100
    ),
control_name='control')
```

[33]:
```
# using all features
features = X_names
r_rf_learner.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds = r_rf_learner.predict(df_test[features].values)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment1 with R-loss
```

[34]:
```
# using top 10 features
features = top_10_features
r_rf_learner.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds_t10 = r_rf_learner.predict(df_test[features].values)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment1 with R-loss
```

[35]:
```
# using top 15 features
features = top_15_features
r_rf_learner.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds_t15 = r_rf_learner.predict(df_test[features].values)
```

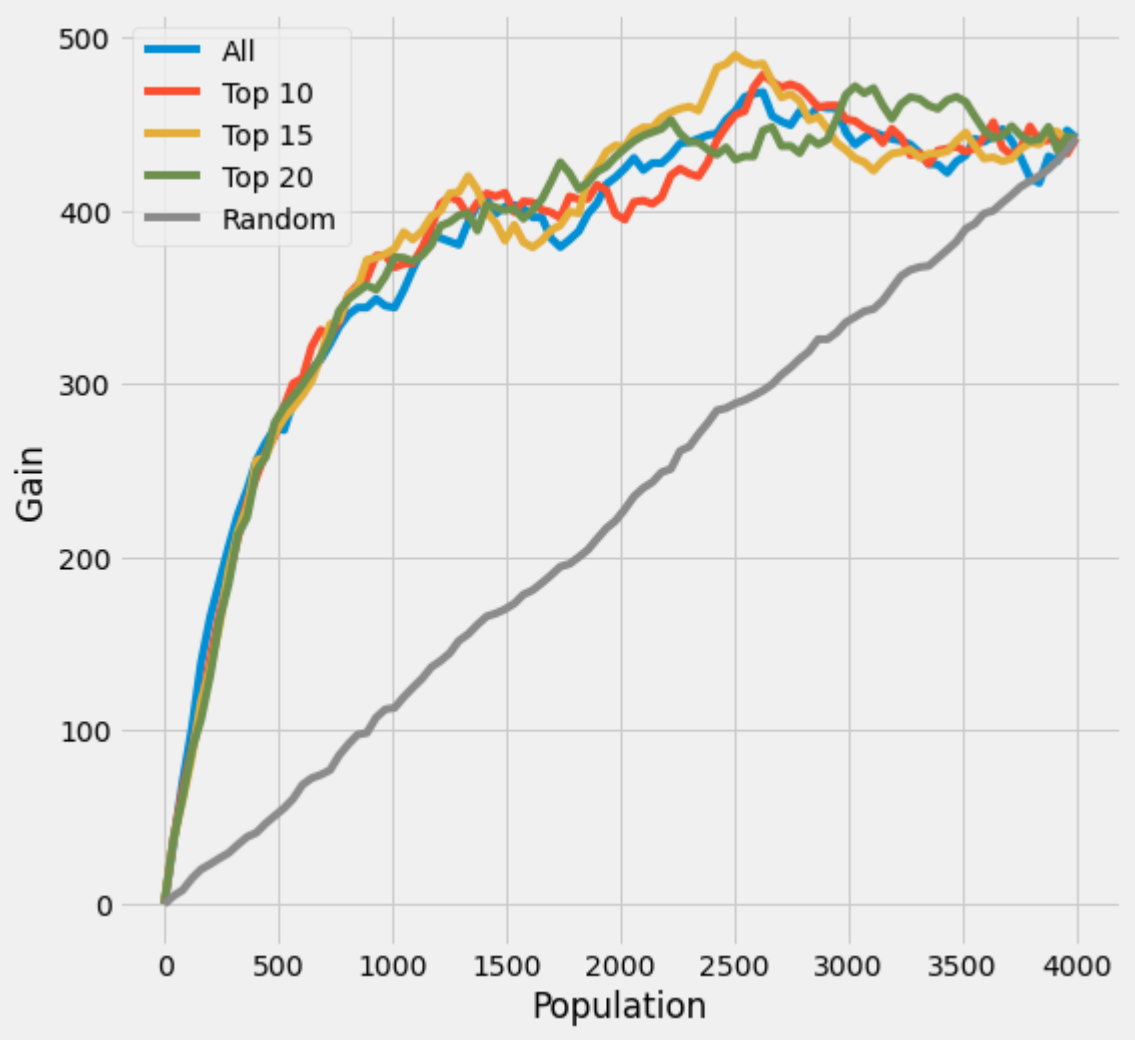
```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment1 with R-loss
```

[36]:
```
# using top 20 features
features = top_20_features
r_rf_learner.fit(X = df_train[features].values,
                 treatment = df_train['treatment_group_key'].values,
                 y = df_train[y_name].values)
y_preds_t20 = r_rf_learner.predict(df_test[features].values)
```

```
INFO:causalml:Generating propensity score
INFO:causalml:Calibrating propensity scores.
INFO:causalml:generating out-of-fold CV outcome estimates
INFO:causalml:training the treatment effect model for treatment1 with R-loss
```

Print results for R Learner

[37]:
```
df_preds = pd.DataFrame([y_preds.ravel(),
                         y_preds_t10.ravel(),
                         y_preds_t15.ravel(),
                         y_preds_t20.ravel(),
                         treatments,
                         df_test[y_name].ravel()],
                        index=['All', 'Top 10', 'Top 15', 'Top 20', 'is_treated', y_
↪name]).T

plot_gain(df_preds, outcome_col=y_name, treatment_col='is_treated')
```



[38]:
```
# print out AUUC score
auuc_score(df_preds, outcome_col=y_name, treatment_col='is_treated')
```

[38]:
```
All        0.859891
Top 10     0.865159
Top 15     0.872650
Top 20     0.870669
Random     0.506801
dtype: float64
```

(a relatively smaller enhancement on the AUUC is observed in this R Learner case)

S Learner as base and feed in Random Forest Regressor

[39]:
```
slearner_rf = BaseSRegressor(
    RandomForestRegressor(
        n_estimators = 100,
        max_depth = 8,
        min_samples_leaf = 100
    ),
    control_name='control')
```

[40]:
```
# using all features
features = X_names
slearner_rf.fit(X = df_train[features].values,
                treatment = df_train['treatment_group_key'].values,
                y = df_train[y_name].values)
y_preds = slearner_rf.predict(df_test[features].values)
```

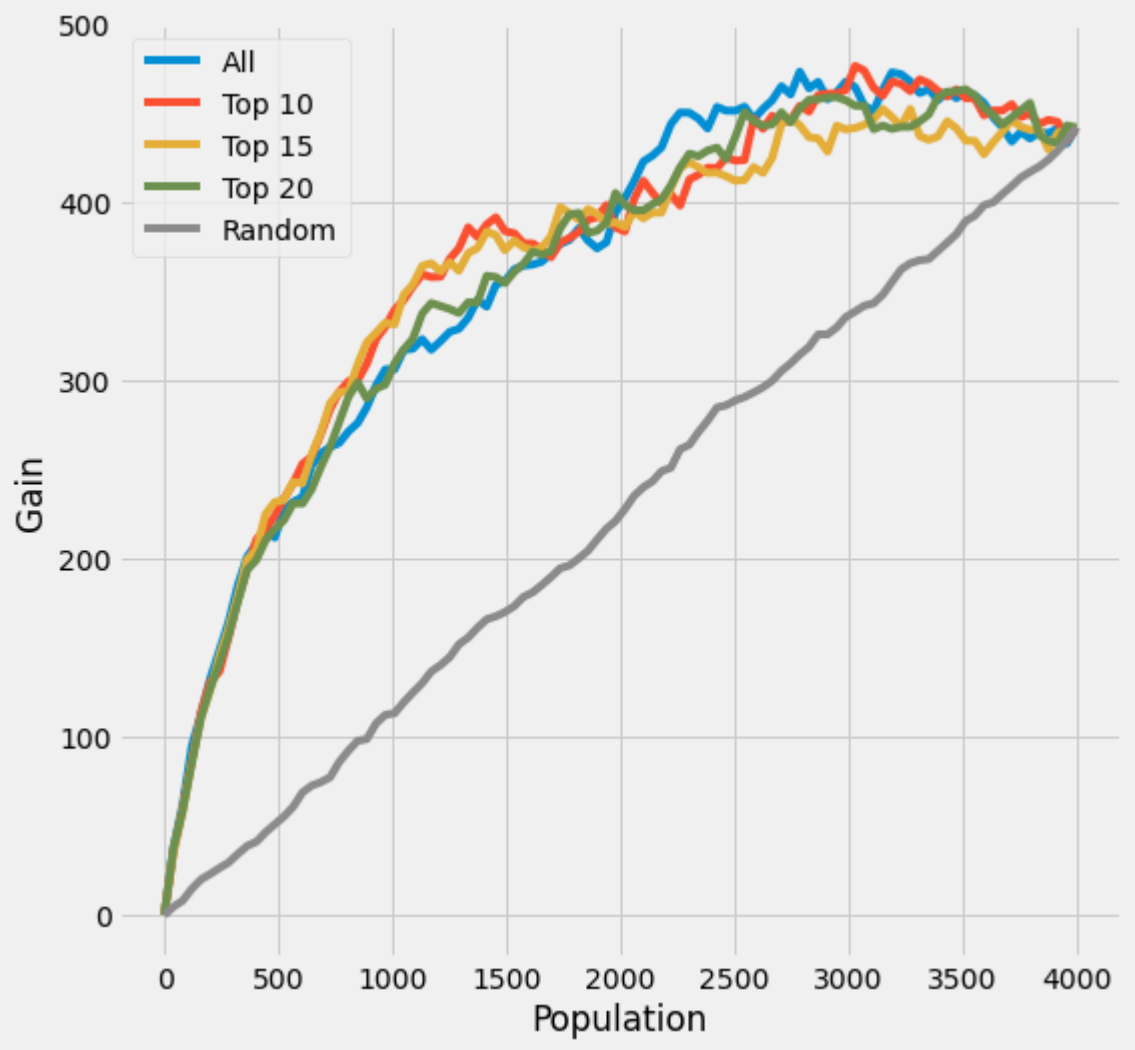
[41]:
```
# using top 10 features
features = top_10_features
slearner_rf.fit(X = df_train[features].values,
                treatment = df_train['treatment_group_key'].values,
                y = df_train[y_name].values)
y_preds_t10 = slearner_rf.predict(df_test[features].values)
```

[42]:
```
# using top 15 features
features = top_15_features
slearner_rf.fit(X = df_train[features].values,
                treatment = df_train['treatment_group_key'].values,
                y = df_train[y_name].values)
y_preds_t15 = slearner_rf.predict(df_test[features].values)
```

[43]:
```
# using top 20 features
features = top_20_features
slearner_rf.fit(X = df_train[features].values,
                treatment = df_train['treatment_group_key'].values,
                y = df_train[y_name].values)
y_preds_t20 = slearner_rf.predict(df_test[features].values)
```

Print results for S Learner

[44]:
```
df_preds = pd.DataFrame([y_preds.ravel(),
                         y_preds_t10.ravel(),
                         y_preds_t15.ravel(),
                         y_preds_t20.ravel(),
                         treatments,
                         df_test[y_name].ravel()],
                        index=['All', 'Top 10', 'Top 15', 'Top 20', 'is_treated', y_
↪name]).T

plot_gain(df_preds, outcome_col=y_name, treatment_col='is_treated')
```



[45]:
```
# print out AUUC score
auuc_score(df_preds, outcome_col=y_name, treatment_col='is_treated')
```

[45]:
```
All        0.824483
Top 10     0.832872
Top 15     0.817835
Top 20     0.816149
Random     0.506801
```

(continues on next page)

(continued from previous page)

```
dtype: float64
```

In this notebook, we demonstrated how our Filter method functions are able to select important features and enhance the AUUC performance (while the results might vary among different datasets, models and hyper-parameters).

[ ]:

5.13 Policy Learner by Athey and Wager (2018) with Binary Treatment

This notebook demonstrates the use of the CausalML implementation of the policy learner by Athey and Wager (2018) (https://arxiv.org/abs/1702.02896).

[1]:
```
%load_ext autoreload
%autoreload 2
```

[2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

[3]:
```
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.ensemble import GradientBoostingRegressor, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
```

[4]:
```
from causalml.optimize import PolicyLearner
from sklearn.tree import plot_tree
from lightgbm import LGBMRegressor
from causalml.inference.meta import BaseXRegressor
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
RuntimeError: module compiled against API version 0xe but this version of numpy is 0xd
```

```
The sklearn.utils.testing module is  deprecated in version 0.22 and will be removed␣
↪in version 0.24. The corresponding classes / functions should instead be imported␣
↪from sklearn.utils. Anything that cannot be imported from sklearn.utils is now part␣
↪of the private API.
```

5.13.1 Binary treatment policy learning

First we generate a synthetic data set with binary treatment. The treatment is random conditioned on covariates. The treatment effect is heterogeneous where for some individuals it is negative. We use a policy learner to classify the individuals into treat/no-treat groups to maximize the total treatment effect.

[5]:
```
np.random.seed(1234)

n = 10000
p = 10

X = np.random.normal(size=(n, p))
ee = 1 / (1 + np.exp(X[:, 2]))
tt = 1 / (1 + np.exp(X[:, 0] + X[:, 1])/2) - 0.5
```

(continues on next page)

(continued from previous page)

```
W = np.random.binomial(1, ee, n)
Y = X[:, 2] + W * tt + np.random.normal(size=n)
```

Use policy learner with default outcome/treatment estimator and a simple policy classifier.

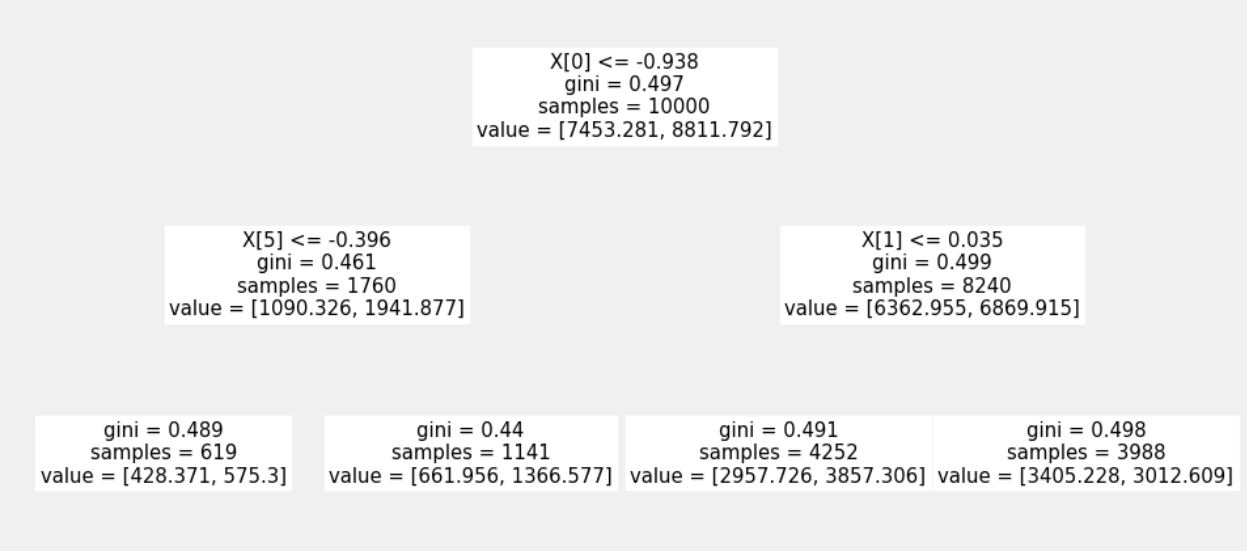
[6]:
```
policy_learner = PolicyLearner(policy_learner=DecisionTreeClassifier(max_depth=2),␣
↪calibration=True)
```

[7]:
```
policy_learner.fit(X, W, Y)
```

[7]:
```
PolicyLearner(model_mu=GradientBoostingRegressor(),
        model_w=GradientBoostingClassifier(),
\model_pi=DecisionTreeClassifier(max_depth=2))
```

[8]:
```
plt.figure(figsize=(15,7))
plot_tree(policy_learner.model_pi)
```

[8]:
```
[Text(469.8, 340.2, 'X[0] <= -0.938\ngini = 0.497\nsamples = 10000\nvalue = [7453.281,
↪ 8811.792]'),
 Text(234.9, 204.12, 'X[5] <= -0.396\ngini = 0.461\nsamples = 1760\nvalue = [1090.326,
↪ 1941.877]'),
 Text(117.45, 68.03999999999996, 'gini = 0.489\nsamples = 619\nvalue = [428.371, 575.
↪3]'),
 Text(352.35, 68.03999999999996, 'gini = 0.44\nsamples = 1141\nvalue = [661.956, 1366.
↪577]'),
 Text(704.7, 204.12, 'X[1] <= 0.035\ngini = 0.499\nsamples = 8240\nvalue = [6362.955,␣
↪6869.915]'),
 Text(587.25, 68.03999999999996, 'gini = 0.491\nsamples = 4252\nvalue = [2957.726,␣
↪3857.306]'),
 Text(822.15, 68.03999999999996, 'gini = 0.498\nsamples = 3988\nvalue = [3405.228,␣
↪3012.609]')]
```



Alternatively, one can construct a policy directly from the ITE estimated from a X-learner.

[9]:
```
learner_x = BaseXRegressor(LGBMRegressor())
ite_x = learner_x.fit_predict(X=X, treatment=W, y=Y)
```

In this example policy learner outperforms the ITE-based policy and gets close to the true optimal.

[10]:
```
pd.DataFrame({
    'DR-DT Optimal': [np.mean((policy_learner.predict(X) + 1) * tt / 2)],
    'True Optimal': [np.mean((np.sign(tt) + 1) * tt / 2)],
    'X Learner': [
        np.mean((np.sign(ite_x) + 1) * tt / 2)
    ],
})
```

[10]:

	DR-DT Optimal	True Optimal	X Learner
0	0.157055	0.183291	0.083172

[ ]:
```
```

5.14 CEVAE vs. Meta-Learners Benchmark with IHDP + Synthetic Datasets

[1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import torch

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error as mse
from scipy.stats import entropy
import warnings
import logging

from causalml.inference.meta import BaseXRegressor, BaseRRegressor, BaseSRegressor,␣
↪BaseTRegressor
from causalml.inference.nn import CEVAE
from causalml.propensity import ElasticNetPropensityModel
from causalml.metrics import *
from causalml.dataset import simulate_hidden_confounder

%matplotlib inline

warnings.filterwarnings('ignore')
logger = logging.getLogger('causalml')
logger.setLevel(logging.DEBUG)

plt.style.use('fivethirtyeight')
sns.set_palette('Paired')
plt.rcParams['figure.figsize'] = (12,8)
```

5.14.1 IHDP semi-synthetic dataset

Hill introduced a semi-synthetic dataset constructed from the Infant Health and Development Program (IHDP). This dataset is based on a randomized experiment investigating the effect of home visits by specialists on future cognitive scores. The IHDP simulation is considered the de-facto standard benchmark for neural network treatment effect estimation methods.

[2]:
```
# load all ihadp data
df = pd.DataFrame()
for i in range(1, 10):
    data = pd.read_csv('./data/ihdp_npci_' + str(i) + '.csv', header=None)
    df = pd.concat([data, df])
cols =  ["treatment", "y_factual", "y_cfactual", "mu0", "mu1"] + [i for i in␣
↪range(25)]
df.columns = cols
print(df.shape)

# replicate the data 100 times
replications = 100
df = pd.concat([df]*replications, ignore_index=True)
print(df.shape)
```

```
(6723, 30)
(672300, 30)
```

[3]:
```
# set which features are binary
binfeats = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
# set which features are continuous
contfeats = [i for i in range(25) if i not in binfeats]

# reorder features with binary first and continuous after
perm = binfeats + contfeats
```

[4]:
```
df = df.reset_index(drop=True)
df.head()
```

[4]:

	treatment	y_factual	y_cfactual	mu0	mu1	0	1	\
0	1	49.647921	34.950762	37.173291	50.383798	-0.528603	-0.343455	
1	0	16.073412	49.435313	16.087249	49.546234	-1.736945	-1.802002	
2	0	19.643007	48.598210	18.044855	49.661068	-0.807451	-0.202946	
3	0	26.368322	49.715204	24.605964	49.971196	0.390083	0.596582	
4	0	20.258893	51.147418	20.612816	49.794120	-1.045229	-0.602710	

	2	3	4	…	15	16	17	18	19	20	21	22	23	24
0	1.128554	0.161703	-0.316603	…	1	1	1	1	0	0	0	0	0	0
1	0.383828	2.244320	-0.629189	…	1	1	1	1	0	0	0	0	0	0
2	-0.360898	-0.879606	0.808706	…	1	0	1	1	0	0	0	0	0	0
3	-1.850350	-0.879606	-0.004017	…	1	0	1	1	0	0	0	0	0	0
4	0.011465	0.161703	0.683672	…	1	1	1	1	0	0	0	0	0	0

[5 rows x 30 columns]

[5]:
```
X = df[perm].values
treatment = df['treatment'].values
y = df['y_factual'].values
y_cf = df['y_cfactual'].values
tau = df.apply(lambda d: d['y_factual'] - d['y_cfactual'] if d['treatment']==1
```

(continues on next page)

(continued from previous page)

```
                else d['y_cfactual'] - d['y_factual'],
                axis=1)
mu_0 = df['mu0'].values
mu_1 = df['mu1'].values
```

[6]:
```
# seperate for train and test
itr, ite = train_test_split(np.arange(X.shape[0]), test_size=0.2, random_state=1)
X_train, treatment_train, y_train, y_cf_train, tau_train, mu_0_train, mu_1_train =␣
↪X[itr], treatment[itr], y[itr], y_cf[itr], tau[itr], mu_0[itr], mu_1[itr]
X_val, treatment_val, y_val, y_cf_val, tau_val, mu_0_val, mu_1_val = X[ite],␣
↪treatment[ite], y[ite], y_cf[ite], tau[ite], mu_0[ite], mu_1[ite]
```

5.14.2 CEVAE Model

[7]:
```
# cevae model settings
outcome_dist = "normal"
latent_dim = 20
hidden_dim = 200
num_epochs = 5
batch_size = 1000
learning_rate = 0.001
learning_rate_decay = 0.01
num_layers = 2
```

[8]:
```
cevae = CEVAE(outcome_dist=outcome_dist,
              latent_dim=latent_dim,
              hidden_dim=hidden_dim,
              num_epochs=num_epochs,
              batch_size=batch_size,
              learning_rate=learning_rate,
              learning_rate_decay=learning_rate_decay,
              num_layers=num_layers)
```

[9]:
```
# fit
losses = cevae.fit(X=torch.tensor(X_train, dtype=torch.float),
                   treatment=torch.tensor(treatment_train, dtype=torch.float),
                   y=torch.tensor(y_train, dtype=torch.float))
```

```
INFO     Training with 538 minibatches per epoch
DEBUG    step     0 loss = 1021.35
DEBUG    step     1 loss = 421.484
DEBUG    step     2 loss = 338.296
DEBUG    step     3 loss = 319.514
DEBUG    step     4 loss = 217.484
DEBUG    step     5 loss = 237.474
DEBUG    step     6 loss = 242.367
DEBUG    step     7 loss = 236.713
DEBUG    step     8 loss = 200.399
DEBUG    step     9 loss = 201.788
DEBUG    step    10 loss = 220.049
DEBUG    step    11 loss = 213.79
DEBUG    step    12 loss = 190.921
DEBUG    step    13 loss = 196.359
DEBUG    step    14 loss = 189.747
```

(continues on next page)

(continued from previous page)

```
DEBUG    step    15 loss = 167.321
DEBUG    step    16 loss = 159.207
DEBUG    step    17 loss = 154.599
DEBUG    step    18 loss = 150.961
DEBUG    step    19 loss = 149.938
DEBUG    step    20 loss = 134.768
DEBUG    step    21 loss = 140.833
DEBUG    step    22 loss = 146.769
DEBUG    step    23 loss = 132.524
DEBUG    step    24 loss = 134.194
DEBUG    step    25 loss = 130.618
DEBUG    step    26 loss = 136.787
DEBUG    step    27 loss = 126.727
DEBUG    step    28 loss = 120.942
DEBUG    step    29 loss = 118.619
DEBUG    step    30 loss = 120.946
DEBUG    step    31 loss = 110.782
DEBUG    step    32 loss = 120.907
DEBUG    step    33 loss = 106.87
DEBUG    step    34 loss = 95.3908
DEBUG    step    35 loss = 104.229
DEBUG    step    36 loss = 100.688
DEBUG    step    37 loss = 102.31
DEBUG    step    38 loss = 96.3181
DEBUG    step    39 loss = 92.0119
DEBUG    step    40 loss = 101.374
DEBUG    step    41 loss = 95.1874
DEBUG    step    42 loss = 91.693
DEBUG    step    43 loss = 83.7838
DEBUG    step    44 loss = 76.9446
DEBUG    step    45 loss = 77.8403
DEBUG    step    46 loss = 81.372
DEBUG    step    47 loss = 82.7198
DEBUG    step    48 loss = 72.8519
DEBUG    step    49 loss = 76.6569
DEBUG    step    50 loss = 75.7397
DEBUG    step    51 loss = 79.6319
DEBUG    step    52 loss = 79.2719
DEBUG    step    53 loss = 74.6354
DEBUG    step    54 loss = 68.5501
DEBUG    step    55 loss = 72.5121
DEBUG    step    56 loss = 65.3819
DEBUG    step    57 loss = 68.0494
DEBUG    step    58 loss = 69.0703
DEBUG    step    59 loss = 67.7917
DEBUG    step    60 loss = 66.9287
DEBUG    step    61 loss = 58.5794
DEBUG    step    62 loss = 59.4718
DEBUG    step    63 loss = 62.9541
DEBUG    step    64 loss = 60.0412
DEBUG    step    65 loss = 57.8926
DEBUG    step    66 loss = 57.5324
DEBUG    step    67 loss = 56.5494
DEBUG    step    68 loss = 52.2587
DEBUG    step    69 loss = 55.7073
DEBUG    step    70 loss = 54.979
DEBUG    step    71 loss = 55.4208
```

(continues on next page)

(continued from previous page)

```
DEBUG    step    72 loss = 54.7927
DEBUG    step    73 loss = 49.0343
DEBUG    step    74 loss = 53.8712
DEBUG    step    75 loss = 50.4505
DEBUG    step    76 loss = 49.2015
DEBUG    step    77 loss = 49.1161
DEBUG    step    78 loss = 51.0351
DEBUG    step    79 loss = 47.8925
DEBUG    step    80 loss = 48.4682
DEBUG    step    81 loss = 47.0941
DEBUG    step    82 loss = 44.807
DEBUG    step    83 loss = 43.6143
DEBUG    step    84 loss = 48.9903
DEBUG    step    85 loss = 46.6454
DEBUG    step    86 loss = 46.2746
DEBUG    step    87 loss = 47.5599
DEBUG    step    88 loss = 45.7764
DEBUG    step    89 loss = 42.9916
DEBUG    step    90 loss = 43.2444
DEBUG    step    91 loss = 43.616
DEBUG    step    92 loss = 41.0364
DEBUG    step    93 loss = 40.7751
DEBUG    step    94 loss = 39.693
DEBUG    step    95 loss = 41.2092
DEBUG    step    96 loss = 41.3535
DEBUG    step    97 loss = 39.0969
DEBUG    step    98 loss = 39.176
DEBUG    step    99 loss = 41.4575
DEBUG    step   100 loss = 40.5371
DEBUG    step   101 loss = 39.4805
DEBUG    step   102 loss = 37.7776
DEBUG    step   103 loss = 36.5425
DEBUG    step   104 loss = 37.3177
DEBUG    step   105 loss = 37.9773
DEBUG    step   106 loss = 36.8961
DEBUG    step   107 loss = 36.6936
DEBUG    step   108 loss = 35.1503
DEBUG    step   109 loss = 37.8622
DEBUG    step   110 loss = 36.6135
DEBUG    step   111 loss = 34.6556
DEBUG    step   112 loss = 32.9034
DEBUG    step   113 loss = 35.928
DEBUG    step   114 loss = 35.6375
DEBUG    step   115 loss = 34.8875
DEBUG    step   116 loss = 32.4369
DEBUG    step   117 loss = 35.5889
DEBUG    step   118 loss = 33.3445
DEBUG    step   119 loss = 35.3891
DEBUG    step   120 loss = 32.7132
DEBUG    step   121 loss = 32.4759
DEBUG    step   122 loss = 33.143
DEBUG    step   123 loss = 31.3498
DEBUG    step   124 loss = 31.6331
DEBUG    step   125 loss = 33.2434
DEBUG    step   126 loss = 31.1028
DEBUG    step   127 loss = 32.8674
DEBUG    step   128 loss = 32.8578
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   129 loss = 32.625
DEBUG    step   130 loss = 31.8448
DEBUG    step   131 loss = 30.8554
DEBUG    step   132 loss = 31.9763
DEBUG    step   133 loss = 29.6616
DEBUG    step   134 loss = 30.0425
DEBUG    step   135 loss = 30.836
DEBUG    step   136 loss = 31.0736
DEBUG    step   137 loss = 30.8878
DEBUG    step   138 loss = 30.43
DEBUG    step   139 loss = 30.6093
DEBUG    step   140 loss = 30.7339
DEBUG    step   141 loss = 30.0207
DEBUG    step   142 loss = 29.3626
DEBUG    step   143 loss = 29.7463
DEBUG    step   144 loss = 29.4184
DEBUG    step   145 loss = 29.2421
DEBUG    step   146 loss = 29.7529
DEBUG    step   147 loss = 29.3111
DEBUG    step   148 loss = 28.7811
DEBUG    step   149 loss = 29.3185
DEBUG    step   150 loss = 28.3709
DEBUG    step   151 loss = 30.2563
DEBUG    step   152 loss = 29.5989
DEBUG    step   153 loss = 28.8563
DEBUG    step   154 loss = 27.3948
DEBUG    step   155 loss = 28.3484
DEBUG    step   156 loss = 29.0616
DEBUG    step   157 loss = 28.8883
DEBUG    step   158 loss = 27.0463
DEBUG    step   159 loss = 27.3796
DEBUG    step   160 loss = 29.0732
DEBUG    step   161 loss = 26.8263
DEBUG    step   162 loss = 27.2883
DEBUG    step   163 loss = 28.6272
DEBUG    step   164 loss = 26.7478
DEBUG    step   165 loss = 27.6244
DEBUG    step   166 loss = 26.3508
DEBUG    step   167 loss = 26.1734
DEBUG    step   168 loss = 26.4877
DEBUG    step   169 loss = 26.9542
DEBUG    step   170 loss = 27.5395
DEBUG    step   171 loss = 26.4924
DEBUG    step   172 loss = 26.2203
DEBUG    step   173 loss = 26.039
DEBUG    step   174 loss = 25.7883
DEBUG    step   175 loss = 25.7104
DEBUG    step   176 loss = 25.9135
DEBUG    step   177 loss = 25.8419
DEBUG    step   178 loss = 26.897
DEBUG    step   179 loss = 24.8235
DEBUG    step   180 loss = 25.8669
DEBUG    step   181 loss = 26.442
DEBUG    step   182 loss = 24.7512
DEBUG    step   183 loss = 25.4444
DEBUG    step   184 loss = 25.7225
DEBUG    step   185 loss = 24.9703
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   186 loss = 25.5197
DEBUG    step   187 loss = 25.3311
DEBUG    step   188 loss = 25.0711
DEBUG    step   189 loss = 25.5542
DEBUG    step   190 loss = 25.2289
DEBUG    step   191 loss = 24.9589
DEBUG    step   192 loss = 24.5436
DEBUG    step   193 loss = 24.4451
DEBUG    step   194 loss = 23.3428
DEBUG    step   195 loss = 24.6046
DEBUG    step   196 loss = 25.1871
DEBUG    step   197 loss = 24.1005
DEBUG    step   198 loss = 24.287
DEBUG    step   199 loss = 24.4165
DEBUG    step   200 loss = 24.5855
DEBUG    step   201 loss = 23.2874
DEBUG    step   202 loss = 23.8787
DEBUG    step   203 loss = 24.5806
DEBUG    step   204 loss = 24.0906
DEBUG    step   205 loss = 25.0818
DEBUG    step   206 loss = 23.9177
DEBUG    step   207 loss = 25.0566
DEBUG    step   208 loss = 23.0722
DEBUG    step   209 loss = 23.8822
DEBUG    step   210 loss = 24.3339
DEBUG    step   211 loss = 24.7321
DEBUG    step   212 loss = 22.9672
DEBUG    step   213 loss = 23.6966
DEBUG    step   214 loss = 23.0869
DEBUG    step   215 loss = 23.5599
DEBUG    step   216 loss = 23.6307
DEBUG    step   217 loss = 23.1928
DEBUG    step   218 loss = 23.9375
DEBUG    step   219 loss = 23.65
DEBUG    step   220 loss = 22.5324
DEBUG    step   221 loss = 23.7082
DEBUG    step   222 loss = 22.854
DEBUG    step   223 loss = 21.8886
DEBUG    step   224 loss = 23.4573
DEBUG    step   225 loss = 22.4752
DEBUG    step   226 loss = 22.2281
DEBUG    step   227 loss = 22.6597
DEBUG    step   228 loss = 22.8313
DEBUG    step   229 loss = 22.8756
DEBUG    step   230 loss = 22.1289
DEBUG    step   231 loss = 22.6235
DEBUG    step   232 loss = 22.0739
DEBUG    step   233 loss = 22.7643
DEBUG    step   234 loss = 21.5396
DEBUG    step   235 loss = 21.5537
DEBUG    step   236 loss = 21.8743
DEBUG    step   237 loss = 22.6117
DEBUG    step   238 loss = 22.8206
DEBUG    step   239 loss = 22.8641
DEBUG    step   240 loss = 22.5666
```

```
DEBUG    step   241 loss = 22.3578
DEBUG    step   242 loss = 23.3638
DEBUG    step   243 loss = 22.1094
DEBUG    step   244 loss = 22.1056
DEBUG    step   245 loss = 22.1651
DEBUG    step   246 loss = 21.4072
DEBUG    step   247 loss = 21.4627
DEBUG    step   248 loss = 21.2096
DEBUG    step   249 loss = 21.3499
DEBUG    step   250 loss = 21.4386
DEBUG    step   251 loss = 21.3385
DEBUG    step   252 loss = 21.3782
DEBUG    step   253 loss = 20.7455
DEBUG    step   254 loss = 22.3244
DEBUG    step   255 loss = 21.1068
DEBUG    step   256 loss = 21.5648
DEBUG    step   257 loss = 21.5746
DEBUG    step   258 loss = 21.6169
DEBUG    step   259 loss = 21.2303
DEBUG    step   260 loss = 21.8207
DEBUG    step   261 loss = 21.2217
DEBUG    step   262 loss = 22.4259
DEBUG    step   263 loss = 21.2911
DEBUG    step   264 loss = 21.9783
DEBUG    step   265 loss = 120.585
DEBUG    step   266 loss = 22.3958
DEBUG    step   267 loss = 21.1204
DEBUG    step   268 loss = 20.3405
DEBUG    step   269 loss = 19.9695
DEBUG    step   270 loss = 21.6718
DEBUG    step   271 loss = 20.8654
DEBUG    step   272 loss = 20.4101
DEBUG    step   273 loss = 20.769
DEBUG    step   274 loss = 20.5526
DEBUG    step   275 loss = 20.026
DEBUG    step   276 loss = 20.2413
DEBUG    step   277 loss = 20.0747
DEBUG    step   278 loss = 20.6848
DEBUG    step   279 loss = 20.0956
DEBUG    step   280 loss = 20.667
DEBUG    step   281 loss = 19.8283
DEBUG    step   282 loss = 19.8651
DEBUG    step   283 loss = 19.4686
DEBUG    step   284 loss = 19.7195
DEBUG    step   285 loss = 20.1469
DEBUG    step   286 loss = 19.8956
DEBUG    step   287 loss = 20.3657
DEBUG    step   288 loss = 20.1624
DEBUG    step   289 loss = 20.8871
DEBUG    step   290 loss = 19.7327
DEBUG    step   291 loss = 19.3476
DEBUG    step   292 loss = 19.841
DEBUG    step   293 loss = 20.0052
DEBUG    step   294 loss = 19.7133
DEBUG    step   295 loss = 19.7911
DEBUG    step   296 loss = 18.6917
DEBUG    step   297 loss = 19.795
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   298 loss = 19.1175
DEBUG    step   299 loss = 20.1492
DEBUG    step   300 loss = 19.7831
DEBUG    step   301 loss = 19.7247
DEBUG    step   302 loss = 19.5755
DEBUG    step   303 loss = 19.9661
DEBUG    step   304 loss = 18.2884
DEBUG    step   305 loss = 19.6565
DEBUG    step   306 loss = 19.6213
DEBUG    step   307 loss = 19.2026
DEBUG    step   308 loss = 19.8699
DEBUG    step   309 loss = 18.7806
DEBUG    step   310 loss = 18.8876
DEBUG    step   311 loss = 19.3982
DEBUG    step   312 loss = 19.1813
DEBUG    step   313 loss = 18.9337
DEBUG    step   314 loss = 18.2574
DEBUG    step   315 loss = 19.0662
DEBUG    step   316 loss = 19.1584
DEBUG    step   317 loss = 18.1926
DEBUG    step   318 loss = 18.7658
DEBUG    step   319 loss = 18.2249
DEBUG    step   320 loss = 19.003
DEBUG    step   321 loss = 19.0593
DEBUG    step   322 loss = 18.9254
DEBUG    step   323 loss = 19.0602
DEBUG    step   324 loss = 18.5273
DEBUG    step   325 loss = 18.2321
DEBUG    step   326 loss = 18.354
DEBUG    step   327 loss = 18.2741
DEBUG    step   328 loss = 18.544
DEBUG    step   329 loss = 18.3197
DEBUG    step   330 loss = 18.8422
DEBUG    step   331 loss = 18.4199
DEBUG    step   332 loss = 17.7382
DEBUG    step   333 loss = 18.1209
DEBUG    step   334 loss = 18.4557
DEBUG    step   335 loss = 18.5937
DEBUG    step   336 loss = 17.7678
DEBUG    step   337 loss = 19.1363
DEBUG    step   338 loss = 18.0725
DEBUG    step   339 loss = 18.3309
DEBUG    step   340 loss = 17.9822
DEBUG    step   341 loss = 17.7317
DEBUG    step   342 loss = 18.1821
DEBUG    step   343 loss = 18.1704
DEBUG    step   344 loss = 18.0436
DEBUG    step   345 loss = 17.3161
DEBUG    step   346 loss = 17.1744
DEBUG    step   347 loss = 18.4531
DEBUG    step   348 loss = 17.097
DEBUG    step   349 loss = 17.2031
DEBUG    step   350 loss = 17.7855
DEBUG    step   351 loss = 17.3887
DEBUG    step   352 loss = 18.1904
DEBUG    step   353 loss = 16.9673
DEBUG    step   354 loss = 17.6665
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   355 loss = 17.9181
DEBUG    step   356 loss = 17.3892
DEBUG    step   357 loss = 18.6147
DEBUG    step   358 loss = 17.0139
DEBUG    step   359 loss = 17.4958
DEBUG    step   360 loss = 16.8143
DEBUG    step   361 loss = 16.8076
DEBUG    step   362 loss = 17.2509
DEBUG    step   363 loss = 16.6091
DEBUG    step   364 loss = 16.5105
DEBUG    step   365 loss = 16.8734
DEBUG    step   366 loss = 16.7367
DEBUG    step   367 loss = 16.3754
DEBUG    step   368 loss = 16.7072
DEBUG    step   369 loss = 16.6687
DEBUG    step   370 loss = 16.4918
DEBUG    step   371 loss = 17.4622
DEBUG    step   372 loss = 16.5902
DEBUG    step   373 loss = 17.0211
DEBUG    step   374 loss = 16.1971
DEBUG    step   375 loss = 17.1127
DEBUG    step   376 loss = 17.0151
DEBUG    step   377 loss = 16.5271
DEBUG    step   378 loss = 15.7553
DEBUG    step   379 loss = 17.5206
DEBUG    step   380 loss = 16.1141
DEBUG    step   381 loss = 16.0002
DEBUG    step   382 loss = 16.7775
DEBUG    step   383 loss = 16.0455
DEBUG    step   384 loss = 16.4851
DEBUG    step   385 loss = 15.9572
DEBUG    step   386 loss = 16.045
DEBUG    step   387 loss = 16.3194
DEBUG    step   388 loss = 16.827
DEBUG    step   389 loss = 16.818
DEBUG    step   390 loss = 16.5154
DEBUG    step   391 loss = 16.4575
DEBUG    step   392 loss = 16.3866
DEBUG    step   393 loss = 16.7649
DEBUG    step   394 loss = 16.3661
DEBUG    step   395 loss = 16.0388
DEBUG    step   396 loss = 16.3603
DEBUG    step   397 loss = 15.9295
DEBUG    step   398 loss = 16.2829
DEBUG    step   399 loss = 15.7255
DEBUG    step   400 loss = 15.9625
DEBUG    step   401 loss = 16.2877
DEBUG    step   402 loss = 15.9384
DEBUG    step   403 loss = 15.7691
DEBUG    step   404 loss = 15.3813
DEBUG    step   405 loss = 16.3497
DEBUG    step   406 loss = 15.6471
DEBUG    step   407 loss = 15.7245
DEBUG    step   408 loss = 15.5237
DEBUG    step   409 loss = 15.4977
DEBUG    step   410 loss = 15.7544
DEBUG    step   411 loss = 16.4454
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   412 loss = 15.8385
DEBUG    step   413 loss = 15.8783
DEBUG    step   414 loss = 14.5518
DEBUG    step   415 loss = 15.248
DEBUG    step   416 loss = 15.4766
DEBUG    step   417 loss = 15.1702
DEBUG    step   418 loss = 15.0027
DEBUG    step   419 loss = 14.7798
DEBUG    step   420 loss = 14.2242
DEBUG    step   421 loss = 14.7344
DEBUG    step   422 loss = 15.3192
DEBUG    step   423 loss = 14.5862
DEBUG    step   424 loss = 14.8549
DEBUG    step   425 loss = 15.1208
DEBUG    step   426 loss = 15.6343
DEBUG    step   427 loss = 14.9648
DEBUG    step   428 loss = 15.8638
DEBUG    step   429 loss = 14.7795
DEBUG    step   430 loss = 15.1229
DEBUG    step   431 loss = 14.9709
DEBUG    step   432 loss = 15.3807
DEBUG    step   433 loss = 14.2497
DEBUG    step   434 loss = 15.0741
DEBUG    step   435 loss = 13.8058
DEBUG    step   436 loss = 15.0915
DEBUG    step   437 loss = 15.2831
DEBUG    step   438 loss = 15.0772
DEBUG    step   439 loss = 15.8433
DEBUG    step   440 loss = 15.3281
DEBUG    step   441 loss = 14.7288
DEBUG    step   442 loss = 15.1505
DEBUG    step   443 loss = 15.3472
DEBUG    step   444 loss = 13.545
DEBUG    step   445 loss = 14.6441
DEBUG    step   446 loss = 14.0351
DEBUG    step   447 loss = 14.0212
DEBUG    step   448 loss = 14.1237
DEBUG    step   449 loss = 14.4073
DEBUG    step   450 loss = 14.4118
DEBUG    step   451 loss = 13.9406
DEBUG    step   452 loss = 15.0758
DEBUG    step   453 loss = 14.9103
DEBUG    step   454 loss = 14.3315
DEBUG    step   455 loss = 13.8796
DEBUG    step   456 loss = 13.9354
DEBUG    step   457 loss = 13.8283
DEBUG    step   458 loss = 14.8273
DEBUG    step   459 loss = 14.4759
DEBUG    step   460 loss = 14.5714
DEBUG    step   461 loss = 14.0121
DEBUG    step   462 loss = 14.393
DEBUG    step   463 loss = 14.4324
DEBUG    step   464 loss = 14.0807
DEBUG    step   465 loss = 14.3522
DEBUG    step   466 loss = 14.4154
DEBUG    step   467 loss = 13.1898
DEBUG    step   468 loss = 14.06
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   469 loss = 20.7401
DEBUG    step   470 loss = 14.2803
DEBUG    step   471 loss = 14.287
DEBUG    step   472 loss = 14.0215
DEBUG    step   473 loss = 13.4496
DEBUG    step   474 loss = 14.033
DEBUG    step   475 loss = 14.4732
DEBUG    step   476 loss = 13.7291
DEBUG    step   477 loss = 13.0513
DEBUG    step   478 loss = 13.6051
DEBUG    step   479 loss = 13.5316
DEBUG    step   480 loss = 13.5474
DEBUG    step   481 loss = 13.7794
DEBUG    step   482 loss = 13.8363
DEBUG    step   483 loss = 13.2939
DEBUG    step   484 loss = 13.3987
DEBUG    step   485 loss = 13.4694
DEBUG    step   486 loss = 13.0736
DEBUG    step   487 loss = 12.9663
DEBUG    step   488 loss = 13.4017
DEBUG    step   489 loss = 13.1387
DEBUG    step   490 loss = 12.8554
DEBUG    step   491 loss = 13.7535
DEBUG    step   492 loss = 13.0516
DEBUG    step   493 loss = 12.9229
DEBUG    step   494 loss = 13.0794
DEBUG    step   495 loss = 12.6742
DEBUG    step   496 loss = 12.5159
DEBUG    step   497 loss = 13.8863
DEBUG    step   498 loss = 13.275
DEBUG    step   499 loss = 13.8195
DEBUG    step   500 loss = 14.2111
DEBUG    step   501 loss = 12.8113
DEBUG    step   502 loss = 13.5611
DEBUG    step   503 loss = 13.1597
DEBUG    step   504 loss = 12.7698
DEBUG    step   505 loss = 12.655
DEBUG    step   506 loss = 13.3424
DEBUG    step   507 loss = 13.0807
DEBUG    step   508 loss = 13.4257
DEBUG    step   509 loss = 12.769
DEBUG    step   510 loss = 13.2426
DEBUG    step   511 loss = 13.7624
DEBUG    step   512 loss = 13.4707
DEBUG    step   513 loss = 12.6719
DEBUG    step   514 loss = 12.7837
DEBUG    step   515 loss = 12.3574
DEBUG    step   516 loss = 12.4319
DEBUG    step   517 loss = 12.2339
DEBUG    step   518 loss = 12.5959
DEBUG    step   519 loss = 12.9824
DEBUG    step   520 loss = 12.7877
DEBUG    step   521 loss = 13.0799
DEBUG    step   522 loss = 12.6134
DEBUG    step   523 loss = 12.0151
DEBUG    step   524 loss = 13.6236
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   525 loss = 13.0926
DEBUG    step   526 loss = 12.7921
DEBUG    step   527 loss = 12.3066
DEBUG    step   528 loss = 12.657
DEBUG    step   529 loss = 12.1989
DEBUG    step   530 loss = 12.6969
DEBUG    step   531 loss = 12.205
DEBUG    step   532 loss = 12.7905
DEBUG    step   533 loss = 12.6645
DEBUG    step   534 loss = 11.9637
DEBUG    step   535 loss = 12.3953
DEBUG    step   536 loss = 12.326
DEBUG    step   537 loss = 12.3011
DEBUG    step   538 loss = 12.3628
DEBUG    step   539 loss = 13.1567
DEBUG    step   540 loss = 12.5927
DEBUG    step   541 loss = 12.5462
DEBUG    step   542 loss = 12.2117
DEBUG    step   543 loss = 11.9447
DEBUG    step   544 loss = 12.5186
DEBUG    step   545 loss = 11.6064
DEBUG    step   546 loss = 12.1038
DEBUG    step   547 loss = 12.4013
DEBUG    step   548 loss = 12.1646
DEBUG    step   549 loss = 11.6217
DEBUG    step   550 loss = 11.7608
DEBUG    step   551 loss = 12.044
DEBUG    step   552 loss = 11.5987
DEBUG    step   553 loss = 12.2336
DEBUG    step   554 loss = 11.6134
DEBUG    step   555 loss = 12.212
DEBUG    step   556 loss = 11.7942
DEBUG    step   557 loss = 11.8134
DEBUG    step   558 loss = 11.8879
DEBUG    step   559 loss = 11.5601
DEBUG    step   560 loss = 11.8819
DEBUG    step   561 loss = 11.2771
DEBUG    step   562 loss = 12.6852
DEBUG    step   563 loss = 11.8853
DEBUG    step   564 loss = 11.8232
DEBUG    step   565 loss = 12.2208
DEBUG    step   566 loss = 11.8434
DEBUG    step   567 loss = 10.8617
DEBUG    step   568 loss = 11.9089
DEBUG    step   569 loss = 12.8768
DEBUG    step   570 loss = 11.7326
DEBUG    step   571 loss = 11.6924
DEBUG    step   572 loss = 12.071
DEBUG    step   573 loss = 11.4507
DEBUG    step   574 loss = 11.9765
DEBUG    step   575 loss = 12.3481
DEBUG    step   576 loss = 10.7076
DEBUG    step   577 loss = 11.2173
DEBUG    step   578 loss = 11.6225
DEBUG    step   579 loss = 11.7975
DEBUG    step   580 loss = 11.4295
DEBUG    step   581 loss = 11.7824
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   582 loss = 12.1286
DEBUG    step   583 loss = 10.932
DEBUG    step   584 loss = 11.9352
DEBUG    step   585 loss = 11.4005
DEBUG    step   586 loss = 11.1264
DEBUG    step   587 loss = 10.3828
DEBUG    step   588 loss = 10.6477
DEBUG    step   589 loss = 11.2266
DEBUG    step   590 loss = 11.7988
DEBUG    step   591 loss = 11.1602
DEBUG    step   592 loss = 11.2809
DEBUG    step   593 loss = 11.0131
DEBUG    step   594 loss = 11.3859
DEBUG    step   595 loss = 11.1015
DEBUG    step   596 loss = 11.4198
DEBUG    step   597 loss = 10.501
DEBUG    step   598 loss = 11.206
DEBUG    step   599 loss = 11.2975
DEBUG    step   600 loss = 10.0333
DEBUG    step   601 loss = 9.98137
DEBUG    step   602 loss = 12.6949
DEBUG    step   603 loss = 11.1914
DEBUG    step   604 loss = 10.2179
DEBUG    step   605 loss = 10.8835
DEBUG    step   606 loss = 10.3426
DEBUG    step   607 loss = 10.9994
DEBUG    step   608 loss = 10.4913
DEBUG    step   609 loss = 10.5934
DEBUG    step   610 loss = 11.2756
DEBUG    step   611 loss = 10.6515
DEBUG    step   612 loss = 10.634
DEBUG    step   613 loss = 10.6894
DEBUG    step   614 loss = 10.4173
DEBUG    step   615 loss = 10.3444
DEBUG    step   616 loss = 16.9274
DEBUG    step   617 loss = 10.6686
DEBUG    step   618 loss = 10.6302
DEBUG    step   619 loss = 11.4147
DEBUG    step   620 loss = 10.4305
DEBUG    step   621 loss = 9.93963
DEBUG    step   622 loss = 10.2567
DEBUG    step   623 loss = 10.4703
DEBUG    step   624 loss = 10.5793
DEBUG    step   625 loss = 10.7117
DEBUG    step   626 loss = 10.6469
DEBUG    step   627 loss = 10.6067
DEBUG    step   628 loss = 10.2047
DEBUG    step   629 loss = 10.7753
DEBUG    step   630 loss = 9.84085
DEBUG    step   631 loss = 9.8512
DEBUG    step   632 loss = 9.90551
DEBUG    step   633 loss = 10.2306
DEBUG    step   634 loss = 10.4
DEBUG    step   635 loss = 9.96456
DEBUG    step   636 loss = 10.0543
DEBUG    step   637 loss = 10.4722
DEBUG    step   638 loss = 10.2624
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   639 loss = 9.8927
DEBUG    step   640 loss = 9.74269
DEBUG    step   641 loss = 10.0714
DEBUG    step   642 loss = 9.4886
DEBUG    step   643 loss = 11.2356
DEBUG    step   644 loss = 10.4613
DEBUG    step   645 loss = 9.92244
DEBUG    step   646 loss = 10.5003
DEBUG    step   647 loss = 9.28321
DEBUG    step   648 loss = 10.0217
DEBUG    step   649 loss = 9.95832
DEBUG    step   650 loss = 9.89816
DEBUG    step   651 loss = 9.97542
DEBUG    step   652 loss = 9.11257
DEBUG    step   653 loss = 9.9837
DEBUG    step   654 loss = 10.1827
DEBUG    step   655 loss = 10.101
DEBUG    step   656 loss = 9.23931
DEBUG    step   657 loss = 8.75782
DEBUG    step   658 loss = 9.40421
DEBUG    step   659 loss = 9.13174
DEBUG    step   660 loss = 9.68286
DEBUG    step   661 loss = 10.4162
DEBUG    step   662 loss = 8.75674
DEBUG    step   663 loss = 10.001
DEBUG    step   664 loss = 9.40904
DEBUG    step   665 loss = 10.1505
DEBUG    step   666 loss = 10.1748
DEBUG    step   667 loss = 10.2148
DEBUG    step   668 loss = 10.2481
DEBUG    step   669 loss = 9.96609
DEBUG    step   670 loss = 9.65714
DEBUG    step   671 loss = 9.60848
DEBUG    step   672 loss = 9.84922
DEBUG    step   673 loss = 10.0371
DEBUG    step   674 loss = 9.28353
DEBUG    step   675 loss = 9.06586
DEBUG    step   676 loss = 9.44504
DEBUG    step   677 loss = 9.66529
DEBUG    step   678 loss = 9.7542
DEBUG    step   679 loss = 9.10189
DEBUG    step   680 loss = 9.36793
DEBUG    step   681 loss = 9.47525
DEBUG    step   682 loss = 9.98902
DEBUG    step   683 loss = 9.58746
DEBUG    step   684 loss = 8.77309
DEBUG    step   685 loss = 9.58264
DEBUG    step   686 loss = 9.774
DEBUG    step   687 loss = 10.1397
DEBUG    step   688 loss = 10.2031
DEBUG    step   689 loss = 8.85642
DEBUG    step   690 loss = 8.65729
DEBUG    step   691 loss = 9.30864
DEBUG    step   692 loss = 9.08819
DEBUG    step   693 loss = 8.79863
DEBUG    step   694 loss = 9.54987
DEBUG    step   695 loss = 8.96493
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   696 loss = 8.57488
DEBUG    step   697 loss = 9.37986
DEBUG    step   698 loss = 9.12005
DEBUG    step   699 loss = 9.55977
DEBUG    step   700 loss = 9.71548
DEBUG    step   701 loss = 8.66767
DEBUG    step   702 loss = 9.24891
DEBUG    step   703 loss = 8.96681
DEBUG    step   704 loss = 8.50462
DEBUG    step   705 loss = 8.97093
DEBUG    step   706 loss = 8.42754
DEBUG    step   707 loss = 8.31459
DEBUG    step   708 loss = 8.92468
DEBUG    step   709 loss = 8.62381
DEBUG    step   710 loss = 8.99014
DEBUG    step   711 loss = 9.12061
DEBUG    step   712 loss = 9.1673
DEBUG    step   713 loss = 8.71748
DEBUG    step   714 loss = 9.10944
DEBUG    step   715 loss = 8.2948
DEBUG    step   716 loss = 9.03157
DEBUG    step   717 loss = 8.86918
DEBUG    step   718 loss = 8.4948
DEBUG    step   719 loss = 8.20143
DEBUG    step   720 loss = 9.02752
DEBUG    step   721 loss = 9.07482
DEBUG    step   722 loss = 8.47549
DEBUG    step   723 loss = 8.6139
DEBUG    step   724 loss = 8.71389
DEBUG    step   725 loss = 8.71019
DEBUG    step   726 loss = 9.34067
DEBUG    step   727 loss = 8.33531
DEBUG    step   728 loss = 8.50657
DEBUG    step   729 loss = 7.92335
DEBUG    step   730 loss = 8.73418
DEBUG    step   731 loss = 7.50367
DEBUG    step   732 loss = 8.30074
DEBUG    step   733 loss = 8.10457
DEBUG    step   734 loss = 8.57933
DEBUG    step   735 loss = 8.29648
DEBUG    step   736 loss = 9.08495
DEBUG    step   737 loss = 9.19558
DEBUG    step   738 loss = 7.57463
DEBUG    step   739 loss = 8.25734
DEBUG    step   740 loss = 8.1562
DEBUG    step   741 loss = 8.13552
DEBUG    step   742 loss = 8.61787
DEBUG    step   743 loss = 7.84507
DEBUG    step   744 loss = 8.50339
DEBUG    step   745 loss = 9.99432
DEBUG    step   746 loss = 8.67392
DEBUG    step   747 loss = 7.62062
DEBUG    step   748 loss = 8.47083
DEBUG    step   749 loss = 7.59856
DEBUG    step   750 loss = 8.73944
DEBUG    step   751 loss = 7.82123
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   752 loss = 8.3673
DEBUG    step   753 loss = 8.05969
DEBUG    step   754 loss = 7.67401
DEBUG    step   755 loss = 8.23807
DEBUG    step   756 loss = 7.85361
DEBUG    step   757 loss = 8.29006
DEBUG    step   758 loss = 7.93663
DEBUG    step   759 loss = 7.14638
DEBUG    step   760 loss = 7.75548
DEBUG    step   761 loss = 7.23605
DEBUG    step   762 loss = 8.39854
DEBUG    step   763 loss = 8.36651
DEBUG    step   764 loss = 8.08217
DEBUG    step   765 loss = 8.51663
DEBUG    step   766 loss = 17.1032
DEBUG    step   767 loss = 8.11124
DEBUG    step   768 loss = 8.07747
DEBUG    step   769 loss = 7.82815
DEBUG    step   770 loss = 9.03203
DEBUG    step   771 loss = 8.53237
DEBUG    step   772 loss = 7.96279
DEBUG    step   773 loss = 8.05574
DEBUG    step   774 loss = 7.76004
DEBUG    step   775 loss = 7.35636
DEBUG    step   776 loss = 8.11715
DEBUG    step   777 loss = 8.26839
DEBUG    step   778 loss = 8.3788
DEBUG    step   779 loss = 8.4216
DEBUG    step   780 loss = 8.70143
DEBUG    step   781 loss = 7.68424
DEBUG    step   782 loss = 7.71564
DEBUG    step   783 loss = 8.99345
DEBUG    step   784 loss = 7.84072
DEBUG    step   785 loss = 7.97106
DEBUG    step   786 loss = 8.17313
DEBUG    step   787 loss = 8.43836
DEBUG    step   788 loss = 8.48604
DEBUG    step   789 loss = 7.89398
DEBUG    step   790 loss = 7.66896
DEBUG    step   791 loss = 7.93176
DEBUG    step   792 loss = 7.50743
DEBUG    step   793 loss = 7.35892
DEBUG    step   794 loss = 8.19966
DEBUG    step   795 loss = 8.04621
DEBUG    step   796 loss = 7.20783
DEBUG    step   797 loss = 7.82553
DEBUG    step   798 loss = 7.99542
DEBUG    step   799 loss = 7.39769
DEBUG    step   800 loss = 7.53701
DEBUG    step   801 loss = 7.24536
DEBUG    step   802 loss = 7.33658
DEBUG    step   803 loss = 7.342
DEBUG    step   804 loss = 7.75321
DEBUG    step   805 loss = 6.91357
DEBUG    step   806 loss = 7.52435
DEBUG    step   807 loss = 7.56103
DEBUG    step   808 loss = 7.79389
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   809 loss = 8.33436
DEBUG    step   810 loss = 7.46276
DEBUG    step   811 loss = 7.03648
DEBUG    step   812 loss = 7.09304
DEBUG    step   813 loss = 7.55697
DEBUG    step   814 loss = 7.74993
DEBUG    step   815 loss = 7.77072
DEBUG    step   816 loss = 7.57071
DEBUG    step   817 loss = 7.87914
DEBUG    step   818 loss = 7.59507
DEBUG    step   819 loss = 7.95819
DEBUG    step   820 loss = 7.26536
DEBUG    step   821 loss = 7.76702
DEBUG    step   822 loss = 6.81672
DEBUG    step   823 loss = 7.69591
DEBUG    step   824 loss = 7.49277
DEBUG    step   825 loss = 7.71589
DEBUG    step   826 loss = 7.54939
DEBUG    step   827 loss = 7.14454
DEBUG    step   828 loss = 6.54073
DEBUG    step   829 loss = 7.31939
DEBUG    step   830 loss = 8.24107
DEBUG    step   831 loss = 7.75897
DEBUG    step   832 loss = 7.0123
DEBUG    step   833 loss = 6.6658
DEBUG    step   834 loss = 7.17121
DEBUG    step   835 loss = 7.8772
DEBUG    step   836 loss = 6.91091
DEBUG    step   837 loss = 7.24767
DEBUG    step   838 loss = 7.3708
DEBUG    step   839 loss = 6.72671
DEBUG    step   840 loss = 6.91319
DEBUG    step   841 loss = 7.38147
DEBUG    step   842 loss = 6.73919
DEBUG    step   843 loss = 7.1541
DEBUG    step   844 loss = 7.09714
DEBUG    step   845 loss = 7.6505
DEBUG    step   846 loss = 6.37122
DEBUG    step   847 loss = 7.15714
DEBUG    step   848 loss = 6.78871
DEBUG    step   849 loss = 6.43234
DEBUG    step   850 loss = 6.64114
DEBUG    step   851 loss = 6.98987
DEBUG    step   852 loss = 7.51277
DEBUG    step   853 loss = 7.34095
DEBUG    step   854 loss = 7.5216
DEBUG    step   855 loss = 6.37953
DEBUG    step   856 loss = 7.08232
DEBUG    step   857 loss = 6.96187
DEBUG    step   858 loss = 6.12791
DEBUG    step   859 loss = 6.71254
DEBUG    step   860 loss = 6.15329
DEBUG    step   861 loss = 6.74574
DEBUG    step   862 loss = 7.24058
DEBUG    step   863 loss = 6.16476
DEBUG    step   864 loss = 7.61778
DEBUG    step   865 loss = 6.35608
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   866 loss = 6.53307
DEBUG    step   867 loss = 6.36949
DEBUG    step   868 loss = 6.71838
DEBUG    step   869 loss = 7.3967
DEBUG    step   870 loss = 6.65597
DEBUG    step   871 loss = 6.77125
DEBUG    step   872 loss = 6.67395
DEBUG    step   873 loss = 6.40736
DEBUG    step   874 loss = 6.35543
DEBUG    step   875 loss = 6.74703
DEBUG    step   876 loss = 6.58434
DEBUG    step   877 loss = 6.62172
DEBUG    step   878 loss = 6.65244
DEBUG    step   879 loss = 6.97937
DEBUG    step   880 loss = 6.42221
DEBUG    step   881 loss = 6.84026
DEBUG    step   882 loss = 6.72631
DEBUG    step   883 loss = 6.90398
DEBUG    step   884 loss = 6.6266
DEBUG    step   885 loss = 6.51678
DEBUG    step   886 loss = 6.65169
DEBUG    step   887 loss = 6.63095
DEBUG    step   888 loss = 6.24306
DEBUG    step   889 loss = 7.46224
DEBUG    step   890 loss = 6.84275
DEBUG    step   891 loss = 6.19764
DEBUG    step   892 loss = 7.16809
DEBUG    step   893 loss = 6.57301
DEBUG    step   894 loss = 6.72905
DEBUG    step   895 loss = 7.3967
DEBUG    step   896 loss = 6.78504
DEBUG    step   897 loss = 6.52102
DEBUG    step   898 loss = 6.07938
DEBUG    step   899 loss = 5.95618
DEBUG    step   900 loss = 6.14126
DEBUG    step   901 loss = 5.67246
DEBUG    step   902 loss = 5.59678
DEBUG    step   903 loss = 6.5394
DEBUG    step   904 loss = 6.4651
DEBUG    step   905 loss = 6.64771
DEBUG    step   906 loss = 6.44477
DEBUG    step   907 loss = 5.17112
DEBUG    step   908 loss = 5.80493
DEBUG    step   909 loss = 6.36914
DEBUG    step   910 loss = 6.68615
DEBUG    step   911 loss = 5.53628
DEBUG    step   912 loss = 6.51742
DEBUG    step   913 loss = 6.95286
DEBUG    step   914 loss = 7.2883
DEBUG    step   915 loss = 6.09494
DEBUG    step   916 loss = 6.74383
DEBUG    step   917 loss = 6.3917
DEBUG    step   918 loss = 6.25799
DEBUG    step   919 loss = 6.55483
DEBUG    step   920 loss = 6.44743
DEBUG    step   921 loss = 5.77905
DEBUG    step   922 loss = 5.98885
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   923 loss = 5.83527
DEBUG    step   924 loss = 5.93447
DEBUG    step   925 loss = 5.9199
DEBUG    step   926 loss = 6.01515
DEBUG    step   927 loss = 6.14634
DEBUG    step   928 loss = 5.77208
DEBUG    step   929 loss = 6.78369
DEBUG    step   930 loss = 6.21236
DEBUG    step   931 loss = 5.98394
DEBUG    step   932 loss = 6.51115
DEBUG    step   933 loss = 6.44652
DEBUG    step   934 loss = 5.83554
DEBUG    step   935 loss = 6.30905
DEBUG    step   936 loss = 5.93238
DEBUG    step   937 loss = 6.50758
DEBUG    step   938 loss = 5.93256
DEBUG    step   939 loss = 6.06647
DEBUG    step   940 loss = 6.03391
DEBUG    step   941 loss = 5.51953
DEBUG    step   942 loss = 6.03728
DEBUG    step   943 loss = 6.18949
DEBUG    step   944 loss = 6.10855
DEBUG    step   945 loss = 5.92263
DEBUG    step   946 loss = 6.72183
DEBUG    step   947 loss = 6.11911
DEBUG    step   948 loss = 5.84314
DEBUG    step   949 loss = 6.02928
DEBUG    step   950 loss = 5.82459
DEBUG    step   951 loss = 5.98588
DEBUG    step   952 loss = 5.75092
DEBUG    step   953 loss = 6.19303
DEBUG    step   954 loss = 5.78729
DEBUG    step   955 loss = 5.9059
DEBUG    step   956 loss = 5.31694
DEBUG    step   957 loss = 5.71936
DEBUG    step   958 loss = 6.06149
DEBUG    step   959 loss = 4.93583
DEBUG    step   960 loss = 5.8746
DEBUG    step   961 loss = 5.81154
DEBUG    step   962 loss = 6.22302
DEBUG    step   963 loss = 4.62915
DEBUG    step   964 loss = 6.26837
DEBUG    step   965 loss = 6.9227
DEBUG    step   966 loss = 5.69589
DEBUG    step   967 loss = 4.89925
DEBUG    step   968 loss = 5.95339
DEBUG    step   969 loss = 5.41167
DEBUG    step   970 loss = 5.61495
DEBUG    step   971 loss = 6.08719
DEBUG    step   972 loss = 5.70671
DEBUG    step   973 loss = 6.29176
DEBUG    step   974 loss = 5.96967
DEBUG    step   975 loss = 5.64207
DEBUG    step   976 loss = 6.11389
DEBUG    step   977 loss = 5.4677
DEBUG    step   978 loss = 5.26326
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   979 loss = 5.63665
DEBUG    step   980 loss = 5.47218
DEBUG    step   981 loss = 5.76207
DEBUG    step   982 loss = 5.25431
DEBUG    step   983 loss = 5.11318
DEBUG    step   984 loss = 5.23281
DEBUG    step   985 loss = 4.9322
DEBUG    step   986 loss = 5.19766
DEBUG    step   987 loss = 5.32089
DEBUG    step   988 loss = 5.56581
DEBUG    step   989 loss = 5.68178
DEBUG    step   990 loss = 4.37302
DEBUG    step   991 loss = 5.50948
DEBUG    step   992 loss = 5.3806
DEBUG    step   993 loss = 6.08309
DEBUG    step   994 loss = 5.74113
DEBUG    step   995 loss = 5.29156
DEBUG    step   996 loss = 6.09862
DEBUG    step   997 loss = 4.34491
DEBUG    step   998 loss = 4.74828
DEBUG    step   999 loss = 5.1352
DEBUG    step  1000 loss = 5.90098
DEBUG    step  1001 loss = 5.65187
DEBUG    step  1002 loss = 4.99241
DEBUG    step  1003 loss = 4.93651
DEBUG    step  1004 loss = 5.71697
DEBUG    step  1005 loss = 5.12284
DEBUG    step  1006 loss = 6.20878
DEBUG    step  1007 loss = 5.12986
DEBUG    step  1008 loss = 4.9672
DEBUG    step  1009 loss = 5.65217
DEBUG    step  1010 loss = 5.48825
DEBUG    step  1011 loss = 5.54487
DEBUG    step  1012 loss = 5.84657
DEBUG    step  1013 loss = 5.74514
DEBUG    step  1014 loss = 5.23785
DEBUG    step  1015 loss = 4.71362
DEBUG    step  1016 loss = 4.36813
DEBUG    step  1017 loss = 5.45256
DEBUG    step  1018 loss = 5.15537
DEBUG    step  1019 loss = 5.42831
DEBUG    step  1020 loss = 5.17
DEBUG    step  1021 loss = 4.94556
DEBUG    step  1022 loss = 5.84439
DEBUG    step  1023 loss = 5.11129
DEBUG    step  1024 loss = 4.68024
DEBUG    step  1025 loss = 4.6169
DEBUG    step  1026 loss = 4.95606
DEBUG    step  1027 loss = 4.74444
DEBUG    step  1028 loss = 4.27131
DEBUG    step  1029 loss = 4.88013
DEBUG    step  1030 loss = 4.77623
DEBUG    step  1031 loss = 5.86898
DEBUG    step  1032 loss = 5.16058
DEBUG    step  1033 loss = 4.97931
DEBUG    step  1034 loss = 5.05067
DEBUG    step  1035 loss = 5.13984
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1036 loss = 5.39295
DEBUG    step  1037 loss = 4.95942
DEBUG    step  1038 loss = 5.33035
DEBUG    step  1039 loss = 4.99434
DEBUG    step  1040 loss = 4.98677
DEBUG    step  1041 loss = 4.65488
DEBUG    step  1042 loss = 4.61823
DEBUG    step  1043 loss = 4.68538
DEBUG    step  1044 loss = 4.55243
DEBUG    step  1045 loss = 4.72619
DEBUG    step  1046 loss = 4.88855
DEBUG    step  1047 loss = 4.91348
DEBUG    step  1048 loss = 4.14682
DEBUG    step  1049 loss = 5.40462
DEBUG    step  1050 loss = 4.9091
DEBUG    step  1051 loss = 4.81781
DEBUG    step  1052 loss = 4.87586
DEBUG    step  1053 loss = 5.02846
DEBUG    step  1054 loss = 5.07139
DEBUG    step  1055 loss = 4.59791
DEBUG    step  1056 loss = 4.63243
DEBUG    step  1057 loss = 5.06353
DEBUG    step  1058 loss = 3.85668
DEBUG    step  1059 loss = 5.28508
DEBUG    step  1060 loss = 5.2355
DEBUG    step  1061 loss = 4.07526
DEBUG    step  1062 loss = 4.13481
DEBUG    step  1063 loss = 5.15536
DEBUG    step  1064 loss = 4.30691
DEBUG    step  1065 loss = 4.27459
DEBUG    step  1066 loss = 4.41401
DEBUG    step  1067 loss = 4.55242
DEBUG    step  1068 loss = 5.11923
DEBUG    step  1069 loss = 4.62136
DEBUG    step  1070 loss = 4.88281
DEBUG    step  1071 loss = 6.58954
DEBUG    step  1072 loss = 4.35964
DEBUG    step  1073 loss = 4.70629
DEBUG    step  1074 loss = 4.33995
DEBUG    step  1075 loss = 4.68683
DEBUG    step  1076 loss = 4.2739
DEBUG    step  1077 loss = 3.67668
DEBUG    step  1078 loss = 4.68557
DEBUG    step  1079 loss = 4.38688
DEBUG    step  1080 loss = 4.37331
DEBUG    step  1081 loss = 4.81933
DEBUG    step  1082 loss = 4.4695
DEBUG    step  1083 loss = 4.97354
DEBUG    step  1084 loss = 4.51781
DEBUG    step  1085 loss = 4.12469
DEBUG    step  1086 loss = 6.42285
DEBUG    step  1087 loss = 5.01891
DEBUG    step  1088 loss = 4.62022
DEBUG    step  1089 loss = 4.87794
DEBUG    step  1090 loss = 4.91586
DEBUG    step  1091 loss = 4.10107
DEBUG    step  1092 loss = 4.64939
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1093 loss = 5.02957
DEBUG    step  1094 loss = 4.41712
DEBUG    step  1095 loss = 4.42776
DEBUG    step  1096 loss = 4.28038
DEBUG    step  1097 loss = 4.93038
DEBUG    step  1098 loss = 4.39647
DEBUG    step  1099 loss = 4.14815
DEBUG    step  1100 loss = 4.47418
DEBUG    step  1101 loss = 4.53913
DEBUG    step  1102 loss = 4.18599
DEBUG    step  1103 loss = 4.42585
DEBUG    step  1104 loss = 4.52254
DEBUG    step  1105 loss = 3.73001
DEBUG    step  1106 loss = 3.80091
DEBUG    step  1107 loss = 4.65234
DEBUG    step  1108 loss = 4.22851
DEBUG    step  1109 loss = 3.80812
DEBUG    step  1110 loss = 4.85446
DEBUG    step  1111 loss = 3.86523
DEBUG    step  1112 loss = 4.18319
DEBUG    step  1113 loss = 4.21953
DEBUG    step  1114 loss = 5.04039
DEBUG    step  1115 loss = 4.80243
DEBUG    step  1116 loss = 4.30441
DEBUG    step  1117 loss = 5.39042
DEBUG    step  1118 loss = 4.25597
DEBUG    step  1119 loss = 5.07854
DEBUG    step  1120 loss = 4.12041
DEBUG    step  1121 loss = 3.47527
DEBUG    step  1122 loss = 4.13058
DEBUG    step  1123 loss = 3.55016
DEBUG    step  1124 loss = 4.84087
DEBUG    step  1125 loss = 4.22556
DEBUG    step  1126 loss = 4.61652
DEBUG    step  1127 loss = 4.38913
DEBUG    step  1128 loss = 4.1752
DEBUG    step  1129 loss = 4.35237
DEBUG    step  1130 loss = 4.11809
DEBUG    step  1131 loss = 4.52757
DEBUG    step  1132 loss = 3.64453
DEBUG    step  1133 loss = 3.92684
DEBUG    step  1134 loss = 4.419
DEBUG    step  1135 loss = 4.53101
DEBUG    step  1136 loss = 4.20247
DEBUG    step  1137 loss = 4.4274
DEBUG    step  1138 loss = 4.00318
DEBUG    step  1139 loss = 6.42864
DEBUG    step  1140 loss = 4.00687
DEBUG    step  1141 loss = 4.74919
DEBUG    step  1142 loss = 3.83376
DEBUG    step  1143 loss = 4.00634
DEBUG    step  1144 loss = 3.43185
DEBUG    step  1145 loss = 3.91977
DEBUG    step  1146 loss = 3.8136
DEBUG    step  1147 loss = 4.02812
DEBUG    step  1148 loss = 4.1181
DEBUG    step  1149 loss = 3.40067
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1150 loss = 3.87853
DEBUG    step  1151 loss = 4.30686
DEBUG    step  1152 loss = 4.22774
DEBUG    step  1153 loss = 4.38618
DEBUG    step  1154 loss = 4.56262
DEBUG    step  1155 loss = 4.45982
DEBUG    step  1156 loss = 4.59891
DEBUG    step  1157 loss = 4.44961
DEBUG    step  1158 loss = 4.0087
DEBUG    step  1159 loss = 4.88411
DEBUG    step  1160 loss = 3.81384
DEBUG    step  1161 loss = 3.60741
DEBUG    step  1162 loss = 4.1445
DEBUG    step  1163 loss = 4.40349
DEBUG    step  1164 loss = 3.83159
DEBUG    step  1165 loss = 3.76538
DEBUG    step  1166 loss = 4.21465
DEBUG    step  1167 loss = 3.94987
DEBUG    step  1168 loss = 4.0818
DEBUG    step  1169 loss = 4.06183
DEBUG    step  1170 loss = 3.47987
DEBUG    step  1171 loss = 3.67692
DEBUG    step  1172 loss = 4.20745
DEBUG    step  1173 loss = 3.84148
DEBUG    step  1174 loss = 3.49437
DEBUG    step  1175 loss = 3.67877
DEBUG    step  1176 loss = 3.95581
DEBUG    step  1177 loss = 4.26368
DEBUG    step  1178 loss = 3.89446
DEBUG    step  1179 loss = 3.66383
DEBUG    step  1180 loss = 4.65264
DEBUG    step  1181 loss = 3.91674
DEBUG    step  1182 loss = 3.80197
DEBUG    step  1183 loss = 3.24795
DEBUG    step  1184 loss = 4.25066
DEBUG    step  1185 loss = 3.59737
DEBUG    step  1186 loss = 4.23543
DEBUG    step  1187 loss = 4.40551
DEBUG    step  1188 loss = 3.06393
DEBUG    step  1189 loss = 3.78871
DEBUG    step  1190 loss = 4.47356
DEBUG    step  1191 loss = 3.01607
DEBUG    step  1192 loss = 3.5921
DEBUG    step  1193 loss = 4.14678
DEBUG    step  1194 loss = 4.06156
DEBUG    step  1195 loss = 3.63912
DEBUG    step  1196 loss = 3.80904
DEBUG    step  1197 loss = 3.94498
DEBUG    step  1198 loss = 4.46766
DEBUG    step  1199 loss = 3.94135
DEBUG    step  1200 loss = 3.16809
DEBUG    step  1201 loss = 4.44084
DEBUG    step  1202 loss = 4.10566
DEBUG    step  1203 loss = 3.80488
DEBUG    step  1204 loss = 3.19777
DEBUG    step  1205 loss = 2.95526
DEBUG    step  1206 loss = 4.49641
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1207 loss = 4.23787
DEBUG    step  1208 loss = 3.70975
DEBUG    step  1209 loss = 3.79127
DEBUG    step  1210 loss = 3.59221
DEBUG    step  1211 loss = 3.88194
DEBUG    step  1212 loss = 3.40576
DEBUG    step  1213 loss = 3.87329
DEBUG    step  1214 loss = 3.49796
DEBUG    step  1215 loss = 3.24266
DEBUG    step  1216 loss = 3.73337
DEBUG    step  1217 loss = 3.64298
DEBUG    step  1218 loss = 3.20159
DEBUG    step  1219 loss = 2.85318
DEBUG    step  1220 loss = 3.73986
DEBUG    step  1221 loss = 3.01543
DEBUG    step  1222 loss = 3.32277
DEBUG    step  1223 loss = 2.74171
DEBUG    step  1224 loss = 3.70805
DEBUG    step  1225 loss = 3.61112
DEBUG    step  1226 loss = 2.88479
DEBUG    step  1227 loss = 3.65801
DEBUG    step  1228 loss = 4.02943
DEBUG    step  1229 loss = 2.83562
DEBUG    step  1230 loss = 3.24228
DEBUG    step  1231 loss = 3.2782
DEBUG    step  1232 loss = 3.59486
DEBUG    step  1233 loss = 3.65803
DEBUG    step  1234 loss = 2.6809
DEBUG    step  1235 loss = 3.3619
DEBUG    step  1236 loss = 3.39297
DEBUG    step  1237 loss = 3.81023
DEBUG    step  1238 loss = 3.22556
DEBUG    step  1239 loss = 3.19648
DEBUG    step  1240 loss = 4.0888
DEBUG    step  1241 loss = 3.74848
DEBUG    step  1242 loss = 2.87371
DEBUG    step  1243 loss = 2.63874
DEBUG    step  1244 loss = 3.5867
DEBUG    step  1245 loss = 2.79683
DEBUG    step  1246 loss = 2.68036
DEBUG    step  1247 loss = 3.90314
DEBUG    step  1248 loss = 2.79271
DEBUG    step  1249 loss = 3.35704
DEBUG    step  1250 loss = 3.22364
DEBUG    step  1251 loss = 4.49007
DEBUG    step  1252 loss = 3.48859
DEBUG    step  1253 loss = 3.53123
DEBUG    step  1254 loss = 3.95726
DEBUG    step  1255 loss = 3.76191
DEBUG    step  1256 loss = 3.16396
DEBUG    step  1257 loss = 3.27892
DEBUG    step  1258 loss = 3.61666
DEBUG    step  1259 loss = 2.60104
DEBUG    step  1260 loss = 3.61282
DEBUG    step  1261 loss = 3.39698
DEBUG    step  1262 loss = 3.25254
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1263 loss = 3.60338
DEBUG    step  1264 loss = 3.24701
DEBUG    step  1265 loss = 2.68532
DEBUG    step  1266 loss = 3.48767
DEBUG    step  1267 loss = 3.38295
DEBUG    step  1268 loss = 3.05102
DEBUG    step  1269 loss = 2.66065
DEBUG    step  1270 loss = 4.91023
DEBUG    step  1271 loss = 3.58709
DEBUG    step  1272 loss = 2.62444
DEBUG    step  1273 loss = 3.1492
DEBUG    step  1274 loss = 2.40123
DEBUG    step  1275 loss = 3.45261
DEBUG    step  1276 loss = 3.09002
DEBUG    step  1277 loss = 3.43325
DEBUG    step  1278 loss = 3.65285
DEBUG    step  1279 loss = 5.20928
DEBUG    step  1280 loss = 3.18166
DEBUG    step  1281 loss = 2.98796
DEBUG    step  1282 loss = 3.51501
DEBUG    step  1283 loss = 3.69819
DEBUG    step  1284 loss = 2.9171
DEBUG    step  1285 loss = 3.58279
DEBUG    step  1286 loss = 3.22799
DEBUG    step  1287 loss = 2.95054
DEBUG    step  1288 loss = 2.73463
DEBUG    step  1289 loss = 2.94937
DEBUG    step  1290 loss = 3.66875
DEBUG    step  1291 loss = 5.37338
DEBUG    step  1292 loss = 3.4862
DEBUG    step  1293 loss = 3.53109
DEBUG    step  1294 loss = 3.13318
DEBUG    step  1295 loss = 3.44508
DEBUG    step  1296 loss = 3.03238
DEBUG    step  1297 loss = 3.20079
DEBUG    step  1298 loss = 2.97329
DEBUG    step  1299 loss = 2.847
DEBUG    step  1300 loss = 2.9055
DEBUG    step  1301 loss = 2.11617
DEBUG    step  1302 loss = 3.67571
DEBUG    step  1303 loss = 3.05302
DEBUG    step  1304 loss = 2.67335
DEBUG    step  1305 loss = 3.19011
DEBUG    step  1306 loss = 2.28169
DEBUG    step  1307 loss = 3.15299
DEBUG    step  1308 loss = 2.48567
DEBUG    step  1309 loss = 3.02921
DEBUG    step  1310 loss = 2.74102
DEBUG    step  1311 loss = 2.92383
DEBUG    step  1312 loss = 3.50952
DEBUG    step  1313 loss = 3.4817
DEBUG    step  1314 loss = 2.90958
DEBUG    step  1315 loss = 3.17264
DEBUG    step  1316 loss = 3.00095
DEBUG    step  1317 loss = 3.28235
DEBUG    step  1318 loss = 3.1123
DEBUG    step  1319 loss = 3.19697
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1320 loss = 3.23534
DEBUG    step  1321 loss = 2.62485
DEBUG    step  1322 loss = 2.39473
DEBUG    step  1323 loss = 2.65671
DEBUG    step  1324 loss = 2.6517
DEBUG    step  1325 loss = 2.83837
DEBUG    step  1326 loss = 2.96297
DEBUG    step  1327 loss = 3.27864
DEBUG    step  1328 loss = 2.8699
DEBUG    step  1329 loss = 2.41302
DEBUG    step  1330 loss = 2.75787
DEBUG    step  1331 loss = 2.02633
DEBUG    step  1332 loss = 2.64443
DEBUG    step  1333 loss = 3.00131
DEBUG    step  1334 loss = 2.90105
DEBUG    step  1335 loss = 2.53407
DEBUG    step  1336 loss = 2.69649
DEBUG    step  1337 loss = 3.10092
DEBUG    step  1338 loss = 2.40056
DEBUG    step  1339 loss = 2.89754
DEBUG    step  1340 loss = 3.58338
DEBUG    step  1341 loss = 2.91623
DEBUG    step  1342 loss = 3.01027
DEBUG    step  1343 loss = 2.88131
DEBUG    step  1344 loss = 2.61064
DEBUG    step  1345 loss = 3.21264
DEBUG    step  1346 loss = 3.68778
DEBUG    step  1347 loss = 3.20522
DEBUG    step  1348 loss = 3.02826
DEBUG    step  1349 loss = 2.26471
DEBUG    step  1350 loss = 1.86408
DEBUG    step  1351 loss = 2.38076
DEBUG    step  1352 loss = 3.04889
DEBUG    step  1353 loss = 2.88127
DEBUG    step  1354 loss = 2.29979
DEBUG    step  1355 loss = 2.32288
DEBUG    step  1356 loss = 2.58144
DEBUG    step  1357 loss = 3.13952
DEBUG    step  1358 loss = 2.64957
DEBUG    step  1359 loss = 2.66308
DEBUG    step  1360 loss = 2.4935
DEBUG    step  1361 loss = 2.44679
DEBUG    step  1362 loss = 2.35046
DEBUG    step  1363 loss = 2.68055
DEBUG    step  1364 loss = 2.70021
DEBUG    step  1365 loss = 2.92847
DEBUG    step  1366 loss = 2.65287
DEBUG    step  1367 loss = 3.36018
DEBUG    step  1368 loss = 3.14083
DEBUG    step  1369 loss = 3.2839
DEBUG    step  1370 loss = 2.87706
DEBUG    step  1371 loss = 2.28323
DEBUG    step  1372 loss = 2.71482
DEBUG    step  1373 loss = 3.14818
DEBUG    step  1374 loss = 1.91019
DEBUG    step  1375 loss = 3.26189
DEBUG    step  1376 loss = 2.32266
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1377 loss = 2.58565
DEBUG    step  1378 loss = 2.78616
DEBUG    step  1379 loss = 2.61887
DEBUG    step  1380 loss = 1.77536
DEBUG    step  1381 loss = 2.46593
DEBUG    step  1382 loss = 2.03291
DEBUG    step  1383 loss = 2.25107
DEBUG    step  1384 loss = 2.02538
DEBUG    step  1385 loss = 2.64462
DEBUG    step  1386 loss = 2.52711
DEBUG    step  1387 loss = 2.82251
DEBUG    step  1388 loss = 1.84549
DEBUG    step  1389 loss = 2.80308
DEBUG    step  1390 loss = 2.50824
DEBUG    step  1391 loss = 2.32621
DEBUG    step  1392 loss = 2.47522
DEBUG    step  1393 loss = 2.25115
DEBUG    step  1394 loss = 2.13335
DEBUG    step  1395 loss = 2.34713
DEBUG    step  1396 loss = 2.70859
DEBUG    step  1397 loss = 2.40365
DEBUG    step  1398 loss = 1.77973
DEBUG    step  1399 loss = 2.20398
DEBUG    step  1400 loss = 2.03752
DEBUG    step  1401 loss = 2.92017
DEBUG    step  1402 loss = 2.30887
DEBUG    step  1403 loss = 2.55533
DEBUG    step  1404 loss = 3.27081
DEBUG    step  1405 loss = 2.00323
DEBUG    step  1406 loss = 2.58616
DEBUG    step  1407 loss = 2.32837
DEBUG    step  1408 loss = 2.62355
DEBUG    step  1409 loss = 2.55319
DEBUG    step  1410 loss = 2.91456
DEBUG    step  1411 loss = 2.51186
DEBUG    step  1412 loss = 2.58023
DEBUG    step  1413 loss = 2.11317
DEBUG    step  1414 loss = 2.72763
DEBUG    step  1415 loss = 2.46438
DEBUG    step  1416 loss = 2.66077
DEBUG    step  1417 loss = 3.45261
DEBUG    step  1418 loss = 1.30968
DEBUG    step  1419 loss = 2.02033
DEBUG    step  1420 loss = 1.66572
DEBUG    step  1421 loss = 2.63344
DEBUG    step  1422 loss = 2.79048
DEBUG    step  1423 loss = 2.36907
DEBUG    step  1424 loss = 2.09989
DEBUG    step  1425 loss = 1.90149
DEBUG    step  1426 loss = 1.62709
DEBUG    step  1427 loss = 1.95195
DEBUG    step  1428 loss = 1.51384
DEBUG    step  1429 loss = 2.89507
DEBUG    step  1430 loss = 2.15085
DEBUG    step  1431 loss = 3.11155
DEBUG    step  1432 loss = 2.44331
DEBUG    step  1433 loss = 2.20407
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1434 loss = 2.08581
DEBUG    step  1435 loss = 2.42461
DEBUG    step  1436 loss = 1.99394
DEBUG    step  1437 loss = 2.04695
DEBUG    step  1438 loss = 2.82294
DEBUG    step  1439 loss = 2.33058
DEBUG    step  1440 loss = 2.10667
DEBUG    step  1441 loss = 2.3715
DEBUG    step  1442 loss = 2.13589
DEBUG    step  1443 loss = 2.0997
DEBUG    step  1444 loss = 2.40378
DEBUG    step  1445 loss = 2.69322
DEBUG    step  1446 loss = 2.3217
DEBUG    step  1447 loss = 3.06968
DEBUG    step  1448 loss = 2.19487
DEBUG    step  1449 loss = 2.62741
DEBUG    step  1450 loss = 1.93388
DEBUG    step  1451 loss = 2.23005
DEBUG    step  1452 loss = 2.05846
DEBUG    step  1453 loss = 2.37242
DEBUG    step  1454 loss = 1.70136
DEBUG    step  1455 loss = 2.47376
DEBUG    step  1456 loss = 2.62243
DEBUG    step  1457 loss = 2.22
DEBUG    step  1458 loss = 2.60625
DEBUG    step  1459 loss = 1.61209
DEBUG    step  1460 loss = 2.40373
DEBUG    step  1461 loss = 3.32855
DEBUG    step  1462 loss = 2.61678
DEBUG    step  1463 loss = 3.63504
DEBUG    step  1464 loss = 2.30637
DEBUG    step  1465 loss = 2.62554
DEBUG    step  1466 loss = 2.52577
DEBUG    step  1467 loss = 2.04929
DEBUG    step  1468 loss = 2.80166
DEBUG    step  1469 loss = 2.27281
DEBUG    step  1470 loss = 2.53645
DEBUG    step  1471 loss = 2.23338
DEBUG    step  1472 loss = 2.09672
DEBUG    step  1473 loss = 2.42459
DEBUG    step  1474 loss = 2.39755
DEBUG    step  1475 loss = 2.70626
DEBUG    step  1476 loss = 2.14803
DEBUG    step  1477 loss = 2.12395
DEBUG    step  1478 loss = 2.0754
DEBUG    step  1479 loss = 2.52702
DEBUG    step  1480 loss = 2.14769
DEBUG    step  1481 loss = 1.52042
DEBUG    step  1482 loss = 2.93158
DEBUG    step  1483 loss = 2.05924
DEBUG    step  1484 loss = 2.20132
DEBUG    step  1485 loss = 2.50342
DEBUG    step  1486 loss = 2.16502
DEBUG    step  1487 loss = 2.30084
DEBUG    step  1488 loss = 1.63317
DEBUG    step  1489 loss = 1.89554
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1490 loss = 1.68024
DEBUG    step  1491 loss = 1.84459
DEBUG    step  1492 loss = 1.63598
DEBUG    step  1493 loss = 1.38678
DEBUG    step  1494 loss = 1.71994
DEBUG    step  1495 loss = 1.81303
DEBUG    step  1496 loss = 2.59038
DEBUG    step  1497 loss = 1.6169
DEBUG    step  1498 loss = 1.90588
DEBUG    step  1499 loss = 2.14643
DEBUG    step  1500 loss = 2.01967
DEBUG    step  1501 loss = 1.91788
DEBUG    step  1502 loss = 1.75204
DEBUG    step  1503 loss = 2.31053
DEBUG    step  1504 loss = 2.12471
DEBUG    step  1505 loss = 2.22645
DEBUG    step  1506 loss = 2.04981
DEBUG    step  1507 loss = 1.88154
DEBUG    step  1508 loss = 1.58932
DEBUG    step  1509 loss = 1.74206
DEBUG    step  1510 loss = 2.37344
DEBUG    step  1511 loss = 1.17495
DEBUG    step  1512 loss = 1.82669
DEBUG    step  1513 loss = 1.3465
DEBUG    step  1514 loss = 1.10967
DEBUG    step  1515 loss = 1.68837
DEBUG    step  1516 loss = 2.49356
DEBUG    step  1517 loss = 1.35455
DEBUG    step  1518 loss = 1.27578
DEBUG    step  1519 loss = 1.65972
DEBUG    step  1520 loss = 1.66863
DEBUG    step  1521 loss = 1.89212
DEBUG    step  1522 loss = 1.54516
DEBUG    step  1523 loss = 1.393
DEBUG    step  1524 loss = 1.88502
DEBUG    step  1525 loss = 2.90167
DEBUG    step  1526 loss = 1.52293
DEBUG    step  1527 loss = 1.99959
DEBUG    step  1528 loss = 1.23991
DEBUG    step  1529 loss = 2.5743
DEBUG    step  1530 loss = 1.36191
DEBUG    step  1531 loss = 1.72816
DEBUG    step  1532 loss = 1.58642
DEBUG    step  1533 loss = 1.48767
DEBUG    step  1534 loss = 1.89661
DEBUG    step  1535 loss = 2.36828
DEBUG    step  1536 loss = 1.07969
DEBUG    step  1537 loss = 1.76135
DEBUG    step  1538 loss = 1.71266
DEBUG    step  1539 loss = 1.89935
DEBUG    step  1540 loss = 1.46401
DEBUG    step  1541 loss = 0.630489
DEBUG    step  1542 loss = 1.97178
DEBUG    step  1543 loss = 1.54882
DEBUG    step  1544 loss = 1.59709
DEBUG    step  1545 loss = 1.05165
DEBUG    step  1546 loss = 1.80869
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1547 loss = 2.13186
DEBUG    step  1548 loss = 2.48523
DEBUG    step  1549 loss = 1.36797
DEBUG    step  1550 loss = 2.11571
DEBUG    step  1551 loss = 1.90579
DEBUG    step  1552 loss = 1.53151
DEBUG    step  1553 loss = 1.99713
DEBUG    step  1554 loss = 2.22942
DEBUG    step  1555 loss = 2.03508
DEBUG    step  1556 loss = 1.91097
DEBUG    step  1557 loss = 1.64553
DEBUG    step  1558 loss = 2.31868
DEBUG    step  1559 loss = 1.88206
DEBUG    step  1560 loss = 1.84929
DEBUG    step  1561 loss = 1.74253
DEBUG    step  1562 loss = 1.55262
DEBUG    step  1563 loss = 1.24187
DEBUG    step  1564 loss = 2.21666
DEBUG    step  1565 loss = 1.54179
DEBUG    step  1566 loss = 1.18126
DEBUG    step  1567 loss = 1.60436
DEBUG    step  1568 loss = 1.62646
DEBUG    step  1569 loss = 1.13235
DEBUG    step  1570 loss = 1.73874
DEBUG    step  1571 loss = 2.98272
DEBUG    step  1572 loss = 1.97496
DEBUG    step  1573 loss = 1.40697
DEBUG    step  1574 loss = 1.75862
DEBUG    step  1575 loss = 2.24646
DEBUG    step  1576 loss = 1.71452
DEBUG    step  1577 loss = 2.13269
DEBUG    step  1578 loss = 1.87098
DEBUG    step  1579 loss = 0.903461
DEBUG    step  1580 loss = 1.25201
DEBUG    step  1581 loss = 1.8638
DEBUG    step  1582 loss = 1.8996
DEBUG    step  1583 loss = 1.43805
DEBUG    step  1584 loss = 1.15156
DEBUG    step  1585 loss = 1.41428
DEBUG    step  1586 loss = 1.13043
DEBUG    step  1587 loss = 0.838783
DEBUG    step  1588 loss = 0.782387
DEBUG    step  1589 loss = 1.6801
DEBUG    step  1590 loss = 2.16813
DEBUG    step  1591 loss = 2.3584
DEBUG    step  1592 loss = 2.03198
DEBUG    step  1593 loss = 1.6852
DEBUG    step  1594 loss = 1.6894
DEBUG    step  1595 loss = 2.05611
DEBUG    step  1596 loss = 2.04665
DEBUG    step  1597 loss = 1.44473
DEBUG    step  1598 loss = 2.35641
DEBUG    step  1599 loss = 1.77884
DEBUG    step  1600 loss = 1.29297
DEBUG    step  1601 loss = 1.44123
DEBUG    step  1602 loss = 1.03164
DEBUG    step  1603 loss = 1.97062
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1604 loss = 1.84778
DEBUG    step  1605 loss = 1.97628
DEBUG    step  1606 loss = 1.80254
DEBUG    step  1607 loss = 1.53044
DEBUG    step  1608 loss = 1.69098
DEBUG    step  1609 loss = 1.92866
DEBUG    step  1610 loss = 1.70258
DEBUG    step  1611 loss = 1.76521
DEBUG    step  1612 loss = 1.52449
DEBUG    step  1613 loss = 1.15307
DEBUG    step  1614 loss = 1.88707
DEBUG    step  1615 loss = 1.61141
DEBUG    step  1616 loss = 1.23801
DEBUG    step  1617 loss = 1.51574
DEBUG    step  1618 loss = 1.26473
DEBUG    step  1619 loss = 1.24652
DEBUG    step  1620 loss = 1.06793
DEBUG    step  1621 loss = 1.89787
DEBUG    step  1622 loss = 1.49286
DEBUG    step  1623 loss = 0.830939
DEBUG    step  1624 loss = 1.66349
DEBUG    step  1625 loss = 1.17004
DEBUG    step  1626 loss = 1.24293
DEBUG    step  1627 loss = 1.90752
DEBUG    step  1628 loss = 2.46158
DEBUG    step  1629 loss = 1.45676
DEBUG    step  1630 loss = 1.70154
DEBUG    step  1631 loss = 1.18527
DEBUG    step  1632 loss = 1.32646
DEBUG    step  1633 loss = 1.34788
DEBUG    step  1634 loss = 1.57518
DEBUG    step  1635 loss = 1.92275
DEBUG    step  1636 loss = 1.85572
DEBUG    step  1637 loss = 1.18637
DEBUG    step  1638 loss = 0.775541
DEBUG    step  1639 loss = 1.3429
DEBUG    step  1640 loss = 1.74344
DEBUG    step  1641 loss = 1.40233
DEBUG    step  1642 loss = 1.9051
DEBUG    step  1643 loss = 1.16771
DEBUG    step  1644 loss = 1.1377
DEBUG    step  1645 loss = 1.73862
DEBUG    step  1646 loss = 0.958234
DEBUG    step  1647 loss = 1.11713
DEBUG    step  1648 loss = 0.944722
DEBUG    step  1649 loss = 3.08687
DEBUG    step  1650 loss = 1.27105
DEBUG    step  1651 loss = 0.857286
DEBUG    step  1652 loss = 1.52856
DEBUG    step  1653 loss = 1.96828
DEBUG    step  1654 loss = 0.92382
DEBUG    step  1655 loss = 2.05783
DEBUG    step  1656 loss = 1.16256
DEBUG    step  1657 loss = 1.42272
DEBUG    step  1658 loss = 1.07507
DEBUG    step  1659 loss = 1.64777
DEBUG    step  1660 loss = 0.919807
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1661 loss = 0.726715
DEBUG    step  1662 loss = 1.57691
DEBUG    step  1663 loss = 1.38782
DEBUG    step  1664 loss = 1.26784
DEBUG    step  1665 loss = 1.64389
DEBUG    step  1666 loss = 0.984072
DEBUG    step  1667 loss = 1.65232
DEBUG    step  1668 loss = 1.8319
DEBUG    step  1669 loss = 1.46141
DEBUG    step  1670 loss = 0.989564
DEBUG    step  1671 loss = 1.60373
DEBUG    step  1672 loss = 1.79838
DEBUG    step  1673 loss = 1.0971
DEBUG    step  1674 loss = 1.6531
DEBUG    step  1675 loss = 0.569279
DEBUG    step  1676 loss = 1.1229
DEBUG    step  1677 loss = 2.09242
DEBUG    step  1678 loss = 1.25957
DEBUG    step  1679 loss = 1.20155
DEBUG    step  1680 loss = 0.445877
DEBUG    step  1681 loss = 1.06367
DEBUG    step  1682 loss = 1.53222
DEBUG    step  1683 loss = 1.46691
DEBUG    step  1684 loss = 1.33858
DEBUG    step  1685 loss = 1.34251
DEBUG    step  1686 loss = 1.41284
DEBUG    step  1687 loss = 1.13937
DEBUG    step  1688 loss = 2.37319
DEBUG    step  1689 loss = 0.934886
DEBUG    step  1690 loss = 0.989814
DEBUG    step  1691 loss = 1.37887
DEBUG    step  1692 loss = 1.40474
DEBUG    step  1693 loss = 1.73022
DEBUG    step  1694 loss = 0.660628
DEBUG    step  1695 loss = 1.47228
DEBUG    step  1696 loss = 1.16098
DEBUG    step  1697 loss = 1.3503
DEBUG    step  1698 loss = 1.31396
DEBUG    step  1699 loss = 2.02182
DEBUG    step  1700 loss = 0.960196
DEBUG    step  1701 loss = 1.45575
DEBUG    step  1702 loss = 1.09297
DEBUG    step  1703 loss = 1.27731
DEBUG    step  1704 loss = 1.63084
DEBUG    step  1705 loss = 1.46701
DEBUG    step  1706 loss = 1.58075
DEBUG    step  1707 loss = 2.77646
DEBUG    step  1708 loss = 1.66917
DEBUG    step  1709 loss = 1.53974
DEBUG    step  1710 loss = 0.746076
DEBUG    step  1711 loss = 0.787667
DEBUG    step  1712 loss = 1.48705
DEBUG    step  1713 loss = 1.15223
DEBUG    step  1714 loss = 0.74432
DEBUG    step  1715 loss = 1.20326
DEBUG    step  1716 loss = 1.05584
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1717 loss = 1.25595
DEBUG    step  1718 loss = 1.63639
DEBUG    step  1719 loss = 1.18738
DEBUG    step  1720 loss = 0.997565
DEBUG    step  1721 loss = 1.59334
DEBUG    step  1722 loss = 1.18497
DEBUG    step  1723 loss = 1.39869
DEBUG    step  1724 loss = 1.13685
DEBUG    step  1725 loss = 0.477479
DEBUG    step  1726 loss = 1.42541
DEBUG    step  1727 loss = 1.47176
DEBUG    step  1728 loss = 2.13344
DEBUG    step  1729 loss = 0.989916
DEBUG    step  1730 loss = 1.00084
DEBUG    step  1731 loss = 1.31844
DEBUG    step  1732 loss = 1.44907
DEBUG    step  1733 loss = 1.14411
DEBUG    step  1734 loss = 0.997098
DEBUG    step  1735 loss = 1.22144
DEBUG    step  1736 loss = 1.65521
DEBUG    step  1737 loss = 1.04064
DEBUG    step  1738 loss = 1.40232
DEBUG    step  1739 loss = 1.21052
DEBUG    step  1740 loss = 0.52208
DEBUG    step  1741 loss = 0.96464
DEBUG    step  1742 loss = 0.922535
DEBUG    step  1743 loss = 0.57069
DEBUG    step  1744 loss = 1.29497
DEBUG    step  1745 loss = 0.764636
DEBUG    step  1746 loss = 0.596204
DEBUG    step  1747 loss = 1.47739
DEBUG    step  1748 loss = 0.704551
DEBUG    step  1749 loss = 1.13051
DEBUG    step  1750 loss = 1.81735
DEBUG    step  1751 loss = 1.15569
DEBUG    step  1752 loss = 0.62525
DEBUG    step  1753 loss = -0.14409
DEBUG    step  1754 loss = 0.819491
DEBUG    step  1755 loss = 0.584971
DEBUG    step  1756 loss = 1.50396
DEBUG    step  1757 loss = 1.12784
DEBUG    step  1758 loss = 1.37416
DEBUG    step  1759 loss = 0.944302
DEBUG    step  1760 loss = 0.708327
DEBUG    step  1761 loss = 1.51183
DEBUG    step  1762 loss = 0.951956
DEBUG    step  1763 loss = 1.13992
DEBUG    step  1764 loss = -0.0584559
DEBUG    step  1765 loss = 0.941625
DEBUG    step  1766 loss = 1.46371
DEBUG    step  1767 loss = 1.36433
DEBUG    step  1768 loss = 0.560516
DEBUG    step  1769 loss = 1.35952
DEBUG    step  1770 loss = 1.01687
DEBUG    step  1771 loss = 1.21911
DEBUG    step  1772 loss = 1.8578
DEBUG    step  1773 loss = 0.774448
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1774 loss = 1.37295
DEBUG    step  1775 loss = 1.18173
DEBUG    step  1776 loss = 1.66936
DEBUG    step  1777 loss = 0.860755
DEBUG    step  1778 loss = 1.32138
DEBUG    step  1779 loss = 0.898082
DEBUG    step  1780 loss = 1.12301
DEBUG    step  1781 loss = 0.960121
DEBUG    step  1782 loss = 1.20348
DEBUG    step  1783 loss = 0.758963
DEBUG    step  1784 loss = 0.862989
DEBUG    step  1785 loss = 1.21436
DEBUG    step  1786 loss = 0.458139
DEBUG    step  1787 loss = 1.46172
DEBUG    step  1788 loss = 0.843393
DEBUG    step  1789 loss = 0.533864
DEBUG    step  1790 loss = 0.960291
DEBUG    step  1791 loss = 0.630529
DEBUG    step  1792 loss = 1.45164
DEBUG    step  1793 loss = 0.664835
DEBUG    step  1794 loss = 0.710118
DEBUG    step  1795 loss = 0.719209
DEBUG    step  1796 loss = 0.810381
DEBUG    step  1797 loss = 0.138259
DEBUG    step  1798 loss = 1.22091
DEBUG    step  1799 loss = 0.446191
DEBUG    step  1800 loss = 1.12451
DEBUG    step  1801 loss = 0.847999
DEBUG    step  1802 loss = 1.09745
DEBUG    step  1803 loss = 1.45925
DEBUG    step  1804 loss = 0.713525
DEBUG    step  1805 loss = 0.953999
DEBUG    step  1806 loss = 1.14265
DEBUG    step  1807 loss = 0.244373
DEBUG    step  1808 loss = 1.06263
DEBUG    step  1809 loss = 0.771337
DEBUG    step  1810 loss = 1.0411
DEBUG    step  1811 loss = 1.37541
DEBUG    step  1812 loss = 1.5398
DEBUG    step  1813 loss = 1.04689
DEBUG    step  1814 loss = 1.50583
DEBUG    step  1815 loss = 0.278969
DEBUG    step  1816 loss = 0.303059
DEBUG    step  1817 loss = 0.843962
DEBUG    step  1818 loss = 0.360989
DEBUG    step  1819 loss = 1.42488
DEBUG    step  1820 loss = 0.334529
DEBUG    step  1821 loss = 1.15429
DEBUG    step  1822 loss = 0.942839
DEBUG    step  1823 loss = -0.0623802
DEBUG    step  1824 loss = 1.2242
DEBUG    step  1825 loss = 0.110633
DEBUG    step  1826 loss = 1.04671
DEBUG    step  1827 loss = 0.814721
DEBUG    step  1828 loss = 0.981389
DEBUG    step  1829 loss = 0.374465
DEBUG    step  1830 loss = 0.682603
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1831 loss = 0.888044
DEBUG    step  1832 loss = 1.00653
DEBUG    step  1833 loss = -0.192628
DEBUG    step  1834 loss = 1.33105
DEBUG    step  1835 loss = -0.292317
DEBUG    step  1836 loss = 1.40156
DEBUG    step  1837 loss = 0.548849
DEBUG    step  1838 loss = 0.733393
DEBUG    step  1839 loss = 0.737875
DEBUG    step  1840 loss = 0.953065
DEBUG    step  1841 loss = 1.35565
DEBUG    step  1842 loss = 0.334132
DEBUG    step  1843 loss = 0.527886
DEBUG    step  1844 loss = 0.728576
DEBUG    step  1845 loss = 0.971659
DEBUG    step  1846 loss = 1.0362
DEBUG    step  1847 loss = 1.1995
DEBUG    step  1848 loss = 0.74542
DEBUG    step  1849 loss = 0.822038
DEBUG    step  1850 loss = 0.14102
DEBUG    step  1851 loss = 0.351881
DEBUG    step  1852 loss = 0.718691
DEBUG    step  1853 loss = 0.454031
DEBUG    step  1854 loss = 1.34327
DEBUG    step  1855 loss = 1.12586
DEBUG    step  1856 loss = 0.794541
DEBUG    step  1857 loss = 0.881259
DEBUG    step  1858 loss = 0.402362
DEBUG    step  1859 loss = 0.490797
DEBUG    step  1860 loss = 0.12956
DEBUG    step  1861 loss = 1.00601
DEBUG    step  1862 loss = 0.0126683
DEBUG    step  1863 loss = 0.367983
DEBUG    step  1864 loss = 0.519085
DEBUG    step  1865 loss = 1.5708
DEBUG    step  1866 loss = 1.47664
DEBUG    step  1867 loss = 0.891001
DEBUG    step  1868 loss = 1.33164
DEBUG    step  1869 loss = 1.43242
DEBUG    step  1870 loss = 1.57703
DEBUG    step  1871 loss = 0.409759
DEBUG    step  1872 loss = 0.481442
DEBUG    step  1873 loss = 0.433702
DEBUG    step  1874 loss = 0.102985
DEBUG    step  1875 loss = 1.07597
DEBUG    step  1876 loss = 0.628031
DEBUG    step  1877 loss = -0.0152627
DEBUG    step  1878 loss = 0.482545
DEBUG    step  1879 loss = 1.55648
DEBUG    step  1880 loss = 0.844998
DEBUG    step  1881 loss = 0.42592
DEBUG    step  1882 loss = -0.0152035
DEBUG    step  1883 loss = -0.0997669
DEBUG    step  1884 loss = 1.01354
DEBUG    step  1885 loss = 0.490207
DEBUG    step  1886 loss = 0.736687
DEBUG    step  1887 loss = 0.433603
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1888 loss = 1.07525
DEBUG    step  1889 loss = 0.678383
DEBUG    step  1890 loss = 0.980835
DEBUG    step  1891 loss = 0.470526
DEBUG    step  1892 loss = 0.591348
DEBUG    step  1893 loss = 0.496179
DEBUG    step  1894 loss = 0.164359
DEBUG    step  1895 loss = 0.505431
DEBUG    step  1896 loss = 0.848054
DEBUG    step  1897 loss = 1.22015
DEBUG    step  1898 loss = 0.21223
DEBUG    step  1899 loss = 0.804585
DEBUG    step  1900 loss = 0.337482
DEBUG    step  1901 loss = 0.380753
DEBUG    step  1902 loss = 1.09557
DEBUG    step  1903 loss = 0.452767
DEBUG    step  1904 loss = 0.505589
DEBUG    step  1905 loss = 0.533463
DEBUG    step  1906 loss = 0.732611
DEBUG    step  1907 loss = 0.457369
DEBUG    step  1908 loss = 0.397615
DEBUG    step  1909 loss = 0.304795
DEBUG    step  1910 loss = 0.832857
DEBUG    step  1911 loss = 0.776005
DEBUG    step  1912 loss = 0.0557357
DEBUG    step  1913 loss = 1.06473
DEBUG    step  1914 loss = 0.621938
DEBUG    step  1915 loss = 3.8174
DEBUG    step  1916 loss = 0.834741
DEBUG    step  1917 loss = 0.432647
DEBUG    step  1918 loss = 1.0107
DEBUG    step  1919 loss = 0.887171
DEBUG    step  1920 loss = 0.214395
DEBUG    step  1921 loss = 0.27015
DEBUG    step  1922 loss = 0.723923
DEBUG    step  1923 loss = 0.0225524
DEBUG    step  1924 loss = 0.311126
DEBUG    step  1925 loss = 0.163129
DEBUG    step  1926 loss = 1.0852
DEBUG    step  1927 loss = 0.845341
DEBUG    step  1928 loss = 0.067302
DEBUG    step  1929 loss = 1.81058
DEBUG    step  1930 loss = 0.711902
DEBUG    step  1931 loss = 0.544337
DEBUG    step  1932 loss = 0.729942
DEBUG    step  1933 loss = 0.281568
DEBUG    step  1934 loss = 0.746916
DEBUG    step  1935 loss = 0.731851
DEBUG    step  1936 loss = 0.861581
DEBUG    step  1937 loss = 0.587285
DEBUG    step  1938 loss = 0.375893
DEBUG    step  1939 loss = 0.52338
DEBUG    step  1940 loss = 0.0507239
DEBUG    step  1941 loss = 0.544204
DEBUG    step  1942 loss = 0.139653
DEBUG    step  1943 loss = 0.603852
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  1944 loss = 0.591492
DEBUG    step  1945 loss = 0.211932
DEBUG    step  1946 loss = 0.632158
DEBUG    step  1947 loss = 0.613739
DEBUG    step  1948 loss = 1.12637
DEBUG    step  1949 loss = 0.655486
DEBUG    step  1950 loss = 0.687108
DEBUG    step  1951 loss = 0.224532
DEBUG    step  1952 loss = 0.675569
DEBUG    step  1953 loss = 1.16836
DEBUG    step  1954 loss = 0.575642
DEBUG    step  1955 loss = 0.314398
DEBUG    step  1956 loss = 0.949717
DEBUG    step  1957 loss = 1.06026
DEBUG    step  1958 loss = 0.894075
DEBUG    step  1959 loss = 0.268737
DEBUG    step  1960 loss = -0.0684191
DEBUG    step  1961 loss = 0.301358
DEBUG    step  1962 loss = 0.670349
DEBUG    step  1963 loss = 0.631736
DEBUG    step  1964 loss = 1.17734
DEBUG    step  1965 loss = -0.0977912
DEBUG    step  1966 loss = 0.872278
DEBUG    step  1967 loss = 0.0835433
DEBUG    step  1968 loss = -0.0705985
DEBUG    step  1969 loss = 0.193565
DEBUG    step  1970 loss = 0.817641
DEBUG    step  1971 loss = 1.54214
DEBUG    step  1972 loss = -0.0112863
DEBUG    step  1973 loss = 0.170732
DEBUG    step  1974 loss = 0.437139
DEBUG    step  1975 loss = -0.0416076
DEBUG    step  1976 loss = 0.201051
DEBUG    step  1977 loss = 0.663106
DEBUG    step  1978 loss = 0.647153
DEBUG    step  1979 loss = 0.138818
DEBUG    step  1980 loss = 0.0719861
DEBUG    step  1981 loss = 1.12457
DEBUG    step  1982 loss = 0.123392
DEBUG    step  1983 loss = 0.35576
DEBUG    step  1984 loss = 0.187577
DEBUG    step  1985 loss = 0.158135
DEBUG    step  1986 loss = 0.172388
DEBUG    step  1987 loss = 0.864039
DEBUG    step  1988 loss = 0.522948
DEBUG    step  1989 loss = 0.218993
DEBUG    step  1990 loss = 0.958601
DEBUG    step  1991 loss = 0.0281422
DEBUG    step  1992 loss = 0.15538
DEBUG    step  1993 loss = 0.298106
DEBUG    step  1994 loss = 0.192198
DEBUG    step  1995 loss = -0.437914
DEBUG    step  1996 loss = 0.17182
DEBUG    step  1997 loss = 0.625345
DEBUG    step  1998 loss = 0.443585
DEBUG    step  1999 loss = -0.0372677
DEBUG    step  2000 loss = 0.0965499
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2001 loss = 0.684757
DEBUG    step  2002 loss = 0.0434506
DEBUG    step  2003 loss = 0.179006
DEBUG    step  2004 loss = 0.585443
DEBUG    step  2005 loss = 0.75187
DEBUG    step  2006 loss = -0.19287
DEBUG    step  2007 loss = 0.753149
DEBUG    step  2008 loss = 0.524784
DEBUG    step  2009 loss = 0.500014
DEBUG    step  2010 loss = 0.68905
DEBUG    step  2011 loss = 0.508104
DEBUG    step  2012 loss = 1.12944
DEBUG    step  2013 loss = 0.636447
DEBUG    step  2014 loss = 1.07191
DEBUG    step  2015 loss = 0.620359
DEBUG    step  2016 loss = -0.0672604
DEBUG    step  2017 loss = 0.12611
DEBUG    step  2018 loss = -0.160067
DEBUG    step  2019 loss = 0.560006
DEBUG    step  2020 loss = -0.0938559
DEBUG    step  2021 loss = 0.2633
DEBUG    step  2022 loss = -0.24172
DEBUG    step  2023 loss = 0.23306
DEBUG    step  2024 loss = -0.119578
DEBUG    step  2025 loss = 0.304582
DEBUG    step  2026 loss = 0.222591
DEBUG    step  2027 loss = 0.47586
DEBUG    step  2028 loss = 0.504828
DEBUG    step  2029 loss = 0.422783
DEBUG    step  2030 loss = 0.346542
DEBUG    step  2031 loss = 0.22548
DEBUG    step  2032 loss = 0.0345138
DEBUG    step  2033 loss = 0.727085
DEBUG    step  2034 loss = 0.438053
DEBUG    step  2035 loss = -0.163181
DEBUG    step  2036 loss = 0.816675
DEBUG    step  2037 loss = 0.0115353
DEBUG    step  2038 loss = 0.768062
DEBUG    step  2039 loss = 0.24584
DEBUG    step  2040 loss = 0.290391
DEBUG    step  2041 loss = 0.955838
DEBUG    step  2042 loss = 0.185171
DEBUG    step  2043 loss = -0.360956
DEBUG    step  2044 loss = 0.12458
DEBUG    step  2045 loss = 0.00191054
DEBUG    step  2046 loss = 0.0451765
DEBUG    step  2047 loss = 0.215519
DEBUG    step  2048 loss = 0.159755
DEBUG    step  2049 loss = 0.917712
DEBUG    step  2050 loss = -0.26462
DEBUG    step  2051 loss = 0.310773
DEBUG    step  2052 loss = -0.0363671
DEBUG    step  2053 loss = 0.0293219
DEBUG    step  2054 loss = -0.00587582
DEBUG    step  2055 loss = 0.471752
DEBUG    step  2056 loss = 0.238597
DEBUG    step  2057 loss = 0.0422264
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2058 loss = -0.543846
DEBUG    step  2059 loss = 0.777388
DEBUG    step  2060 loss = -0.693749
DEBUG    step  2061 loss = 0.0994059
DEBUG    step  2062 loss = -0.286047
DEBUG    step  2063 loss = 0.766898
DEBUG    step  2064 loss = -0.142116
DEBUG    step  2065 loss = 0.883171
DEBUG    step  2066 loss = 0.180947
DEBUG    step  2067 loss = 0.210857
DEBUG    step  2068 loss = 0.118777
DEBUG    step  2069 loss = -0.141074
DEBUG    step  2070 loss = 0.363284
DEBUG    step  2071 loss = 0.39178
DEBUG    step  2072 loss = 0.305299
DEBUG    step  2073 loss = 0.545026
DEBUG    step  2074 loss = -0.226126
DEBUG    step  2075 loss = 0.169667
DEBUG    step  2076 loss = -0.336501
DEBUG    step  2077 loss = 0.965252
DEBUG    step  2078 loss = -0.170774
DEBUG    step  2079 loss = 0.0928747
DEBUG    step  2080 loss = 0.134985
DEBUG    step  2081 loss = 0.0768925
DEBUG    step  2082 loss = 0.207024
DEBUG    step  2083 loss = -0.157205
DEBUG    step  2084 loss = -0.13322
DEBUG    step  2085 loss = 0.262412
DEBUG    step  2086 loss = 0.327786
DEBUG    step  2087 loss = -0.0993449
DEBUG    step  2088 loss = 0.244769
DEBUG    step  2089 loss = -0.0589051
DEBUG    step  2090 loss = 0.332496
DEBUG    step  2091 loss = 0.925634
DEBUG    step  2092 loss = -0.257988
DEBUG    step  2093 loss = 0.518207
DEBUG    step  2094 loss = 0.286856
DEBUG    step  2095 loss = -0.300405
DEBUG    step  2096 loss = -0.0130847
DEBUG    step  2097 loss = 0.519027
DEBUG    step  2098 loss = 0.318041
DEBUG    step  2099 loss = -0.133822
DEBUG    step  2100 loss = -0.076749
DEBUG    step  2101 loss = 0.0152595
DEBUG    step  2102 loss = 0.678585
DEBUG    step  2103 loss = -0.164601
DEBUG    step  2104 loss = 0.384856
DEBUG    step  2105 loss = 0.0680997
DEBUG    step  2106 loss = -0.0351076
DEBUG    step  2107 loss = 0.231791
DEBUG    step  2108 loss = -0.117496
DEBUG    step  2109 loss = -0.0222189
DEBUG    step  2110 loss = -0.0573999
DEBUG    step  2111 loss = 0.524485
DEBUG    step  2112 loss = 0.0913248
DEBUG    step  2113 loss = 0.280226
DEBUG    step  2114 loss = 0.318695
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2115 loss = 0.039408
DEBUG    step  2116 loss = 0.0231956
DEBUG    step  2117 loss = -0.144188
DEBUG    step  2118 loss = -0.249522
DEBUG    step  2119 loss = 0.182491
DEBUG    step  2120 loss = -0.137275
DEBUG    step  2121 loss = -0.116535
DEBUG    step  2122 loss = -0.502473
DEBUG    step  2123 loss = 0.106871
DEBUG    step  2124 loss = 0.219624
DEBUG    step  2125 loss = 0.236981
DEBUG    step  2126 loss = 0.308991
DEBUG    step  2127 loss = 0.361933
DEBUG    step  2128 loss = -0.0891354
DEBUG    step  2129 loss = 0.375717
DEBUG    step  2130 loss = 0.458
DEBUG    step  2131 loss = 0.804599
DEBUG    step  2132 loss = -0.850078
DEBUG    step  2133 loss = -0.565978
DEBUG    step  2134 loss = 0.395504
DEBUG    step  2135 loss = 0.0360778
DEBUG    step  2136 loss = 0.262763
DEBUG    step  2137 loss = 0.173679
DEBUG    step  2138 loss = 0.245434
DEBUG    step  2139 loss = -0.325045
DEBUG    step  2140 loss = 0.197687
DEBUG    step  2141 loss = 0.10554
DEBUG    step  2142 loss = 0.629076
DEBUG    step  2143 loss = -0.444622
DEBUG    step  2144 loss = 0.29245
DEBUG    step  2145 loss = -0.169153
DEBUG    step  2146 loss = -0.122091
DEBUG    step  2147 loss = -0.482058
DEBUG    step  2148 loss = -0.145807
DEBUG    step  2149 loss = -0.321955
DEBUG    step  2150 loss = -0.204977
DEBUG    step  2151 loss = 0.260222
DEBUG    step  2152 loss = -0.0221428
DEBUG    step  2153 loss = -0.299182
DEBUG    step  2154 loss = 0.492136
DEBUG    step  2155 loss = -0.512058
DEBUG    step  2156 loss = -0.701374
DEBUG    step  2157 loss = 0.616286
DEBUG    step  2158 loss = -0.580705
DEBUG    step  2159 loss = 0.543072
DEBUG    step  2160 loss = -0.271091
DEBUG    step  2161 loss = -0.152006
DEBUG    step  2162 loss = -0.0906625
DEBUG    step  2163 loss = -0.341321
DEBUG    step  2164 loss = -0.0973744
DEBUG    step  2165 loss = 0.335691
DEBUG    step  2166 loss = -0.513224
DEBUG    step  2167 loss = 0.441127
DEBUG    step  2168 loss = -0.195149
DEBUG    step  2169 loss = -0.155654
DEBUG    step  2170 loss = 0.146065
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2171 loss = -0.157879
DEBUG    step  2172 loss = 0.427397
DEBUG    step  2173 loss = -0.264271
DEBUG    step  2174 loss = 0.255104
DEBUG    step  2175 loss = 0.143516
DEBUG    step  2176 loss = -0.144723
DEBUG    step  2177 loss = 0.362921
DEBUG    step  2178 loss = 0.085199
DEBUG    step  2179 loss = 0.166598
DEBUG    step  2180 loss = -0.529532
DEBUG    step  2181 loss = -0.318048
DEBUG    step  2182 loss = -0.0852365
DEBUG    step  2183 loss = -0.226952
DEBUG    step  2184 loss = 0.372169
DEBUG    step  2185 loss = 0.46677
DEBUG    step  2186 loss = -0.0550372
DEBUG    step  2187 loss = 0.123473
DEBUG    step  2188 loss = -0.709439
DEBUG    step  2189 loss = 0.627293
DEBUG    step  2190 loss = -0.932047
DEBUG    step  2191 loss = -0.0653693
DEBUG    step  2192 loss = 0.694153
DEBUG    step  2193 loss = -0.0535071
DEBUG    step  2194 loss = -0.691768
DEBUG    step  2195 loss = -0.0777673
DEBUG    step  2196 loss = -0.0291022
DEBUG    step  2197 loss = 0.0775634
DEBUG    step  2198 loss = -0.00225392
DEBUG    step  2199 loss = 0.467416
DEBUG    step  2200 loss = -0.0729818
DEBUG    step  2201 loss = -0.174586
DEBUG    step  2202 loss = -0.0735762
DEBUG    step  2203 loss = -0.291103
DEBUG    step  2204 loss = 0.206642
DEBUG    step  2205 loss = -0.35946
DEBUG    step  2206 loss = 0.0623758
DEBUG    step  2207 loss = -0.0335207
DEBUG    step  2208 loss = -0.322341
DEBUG    step  2209 loss = -0.164268
DEBUG    step  2210 loss = -0.298333
DEBUG    step  2211 loss = -0.542928
DEBUG    step  2212 loss = 0.818519
DEBUG    step  2213 loss = -0.175861
DEBUG    step  2214 loss = -1.18826
DEBUG    step  2215 loss = 0.020086
DEBUG    step  2216 loss = -1.07731
DEBUG    step  2217 loss = 0.861459
DEBUG    step  2218 loss = -0.30791
DEBUG    step  2219 loss = 12.8663
DEBUG    step  2220 loss = 0.110738
DEBUG    step  2221 loss = 0.415476
DEBUG    step  2222 loss = -0.0830224
DEBUG    step  2223 loss = 0.026601
DEBUG    step  2224 loss = -0.484626
DEBUG    step  2225 loss = -0.643493
DEBUG    step  2226 loss = -0.531596
DEBUG    step  2227 loss = -0.159798
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2228 loss = 0.444723
DEBUG    step  2229 loss = -0.209576
DEBUG    step  2230 loss = -0.117957
DEBUG    step  2231 loss = 0.26718
DEBUG    step  2232 loss = -0.623983
DEBUG    step  2233 loss = -0.134441
DEBUG    step  2234 loss = -1.03047
DEBUG    step  2235 loss = 0.10526
DEBUG    step  2236 loss = -0.168391
DEBUG    step  2237 loss = -0.325326
DEBUG    step  2238 loss = -0.636917
DEBUG    step  2239 loss = -1.01447
DEBUG    step  2240 loss = -0.137275
DEBUG    step  2241 loss = -0.0928798
DEBUG    step  2242 loss = 0.521724
DEBUG    step  2243 loss = -0.726267
DEBUG    step  2244 loss = -0.151048
DEBUG    step  2245 loss = -0.0553814
DEBUG    step  2246 loss = -0.0806889
DEBUG    step  2247 loss = -0.265405
DEBUG    step  2248 loss = -0.605389
DEBUG    step  2249 loss = 0.609598
DEBUG    step  2250 loss = 0.201578
DEBUG    step  2251 loss = -0.301686
DEBUG    step  2252 loss = 0.254437
DEBUG    step  2253 loss = 0.53236
DEBUG    step  2254 loss = -0.405195
DEBUG    step  2255 loss = -0.0701203
DEBUG    step  2256 loss = -0.2183
DEBUG    step  2257 loss = -0.766243
DEBUG    step  2258 loss = -0.732259
DEBUG    step  2259 loss = -0.142207
DEBUG    step  2260 loss = -0.15166
DEBUG    step  2261 loss = -0.700015
DEBUG    step  2262 loss = 0.0802323
DEBUG    step  2263 loss = 0.313499
DEBUG    step  2264 loss = 0.283268
DEBUG    step  2265 loss = -0.458733
DEBUG    step  2266 loss = 0.169434
DEBUG    step  2267 loss = 0.0517936
DEBUG    step  2268 loss = -0.303608
DEBUG    step  2269 loss = 0.273257
DEBUG    step  2270 loss = -0.392904
DEBUG    step  2271 loss = 0.44848
DEBUG    step  2272 loss = -0.703877
DEBUG    step  2273 loss = -1.01002
DEBUG    step  2274 loss = 0.359133
DEBUG    step  2275 loss = 0.212775
DEBUG    step  2276 loss = -0.519192
DEBUG    step  2277 loss = -0.2437
DEBUG    step  2278 loss = -0.667431
DEBUG    step  2279 loss = -0.996026
DEBUG    step  2280 loss = 0.273185
DEBUG    step  2281 loss = -0.00770547
DEBUG    step  2282 loss = -0.162126
DEBUG    step  2283 loss = 0.175816
DEBUG    step  2284 loss = 0.0773304
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2285 loss = -0.512412
DEBUG    step  2286 loss = -0.607146
DEBUG    step  2287 loss = 0.182539
DEBUG    step  2288 loss = -0.694855
DEBUG    step  2289 loss = 0.335107
DEBUG    step  2290 loss = 0.351011
DEBUG    step  2291 loss = -0.367074
DEBUG    step  2292 loss = 0.961813
DEBUG    step  2293 loss = 0.319814
DEBUG    step  2294 loss = -0.0375465
DEBUG    step  2295 loss = -0.685502
DEBUG    step  2296 loss = 0.702536
DEBUG    step  2297 loss = -0.0365256
DEBUG    step  2298 loss = 0.297325
DEBUG    step  2299 loss = -0.161133
DEBUG    step  2300 loss = -0.0621092
DEBUG    step  2301 loss = -0.524049
DEBUG    step  2302 loss = -0.428477
DEBUG    step  2303 loss = -0.481184
DEBUG    step  2304 loss = -0.582241
DEBUG    step  2305 loss = -0.22409
DEBUG    step  2306 loss = -0.0466428
DEBUG    step  2307 loss = -0.807201
DEBUG    step  2308 loss = -0.418819
DEBUG    step  2309 loss = -0.11762
DEBUG    step  2310 loss = -0.00959172
DEBUG    step  2311 loss = -0.00444585
DEBUG    step  2312 loss = 0.043913
DEBUG    step  2313 loss = 0.571166
DEBUG    step  2314 loss = -0.537292
DEBUG    step  2315 loss = 0.270969
DEBUG    step  2316 loss = -0.212546
DEBUG    step  2317 loss = 0.112569
DEBUG    step  2318 loss = -0.455186
DEBUG    step  2319 loss = -0.424695
DEBUG    step  2320 loss = -0.464438
DEBUG    step  2321 loss = -0.473156
DEBUG    step  2322 loss = -0.105536
DEBUG    step  2323 loss = -0.198469
DEBUG    step  2324 loss = 0.422803
DEBUG    step  2325 loss = 0.887627
DEBUG    step  2326 loss = -0.685745
DEBUG    step  2327 loss = -0.656979
DEBUG    step  2328 loss = -1.1468
DEBUG    step  2329 loss = -0.416101
DEBUG    step  2330 loss = -0.0506251
DEBUG    step  2331 loss = 0.38371
DEBUG    step  2332 loss = -0.410896
DEBUG    step  2333 loss = -0.490316
DEBUG    step  2334 loss = -0.148082
DEBUG    step  2335 loss = -1.2066
DEBUG    step  2336 loss = -0.480291
DEBUG    step  2337 loss = -0.564195
DEBUG    step  2338 loss = -0.051699
DEBUG    step  2339 loss = 0.554887
DEBUG    step  2340 loss = 0.464537
DEBUG    step  2341 loss = -0.586118
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2342 loss = -0.224842
DEBUG    step  2343 loss = 0.140776
DEBUG    step  2344 loss = 0.0989285
DEBUG    step  2345 loss = -0.140234
DEBUG    step  2346 loss = -0.220834
DEBUG    step  2347 loss = 0.358295
DEBUG    step  2348 loss = -0.935413
DEBUG    step  2349 loss = -0.797103
DEBUG    step  2350 loss = -0.370552
DEBUG    step  2351 loss = -0.255635
DEBUG    step  2352 loss = 0.0331677
DEBUG    step  2353 loss = -0.0654061
DEBUG    step  2354 loss = -0.792516
DEBUG    step  2355 loss = 1.00517
DEBUG    step  2356 loss = -0.0650678
DEBUG    step  2357 loss = 0.100208
DEBUG    step  2358 loss = 0.315501
DEBUG    step  2359 loss = -0.196945
DEBUG    step  2360 loss = -0.706372
DEBUG    step  2361 loss = 0.134541
DEBUG    step  2362 loss = -0.114532
DEBUG    step  2363 loss = -0.661938
DEBUG    step  2364 loss = -0.826783
DEBUG    step  2365 loss = 0.561703
DEBUG    step  2366 loss = -0.380749
DEBUG    step  2367 loss = -0.599982
DEBUG    step  2368 loss = -0.552984
DEBUG    step  2369 loss = -0.809876
DEBUG    step  2370 loss = -0.41806
DEBUG    step  2371 loss = -0.293652
DEBUG    step  2372 loss = 0.019794
DEBUG    step  2373 loss = 0.366571
DEBUG    step  2374 loss = -0.330331
DEBUG    step  2375 loss = -0.108959
DEBUG    step  2376 loss = 0.0823981
DEBUG    step  2377 loss = -0.122074
DEBUG    step  2378 loss = 0.104684
DEBUG    step  2379 loss = -0.245806
DEBUG    step  2380 loss = -0.458836
DEBUG    step  2381 loss = -0.728625
DEBUG    step  2382 loss = 0.366162
DEBUG    step  2383 loss = -0.402356
DEBUG    step  2384 loss = -0.915713
DEBUG    step  2385 loss = 0.25255
DEBUG    step  2386 loss = -0.596414
DEBUG    step  2387 loss = 0.191845
DEBUG    step  2388 loss = 0.173331
DEBUG    step  2389 loss = -0.235943
DEBUG    step  2390 loss = -0.578616
DEBUG    step  2391 loss = -0.387393
DEBUG    step  2392 loss = -0.509603
DEBUG    step  2393 loss = -0.0789079
DEBUG    step  2394 loss = -0.146879
DEBUG    step  2395 loss = -0.162622
DEBUG    step  2396 loss = -0.580962
DEBUG    step  2397 loss = -0.704767
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2398 loss = -0.471613
DEBUG    step  2399 loss = -0.18096
DEBUG    step  2400 loss = -0.162947
DEBUG    step  2401 loss = 0.0571842
DEBUG    step  2402 loss = -0.707115
DEBUG    step  2403 loss = -0.812926
DEBUG    step  2404 loss = 0.680889
DEBUG    step  2405 loss = 0.158955
DEBUG    step  2406 loss = -0.636955
DEBUG    step  2407 loss = -0.821936
DEBUG    step  2408 loss = 0.0161349
DEBUG    step  2409 loss = 2.05343
DEBUG    step  2410 loss = -0.449846
DEBUG    step  2411 loss = -0.112297
DEBUG    step  2412 loss = 0.23516
DEBUG    step  2413 loss = 0.598729
DEBUG    step  2414 loss = -0.637791
DEBUG    step  2415 loss = -0.0771543
DEBUG    step  2416 loss = -0.720933
DEBUG    step  2417 loss = -0.324247
DEBUG    step  2418 loss = -0.615081
DEBUG    step  2419 loss = -0.489061
DEBUG    step  2420 loss = -0.81913
DEBUG    step  2421 loss = -0.291852
DEBUG    step  2422 loss = -0.279411
DEBUG    step  2423 loss = -0.168712
DEBUG    step  2424 loss = -0.823371
DEBUG    step  2425 loss = -0.956634
DEBUG    step  2426 loss = 0.283457
DEBUG    step  2427 loss = 0.194569
DEBUG    step  2428 loss = -0.838871
DEBUG    step  2429 loss = -0.0047413
DEBUG    step  2430 loss = -0.559076
DEBUG    step  2431 loss = -0.689148
DEBUG    step  2432 loss = -0.299682
DEBUG    step  2433 loss = -0.884385
DEBUG    step  2434 loss = -0.595315
DEBUG    step  2435 loss = -1.11435
DEBUG    step  2436 loss = 0.0495753
DEBUG    step  2437 loss = -0.0852002
DEBUG    step  2438 loss = -0.15404
DEBUG    step  2439 loss = -0.266736
DEBUG    step  2440 loss = 0.195932
DEBUG    step  2441 loss = 0.185633
DEBUG    step  2442 loss = -0.863258
DEBUG    step  2443 loss = -0.382026
DEBUG    step  2444 loss = 0.252158
DEBUG    step  2445 loss = -0.448511
DEBUG    step  2446 loss = -0.179625
DEBUG    step  2447 loss = -0.114999
DEBUG    step  2448 loss = -1.00638
DEBUG    step  2449 loss = -0.0562548
DEBUG    step  2450 loss = 0.120608
DEBUG    step  2451 loss = -0.248703
DEBUG    step  2452 loss = 0.580167
DEBUG    step  2453 loss = -0.403365
DEBUG    step  2454 loss = -0.427596
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2455 loss = -0.386274
DEBUG    step  2456 loss = -0.0709784
DEBUG    step  2457 loss = -0.478124
DEBUG    step  2458 loss = -0.427781
DEBUG    step  2459 loss = 0.213299
DEBUG    step  2460 loss = 0.185551
DEBUG    step  2461 loss = -1.15001
DEBUG    step  2462 loss = -0.908913
DEBUG    step  2463 loss = -0.296839
DEBUG    step  2464 loss = -0.213982
DEBUG    step  2465 loss = -0.139768
DEBUG    step  2466 loss = -0.554577
DEBUG    step  2467 loss = -1.29373
DEBUG    step  2468 loss = 0.168238
DEBUG    step  2469 loss = 0.134877
DEBUG    step  2470 loss = -0.255521
DEBUG    step  2471 loss = -0.750256
DEBUG    step  2472 loss = -0.0114451
DEBUG    step  2473 loss = -0.410735
DEBUG    step  2474 loss = 0.218873
DEBUG    step  2475 loss = -0.141217
DEBUG    step  2476 loss = -0.78113
DEBUG    step  2477 loss = -0.143108
DEBUG    step  2478 loss = -0.0878578
DEBUG    step  2479 loss = 0.498992
DEBUG    step  2480 loss = -0.385873
DEBUG    step  2481 loss = 0.697456
DEBUG    step  2482 loss = -0.330902
DEBUG    step  2483 loss = -0.416052
DEBUG    step  2484 loss = -0.0582824
DEBUG    step  2485 loss = -0.749726
DEBUG    step  2486 loss = -0.705093
DEBUG    step  2487 loss = -0.366732
DEBUG    step  2488 loss = 0.0636343
DEBUG    step  2489 loss = -0.428274
DEBUG    step  2490 loss = -0.97996
DEBUG    step  2491 loss = -0.721423
DEBUG    step  2492 loss = -0.901971
DEBUG    step  2493 loss = -0.821726
DEBUG    step  2494 loss = -0.48277
DEBUG    step  2495 loss = 0.159761
DEBUG    step  2496 loss = -0.802472
DEBUG    step  2497 loss = -0.687559
DEBUG    step  2498 loss = -0.256268
DEBUG    step  2499 loss = -0.571636
DEBUG    step  2500 loss = -0.184076
DEBUG    step  2501 loss = -0.0532485
DEBUG    step  2502 loss = 0.0489593
DEBUG    step  2503 loss = -0.699592
DEBUG    step  2504 loss = -0.964232
DEBUG    step  2505 loss = -0.33835
DEBUG    step  2506 loss = -0.425566
DEBUG    step  2507 loss = -0.0965802
DEBUG    step  2508 loss = -0.745661
DEBUG    step  2509 loss = -0.103916
DEBUG    step  2510 loss = -0.489986
DEBUG    step  2511 loss = -1.22721
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2512 loss = -0.573065
DEBUG    step  2513 loss = -0.8967
DEBUG    step  2514 loss = -0.714046
DEBUG    step  2515 loss = -0.893781
DEBUG    step  2516 loss = 0.465743
DEBUG    step  2517 loss = -0.941392
DEBUG    step  2518 loss = -0.858442
DEBUG    step  2519 loss = -0.18183
DEBUG    step  2520 loss = -0.380441
DEBUG    step  2521 loss = -0.374258
DEBUG    step  2522 loss = -0.682367
DEBUG    step  2523 loss = -0.821137
DEBUG    step  2524 loss = -0.445525
DEBUG    step  2525 loss = -0.97567
DEBUG    step  2526 loss = -0.547556
DEBUG    step  2527 loss = -0.853315
DEBUG    step  2528 loss = 0.114161
DEBUG    step  2529 loss = -0.579036
DEBUG    step  2530 loss = 0.0135827
DEBUG    step  2531 loss = -0.0582753
DEBUG    step  2532 loss = -0.140801
DEBUG    step  2533 loss = -0.182517
DEBUG    step  2534 loss = -0.829945
DEBUG    step  2535 loss = -0.0669306
DEBUG    step  2536 loss = -0.467228
DEBUG    step  2537 loss = -0.584846
DEBUG    step  2538 loss = -0.273549
DEBUG    step  2539 loss = -0.00248221
DEBUG    step  2540 loss = -0.345479
DEBUG    step  2541 loss = -0.515946
DEBUG    step  2542 loss = -0.103854
DEBUG    step  2543 loss = 0.187452
DEBUG    step  2544 loss = -0.154338
DEBUG    step  2545 loss = -0.915668
DEBUG    step  2546 loss = -0.75074
DEBUG    step  2547 loss = -0.235062
DEBUG    step  2548 loss = -0.615748
DEBUG    step  2549 loss = 0.163511
DEBUG    step  2550 loss = -0.558204
DEBUG    step  2551 loss = -0.429658
DEBUG    step  2552 loss = -0.527625
DEBUG    step  2553 loss = -0.663658
DEBUG    step  2554 loss = -0.866039
DEBUG    step  2555 loss = -0.0667327
DEBUG    step  2556 loss = -1.14744
DEBUG    step  2557 loss = -0.599862
DEBUG    step  2558 loss = -0.628051
DEBUG    step  2559 loss = -1.02429
DEBUG    step  2560 loss = -0.812641
DEBUG    step  2561 loss = -0.207669
DEBUG    step  2562 loss = -0.346239
DEBUG    step  2563 loss = -0.42864
DEBUG    step  2564 loss = -0.769289
DEBUG    step  2565 loss = -0.442619
DEBUG    step  2566 loss = -0.551839
DEBUG    step  2567 loss = -0.434892
DEBUG    step  2568 loss = -0.822885
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2569 loss = -0.0774252
DEBUG    step  2570 loss = -0.962704
DEBUG    step  2571 loss = 0.382489
DEBUG    step  2572 loss = -0.340682
DEBUG    step  2573 loss = -0.42353
DEBUG    step  2574 loss = -0.0114898
DEBUG    step  2575 loss = -0.210306
DEBUG    step  2576 loss = -0.625316
DEBUG    step  2577 loss = -0.61977
DEBUG    step  2578 loss = -0.641895
DEBUG    step  2579 loss = -0.158468
DEBUG    step  2580 loss = -0.376173
DEBUG    step  2581 loss = -0.562516
DEBUG    step  2582 loss = -0.606728
DEBUG    step  2583 loss = -0.486623
DEBUG    step  2584 loss = -0.253736
DEBUG    step  2585 loss = 0.342148
DEBUG    step  2586 loss = -0.165116
DEBUG    step  2587 loss = -0.173551
DEBUG    step  2588 loss = -1.46536
DEBUG    step  2589 loss = 0.0896398
DEBUG    step  2590 loss = -0.545322
DEBUG    step  2591 loss = -0.406094
DEBUG    step  2592 loss = -0.918525
DEBUG    step  2593 loss = -0.894497
DEBUG    step  2594 loss = -0.578103
DEBUG    step  2595 loss = -0.553256
DEBUG    step  2596 loss = -0.593555
DEBUG    step  2597 loss = 0.00266581
DEBUG    step  2598 loss = -0.0584986
DEBUG    step  2599 loss = -0.607323
DEBUG    step  2600 loss = 0.38463
DEBUG    step  2601 loss = -0.481794
DEBUG    step  2602 loss = -0.902755
DEBUG    step  2603 loss = -0.823573
DEBUG    step  2604 loss = -0.581352
DEBUG    step  2605 loss = -0.546566
DEBUG    step  2606 loss = -1.1963
DEBUG    step  2607 loss = -0.66562
DEBUG    step  2608 loss = -0.885256
DEBUG    step  2609 loss = -0.510776
DEBUG    step  2610 loss = -0.414367
DEBUG    step  2611 loss = -0.63994
DEBUG    step  2612 loss = -0.993912
DEBUG    step  2613 loss = -1.01504
DEBUG    step  2614 loss = 0.596202
DEBUG    step  2615 loss = 0.482037
DEBUG    step  2616 loss = -0.301577
DEBUG    step  2617 loss = -1.49396
DEBUG    step  2618 loss = -0.392669
DEBUG    step  2619 loss = -0.324627
DEBUG    step  2620 loss = 0.619205
DEBUG    step  2621 loss = -0.269684
DEBUG    step  2622 loss = -0.661252
DEBUG    step  2623 loss = -0.774471
DEBUG    step  2624 loss = -1.18561
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2625 loss = -0.275053
DEBUG    step  2626 loss = -0.887767
DEBUG    step  2627 loss = 0.287073
DEBUG    step  2628 loss = -0.905378
DEBUG    step  2629 loss = 0.0570901
DEBUG    step  2630 loss = -0.351999
DEBUG    step  2631 loss = -0.118707
DEBUG    step  2632 loss = -0.671623
DEBUG    step  2633 loss = -0.681996
DEBUG    step  2634 loss = -0.521377
DEBUG    step  2635 loss = -0.617793
DEBUG    step  2636 loss = -0.603524
DEBUG    step  2637 loss = -0.821486
DEBUG    step  2638 loss = -0.356088
DEBUG    step  2639 loss = -0.536534
DEBUG    step  2640 loss = -0.747998
DEBUG    step  2641 loss = -0.439992
DEBUG    step  2642 loss = -0.0627628
DEBUG    step  2643 loss = 0.331022
DEBUG    step  2644 loss = -0.441603
DEBUG    step  2645 loss = -0.515788
DEBUG    step  2646 loss = -0.475961
DEBUG    step  2647 loss = -0.401744
DEBUG    step  2648 loss = 0.262217
DEBUG    step  2649 loss = -0.831643
DEBUG    step  2650 loss = -1.12754
DEBUG    step  2651 loss = -0.829439
DEBUG    step  2652 loss = 0.0111126
DEBUG    step  2653 loss = 0.0545446
DEBUG    step  2654 loss = -0.34779
DEBUG    step  2655 loss = -0.239686
DEBUG    step  2656 loss = -0.0659961
DEBUG    step  2657 loss = -0.0800167
DEBUG    step  2658 loss = -0.56742
DEBUG    step  2659 loss = -1.12966
DEBUG    step  2660 loss = -0.735846
DEBUG    step  2661 loss = -0.857747
DEBUG    step  2662 loss = -0.626603
DEBUG    step  2663 loss = 0.501296
DEBUG    step  2664 loss = -0.345909
DEBUG    step  2665 loss = -0.48826
DEBUG    step  2666 loss = -0.425832
DEBUG    step  2667 loss = -0.622227
DEBUG    step  2668 loss = 0.0905803
DEBUG    step  2669 loss = -0.934806
DEBUG    step  2670 loss = -0.55195
DEBUG    step  2671 loss = 0.285835
DEBUG    step  2672 loss = -0.62289
DEBUG    step  2673 loss = -0.438078
DEBUG    step  2674 loss = -0.351686
DEBUG    step  2675 loss = -0.476577
DEBUG    step  2676 loss = -0.894385
DEBUG    step  2677 loss = -0.258823
DEBUG    step  2678 loss = -0.413825
DEBUG    step  2679 loss = -0.737152
DEBUG    step  2680 loss = -0.756135
DEBUG    step  2681 loss = -0.475365
```

(continues on next page)

(continued from previous page)

```
DEBUG    step  2682 loss = -0.271527
DEBUG    step  2683 loss = -0.628242
DEBUG    step  2684 loss = -1.36686
DEBUG    step  2685 loss = -0.608447
DEBUG    step  2686 loss = -0.685795
DEBUG    step  2687 loss = -0.240269
DEBUG    step  2688 loss = 0.146378
DEBUG    step  2689 loss = -1.10885
```

[10]:
```
# predict
ite_train = cevae.predict(X_train)
ite_val = cevae.predict(X_val)
```

```
INFO     Evaluating 538 minibatches
DEBUG    batch ate = 0.62191
DEBUG    batch ate = 0.613137
DEBUG    batch ate = 0.688279
DEBUG    batch ate = 0.530233
DEBUG    batch ate = 0.814089
DEBUG    batch ate = 0.623182
DEBUG    batch ate = 0.657884
DEBUG    batch ate = 0.594205
DEBUG    batch ate = 0.319953
DEBUG    batch ate = 0.557599
DEBUG    batch ate = 0.718177
DEBUG    batch ate = 0.441256
DEBUG    batch ate = 0.654653
DEBUG    batch ate = 0.70725
DEBUG    batch ate = 0.715862
DEBUG    batch ate = 0.193786
DEBUG    batch ate = 0.557451
DEBUG    batch ate = 0.788378
DEBUG    batch ate = 0.605489
DEBUG    batch ate = 0.669786
DEBUG    batch ate = 0.852794
DEBUG    batch ate = 0.755987
DEBUG    batch ate = 0.510262
DEBUG    batch ate = 0.502153
DEBUG    batch ate = 0.254691
DEBUG    batch ate = 0.369999
DEBUG    batch ate = 0.59401
DEBUG    batch ate = 0.608015
DEBUG    batch ate = 0.661765
DEBUG    batch ate = 0.25462
DEBUG    batch ate = 0.771231
DEBUG    batch ate = 0.530303
DEBUG    batch ate = 0.566246
DEBUG    batch ate = 0.683882
DEBUG    batch ate = 0.616635
DEBUG    batch ate = 0.324804
DEBUG    batch ate = 0.383451
DEBUG    batch ate = 0.690402
DEBUG    batch ate = 0.558513
DEBUG    batch ate = 0.618007
DEBUG    batch ate = 0.551096
DEBUG    batch ate = 0.462644
DEBUG    batch ate = 0.615761
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.543891
DEBUG    batch ate = 0.432806
DEBUG    batch ate = 0.562174
DEBUG    batch ate = 0.654926
DEBUG    batch ate = 0.421796
DEBUG    batch ate = 0.719893
DEBUG    batch ate = 0.454017
DEBUG    batch ate = 0.699385
DEBUG    batch ate = 0.54048
DEBUG    batch ate = 0.333772
DEBUG    batch ate = 0.737522
DEBUG    batch ate = 0.5696
DEBUG    batch ate = 0.467629
DEBUG    batch ate = 0.601579
DEBUG    batch ate = 0.509313
DEBUG    batch ate = 0.385523
DEBUG    batch ate = 0.510085
DEBUG    batch ate = 0.661952
DEBUG    batch ate = 0.600664
DEBUG    batch ate = 0.066584
DEBUG    batch ate = 0.552528
DEBUG    batch ate = 0.467475
DEBUG    batch ate = 0.539326
DEBUG    batch ate = 0.694311
DEBUG    batch ate = 0.198014
DEBUG    batch ate = 0.61709
DEBUG    batch ate = 0.408558
DEBUG    batch ate = 0.684187
DEBUG    batch ate = 0.447501
DEBUG    batch ate = 0.347885
DEBUG    batch ate = 0.561035
DEBUG    batch ate = 0.617192
DEBUG    batch ate = 0.81278
DEBUG    batch ate = 0.61961
DEBUG    batch ate = 1.01213
DEBUG    batch ate = 0.345585
DEBUG    batch ate = 0.51818
DEBUG    batch ate = 0.436719
DEBUG    batch ate = 0.604546
DEBUG    batch ate = 0.706353
DEBUG    batch ate = 0.661419
DEBUG    batch ate = 0.787418
DEBUG    batch ate = 0.61231
DEBUG    batch ate = 0.629355
DEBUG    batch ate = 0.550861
DEBUG    batch ate = 0.472948
DEBUG    batch ate = 0.594738
DEBUG    batch ate = 0.844747
DEBUG    batch ate = 0.682486
DEBUG    batch ate = 0.607738
DEBUG    batch ate = 0.49322
DEBUG    batch ate = 0.547857
DEBUG    batch ate = 0.255665
DEBUG    batch ate = 0.564768
DEBUG    batch ate = 0.34345
DEBUG    batch ate = 0.40075
DEBUG    batch ate = 0.72982
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.878728
DEBUG    batch ate = 0.860621
DEBUG    batch ate = 0.544359
DEBUG    batch ate = 0.777127
DEBUG    batch ate = 0.590297
DEBUG    batch ate = 0.880415
DEBUG    batch ate = 0.67375
DEBUG    batch ate = 0.784914
DEBUG    batch ate = 0.511374
DEBUG    batch ate = 0.327954
DEBUG    batch ate = 0.628989
DEBUG    batch ate = 0.529468
DEBUG    batch ate = 0.688235
DEBUG    batch ate = 0.872871
DEBUG    batch ate = 0.3485
DEBUG    batch ate = 0.572016
DEBUG    batch ate = 0.565154
DEBUG    batch ate = 0.588927
DEBUG    batch ate = 0.520636
DEBUG    batch ate = 0.345301
DEBUG    batch ate = 0.611386
DEBUG    batch ate = 0.702772
DEBUG    batch ate = 0.764302
DEBUG    batch ate = 0.638517
DEBUG    batch ate = 0.498749
DEBUG    batch ate = 0.922372
DEBUG    batch ate = 0.648347
DEBUG    batch ate = 0.930839
DEBUG    batch ate = 0.841956
DEBUG    batch ate = 0.687886
DEBUG    batch ate = 0.804776
DEBUG    batch ate = 0.550305
DEBUG    batch ate = 0.625526
DEBUG    batch ate = 0.856957
DEBUG    batch ate = 0.470616
DEBUG    batch ate = 0.507122
DEBUG    batch ate = 0.358198
DEBUG    batch ate = 0.6335
DEBUG    batch ate = 0.473881
DEBUG    batch ate = 0.415356
DEBUG    batch ate = 0.309733
DEBUG    batch ate = 0.290068
DEBUG    batch ate = 0.470317
DEBUG    batch ate = 0.668486
DEBUG    batch ate = 0.580281
DEBUG    batch ate = 0.772137
DEBUG    batch ate = 0.490976
DEBUG    batch ate = 0.511012
DEBUG    batch ate = 0.441551
DEBUG    batch ate = 0.575225
DEBUG    batch ate = 0.591247
DEBUG    batch ate = 0.368313
DEBUG    batch ate = 0.350138
DEBUG    batch ate = 0.603038
DEBUG    batch ate = 0.241947
DEBUG    batch ate = 0.599275
DEBUG    batch ate = 0.41003
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.447525
DEBUG    batch ate = 0.79099
DEBUG    batch ate = 0.506499
DEBUG    batch ate = 0.61826
DEBUG    batch ate = 0.651964
DEBUG    batch ate = 0.52761
DEBUG    batch ate = 0.888067
DEBUG    batch ate = 0.367077
DEBUG    batch ate = 0.524761
DEBUG    batch ate = 0.6165
DEBUG    batch ate = 0.72863
DEBUG    batch ate = 0.516559
DEBUG    batch ate = 0.385291
DEBUG    batch ate = 0.660073
DEBUG    batch ate = 0.465947
DEBUG    batch ate = 0.586065
DEBUG    batch ate = 0.533599
DEBUG    batch ate = 0.916433
DEBUG    batch ate = 0.658235
DEBUG    batch ate = 0.770213
DEBUG    batch ate = 0.634768
DEBUG    batch ate = 0.887955
DEBUG    batch ate = 0.374664
DEBUG    batch ate = 0.649699
DEBUG    batch ate = 0.550386
DEBUG    batch ate = 0.516355
DEBUG    batch ate = 0.425265
DEBUG    batch ate = 0.264789
DEBUG    batch ate = 0.775339
DEBUG    batch ate = 0.636203
DEBUG    batch ate = 0.507562
DEBUG    batch ate = 0.885973
DEBUG    batch ate = 0.951861
DEBUG    batch ate = 0.370282
DEBUG    batch ate = 0.69922
DEBUG    batch ate = 0.956577
DEBUG    batch ate = 0.789856
DEBUG    batch ate = 0.726278
DEBUG    batch ate = 0.165073
DEBUG    batch ate = 0.530907
DEBUG    batch ate = 0.602567
DEBUG    batch ate = 0.682041
DEBUG    batch ate = 0.54427
DEBUG    batch ate = 0.787318
DEBUG    batch ate = 0.491623
DEBUG    batch ate = 0.794449
DEBUG    batch ate = 0.928849
DEBUG    batch ate = 0.771662
DEBUG    batch ate = 0.722534
DEBUG    batch ate = 0.611424
DEBUG    batch ate = 0.754558
DEBUG    batch ate = 0.466829
DEBUG    batch ate = 0.623566
DEBUG    batch ate = 0.595247
DEBUG    batch ate = 0.790067
DEBUG    batch ate = 0.218814
DEBUG    batch ate = 0.551078
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.561368
DEBUG    batch ate = 0.823733
DEBUG    batch ate = 0.725582
DEBUG    batch ate = 0.685417
DEBUG    batch ate = 0.573616
DEBUG    batch ate = 0.408314
DEBUG    batch ate = 0.420605
DEBUG    batch ate = 0.699393
DEBUG    batch ate = 0.485361
DEBUG    batch ate = 0.470607
DEBUG    batch ate = 0.672379
DEBUG    batch ate = 0.515571
DEBUG    batch ate = 0.837184
DEBUG    batch ate = 0.383294
DEBUG    batch ate = 0.631237
DEBUG    batch ate = 0.660588
DEBUG    batch ate = 0.454409
DEBUG    batch ate = 0.277474
DEBUG    batch ate = 1.08705
DEBUG    batch ate = 0.542072
DEBUG    batch ate = 0.667987
DEBUG    batch ate = 0.474515
DEBUG    batch ate = 0.462981
DEBUG    batch ate = 0.581607
DEBUG    batch ate = 0.539565
DEBUG    batch ate = 0.740687
DEBUG    batch ate = 0.672987
DEBUG    batch ate = 0.725537
DEBUG    batch ate = 0.683099
DEBUG    batch ate = 0.695347
DEBUG    batch ate = 0.533302
DEBUG    batch ate = 0.625668
DEBUG    batch ate = 0.744886
DEBUG    batch ate = 0.686994
DEBUG    batch ate = 0.572683
DEBUG    batch ate = 0.431316
DEBUG    batch ate = 0.521101
DEBUG    batch ate = 0.651604
DEBUG    batch ate = 0.514384
DEBUG    batch ate = 0.471155
DEBUG    batch ate = 0.759972
DEBUG    batch ate = 0.633456
DEBUG    batch ate = 0.52144
DEBUG    batch ate = 0.675739
DEBUG    batch ate = 0.713319
DEBUG    batch ate = 0.749301
DEBUG    batch ate = 0.637229
DEBUG    batch ate = 0.690767
DEBUG    batch ate = 0.638464
DEBUG    batch ate = 0.804409
DEBUG    batch ate = 0.379763
DEBUG    batch ate = 0.939645
DEBUG    batch ate = 0.566416
DEBUG    batch ate = 0.722778
DEBUG    batch ate = 0.875249
DEBUG    batch ate = 0.585553
DEBUG    batch ate = 0.452997
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.660046
DEBUG    batch ate = 0.523958
DEBUG    batch ate = 0.743689
DEBUG    batch ate = 0.281901
DEBUG    batch ate = 0.79823
DEBUG    batch ate = 0.501476
DEBUG    batch ate = 0.27024
DEBUG    batch ate = 0.661638
DEBUG    batch ate = 0.530568
DEBUG    batch ate = 0.276738
DEBUG    batch ate = 0.734873
DEBUG    batch ate = 0.547245
```

```
DEBUG    batch ate = 0.642462
DEBUG    batch ate = 0.69965
DEBUG    batch ate = 0.544179
DEBUG    batch ate = 0.501292
DEBUG    batch ate = 0.782594
DEBUG    batch ate = 0.718873
DEBUG    batch ate = 0.53492
DEBUG    batch ate = 0.602767
DEBUG    batch ate = 0.642604
DEBUG    batch ate = 0.899802
DEBUG    batch ate = 0.345271
DEBUG    batch ate = 0.408736
DEBUG    batch ate = 0.503462
DEBUG    batch ate = 0.548023
DEBUG    batch ate = 0.869944
DEBUG    batch ate = 0.712165
DEBUG    batch ate = 0.840788
DEBUG    batch ate = 0.802797
DEBUG    batch ate = 0.448752
DEBUG    batch ate = 0.489339
DEBUG    batch ate = 0.760921
DEBUG    batch ate = 0.549896
DEBUG    batch ate = 0.337833
DEBUG    batch ate = 0.489319
DEBUG    batch ate = 0.349298
DEBUG    batch ate = 0.0851573
DEBUG    batch ate = 0.701312
DEBUG    batch ate = 0.426929
DEBUG    batch ate = 0.52591
DEBUG    batch ate = 0.45672
DEBUG    batch ate = 0.691007
DEBUG    batch ate = 0.681652
DEBUG    batch ate = 0.414373
DEBUG    batch ate = 0.43001
DEBUG    batch ate = 0.698964
DEBUG    batch ate = 0.569967
DEBUG    batch ate = 0.670148
DEBUG    batch ate = 0.612077
DEBUG    batch ate = 0.559155
DEBUG    batch ate = 0.839547
DEBUG    batch ate = 0.704653
DEBUG    batch ate = 0.44604
DEBUG    batch ate = 0.608618
DEBUG    batch ate = 0.744417
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.340019
DEBUG    batch ate = 0.469705
DEBUG    batch ate = 0.859227
DEBUG    batch ate = 0.732652
DEBUG    batch ate = 0.624253
DEBUG    batch ate = 0.767217
DEBUG    batch ate = 0.431167
DEBUG    batch ate = 0.712165
DEBUG    batch ate = 0.576947
DEBUG    batch ate = 0.546332
DEBUG    batch ate = 0.52999
DEBUG    batch ate = 0.349895
DEBUG    batch ate = 0.625377
DEBUG    batch ate = 0.564784
DEBUG    batch ate = 0.827983
DEBUG    batch ate = 0.402039
DEBUG    batch ate = 0.732634
DEBUG    batch ate = 0.828913
DEBUG    batch ate = 0.580144
DEBUG    batch ate = 0.568022
DEBUG    batch ate = 0.561761
DEBUG    batch ate = 0.294596
DEBUG    batch ate = 0.636919
DEBUG    batch ate = 0.655477
DEBUG    batch ate = 0.925995
DEBUG    batch ate = 0.729636
DEBUG    batch ate = 0.550091
DEBUG    batch ate = 0.558647
DEBUG    batch ate = 0.673149
DEBUG    batch ate = 0.657379
DEBUG    batch ate = 0.553136
DEBUG    batch ate = 0.784905
DEBUG    batch ate = 0.72343
DEBUG    batch ate = 0.872444
DEBUG    batch ate = 0.594647
DEBUG    batch ate = 0.815522
DEBUG    batch ate = 0.882869
DEBUG    batch ate = 0.505135
DEBUG    batch ate = 0.608259
DEBUG    batch ate = 0.438947
DEBUG    batch ate = 0.642148
DEBUG    batch ate = 0.42703
DEBUG    batch ate = 0.492255
DEBUG    batch ate = 1.01806
DEBUG    batch ate = 0.488789
DEBUG    batch ate = 0.353427
DEBUG    batch ate = 0.697426
DEBUG    batch ate = 0.454108
DEBUG    batch ate = 0.585995
DEBUG    batch ate = 0.898554
DEBUG    batch ate = 0.462355
DEBUG    batch ate = 0.847193
DEBUG    batch ate = 0.435861
DEBUG    batch ate = 0.350475
DEBUG    batch ate = 0.494122
DEBUG    batch ate = 0.641375
DEBUG    batch ate = 1.05287
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.560613
DEBUG    batch ate = 0.622122
DEBUG    batch ate = 0.617646
DEBUG    batch ate = 0.438831
DEBUG    batch ate = 0.413241
DEBUG    batch ate = 0.709999
DEBUG    batch ate = 0.393058
DEBUG    batch ate = 0.577082
DEBUG    batch ate = 0.449773
DEBUG    batch ate = 0.409307
DEBUG    batch ate = 0.717688
DEBUG    batch ate = 0.680811
DEBUG    batch ate = 0.636654
DEBUG    batch ate = 0.537257
DEBUG    batch ate = 0.485248
DEBUG    batch ate = 0.611201
DEBUG    batch ate = 0.66029
DEBUG    batch ate = 0.621785
DEBUG    batch ate = 0.656557
DEBUG    batch ate = 0.50069
DEBUG    batch ate = 0.531677
DEBUG    batch ate = 0.539529
DEBUG    batch ate = 0.7621
DEBUG    batch ate = 0.34175
DEBUG    batch ate = 0.573927
DEBUG    batch ate = 0.698847
DEBUG    batch ate = 0.687271
DEBUG    batch ate = 0.625974
DEBUG    batch ate = 0.623745
DEBUG    batch ate = 0.542737
DEBUG    batch ate = 0.203161
DEBUG    batch ate = 0.656258
DEBUG    batch ate = 0.20316
DEBUG    batch ate = 0.333921
DEBUG    batch ate = 0.503528
DEBUG    batch ate = 0.274319
DEBUG    batch ate = 0.435086
DEBUG    batch ate = 0.577274
DEBUG    batch ate = 0.404617
DEBUG    batch ate = 0.488066
DEBUG    batch ate = 0.804592
DEBUG    batch ate = 0.731865
DEBUG    batch ate = 0.751529
DEBUG    batch ate = 0.847831
DEBUG    batch ate = 0.737108
DEBUG    batch ate = 0.403549
DEBUG    batch ate = 0.659598
DEBUG    batch ate = 0.777456
DEBUG    batch ate = 0.655091
DEBUG    batch ate = 0.805262
DEBUG    batch ate = 0.578173
DEBUG    batch ate = 0.749979
DEBUG    batch ate = 0.645467
DEBUG    batch ate = 0.765642
DEBUG    batch ate = 0.221318
DEBUG    batch ate = 0.566684
DEBUG    batch ate = 0.885021
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.798495
DEBUG    batch ate = 0.749958
DEBUG    batch ate = 0.404101
DEBUG    batch ate = 0.597844
DEBUG    batch ate = 0.548862
DEBUG    batch ate = 0.633423
DEBUG    batch ate = 0.58442
DEBUG    batch ate = 0.406284
DEBUG    batch ate = 0.497425
DEBUG    batch ate = 0.64323
DEBUG    batch ate = 0.764823
DEBUG    batch ate = 0.719326
DEBUG    batch ate = 0.850669
DEBUG    batch ate = 0.567251
DEBUG    batch ate = 0.531746
DEBUG    batch ate = 0.422011
DEBUG    batch ate = 0.469137
DEBUG    batch ate = 0.568481
DEBUG    batch ate = 0.336506
DEBUG    batch ate = 0.785506
DEBUG    batch ate = 0.771601
DEBUG    batch ate = 0.790584
DEBUG    batch ate = 0.756722
DEBUG    batch ate = 0.558484
DEBUG    batch ate = 0.565823
DEBUG    batch ate = 0.85092
DEBUG    batch ate = 0.836311
DEBUG    batch ate = 0.36647
DEBUG    batch ate = 0.671067
DEBUG    batch ate = 0.678834
DEBUG    batch ate = 0.7427
DEBUG    batch ate = 0.380171
DEBUG    batch ate = 0.702751
DEBUG    batch ate = 0.821684
DEBUG    batch ate = 0.183044
DEBUG    batch ate = 0.71705
DEBUG    batch ate = 0.650429
DEBUG    batch ate = 0.647615
DEBUG    batch ate = 0.590948
DEBUG    batch ate = 0.32329
DEBUG    batch ate = 0.8901
DEBUG    batch ate = 0.56427
DEBUG    batch ate = 0.335077
DEBUG    batch ate = 0.777793
DEBUG    batch ate = 0.669449
DEBUG    batch ate = 0.794569
DEBUG    batch ate = 0.455826
DEBUG    batch ate = 0.237244
DEBUG    batch ate = 0.449816
DEBUG    batch ate = 0.544514
DEBUG    batch ate = 0.426984
DEBUG    batch ate = 0.440946
DEBUG    batch ate = 0.331075
DEBUG    batch ate = 0.486034
DEBUG    batch ate = 0.518074
DEBUG    batch ate = 0.508189
DEBUG    batch ate = 0.7412
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.744264
DEBUG    batch ate = 0.23702
DEBUG    batch ate = 0.724052
DEBUG    batch ate = 0.26753
DEBUG    batch ate = 0.45962
DEBUG    batch ate = 0.447174
DEBUG    batch ate = 0.615098
DEBUG    batch ate = 0.665408
DEBUG    batch ate = 0.227405
DEBUG    batch ate = 0.567846
DEBUG    batch ate = 0.642301
DEBUG    batch ate = 0.572763
DEBUG    batch ate = 0.492713
DEBUG    batch ate = 0.495091
DEBUG    batch ate = 0.387373
DEBUG    batch ate = 0.536913
DEBUG    batch ate = 0.70732
DEBUG    batch ate = 0.57493
DEBUG    batch ate = 0.575226
DEBUG    batch ate = 0.820646
DEBUG    batch ate = 0.299924
DEBUG    batch ate = 0.521718
DEBUG    batch ate = 0.201825
DEBUG    batch ate = 0.575455
DEBUG    batch ate = 0.34346
DEBUG    batch ate = 0.511799
DEBUG    batch ate = 0.577593
DEBUG    batch ate = 0.606313
DEBUG    batch ate = 0.479831
DEBUG    batch ate = 0.430969
DEBUG    batch ate = 0.68106
DEBUG    batch ate = 0.393857
DEBUG    batch ate = 0.592259
DEBUG    batch ate = 0.904887
DEBUG    batch ate = 1.1646
DEBUG    batch ate = 0.462751
DEBUG    batch ate = 0.849577
DEBUG    batch ate = 0.675505
DEBUG    batch ate = 0.655771
DEBUG    batch ate = 0.433719
INFO     Evaluating 135 minibatches
DEBUG    batch ate = 0.228577
DEBUG    batch ate = 0.602583
DEBUG    batch ate = 0.802412
DEBUG    batch ate = 0.445214
DEBUG    batch ate = 0.569569
DEBUG    batch ate = 0.816098
DEBUG    batch ate = 0.799774
DEBUG    batch ate = 0.580379
DEBUG    batch ate = 0.705277
DEBUG    batch ate = 0.472644
DEBUG    batch ate = 0.425481
DEBUG    batch ate = 0.529719
DEBUG    batch ate = 1.03265
DEBUG    batch ate = 0.702212
DEBUG    batch ate = 0.716867
DEBUG    batch ate = 0.732634
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.479447
DEBUG    batch ate = 0.751748
DEBUG    batch ate = 0.372753
DEBUG    batch ate = 0.743915
DEBUG    batch ate = 0.695771
DEBUG    batch ate = 0.486699
DEBUG    batch ate = 0.617069
DEBUG    batch ate = 0.924266
DEBUG    batch ate = 0.41445
DEBUG    batch ate = 0.51611
DEBUG    batch ate = 0.570871
DEBUG    batch ate = 0.52222
```

```
DEBUG    batch ate = 0.550225
DEBUG    batch ate = 0.827474
DEBUG    batch ate = 0.660622
DEBUG    batch ate = 0.435264
DEBUG    batch ate = 0.252852
DEBUG    batch ate = 0.521581
DEBUG    batch ate = 0.620552
DEBUG    batch ate = 0.46738
DEBUG    batch ate = 0.469133
DEBUG    batch ate = 0.769782
DEBUG    batch ate = 0.641767
DEBUG    batch ate = 0.61662
DEBUG    batch ate = 0.497127
DEBUG    batch ate = 0.541457
DEBUG    batch ate = 0.950244
DEBUG    batch ate = 0.475156
DEBUG    batch ate = 0.752711
DEBUG    batch ate = 0.301103
DEBUG    batch ate = 0.843295
DEBUG    batch ate = 0.374278
DEBUG    batch ate = 0.686422
DEBUG    batch ate = 0.558687
DEBUG    batch ate = 0.66816
DEBUG    batch ate = 0.756011
DEBUG    batch ate = 0.268842
DEBUG    batch ate = 0.467443
DEBUG    batch ate = 0.7511
DEBUG    batch ate = 0.644642
DEBUG    batch ate = 0.763036
DEBUG    batch ate = 0.590393
DEBUG    batch ate = 0.693136
DEBUG    batch ate = 0.486587
DEBUG    batch ate = 0.604928
DEBUG    batch ate = 0.711657
DEBUG    batch ate = 0.606803
DEBUG    batch ate = 0.514715
DEBUG    batch ate = 0.755621
DEBUG    batch ate = 0.563381
DEBUG    batch ate = 0.658584
DEBUG    batch ate = 0.309254
DEBUG    batch ate = 0.186426
DEBUG    batch ate = 0.642211
DEBUG    batch ate = 0.726449
DEBUG    batch ate = 0.609017
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.693574
DEBUG    batch ate = 0.619707
DEBUG    batch ate = 0.711907
DEBUG    batch ate = 0.763202
DEBUG    batch ate = 0.583925
DEBUG    batch ate = 0.732382
DEBUG    batch ate = 0.598957
DEBUG    batch ate = 0.61077
DEBUG    batch ate = 0.407628
DEBUG    batch ate = 0.813409
DEBUG    batch ate = 0.879196
DEBUG    batch ate = 0.59526
DEBUG    batch ate = 0.597031
DEBUG    batch ate = 0.404295
DEBUG    batch ate = 0.444806
DEBUG    batch ate = 0.976863
DEBUG    batch ate = 0.191305
DEBUG    batch ate = 0.55377
DEBUG    batch ate = 1.03828
DEBUG    batch ate = 0.478516
DEBUG    batch ate = 0.925168
DEBUG    batch ate = 0.605732
DEBUG    batch ate = 0.321156
DEBUG    batch ate = 0.47538
DEBUG    batch ate = 0.750148
DEBUG    batch ate = 0.468002
DEBUG    batch ate = 0.483354
DEBUG    batch ate = 0.727932
DEBUG    batch ate = 0.499526
DEBUG    batch ate = 0.505064
DEBUG    batch ate = 1.03597
DEBUG    batch ate = 0.528672
DEBUG    batch ate = 0.713761
DEBUG    batch ate = 0.657063
DEBUG    batch ate = 0.677198
DEBUG    batch ate = 0.761366
DEBUG    batch ate = 0.569046
DEBUG    batch ate = 0.806944
DEBUG    batch ate = 0.512402
DEBUG    batch ate = 0.638473
DEBUG    batch ate = 0.594415
DEBUG    batch ate = 0.662585
DEBUG    batch ate = 0.815776
DEBUG    batch ate = 0.547243
DEBUG    batch ate = 0.446772
DEBUG    batch ate = 0.609724
DEBUG    batch ate = 0.672535
DEBUG    batch ate = 0.294262
DEBUG    batch ate = 0.650225
DEBUG    batch ate = 0.437027
DEBUG    batch ate = 0.395884
DEBUG    batch ate = 0.457884
DEBUG    batch ate = 0.381654
DEBUG    batch ate = 0.474322
DEBUG    batch ate = 0.636114
DEBUG    batch ate = 0.433205
DEBUG    batch ate = 0.340026
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.631428
DEBUG    batch ate = 0.465448
DEBUG    batch ate = 0.438805
DEBUG    batch ate = 0.50323
DEBUG    batch ate = 0.522954
DEBUG    batch ate = 0.58916
```

[11]:
```
ate_train = ite_train.mean()
ate_val = ite_val.mean()
print(ate_train, ate_val)
```

```
0.58953923 0.5956359
```

5.14.3 Meta Learners

[12]:
```
# fit propensity model
p_model = ElasticNetPropensityModel()
p_train = p_model.fit_predict(X_train, treatment_train)
p_val = p_model.fit_predict(X_val, treatment_val)
```

[13]:
```
s_learner = BaseSRegressor(LGBMRegressor())
s_ate = s_learner.estimate_ate(X_train, treatment_train, y_train)[0]
s_ite_train = s_learner.fit_predict(X_train, treatment_train, y_train)
s_ite_val = s_learner.predict(X_val)

t_learner = BaseTRegressor(LGBMRegressor())
t_ate = t_learner.estimate_ate(X_train, treatment_train, y_train)[0][0]
t_ite_train = t_learner.fit_predict(X_train, treatment_train, y_train)
t_ite_val = t_learner.predict(X_val, treatment_val, y_val)

x_learner = BaseXRegressor(LGBMRegressor())
x_ate = x_learner.estimate_ate(X_train, treatment_train, y_train, p_train)[0][0]
x_ite_train = x_learner.fit_predict(X_train, treatment_train, y_train, p_train)
x_ite_val = x_learner.predict(X_val, treatment_val, y_val, p_val)

r_learner = BaseRRegressor(LGBMRegressor())
r_ate = r_learner.estimate_ate(X_train, treatment_train, y_train, p_train)[0][0]
r_ite_train = r_learner.fit_predict(X_train, treatment_train, y_train, p_train)
r_ite_val = r_learner.predict(X_val)
```

5.14.4 Model Results Comparsion

Training

[14]:
```
df_preds_train = pd.DataFrame([s_ite_train.ravel(),
                               t_ite_train.ravel(),
                               x_ite_train.ravel(),
                               r_ite_train.ravel(),
                               ite_train.ravel(),
                               tau_train.ravel(),
                               treatment_train.ravel(),
                               y_train.ravel()],
```

(continues on next page)

(continued from previous page)

```
                                index=['S','T','X','R','CEVAE','tau','w','y']).T

df_cumgain_train = get_cumgain(df_preds_train)
```

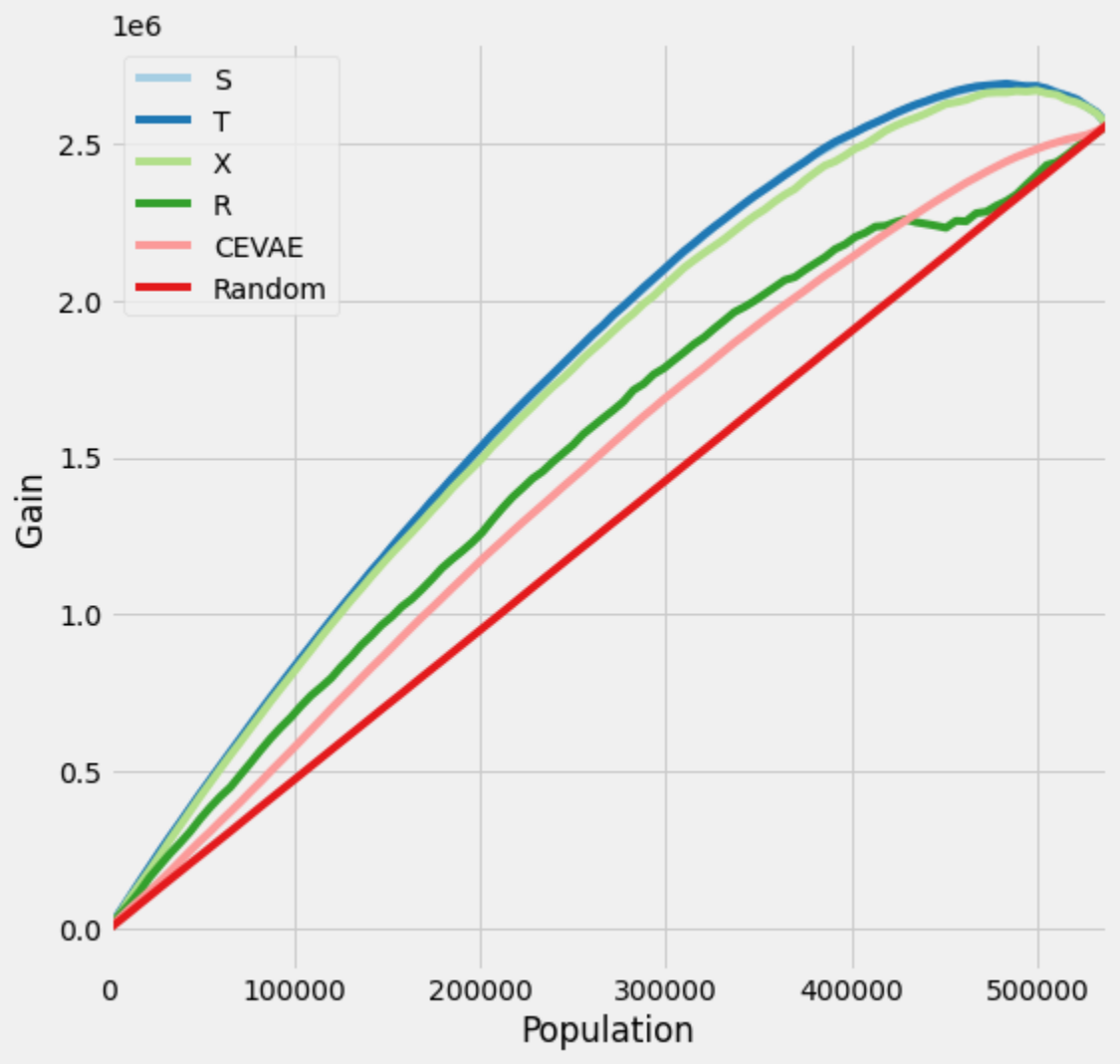
[15]:
```
df_result_train = pd.DataFrame([s_ate, t_ate, x_ate, r_ate, ate_train, tau_train.
↪mean()],
                                index=['S','T','X','R','CEVAE','actual'], columns=['ATE
↪'])
df_result_train['MAE'] = [mean_absolute_error(t,p) for t,p in zip([s_ite_train, t_ite_
↪train, x_ite_train, r_ite_train, ite_train],
                                                                  [tau_train.values.
↪reshape(-1,1)]*5 )
                          ] + [None]
df_result_train['AUUC'] = auuc_score(df_preds_train)
```

[16]:
```
df_result_train
```

[16]:

	ATE	MAE	AUUC
S	4.690540	4.581416	0.684130
T	4.708557	4.715296	0.684878
X	4.555315	4.549527	0.671956
R	0.714936	5.991034	0.586835
CEVAE	0.589539	6.238858	0.566627
actual	4.755900	NaN	NaN

[17]:
```
plot_gain(df_preds_train)
```



Validation

[18]:
```
df_preds_val = pd.DataFrame([s_ite_val.ravel(),
                             t_ite_val.ravel(),
                             x_ite_val.ravel(),
                             r_ite_val.ravel(),
                             ite_val.ravel(),
                             tau_val.ravel(),
                             treatment_val.ravel(),
                             y_val.ravel()],
                             index=['S','T','X','R','CEVAE','tau','w','y']).T

df_cumgain_val = get_cumgain(df_preds_val)
```

[19]:
```
df_result_val = pd.DataFrame([s_ite_val.mean(), t_ite_val.mean(), x_ite_val.mean(), r_
↪ite_val.mean(), ate_val, tau_val.mean()],
                              index=['S','T','X','R','CEVAE','actual'], columns=['ATE
```

(continues on next page)

(continued from previous page)

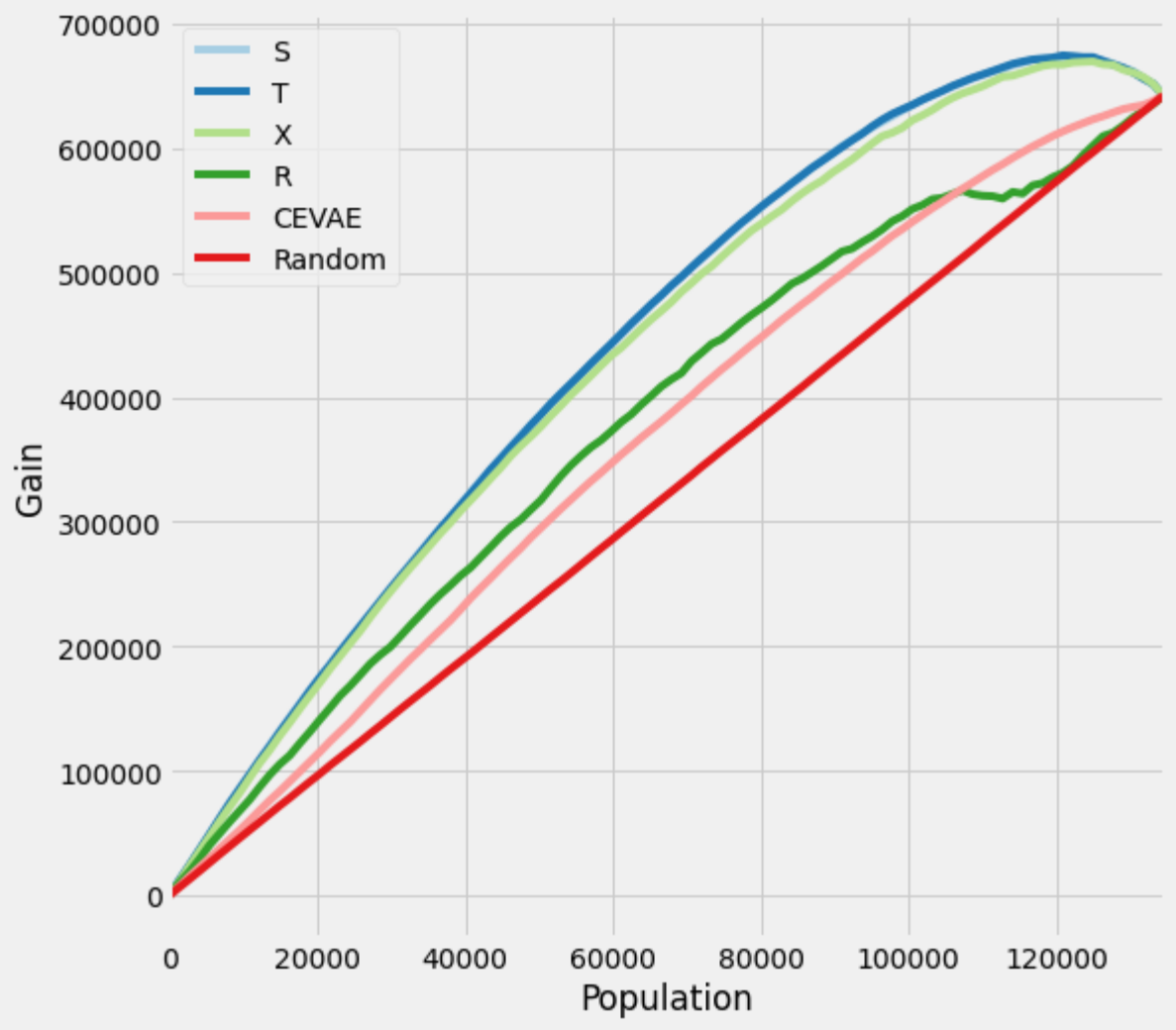
```
↪'])
df_result_val['MAE'] = [mean_absolute_error(t,p) for t,p in zip([s_ite_val, t_ite_val,
↪ x_ite_val, r_ite_val, ite_val],
                                                                [tau_val.values.
↪reshape(-1,1)]*5 )
                                ] + [None]
df_result_val['AUUC'] = auuc_score(df_preds_val)
```

[20]:
```
df_result_val
```

[20]:

	ATE	MAE	AUUC
S	4.690676	4.582191	0.683782
T	4.709923	4.717909	0.684032
X	4.560680	4.544644	0.671907
R	0.761550	5.997526	0.586110
CEVAE	0.595636	6.241192	0.566356
actual	4.774991	NaN	NaN

[21]:
```
plot_gain(df_preds_val)
```



5.14.5 Synthetic Data

[23]:
```
y, X, w, tau, b, e = simulate_hidden_confounder(n=100000, p=5, sigma=1.0, adj=0.)

X_train, X_val, y_train, y_val, w_train, w_val, tau_train, tau_val, b_train, b_val, e_
↪train, e_val = \
    train_test_split(X, y, w, tau, b, e, test_size=0.2, random_state=123,␣
↪shuffle=True)

preds_dict_train = {}
preds_dict_valid = {}

preds_dict_train['Actuals'] = tau_train
preds_dict_valid['Actuals'] = tau_val

preds_dict_train['generated_data'] = {
    'y': y_train,
    'X': X_train,
    'w': w_train,
    'tau': tau_train,
    'b': b_train,
    'e': e_train}
preds_dict_valid['generated_data'] = {
    'y': y_val,
    'X': X_val,
    'w': w_val,
    'tau': tau_val,
    'b': b_val,
    'e': e_val}

# Predict p_hat because e would not be directly observed in real-life
p_model = ElasticNetPropensityModel()
p_hat_train = p_model.fit_predict(X_train, w_train)
p_hat_val = p_model.fit_predict(X_val, w_val)

for base_learner, label_l in zip([BaseSRegressor, BaseTRegressor, BaseXRegressor,␣
↪BaseRRegressor],
                                 ['S', 'T', 'X', 'R']):
    for model, label_m in zip([LinearRegression, XGBRegressor], ['LR', 'XGB']):
        # RLearner will need to fit on the p_hat
        if label_l != 'R':
            learner = base_learner(model())
            # fit the model on training data only
            learner.fit(X=X_train, treatment=w_train, y=y_train)
            try:
                preds_dict_train['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_train, p=p_hat_train).
↪flatten()
                preds_dict_valid['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_val, p=p_hat_val).
↪flatten()
            except TypeError:
                preds_dict_train['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_train, treatment=w_train,
↪ y=y_train).flatten()
                preds_dict_valid['{} Learner ({})'.format(
                    label_l, label_m)] = learner.predict(X=X_val, treatment=w_val,␣
```

(continues on next page)

(continued from previous page)

```
↪y=y_val).flatten()
        else:
            learner = base_learner(model())
            learner.fit(X=X_train, p=p_hat_train, treatment=w_train, y=y_train)
            preds_dict_train['{} Learner ({})'.format(
                label_l, label_m)] = learner.predict(X=X_train).flatten()
            preds_dict_valid['{} Learner ({})'.format(
                label_l, label_m)] = learner.predict(X=X_val).flatten()

# cevae model settings
outcome_dist = "normal"
latent_dim = 20
hidden_dim = 200
num_epochs = 5
batch_size = 1000
learning_rate = 1e-3
learning_rate_decay = 0.1
num_layers = 3
num_samples = 10

cevae = CEVAE(outcome_dist=outcome_dist,
              latent_dim=latent_dim,
              hidden_dim=hidden_dim,
              num_epochs=num_epochs,
              batch_size=batch_size,
              learning_rate=learning_rate,
              learning_rate_decay=learning_rate_decay,
              num_layers=num_layers,
              num_samples=num_samples)

# fit
losses = cevae.fit(X=torch.tensor(X_train, dtype=torch.float),
                   treatment=torch.tensor(w_train, dtype=torch.float),
                   y=torch.tensor(y_train, dtype=torch.float))

preds_dict_train['CEVAE'] = cevae.predict(X_train).flatten()
preds_dict_valid['CEVAE'] = cevae.predict(X_val).flatten()
```

```
INFO     Training with 80 minibatches per epoch
DEBUG    step     0 loss = 14.0534
DEBUG    step     1 loss = 13.2864
DEBUG    step     2 loss = 13.0712
DEBUG    step     3 loss = 12.4646
DEBUG    step     4 loss = 12.0247
DEBUG    step     5 loss = 11.5239
DEBUG    step     6 loss = 11.2934
DEBUG    step     7 loss = 11.3141
DEBUG    step     8 loss = 10.8347
DEBUG    step     9 loss = 10.7364
DEBUG    step    10 loss = 10.5978
DEBUG    step    11 loss = 10.2533
DEBUG    step    12 loss = 10.131
DEBUG    step    13 loss = 10.0307
DEBUG    step    14 loss = 9.57977
DEBUG    step    15 loss = 9.79295
DEBUG    step    16 loss = 9.46927
DEBUG    step    17 loss = 9.57581
```

(continues on next page)

(continued from previous page)

```
DEBUG    step    18 loss = 9.24119
DEBUG    step    19 loss = 9.34084
DEBUG    step    20 loss = 9.32529
DEBUG    step    21 loss = 9.40313
DEBUG    step    22 loss = 9.27057
DEBUG    step    23 loss = 9.05239
DEBUG    step    24 loss = 9.17952
DEBUG    step    25 loss = 8.93083
DEBUG    step    26 loss = 8.88059
DEBUG    step    27 loss = 9.06328
DEBUG    step    28 loss = 8.97881
DEBUG    step    29 loss = 8.7639
DEBUG    step    30 loss = 8.80499
DEBUG    step    31 loss = 8.87173
DEBUG    step    32 loss = 8.56747
DEBUG    step    33 loss = 8.61066
DEBUG    step    34 loss = 8.79932
DEBUG    step    35 loss = 8.62871
DEBUG    step    36 loss = 8.54852
DEBUG    step    37 loss = 8.38022
DEBUG    step    38 loss = 8.31573
DEBUG    step    39 loss = 8.53857
DEBUG    step    40 loss = 8.57149
DEBUG    step    41 loss = 8.25793
DEBUG    step    42 loss = 8.54684
DEBUG    step    43 loss = 8.47699
DEBUG    step    44 loss = 8.3233
DEBUG    step    45 loss = 8.40228
DEBUG    step    46 loss = 8.14949
DEBUG    step    47 loss = 8.2015
DEBUG    step    48 loss = 8.07472
DEBUG    step    49 loss = 8.16795
DEBUG    step    50 loss = 8.34108
DEBUG    step    51 loss = 8.57682
DEBUG    step    52 loss = 8.24426
DEBUG    step    53 loss = 8.33251
DEBUG    step    54 loss = 8.10115
DEBUG    step    55 loss = 8.67902
DEBUG    step    56 loss = 8.14677
DEBUG    step    57 loss = 8.1041
DEBUG    step    58 loss = 8.15102
DEBUG    step    59 loss = 8.00679
DEBUG    step    60 loss = 8.0271
DEBUG    step    61 loss = 7.96041
DEBUG    step    62 loss = 7.82294
DEBUG    step    63 loss = 8.13456
DEBUG    step    64 loss = 8.23367
DEBUG    step    65 loss = 8.1886
DEBUG    step    66 loss = 8.11654
DEBUG    step    67 loss = 8.22645
DEBUG    step    68 loss = 8.29743
DEBUG    step    69 loss = 8.24127
DEBUG    step    70 loss = 7.86166
DEBUG    step    71 loss = 8.22115
DEBUG    step    72 loss = 7.8913
DEBUG    step    73 loss = 7.96265
DEBUG    step    74 loss = 7.96243
```

(continues on next page)

(continued from previous page)

```
DEBUG    step    75 loss = 7.99336
DEBUG    step    76 loss = 7.97742
DEBUG    step    77 loss = 7.90728
DEBUG    step    78 loss = 7.79539
DEBUG    step    79 loss = 8.1732
DEBUG    step    80 loss = 8.05217
DEBUG    step    81 loss = 8.34642
DEBUG    step    82 loss = 8.03199
DEBUG    step    83 loss = 7.64226
DEBUG    step    84 loss = 7.60438
DEBUG    step    85 loss = 7.5962
DEBUG    step    86 loss = 7.85927
DEBUG    step    87 loss = 7.98567
DEBUG    step    88 loss = 7.82793
DEBUG    step    89 loss = 7.90716
DEBUG    step    90 loss = 7.71277
DEBUG    step    91 loss = 7.97724
DEBUG    step    92 loss = 7.84886
DEBUG    step    93 loss = 7.88323
DEBUG    step    94 loss = 7.58179
DEBUG    step    95 loss = 7.89912
DEBUG    step    96 loss = 7.67735
DEBUG    step    97 loss = 7.84808
DEBUG    step    98 loss = 7.66705
DEBUG    step    99 loss = 7.65615
DEBUG    step   100 loss = 7.73811
DEBUG    step   101 loss = 7.64997
DEBUG    step   102 loss = 8.36613
DEBUG    step   103 loss = 7.72687
DEBUG    step   104 loss = 7.68498
DEBUG    step   105 loss = 7.50849
DEBUG    step   106 loss = 7.63987
DEBUG    step   107 loss = 7.75501
DEBUG    step   108 loss = 7.62423
DEBUG    step   109 loss = 7.66921
DEBUG    step   110 loss = 7.50166
DEBUG    step   111 loss = 7.62314
DEBUG    step   112 loss = 7.80907
DEBUG    step   113 loss = 7.65659
DEBUG    step   114 loss = 7.55159
DEBUG    step   115 loss = 7.60577
DEBUG    step   116 loss = 7.36759
DEBUG    step   117 loss = 7.43037
DEBUG    step   118 loss = 7.41372
DEBUG    step   119 loss = 7.58245
DEBUG    step   120 loss = 7.75382
DEBUG    step   121 loss = 7.75345
DEBUG    step   122 loss = 7.71091
DEBUG    step   123 loss = 7.61762
DEBUG    step   124 loss = 7.5415
DEBUG    step   125 loss = 7.70995
DEBUG    step   126 loss = 7.43083
DEBUG    step   127 loss = 7.62284
DEBUG    step   128 loss = 7.57494
DEBUG    step   129 loss = 7.43229
DEBUG    step   130 loss = 7.417
DEBUG    step   131 loss = 7.36716
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   132 loss = 7.58527
DEBUG    step   133 loss = 7.61684
DEBUG    step   134 loss = 7.55247
DEBUG    step   135 loss = 7.54181
DEBUG    step   136 loss = 7.47493
DEBUG    step   137 loss = 7.65583
DEBUG    step   138 loss = 7.33769
DEBUG    step   139 loss = 7.36649
DEBUG    step   140 loss = 7.3634
DEBUG    step   141 loss = 7.50731
DEBUG    step   142 loss = 7.60657
DEBUG    step   143 loss = 7.38694
DEBUG    step   144 loss = 7.3596
DEBUG    step   145 loss = 7.42744
DEBUG    step   146 loss = 7.46609
DEBUG    step   147 loss = 7.44444
DEBUG    step   148 loss = 7.44656
DEBUG    step   149 loss = 7.32834
DEBUG    step   150 loss = 7.63049
DEBUG    step   151 loss = 7.43903
DEBUG    step   152 loss = 7.28372
DEBUG    step   153 loss = 7.28897
DEBUG    step   154 loss = 7.3515
DEBUG    step   155 loss = 7.29871
DEBUG    step   156 loss = 7.47948
DEBUG    step   157 loss = 7.56888
DEBUG    step   158 loss = 7.50302
DEBUG    step   159 loss = 7.14918
DEBUG    step   160 loss = 7.34611
DEBUG    step   161 loss = 7.04855
DEBUG    step   162 loss = 7.38615
DEBUG    step   163 loss = 7.39172
DEBUG    step   164 loss = 7.35778
DEBUG    step   165 loss = 7.39445
DEBUG    step   166 loss = 7.41489
DEBUG    step   167 loss = 7.36096
DEBUG    step   168 loss = 7.49107
DEBUG    step   169 loss = 7.31799
DEBUG    step   170 loss = 7.34851
DEBUG    step   171 loss = 7.17355
DEBUG    step   172 loss = 7.38851
DEBUG    step   173 loss = 7.35425
DEBUG    step   174 loss = 7.39068
DEBUG    step   175 loss = 7.08015
DEBUG    step   176 loss = 7.05245
DEBUG    step   177 loss = 7.43696
DEBUG    step   178 loss = 7.32325
DEBUG    step   179 loss = 7.31021
DEBUG    step   180 loss = 7.32132
DEBUG    step   181 loss = 7.34862
DEBUG    step   182 loss = 7.2863
DEBUG    step   183 loss = 7.04851
DEBUG    step   184 loss = 7.09608
DEBUG    step   185 loss = 7.30419
DEBUG    step   186 loss = 7.57377
DEBUG    step   187 loss = 7.17361
DEBUG    step   188 loss = 7.14099
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   189 loss = 7.0449
DEBUG    step   190 loss = 7.33529
DEBUG    step   191 loss = 8.26479
DEBUG    step   192 loss = 7.07407
DEBUG    step   193 loss = 7.17149
DEBUG    step   194 loss = 7.18364
DEBUG    step   195 loss = 7.27539
DEBUG    step   196 loss = 7.32838
DEBUG    step   197 loss = 7.26303
DEBUG    step   198 loss = 7.17846
DEBUG    step   199 loss = 7.43274
DEBUG    step   200 loss = 7.05834
DEBUG    step   201 loss = 7.06987
DEBUG    step   202 loss = 7.23815
DEBUG    step   203 loss = 7.2454
DEBUG    step   204 loss = 7.29509
DEBUG    step   205 loss = 7.13663
DEBUG    step   206 loss = 6.96725
DEBUG    step   207 loss = 7.11374
DEBUG    step   208 loss = 6.93604
DEBUG    step   209 loss = 7.14596
DEBUG    step   210 loss = 7.12832
DEBUG    step   211 loss = 7.16911
DEBUG    step   212 loss = 6.9426
DEBUG    step   213 loss = 7.18095
DEBUG    step   214 loss = 7.06178
DEBUG    step   215 loss = 7.10941
DEBUG    step   216 loss = 7.11186
DEBUG    step   217 loss = 7.20186
DEBUG    step   218 loss = 7.27586
DEBUG    step   219 loss = 7.1021
DEBUG    step   220 loss = 6.94478
DEBUG    step   221 loss = 7.09795
DEBUG    step   222 loss = 6.88571
DEBUG    step   223 loss = 7.03089
DEBUG    step   224 loss = 7.23866
DEBUG    step   225 loss = 7.10442
DEBUG    step   226 loss = 6.95982
DEBUG    step   227 loss = 8.71509
DEBUG    step   228 loss = 6.93005
DEBUG    step   229 loss = 7.2101
DEBUG    step   230 loss = 7.23326
DEBUG    step   231 loss = 6.94798
DEBUG    step   232 loss = 6.83511
DEBUG    step   233 loss = 6.99621
DEBUG    step   234 loss = 6.79696
DEBUG    step   235 loss = 7.21458
DEBUG    step   236 loss = 6.97841
DEBUG    step   237 loss = 7.12467
DEBUG    step   238 loss = 6.98927
DEBUG    step   239 loss = 7.13294
DEBUG    step   240 loss = 7.17033
DEBUG    step   241 loss = 7.09788
DEBUG    step   242 loss = 6.98868
DEBUG    step   243 loss = 7.0711
DEBUG    step   244 loss = 7.10628
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   245 loss = 7.12893
DEBUG    step   246 loss = 6.94537
DEBUG    step   247 loss = 6.98222
DEBUG    step   248 loss = 7.12801
DEBUG    step   249 loss = 6.94684
DEBUG    step   250 loss = 7.01901
DEBUG    step   251 loss = 7.03228
DEBUG    step   252 loss = 7.14612
DEBUG    step   253 loss = 7.04241
DEBUG    step   254 loss = 6.92232
DEBUG    step   255 loss = 7.02093
DEBUG    step   256 loss = 6.98689
DEBUG    step   257 loss = 6.97682
DEBUG    step   258 loss = 6.99232
DEBUG    step   259 loss = 7.01528
DEBUG    step   260 loss = 6.86835
DEBUG    step   261 loss = 7.00633
DEBUG    step   262 loss = 7.06246
DEBUG    step   263 loss = 6.90189
DEBUG    step   264 loss = 7.07629
DEBUG    step   265 loss = 6.88559
DEBUG    step   266 loss = 6.92606
DEBUG    step   267 loss = 6.8929
DEBUG    step   268 loss = 6.83142
DEBUG    step   269 loss = 6.73955
DEBUG    step   270 loss = 6.81085
DEBUG    step   271 loss = 6.87084
DEBUG    step   272 loss = 6.88125
DEBUG    step   273 loss = 6.94562
DEBUG    step   274 loss = 6.9711
DEBUG    step   275 loss = 7.01001
DEBUG    step   276 loss = 6.91986
DEBUG    step   277 loss = 6.92239
DEBUG    step   278 loss = 6.70706
DEBUG    step   279 loss = 6.84017
DEBUG    step   280 loss = 7.09178
DEBUG    step   281 loss = 6.7313
DEBUG    step   282 loss = 6.79816
DEBUG    step   283 loss = 6.86953
DEBUG    step   284 loss = 6.92598
DEBUG    step   285 loss = 7.0731
DEBUG    step   286 loss = 6.91421
DEBUG    step   287 loss = 6.76945
DEBUG    step   288 loss = 6.74834
DEBUG    step   289 loss = 6.84824
DEBUG    step   290 loss = 6.88344
DEBUG    step   291 loss = 6.85244
DEBUG    step   292 loss = 6.922
DEBUG    step   293 loss = 9.57555
DEBUG    step   294 loss = 6.83098
DEBUG    step   295 loss = 7.43121
DEBUG    step   296 loss = 6.95061
DEBUG    step   297 loss = 6.79967
DEBUG    step   298 loss = 6.7929
DEBUG    step   299 loss = 6.7355
DEBUG    step   300 loss = 7.01345
DEBUG    step   301 loss = 6.83328
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   302 loss = 6.62454
DEBUG    step   303 loss = 6.84473
DEBUG    step   304 loss = 9.05065
DEBUG    step   305 loss = 7.038
DEBUG    step   306 loss = 6.60419
DEBUG    step   307 loss = 6.80575
DEBUG    step   308 loss = 6.73912
DEBUG    step   309 loss = 6.47463
DEBUG    step   310 loss = 6.84484
DEBUG    step   311 loss = 6.73429
DEBUG    step   312 loss = 6.89219
DEBUG    step   313 loss = 7.05905
DEBUG    step   314 loss = 6.82365
DEBUG    step   315 loss = 6.72354
DEBUG    step   316 loss = 6.54532
DEBUG    step   317 loss = 6.95339
DEBUG    step   318 loss = 7.0503
DEBUG    step   319 loss = 6.78209
DEBUG    step   320 loss = 6.59514
DEBUG    step   321 loss = 6.89779
DEBUG    step   322 loss = 6.72151
DEBUG    step   323 loss = 6.90015
DEBUG    step   324 loss = 7.00599
DEBUG    step   325 loss = 6.85437
DEBUG    step   326 loss = 6.89033
DEBUG    step   327 loss = 6.7871
DEBUG    step   328 loss = 6.8493
DEBUG    step   329 loss = 6.80922
DEBUG    step   330 loss = 6.96322
DEBUG    step   331 loss = 6.84506
DEBUG    step   332 loss = 6.87015
DEBUG    step   333 loss = 6.88979
DEBUG    step   334 loss = 6.64982
DEBUG    step   335 loss = 6.86292
DEBUG    step   336 loss = 6.92489
DEBUG    step   337 loss = 6.62396
DEBUG    step   338 loss = 6.84564
DEBUG    step   339 loss = 6.62305
DEBUG    step   340 loss = 7.36375
DEBUG    step   341 loss = 6.73599
DEBUG    step   342 loss = 6.80353
DEBUG    step   343 loss = 6.96371
DEBUG    step   344 loss = 6.89915
DEBUG    step   345 loss = 6.64238
DEBUG    step   346 loss = 6.51934
DEBUG    step   347 loss = 6.78445
DEBUG    step   348 loss = 6.94965
DEBUG    step   349 loss = 6.78796
DEBUG    step   350 loss = 6.77106
DEBUG    step   351 loss = 6.7466
DEBUG    step   352 loss = 6.77313
DEBUG    step   353 loss = 6.70463
DEBUG    step   354 loss = 6.96683
DEBUG    step   355 loss = 6.73415
DEBUG    step   356 loss = 6.73694
DEBUG    step   357 loss = 6.60738
DEBUG    step   358 loss = 9.84151
```

(continues on next page)

(continued from previous page)

```
DEBUG    step   359 loss = 6.84548
DEBUG    step   360 loss = 6.57425
DEBUG    step   361 loss = 6.78442
DEBUG    step   362 loss = 6.68523
DEBUG    step   363 loss = 6.93113
DEBUG    step   364 loss = 9.26669
DEBUG    step   365 loss = 6.71749
DEBUG    step   366 loss = 6.60656
DEBUG    step   367 loss = 6.7795
DEBUG    step   368 loss = 6.55477
DEBUG    step   369 loss = 6.73777
DEBUG    step   370 loss = 6.80791
DEBUG    step   371 loss = 6.75802
DEBUG    step   372 loss = 6.80779
DEBUG    step   373 loss = 6.82983
DEBUG    step   374 loss = 6.5821
DEBUG    step   375 loss = 6.81309
DEBUG    step   376 loss = 6.58409
DEBUG    step   377 loss = 6.59094
DEBUG    step   378 loss = 6.59232
DEBUG    step   379 loss = 7.0035
DEBUG    step   380 loss = 6.65775
DEBUG    step   381 loss = 6.61621
DEBUG    step   382 loss = 6.6329
DEBUG    step   383 loss = 6.63025
DEBUG    step   384 loss = 6.61858
DEBUG    step   385 loss = 6.63814
DEBUG    step   386 loss = 6.50298
DEBUG    step   387 loss = 6.62591
DEBUG    step   388 loss = 6.56514
DEBUG    step   389 loss = 6.67944
DEBUG    step   390 loss = 6.80612
DEBUG    step   391 loss = 6.61369
DEBUG    step   392 loss = 6.85104
DEBUG    step   393 loss = 6.61612
DEBUG    step   394 loss = 6.55337
DEBUG    step   395 loss = 6.76919
DEBUG    step   396 loss = 6.66491
DEBUG    step   397 loss = 6.57224
DEBUG    step   398 loss = 6.54065
DEBUG    step   399 loss = 6.73794
INFO     Evaluating 80 minibatches
DEBUG    batch ate = 0.823513
DEBUG    batch ate = 0.824189
DEBUG    batch ate = 0.820978
DEBUG    batch ate = 0.822631
DEBUG    batch ate = 0.823555
DEBUG    batch ate = 0.822441
DEBUG    batch ate = 0.823683
DEBUG    batch ate = 0.822339
DEBUG    batch ate = 0.823964
DEBUG    batch ate = 0.823921
DEBUG    batch ate = 0.825266
DEBUG    batch ate = 0.822931
DEBUG    batch ate = 0.823049
DEBUG    batch ate = 0.824161
DEBUG    batch ate = 0.821918
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.824303
DEBUG    batch ate = 0.823845
DEBUG    batch ate = 0.822578
DEBUG    batch ate = 0.825122
DEBUG    batch ate = 0.823321
DEBUG    batch ate = 0.823198
DEBUG    batch ate = 0.823159
DEBUG    batch ate = 0.823571
DEBUG    batch ate = 0.822972
DEBUG    batch ate = 0.82311
DEBUG    batch ate = 0.821233
DEBUG    batch ate = 0.824326
DEBUG    batch ate = 0.823645
DEBUG    batch ate = 0.8233
DEBUG    batch ate = 0.821567
DEBUG    batch ate = 0.820404
DEBUG    batch ate = 0.821521
DEBUG    batch ate = 0.82027
DEBUG    batch ate = 0.824084
DEBUG    batch ate = 0.824593
DEBUG    batch ate = 0.823614
DEBUG    batch ate = 0.820698
DEBUG    batch ate = 0.824454
DEBUG    batch ate = 0.819246
DEBUG    batch ate = 0.823614
DEBUG    batch ate = 0.822471
DEBUG    batch ate = 0.822809
DEBUG    batch ate = 0.82155
DEBUG    batch ate = 0.822985
DEBUG    batch ate = 0.821966
DEBUG    batch ate = 0.822152
DEBUG    batch ate = 0.824818
DEBUG    batch ate = 0.821926
DEBUG    batch ate = 0.821183
DEBUG    batch ate = 0.821644
DEBUG    batch ate = 0.823652
DEBUG    batch ate = 0.822925
DEBUG    batch ate = 0.822612
DEBUG    batch ate = 0.824216
DEBUG    batch ate = 0.824456
DEBUG    batch ate = 0.822995
DEBUG    batch ate = 0.823972
DEBUG    batch ate = 0.821021
DEBUG    batch ate = 0.822201
DEBUG    batch ate = 0.821493
DEBUG    batch ate = 0.823859
DEBUG    batch ate = 0.819778
DEBUG    batch ate = 0.822789
DEBUG    batch ate = 0.825457
DEBUG    batch ate = 0.824181
DEBUG    batch ate = 0.821647
DEBUG    batch ate = 0.82509
DEBUG    batch ate = 0.821287
DEBUG    batch ate = 0.824007
DEBUG    batch ate = 0.821076
DEBUG    batch ate = 0.823777
DEBUG    batch ate = 0.822884
```

(continues on next page)

(continued from previous page)

```
DEBUG    batch ate = 0.824057
DEBUG    batch ate = 0.820844
DEBUG    batch ate = 0.821426
DEBUG    batch ate = 0.82413
DEBUG    batch ate = 0.822516
DEBUG    batch ate = 0.823242
DEBUG    batch ate = 0.820823
DEBUG    batch ate = 0.822049
INFO     Evaluating 20 minibatches
DEBUG    batch ate = 0.823355
DEBUG    batch ate = 0.826493
DEBUG    batch ate = 0.825423
DEBUG    batch ate = 0.825241
DEBUG    batch ate = 0.823623
DEBUG    batch ate = 0.823627
DEBUG    batch ate = 0.821589
DEBUG    batch ate = 0.824463
DEBUG    batch ate = 0.821071
DEBUG    batch ate = 0.820596
DEBUG    batch ate = 0.823198
DEBUG    batch ate = 0.820816
DEBUG    batch ate = 0.823484
DEBUG    batch ate = 0.823282
DEBUG    batch ate = 0.825439
```

```
DEBUG    batch ate = 0.822407
DEBUG    batch ate = 0.822365
DEBUG    batch ate = 0.825534
DEBUG    batch ate = 0.822151
DEBUG    batch ate = 0.823306
```

[24]:
```
actuals_train = preds_dict_train['Actuals']
actuals_validation = preds_dict_valid['Actuals']

synthetic_summary_train = pd.DataFrame({label: [preds.mean(), mse(preds, actuals_
↪train)] for label, preds
                                        in preds_dict_train.items() if 'generated'␣
↪not in label.lower()},
                                       index=['ATE', 'MSE']).T
synthetic_summary_train['Abs % Error of ATE'] = np.abs(
    (synthetic_summary_train['ATE']/synthetic_summary_train.loc['Actuals', 'ATE']) -␣
↪1)

synthetic_summary_validation = pd.DataFrame({label: [preds.mean(), mse(preds, actuals_
↪validation)]
                                             for label, preds in preds_dict_valid.
↪items()
                                             if 'generated' not in label.lower()},
                                            index=['ATE', 'MSE']).T
synthetic_summary_validation['Abs % Error of ATE'] = np.abs(
    (synthetic_summary_validation['ATE']/synthetic_summary_validation.loc['Actuals',
↪'ATE']) - 1)

# calculate kl divergence for training
for label in synthetic_summary_train.index:
    stacked_values = np.hstack((preds_dict_train[label], actuals_train))
    stacked_low = np.percentile(stacked_values, 0.1)
```

(continues on next page)

(continued from previous page)

```
    stacked_high = np.percentile(stacked_values, 99.9)
    bins = np.linspace(stacked_low, stacked_high, 100)

    distr = np.histogram(preds_dict_train[label], bins=bins)[0]
    distr = np.clip(distr/distr.sum(), 0.001, 0.999)
    true_distr = np.histogram(actuals_train, bins=bins)[0]
    true_distr = np.clip(true_distr/true_distr.sum(), 0.001, 0.999)

    kl = entropy(distr, true_distr)
    synthetic_summary_train.loc[label, 'KL Divergence'] = kl

# calculate kl divergence for validation
for label in synthetic_summary_validation.index:
    stacked_values = np.hstack((preds_dict_valid[label], actuals_validation))
    stacked_low = np.percentile(stacked_values, 0.1)
    stacked_high = np.percentile(stacked_values, 99.9)
    bins = np.linspace(stacked_low, stacked_high, 100)

    distr = np.histogram(preds_dict_valid[label], bins=bins)[0]
    distr = np.clip(distr/distr.sum(), 0.001, 0.999)
    true_distr = np.histogram(actuals_validation, bins=bins)[0]
    true_distr = np.clip(true_distr/true_distr.sum(), 0.001, 0.999)

    kl = entropy(distr, true_distr)
    synthetic_summary_validation.loc[label, 'KL Divergence'] = kl
```

[25]:
```
df_preds_train = pd.DataFrame([preds_dict_train['S Learner (LR)'].ravel(),
                               preds_dict_train['S Learner (XGB)'].ravel(),
                               preds_dict_train['T Learner (LR)'].ravel(),
                               preds_dict_train['T Learner (XGB)'].ravel(),
                               preds_dict_train['X Learner (LR)'].ravel(),
                               preds_dict_train['X Learner (XGB)'].ravel(),
                               preds_dict_train['R Learner (LR)'].ravel(),
                               preds_dict_train['R Learner (XGB)'].ravel(),
                               preds_dict_train['CEVAE'].ravel(),
                               preds_dict_train['generated_data']['tau'].ravel(),
                               preds_dict_train['generated_data']['w'].ravel(),
                               preds_dict_train['generated_data']['y'].ravel()],
                              index=['S Learner (LR)','S Learner (XGB)',
                                     'T Learner (LR)','T Learner (XGB)',
                                     'X Learner (LR)','X Learner (XGB)',
                                     'R Learner (LR)','R Learner (XGB)',
                                     'CEVAE','tau','w','y']).T

synthetic_summary_train['AUUC'] = auuc_score(df_preds_train).iloc[:-1]
```

[26]:
```
df_preds_validation = pd.DataFrame([preds_dict_valid['S Learner (LR)'].ravel(),
                                    preds_dict_valid['S Learner (XGB)'].ravel(),
                                    preds_dict_valid['T Learner (LR)'].ravel(),
                                    preds_dict_valid['T Learner (XGB)'].ravel(),
                                    preds_dict_valid['X Learner (LR)'].ravel(),
                                    preds_dict_valid['X Learner (XGB)'].ravel(),
                                    preds_dict_valid['R Learner (LR)'].ravel(),
                                    preds_dict_valid['R Learner (XGB)'].ravel(),
                                    preds_dict_valid['CEVAE'].ravel(),
                                    preds_dict_valid['generated_data']['tau'].ravel(),
```

(continues on next page)

(continued from previous page)

```
                                preds_dict_valid['generated_data']['w'].ravel(),
                                preds_dict_valid['generated_data']['y'].ravel()],
                               index=['S Learner (LR)','S Learner (XGB)',
                                      'T Learner (LR)','T Learner (XGB)',
                                      'X Learner (LR)','X Learner (XGB)',
                                      'R Learner (LR)','R Learner (XGB)',
                                      'CEVAE','tau','w','y']).T

synthetic_summary_validation['AUUC'] = auuc_score(df_preds_validation).iloc[:-1]
```

[27]:
```
synthetic_summary_train
```

[27]:

	ATE	MSE	Abs % Error of ATE	KL Divergence	\
Actuals	0.726115	0.000000	0.000000	0.000000	
S Learner (LR)	0.832336	0.062462	0.146287	6.278413	
S Learner (XGB)	0.807743	0.039735	0.112417	2.551297	
T Learner (LR)	0.833364	0.059665	0.147703	3.312696	
T Learner (XGB)	0.803592	0.040524	0.106701	2.565715	
X Learner (LR)	0.833364	0.059665	0.147703	3.312696	
X Learner (XGB)	0.803349	0.038580	0.106367	2.500947	
R Learner (LR)	0.833845	0.060239	0.148365	3.511157	
R Learner (XGB)	0.735442	0.046848	0.012845	2.836128	
CEVAE	0.822853	0.058177	0.133227	3.157059	

	AUUC
Actuals	NaN
S Learner (LR)	0.499991
S Learner (XGB)	0.554885
T Learner (LR)	0.523272
T Learner (XGB)	0.553197
X Learner (LR)	0.523272
X Learner (XGB)	0.555391
R Learner (LR)	0.523214
R Learner (XGB)	0.539213
CEVAE	0.519150

[28]:
```
synthetic_summary_validation
```

[28]:

	ATE	MSE	Abs % Error of ATE	KL Divergence	\
Actuals	0.728371	0.000000	0.000000	0.000000	
S Learner (LR)	0.832336	0.061983	0.142736	6.278413	
S Learner (XGB)	0.808844	0.040638	0.110483	2.548714	
T Learner (LR)	0.833805	0.059305	0.144753	3.316884	
T Learner (XGB)	0.803766	0.042424	0.103512	2.561688	
X Learner (LR)	0.833805	0.059305	0.144753	3.316884	
X Learner (XGB)	0.803530	0.039699	0.103187	2.489822	
R Learner (LR)	0.834179	0.059851	0.145266	3.512746	
R Learner (XGB)	0.736147	0.046685	0.010675	2.747596	
CEVAE	0.823373	0.057690	0.130430	3.152161	

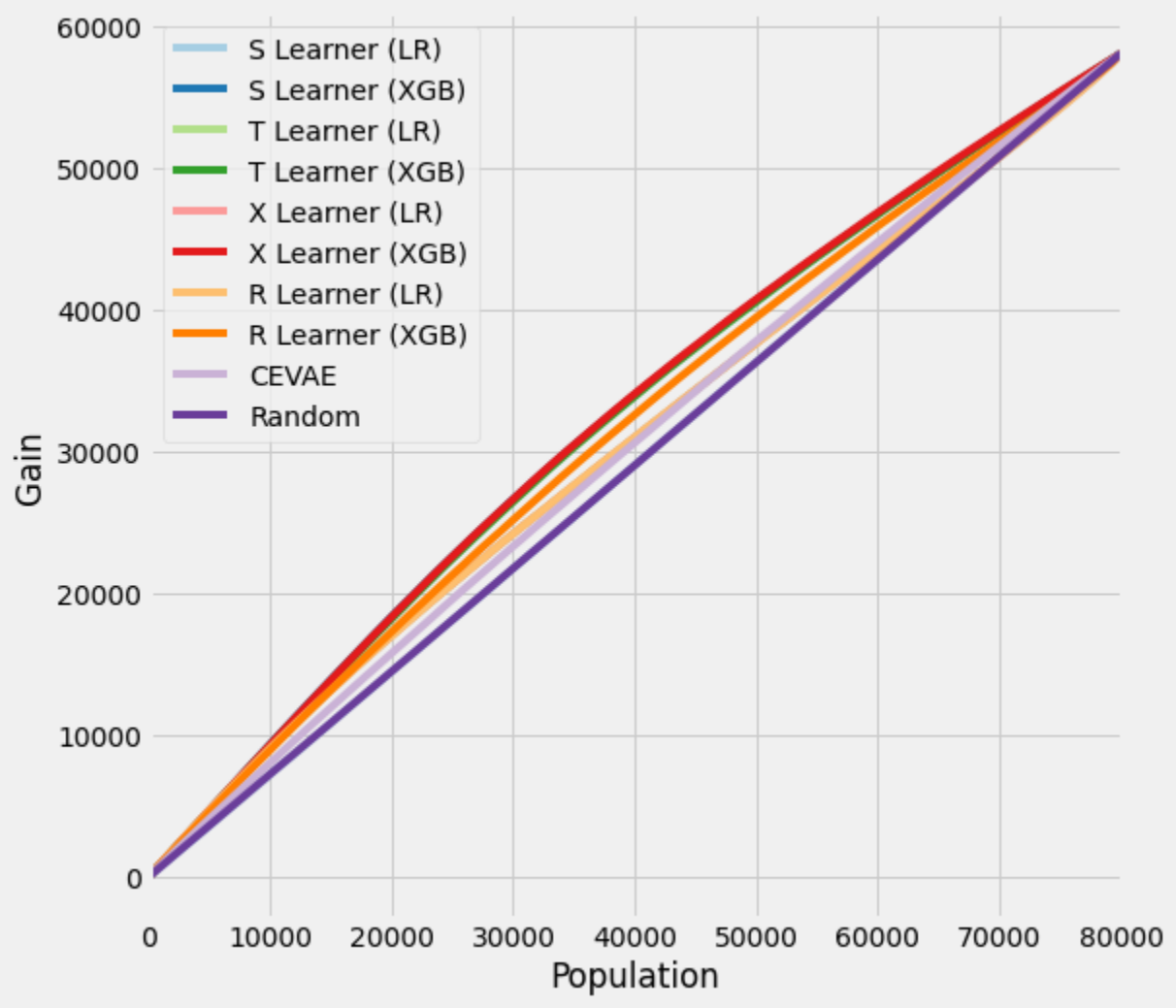
	AUUC
Actuals	NaN
S Learner (LR)	0.499967
S Learner (XGB)	0.553011
T Learner (LR)	0.522972
T Learner (XGB)	0.549279
X Learner (LR)	0.522972

(continues on next page)

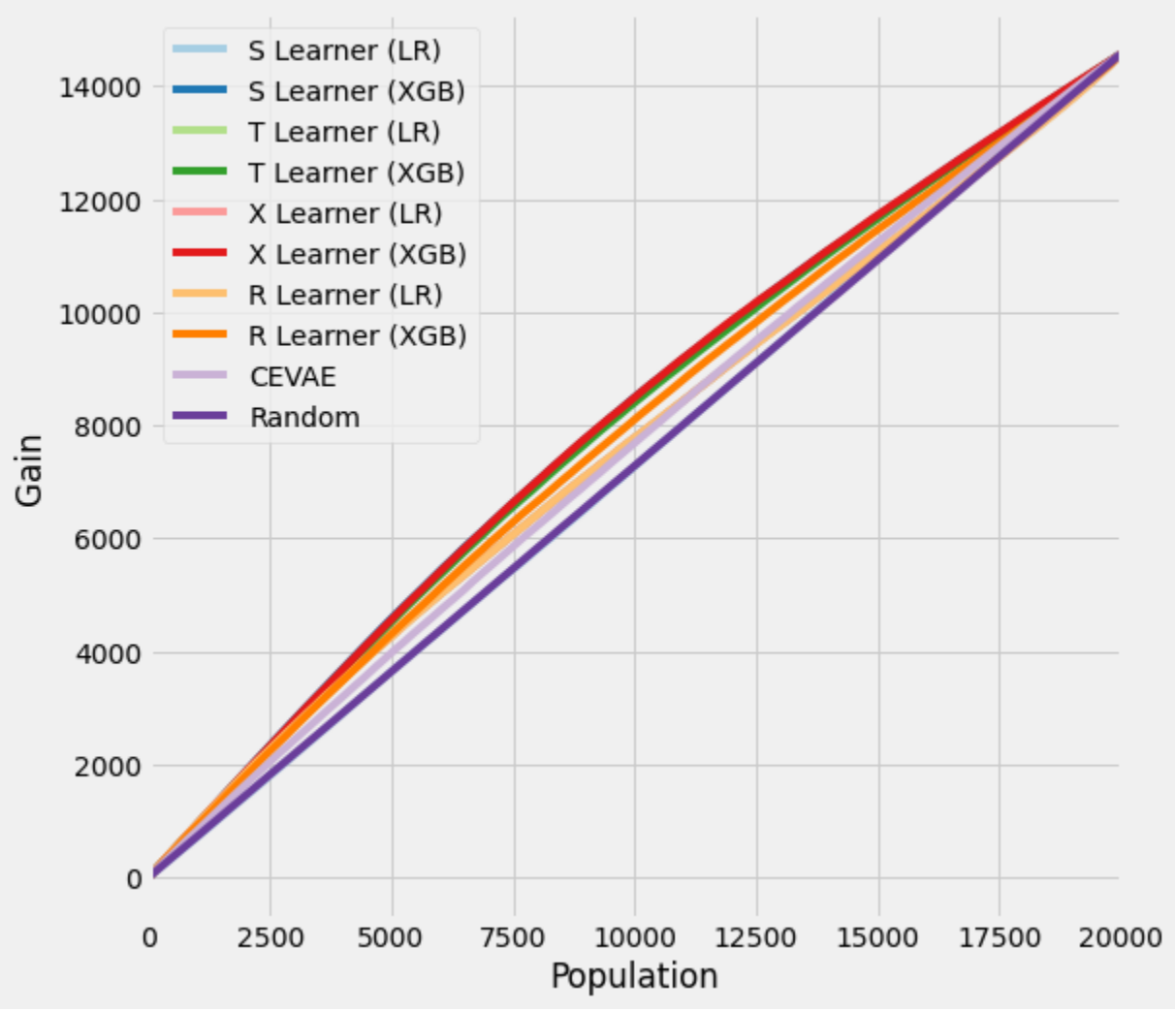
(continued from previous page)

```
X Learner (XGB)   0.553039
R Learner (LR)    0.522887
R Learner (XGB)   0.536579
CEVAE             0.519573
```

[29]:
```
plot_gain(df_preds_train)
```



[30]:
```
plot_gain(df_preds_validation)
```



5.15 DR Learner vs. DR-IV Learner vs. X-Learner Benchmark with Synthetic Data

This notebook demonstrates the use of the CausalML implemented DR Learner by Kennedy (2020) (https://arxiv.org/abs/2004.14497) for the Individual Treatment Effect (ITE) estimation.

[2]:
```
%load_ext autoreload
%autoreload 2
```

[3]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from xgboost import XGBRegressor
import warnings
```

(continues on next page)

(continued from previous page)

```
from causalml.inference.meta import BaseXRegressor, BaseDRRegressor
from causalml.inference.iv import BaseDRIVRegressor
from causalml.dataset import synthetic_data
from causalml.metrics import *

warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')

%matplotlib inline
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
RuntimeError: module compiled against API version 0xe but this version of numpy is 0xd
```

```
The sklearn.utils.testing module is  deprecated in version 0.22 and will be removed␣
↪in version 0.24. The corresponding classes / functions should instead be imported␣
↪from sklearn.utils. Anything that cannot be imported from sklearn.utils is now part␣
↪of the private API.
```

5.15.1 Synthetic Data Generation

[4]:
```
y, X, treatment, tau, b, e = synthetic_data(mode=1, n=10000, p=8, sigma=1.0)
```

5.15.2 Comparing DR Learner with X Learner

We use a flexible ML estimator to estimate the outcome model but a simple linear regression model to estimate the ITE, since the ITE estimate is often noisy and prone to overfit with a flexible estimator.

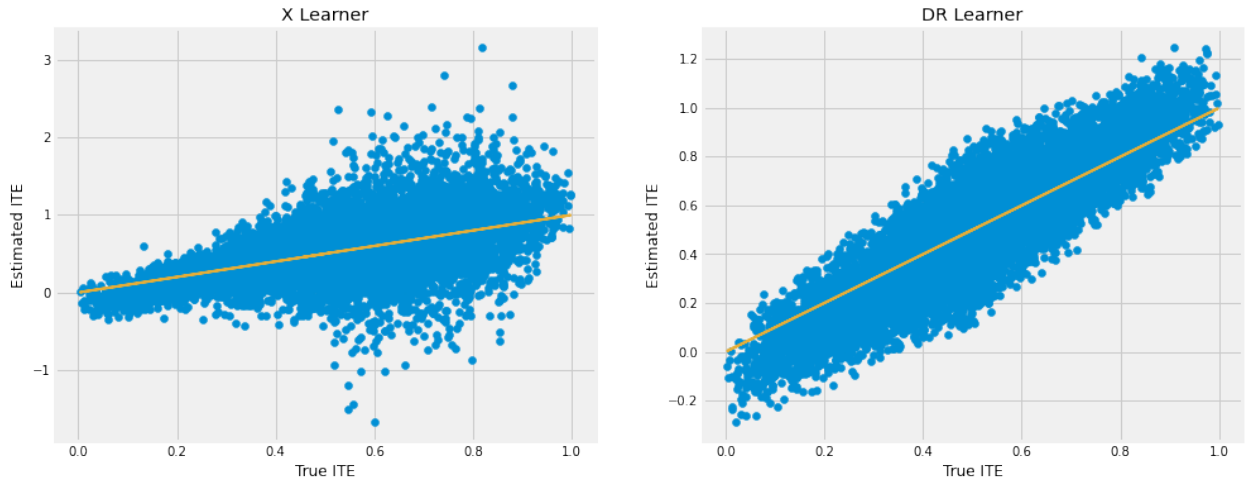
[5]:
```
learner_x = BaseXRegressor(learner=XGBRegressor(), treatment_effect_
↪learner=LinearRegression())
cate_x = learner_x.fit_predict(X=X, treatment=treatment, y=y)
```

[6]:
```
learner_dr = BaseDRRegressor(learner=XGBRegressor(), treatment_effect_
↪learner=LinearRegression())
cate_dr = learner_dr.fit_predict(X=X, treatment=treatment, y=y)
```

DR Learner outforms X Learner in this dataset. Even with built-in mechanism to counteract the unbalancedness between the treatment and control samples, X Learner still suffers from the regime where the treatment probability is close to 1 in this case.

[7]:
```
fig, ax = plt.subplots(1, 2, figsize=(15, 6))
ax[0].scatter(tau, cate_x)
ax[0].plot(tau, tau, color='C2', linewidth=2)
ax[0].set_xlabel('True ITE')
ax[0].set_ylabel('Estimated ITE')
ax[0].set_title('X Learner')
ax[1].scatter(tau, cate_dr)
ax[1].plot(tau, tau, color='C2', linewidth=2)
ax[1].set_xlabel('True ITE')
ax[1].set_ylabel('Estimated ITE')
ax[1].set_title('DR Learner')
```

[7]:
```
Text(0.5, 1.0, 'DR Learner')
```



5.15.3 Synthetic Data with Hidden Confounder

Now we tweaked the previous synthetic data generation by the following 2 changes - Adding a random assignment mechanism. Only assigned units may potentially have a treatment, though whether a unit gets treatment in the assigned group depends on its confounding variables. Therefore this is a situation of one-sided non-compliance. - One of the confounding variables that affects both the propensity to receive treatment and the treatment effect is not observed by analyst. Therefore it is a problem of hidden confounder.

[8]:
```
n = 10000
p = 8
sigma = 1.0

X = np.random.uniform(size=n*p).reshape((n, -1))
b = np.sin(np.pi * X[:, 0] * X[:, 1]) + 2 * (X[:, 2] - 0.5) ** 2 + X[:, 3] + 0.5 * X[:
↪, 4]
assignment = (np.random.uniform(size=10000)>0.5).astype(int)
eta = 0.1
e = np.maximum(np.repeat(eta, n), np.minimum(np.sin(np.pi * X[:, 0] * X[:, 1]), np.
↪repeat(1-eta, n)))
e[assignment == 0] = 0
tau = (X[:, 0] + X[:, 1]) / 2
X_obs = X[:, [i for i in range(8) if i!=1]]

w = np.random.binomial(1, e, size=n)
treatment = w
y = b + (w - 0.5) * tau + sigma * np.random.normal(size=n)
```

5.15.4 Comparing X Learner, DR Learner, and DRIV Learner

We use 3 learners, X Learner, DR Learner, and DRIV Learner, to estimate the ITE of the compliers, i.e. those who only receive treatment when they are assigned.

[9]:
```
learner_x = BaseXRegressor(learner=XGBRegressor(), treatment_effect_
↪learner=LinearRegression())
cate_x = learner_x.fit_predict(X=X_obs, treatment=treatment, y=y)
```

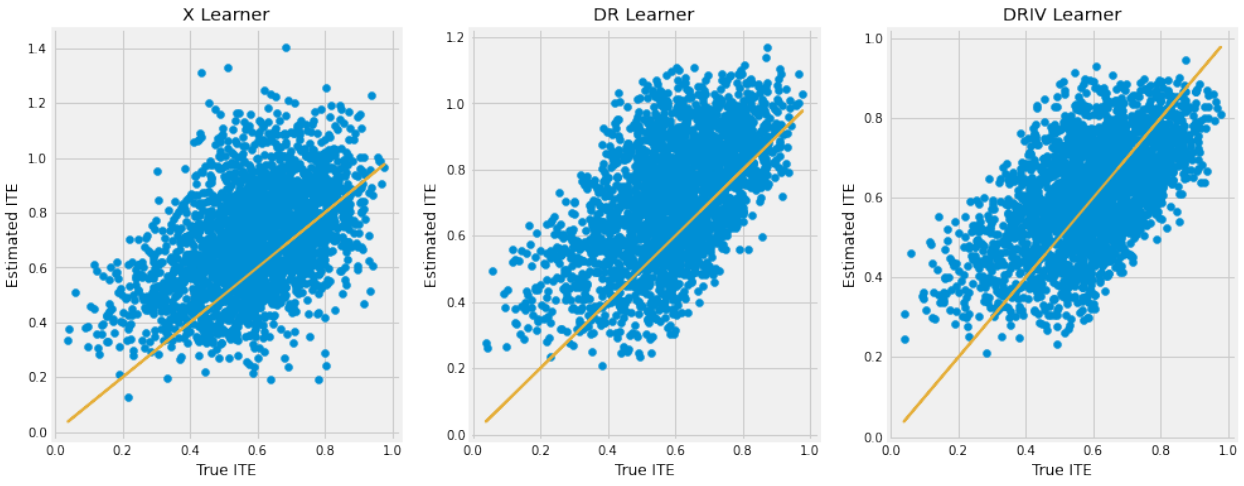
[10]:
```
learner_dr = BaseDRRegressor(learner=XGBRegressor(), treatment_effect_
↪learner=LinearRegression())
cate_dr = learner_dr.fit_predict(X=X_obs, treatment=treatment, y=y)
```

[12]:
```
learner_driv = BaseDRIVRegressor(learner=XGBRegressor(), treatment_effect_
↪learner=LinearRegression())
cate_driv = learner_driv.fit_predict(X=X_obs, assignment=assignment,␣
↪treatment=treatment, y=y)
```

We continue to see that X Learner generates a noisier ITE estimate than DR Learner, though both of them have a upward bias. But DRIV Learner is able to alleviate the bias significantly.

[13]:
```
fig, ax = plt.subplots(1, 3, figsize=(15, 6))
ax[0].scatter(tau[treatment==1], cate_x[treatment==1])
ax[0].plot(tau[treatment==1], tau[treatment==1], color='C2', linewidth=2)
ax[0].set_xlabel('True ITE')
ax[0].set_ylabel('Estimated ITE')
ax[0].set_title('X Learner')
ax[1].scatter(tau[treatment==1], cate_dr[treatment==1])
ax[1].plot(tau[treatment==1], tau[treatment==1], color='C2', linewidth=2)
ax[1].set_xlabel('True ITE')
ax[1].set_ylabel('Estimated ITE')
ax[1].set_title('DR Learner')
ax[2].scatter(tau[treatment==1], cate_driv[treatment==1])
ax[2].plot(tau[treatment==1], tau[treatment==1], color='C2', linewidth=2)
ax[2].set_xlabel('True ITE')
ax[2].set_ylabel('Estimated ITE')
ax[2].set_title('DRIV Learner')
```

[13]:
```
Text(0.5, 1.0, 'DRIV Learner')
```



[ ]:

5.16 Meta-Learner Benchmarks with Synthetic Data in Nie and Wager (2020)

This notebook compares X-, R-, T- and S-learners across the simulation setups discussed by Nie and Wager (2020). Note that the experiments don't include the parameter tuning described in the paper.

[1]:
```
import numpy as np
import pandas as pd

from causalml.inference.meta import BaseSRegressor
from causalml.inference.meta import BaseTRegressor
from causalml.inference.meta import BaseXRegressor
from causalml.inference.meta import BaseRRegressor

from causalml.dataset import synthetic_data

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.base import clone

from sklearn.linear_model import LogisticRegression, Lasso
from xgboost import XGBRegressor

from copy import deepcopy
from itertools import product

from tqdm import tqdm

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
```

```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force␣
↪console mode (e.g. in jupyter console)
Failed to import duecredit due to No module named 'duecredit'
```

[2]:
```
import importlib
print(importlib.metadata.version('causalml') )
```

```
0.14.0
```

[3]:
```
def run_experiments(n_list, p_list, s_list, m_list, learner_dict, num_iter,
                    propensity_learner):

    result_list = []

    for i in tqdm(range(num_iter)):

        for n, p, s, m, learner in product(n_list, p_list, s_list, m_list, learner_
↪dict.keys()):

            y, X, W, tau, _, _ = synthetic_data(mode=m, n=n, p=p, sigma=s)
```

(continues on next page)

(continued from previous page)

```
            X_train, X_test, W_train, _, y_train, _, _, tau_test = train_test_split(
                X, W, y, tau, test_size=0.2, random_state=111)

            if propensity_learner is not None:
                em = clone(propensity_learner)
                em.fit(X_train, W_train)
                e_hat_train = cross_val_predict(em, X_train, W_train, method='predict_
↪proba')[:, 1]
                e_hat_test = em.predict_proba(X_test)[:, 1]

            model = deepcopy(learner_dict[learner])
            model.fit(X=X_train, treatment=W_train, y=y_train, p=e_hat_train)
            hat_tau = model.predict(X_test, p=e_hat_test)

            pehe = mean_squared_error(tau_test, hat_tau)

            result_list.append([n, p, s, m, learner, pehe])

    cols = ['num_samples', 'num_features', 'sigma', 'sim_mode', 'learner', 'pehe']
    df_res = pd.DataFrame(result_list, columns=cols)
    return df_res
```

5.16.1 Lasso based experiments

[4]:
```
# Simulation params from Nie and Wager (2020)
n_list = [100, 500]
p_list = [6, 12]
s_list = [0.5, 1, 2, 4]
m_list = [1, 2, 3, 4]
num_iter = 100

learner_dict = {
    'S-Learner': BaseSRegressor(learner=Lasso()),
    'T-Learner': BaseTRegressor(learner=Lasso()),
    'X-Learner': BaseXRegressor(learner=Lasso()),
    'R-Learner': BaseRRegressor(learner=Lasso())
}

propensity_learner = LogisticRegression(penalty='l1', solver='liblinear')

df_res_lasso = run_experiments(n_list, p_list, s_list, m_list, learner_dict, num_iter,
↪ propensity_learner=propensity_learner)
```

```
100%|██████████| 100/100 [04:00<00:00,  2.40s/it]
```

[5]:
```
df_res_lasso.groupby(['learner', 'sim_mode'])['pehe'].median()
```

[5]:
```
learner    sim_mode
R-Learner  1           0.135057
           2           1.228229
           3           0.056223
           4           1.769802
S-Learner  1           0.290226
           2           1.911610
           3           1.000000
```

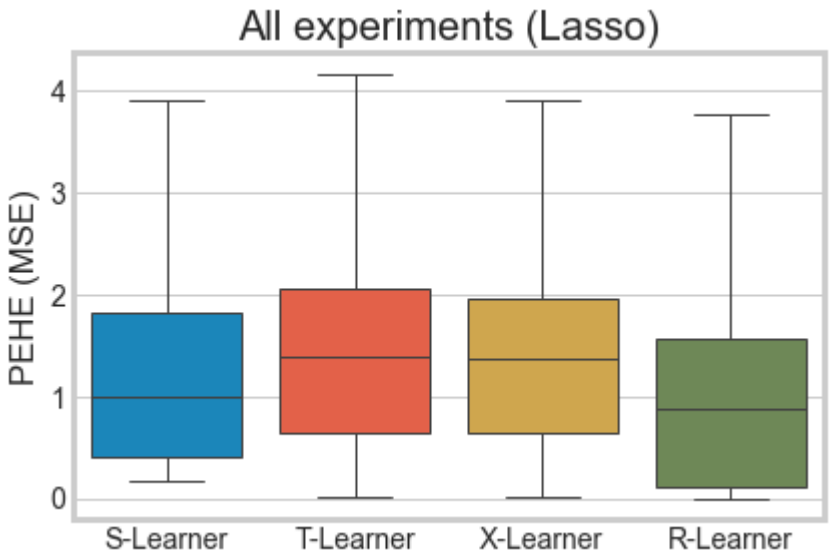
(continues on next page)

(continued from previous page)

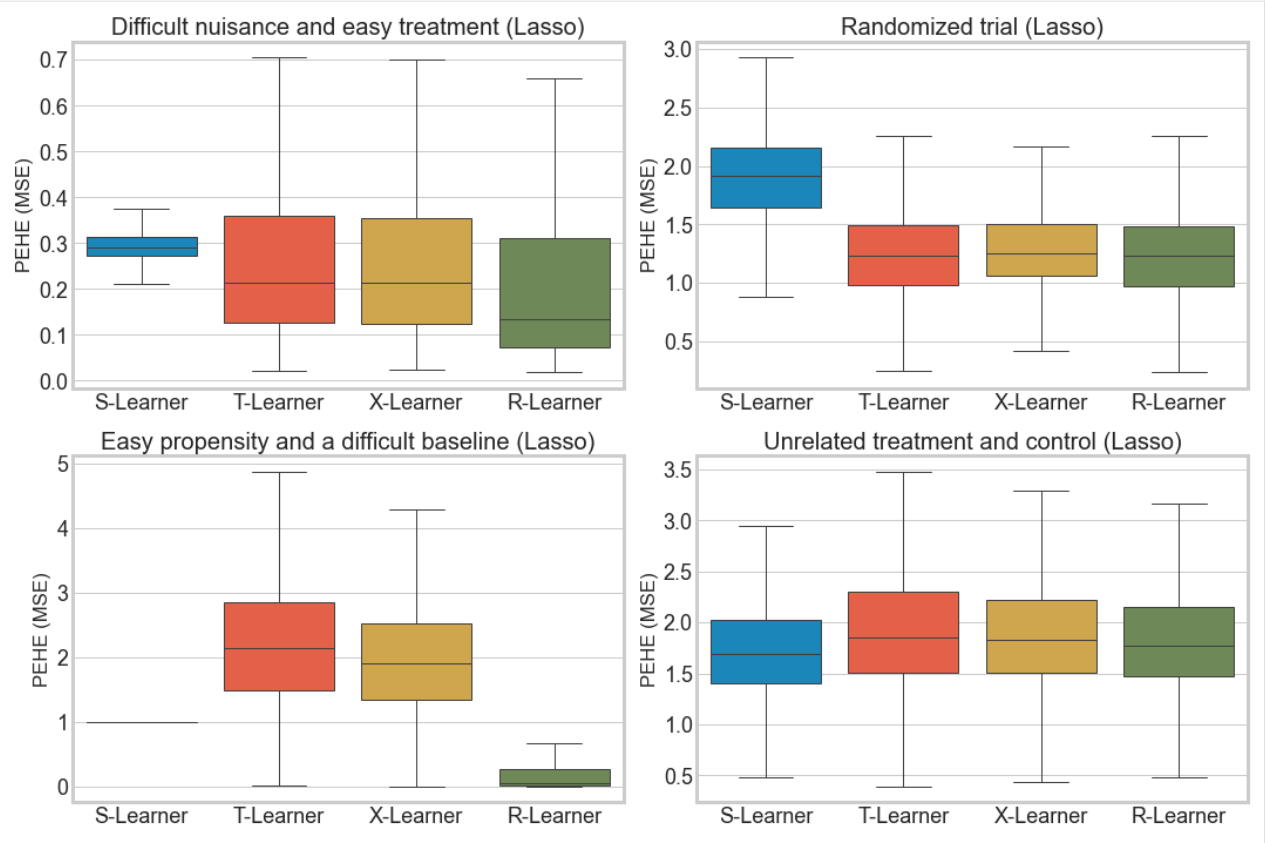
```
            4           1.696009
T-Learner   1           0.214197
            2           1.229950
            3           2.133935
            4           1.848652
X-Learner   1           0.213853
            2           1.257568
            3           1.910579
            4           1.826252
Name: pehe, dtype: float64
```

[6]:
```
data_generation_descs = {
    1: 'Difficult nuisance and easy treatment',
    2: 'Randomized trial',
    3: 'Easy propensity and a difficult baseline',
    4: 'Unrelated treatment and control'
}
```

[7]:
```
sns.boxplot(x='learner', y='pehe', data=df_res_lasso, linewidth=1, showfliers=False)
plt.ylabel('PEHE (MSE)')
plt.xlabel('')
plt.title('All experiments (Lasso)')
plt.show()
```



[8]:
```
fig, axs = plt.subplots(2, 2, figsize=(15, 10))
axs = axs.ravel()
for i, m in zip(range(4), m_list):
    sns.boxplot(x='learner', y='pehe', data=df_res_lasso.loc[df_res_lasso['sim_mode']␣
↪== m], linewidth=1, showfliers=False, ax=axs[i])
    axs[i].title.set_text(data_generation_descs[m] + ' (Lasso)')
    axs[i].set_ylabel('PEHE (MSE)')
    axs[i].set_xlabel('') # Hack
    axs[i].tick_params(labelsize=18)
plt.tight_layout()
```



5.16.2 Gradient boosting based experiments

[9]:
```
n_list = [500, 1000]
p_list = [6, 12]
s_list = [0.5, 1, 2, 4]
m_list = [1, 2, 3, 4]
num_iter = 100

learner_dict = {
    'S-Learner': BaseSRegressor(learner=XGBRegressor(n_jobs=-1)),
    'T-Learner': BaseTRegressor(learner=XGBRegressor(n_jobs=-1)),
    'X-Learner': BaseXRegressor(learner=XGBRegressor(n_jobs=-1)),
    'R-Learner': BaseRRegressor(learner=XGBRegressor(n_jobs=-1))
}

propensity_learner = LogisticRegression(penalty='l1', solver='liblinear')

df_res_xgb = run_experiments(n_list, p_list, s_list, m_list, learner_dict, num_iter,␣
↪propensity_learner=propensity_learner)
```

```
100%|██████████| 100/100 [17:46:40<00:00, 640.00s/it]
```

[10]:
```
df_res_xgb.groupby(['learner', 'sim_mode'])['pehe'].median()
```

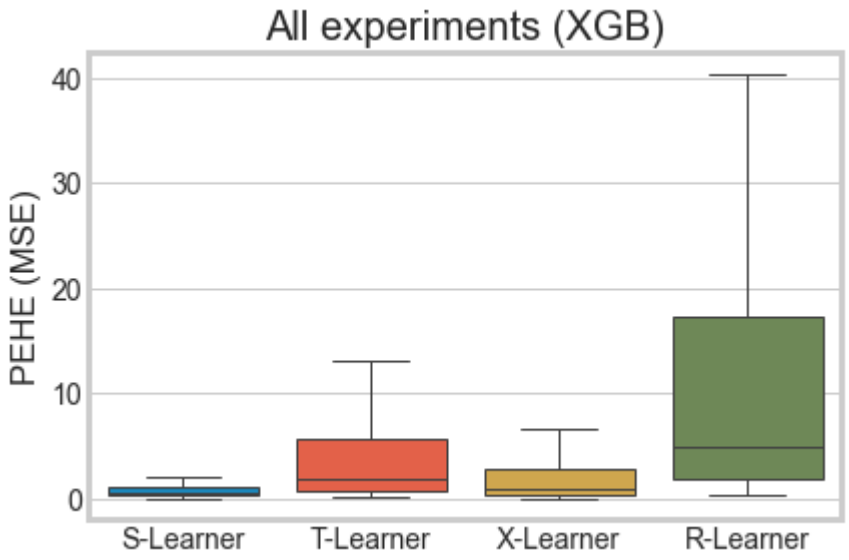
[10]:
```
learner    sim_mode
R-Learner  1           5.178797
```

(continues on next page)

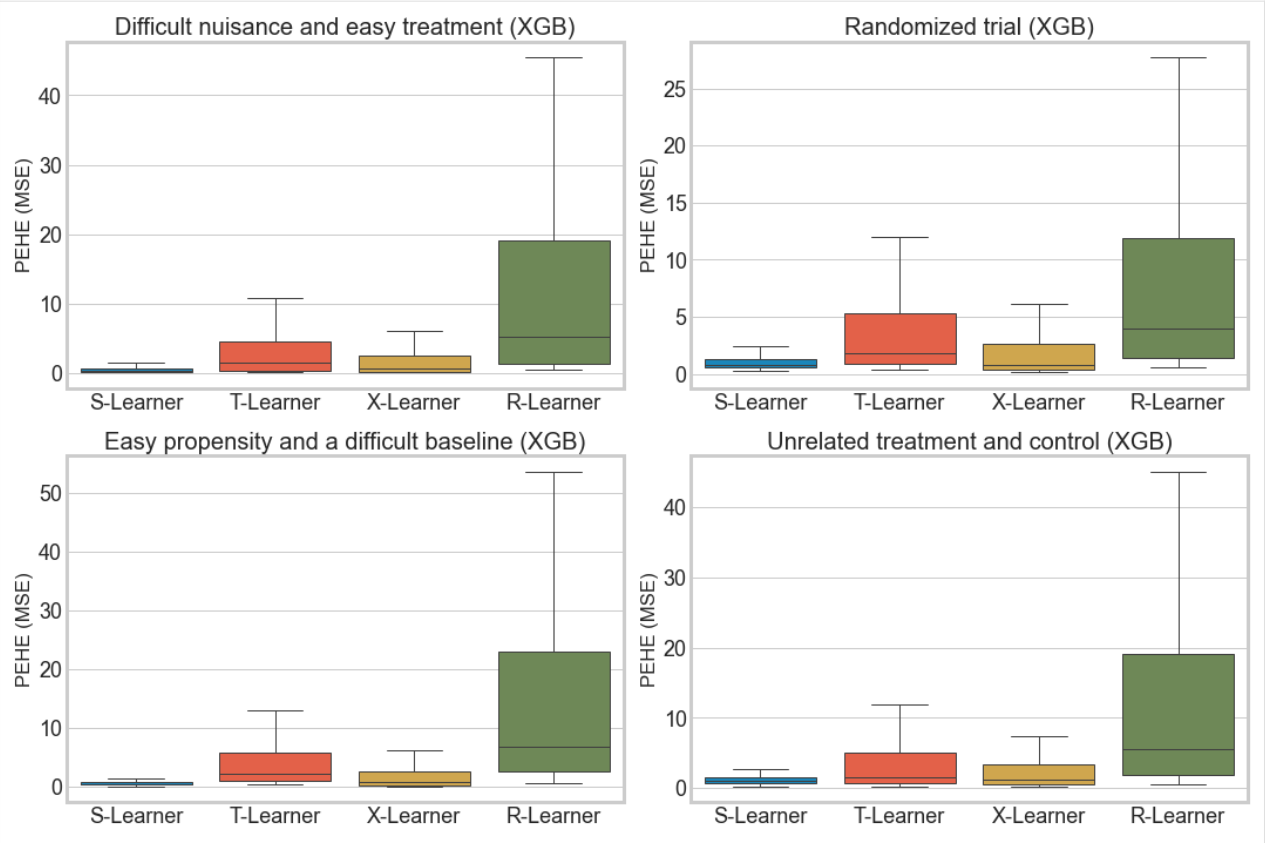
(continued from previous page)

```
            2           3.969635
            3           6.766369
            4           5.581396
S-Learner   1           0.364403
            2           0.802687
            3           0.507753
            4           1.030971
T-Learner   1           1.401733
            2           1.829172
            3           2.266735
            4           1.623793
X-Learner   1           0.712560
            2           0.818282
            3           0.864205
            4           1.196562
Name: pehe, dtype: float64
```

[11]:
```
sns.boxplot(x='learner', y='pehe', data=df_res_xgb, linewidth=1, showfliers=False)
plt.ylabel('PEHE (MSE)')
plt.xlabel('')
plt.title('All experiments (XGB)')
plt.show()
```



[12]:
```
fig, axs = plt.subplots(2, 2, figsize=(15, 10))
axs = axs.ravel()
for i, m in zip(range(4), m_list):
    sns.boxplot(x='learner', y='pehe', data=df_res_xgb.loc[df_res_xgb['sim_mode'] ==␣
↪m], linewidth=1, showfliers=False, ax=axs[i])
    axs[i].title.set_text(data_generation_descs[m] + ' (XGB)')
    axs[i].set_ylabel('PEHE (MSE)')
    axs[i].set_xlabel('') # Hack
    axs[i].tick_params(labelsize=18)
plt.tight_layout()
```



5.17 Causal Trees/Forests Treatment Effects Estimation and Tree Visualization

[1]:
```
import pandas as pd
import numpy as np
import multiprocessing as mp
from collections import defaultdict

np.random.seed(42)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

import causalml
from causalml.metrics import plot_gain, plot_qini, qini_score
from causalml.dataset import synthetic_data
from causalml.inference.tree import plot_dist_tree_leaves_values, get_tree_leaves_mask
from causalml.inference.meta import BaseSRegressor, BaseXRegressor, BaseTRegressor,␣
↪BaseDRRegressor
from causalml.inference.tree import CausalRandomForestRegressor
from causalml.inference.tree import CausalTreeRegressor
from causalml.inference.tree.plot import plot_causal_tree

import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import seaborn as sns

%config InlineBackend.figure_format = 'retina'
```

```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force␣
↪console mode (e.g. in jupyter console)
Failed to import duecredit due to No module named 'duecredit'
```

[2]:
```
import importlib
print(importlib.metadata.version('causalml') )
```

```
0.14.0
```

[3]:
```
# Simulate randomized trial: mode=2
y, X, w, tau, b, e = synthetic_data(mode=2, n=15000, p=20, sigma=5.5)

df = pd.DataFrame(X)
feature_names = [f'feature_{i}' for i in range(X.shape[1])]
df.columns = feature_names
df['outcome'] = y
df['treatment'] = w
df['treatment_effect'] = tau
```

[4]:
```
df.head()
```

[4]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	\
0	0.496714	-0.138264	0.358450	1.523030	-0.234153	-0.234137	
1	1.465649	-0.225776	1.239872	-1.424748	-0.544383	0.110923	
2	0.738467	0.171368	0.909835	-0.301104	-1.478522	-0.719844	
3	-0.479174	-0.185659	0.000000	-1.196207	0.812526	1.356240	
4	-0.219672	0.357113	0.137441	-0.518270	-0.808494	-0.501757	

	feature_6	feature_7	feature_8	feature_9	…	feature_13	feature_14	\
0	1.579213	0.767435	-0.469474	0.542560	…	-1.913280	-1.724918	
1	-1.150994	0.375698	-0.600639	-0.291694	…	-1.057711	0.822545	
2	-0.460639	1.057122	0.343618	-1.763040	…	0.611676	1.031000	
3	-0.072010	1.003533	0.361636	-0.645120	…	1.564644	-2.619745	
4	0.915402	0.328751	-0.529760	0.513267	…	-0.327662	-0.392108	

	feature_15	feature_16	feature_17	feature_18	feature_19	outcome	\
0	-0.562288	-1.012831	0.314247	-0.908024	-1.412304	7.124356	
1	-1.220844	0.208864	-1.959670	-1.328186	0.196861	-11.263144	
2	0.931280	-0.839218	-0.309212	0.331263	0.975545	0.269378	
3	0.821903	0.087047	-0.299007	0.091761	-1.987569	-0.976893	
4	-1.463515	0.296120	0.261055	0.005113	-0.234587	-1.949163	

	treatment	treatment_effect
0	1	1.123117
1	0	2.052266
2	0	1.520964
3	0	0.125446
4	1	0.667889

[5 rows x 23 columns]

[5]:
```
# Look at the conversion rate and sample size in each group
df.pivot_table(values='outcome',
```

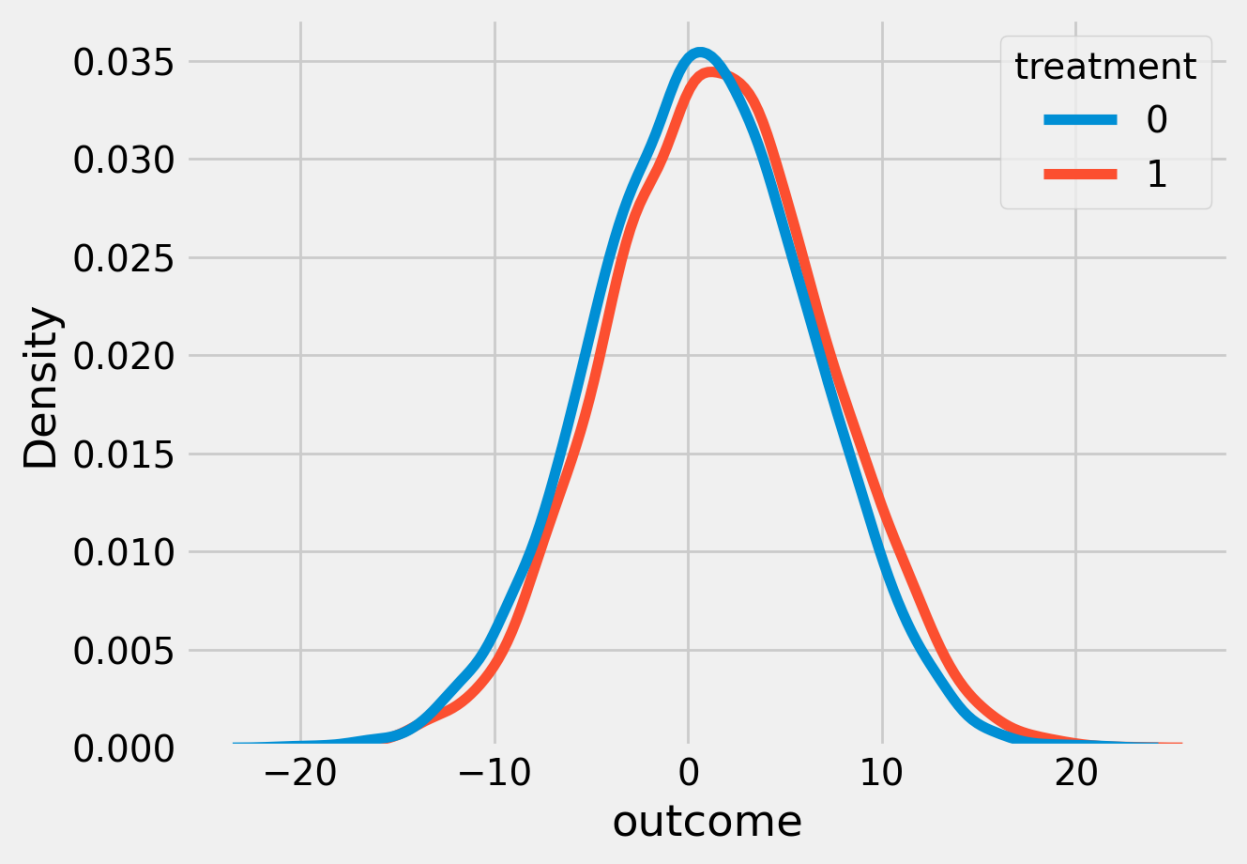
(continues on next page)

(continued from previous page)

```
                index='treatment',
                aggfunc=[np.mean, np.size],
                margins=True)
```

[5]:
```
                mean    size
             outcome outcome
treatment
0           0.736125    7502
1           1.543688    7498
All         1.139799   15000
```

[6]:
```
sns.kdeplot(data=df, x='outcome', hue='treatment')
plt.show()
```



[7]:
```
# Split data to training and testing samples for model validation (next section)
df_train, df_test = train_test_split(df, test_size=0.2, random_state=11101)
n_test = df_test.shape[0]
n_train = df_train.shape[0]
```

[8]:
```
# Table to gather estimated ITEs by models
df_result = pd.DataFrame({
    'outcome': df_test['outcome'],
    'is_treated': df_test['treatment'],
    'treatment_effect': df_test['treatment_effect']
})
```

5.17.1 CausalTreeRegressor

Available criteria for causal trees:

`standard_mse`: scikit-learn MSE where node values store $E_{node_i}(X|T=1) - E_{node_i}(X|T=0)$, treatment effects.

`causal_mse`: *The criteria reward a partition for finding strong heterogeneity in treatment effects and penalize a partition that creates variance in leaf estimates.* https://www.pnas.org/doi/10.1073/pnas.1510489113

[9]:
```
ctrees = {
    'ctree_mse': {
        'params':
        dict(criterion='standard_mse',
             control_name=0,
             min_impurity_decrease=0,
             min_samples_leaf=400,
             groups_penalty=0.,
             groups_cnt=True),
    },
    'ctree_cmse': {
        'params':
        dict(
            criterion='causal_mse',
            control_name=0,
            min_samples_leaf=400,
            groups_penalty=0.,
            groups_cnt=True,
        ),
    },
    'ctree_cmse_p=0.1': {
        'params':
        dict(
            criterion='causal_mse',
            control_name=0,
            min_samples_leaf=400,
            groups_penalty=0.1,
            groups_cnt=True,
        ),
    },
    'ctree_cmse_p=0.25': {
        'params':
        dict(
            criterion='causal_mse',
            control_name=0,
            min_samples_leaf=400,
            groups_penalty=0.25,
            groups_cnt=True,
        ),
    },
    'ctree_cmse_p=0.5': {
        'params':
        dict(
            criterion='causal_mse',
            control_name=0,
            min_samples_leaf=400,
            groups_penalty=0.5,
            groups_cnt=True,
        ),
```

(continues on next page)

(continued from previous page)

```
    },
    'ctree_ttest': {
        'params':
        dict(criterion='t_test',
             control_name=0,
             min_samples_leaf=400,
             groups_penalty=0.,
             groups_cnt=True),
    },
}
```

[10]:
```
# Model treatment effect
for ctree_name, ctree_info in ctrees.items():
    print(f"Fitting: {ctree_name}")
    ctree = CausalTreeRegressor(**ctree_info['params'])
    ctree.fit(X=df_train[feature_names].values,
              treatment=df_train['treatment'].values,
              y=df_train['outcome'].values)

    ctrees[ctree_name].update({'model': ctree})
    df_result[ctree_name] = ctree.predict(df_test[feature_names].values)
```

```
Fitting: ctree_mse
Fitting: ctree_cmse
Fitting: ctree_cmse_p=0.1
Fitting: ctree_cmse_p=0.25
Fitting: ctree_cmse_p=0.5
Fitting: ctree_ttest
```

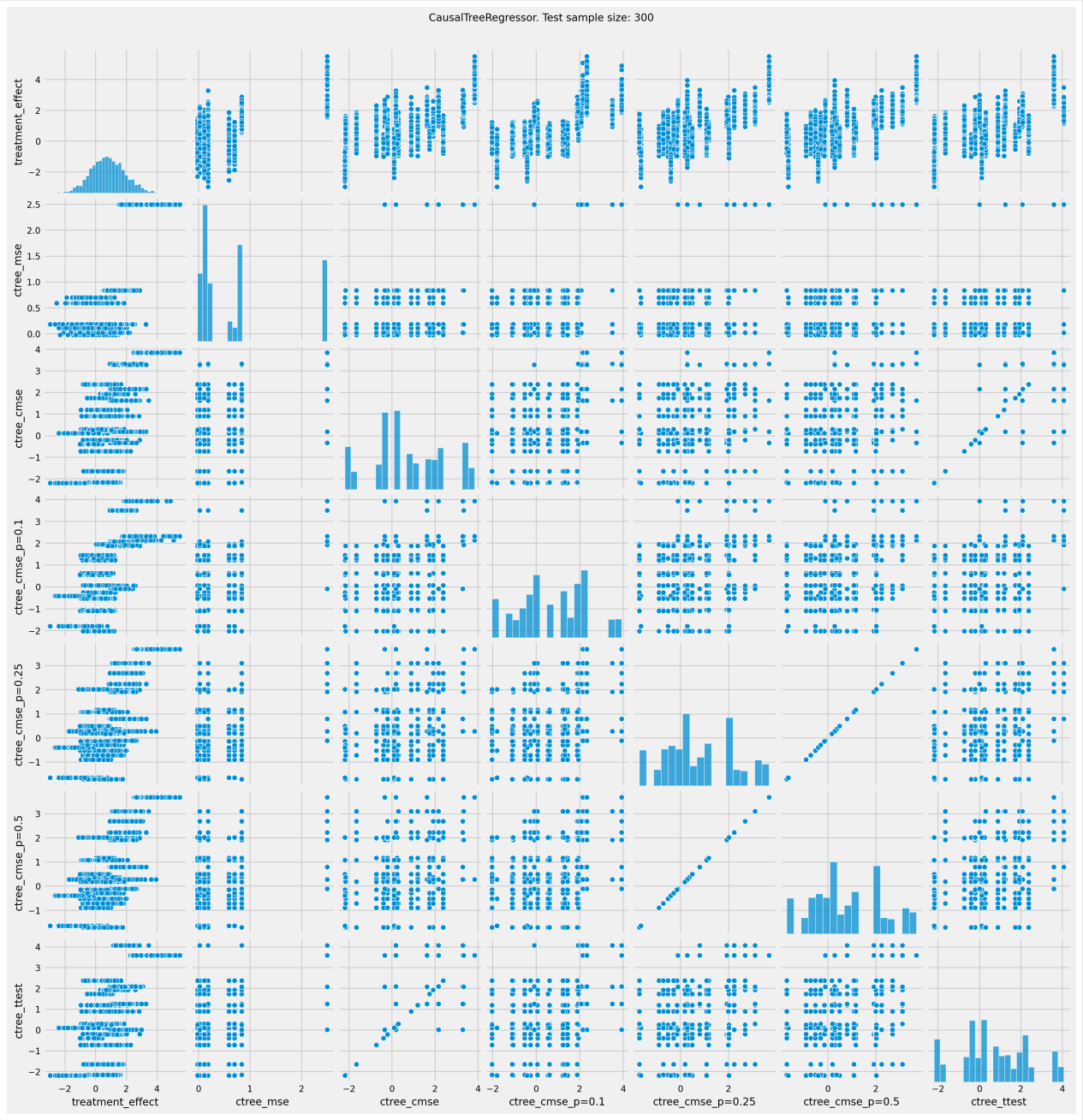
[11]:
```
df_result.head()
```

[11]:

	outcome	is_treated	treatment_effect	ctree_mse	ctree_cmse	\
625	3.519424	1	0.819201	0.110685	0.895132	
5717	-1.175555	0	1.131599	0.183293	3.286218	
14801	4.361167	0	1.969727	0.834163	3.329034	
13605	4.523891	0	0.884079	0.183293	-0.727168	
4208	-6.077212	0	1.179124	0.183293	3.329034	

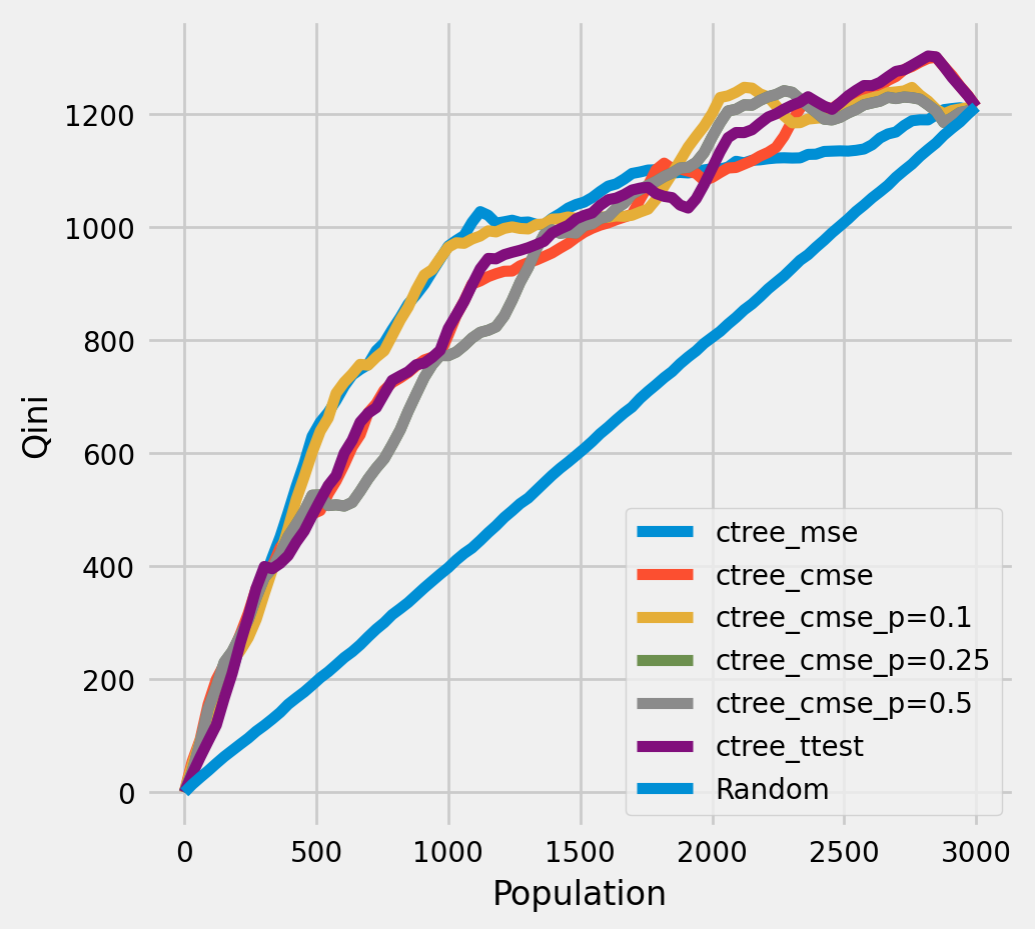
	ctree_cmse_p=0.1	ctree_cmse_p=0.25	ctree_cmse_p=0.5	ctree_ttest
625	-1.104810	1.166407	1.166407	0.895132
5717	2.071375	0.794040	0.794040	2.096099
14801	3.497691	3.097263	3.097263	2.077012
13605	-2.025098	-0.902955	-0.902955	-0.727168
4208	3.497691	1.916049	1.916049	1.257711

[12]:
```
# See treatment effect estimation with CausalTreeRegressor vs true treatment effect

n_obs = 300

indxs = df_result.index.values
np.random.shuffle(indxs)
indxs = indxs[:n_obs]

plt.rcParams.update({'font.size': 10})
pairplot = sns.pairplot(df_result[['treatment_effect', *list(ctrees)]])
pairplot.fig.suptitle(f"CausalTreeRegressor. Test sample size: {n_obs}" , y=1.02)
plt.show()
```



Plot the Qini chart

[13]:
```
plot_qini(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          treatment_effect_col='treatment_effect',
          figsize=(5,5)
         )
```

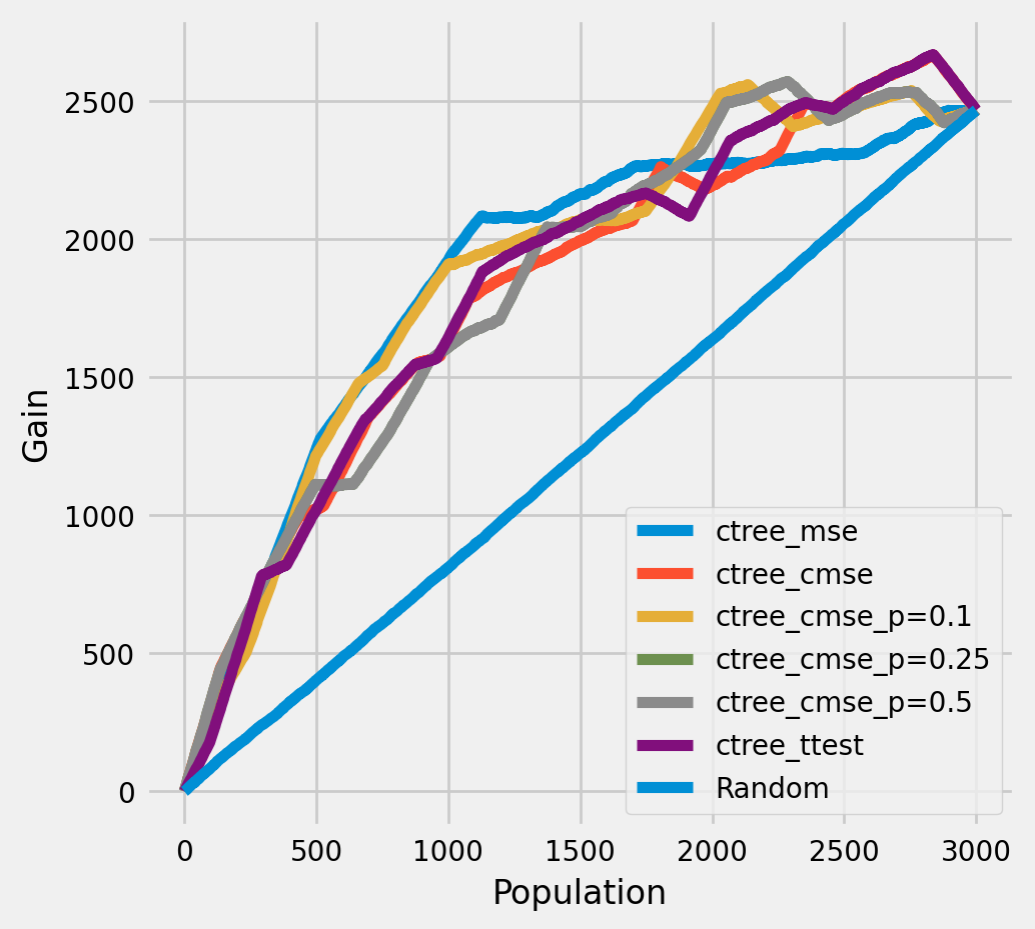


[14]:
```
df_qini = qini_score(df_result,
           outcome_col='outcome',
           treatment_col='is_treated',
           treatment_effect_col='treatment_effect')
df_qini.sort_values(ascending=False)
```

[14]:
```
ctree_cmse_p=0.1     0.273112
ctree_mse            0.260141
ctree_ttest          0.247668
ctree_cmse           0.242333
ctree_cmse_p=0.25    0.231232
ctree_cmse_p=0.5     0.231232
Random               0.000000
dtype: float64
```

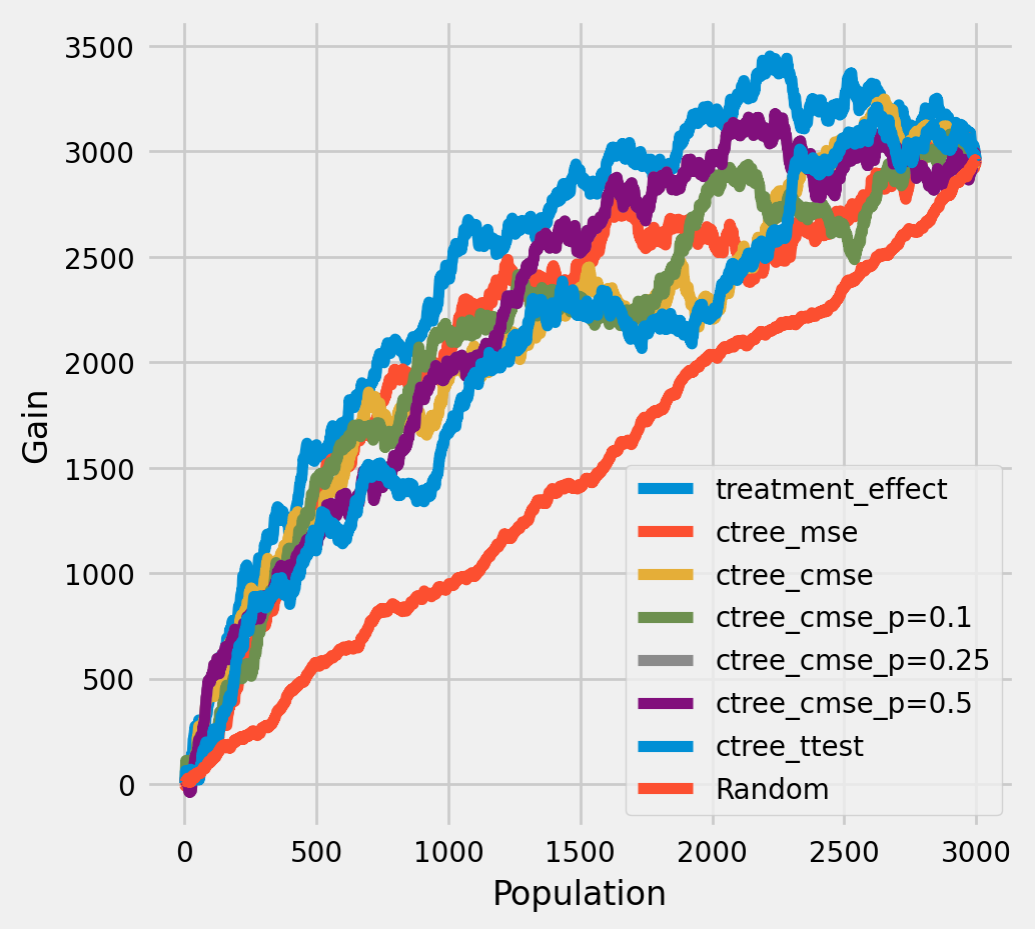
The cumulative gain of the true treatment effect in each population

[15]:
```
plot_gain(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          treatment_effect_col='treatment_effect',
          n = n_test,
          figsize=(5,5)
         )
```



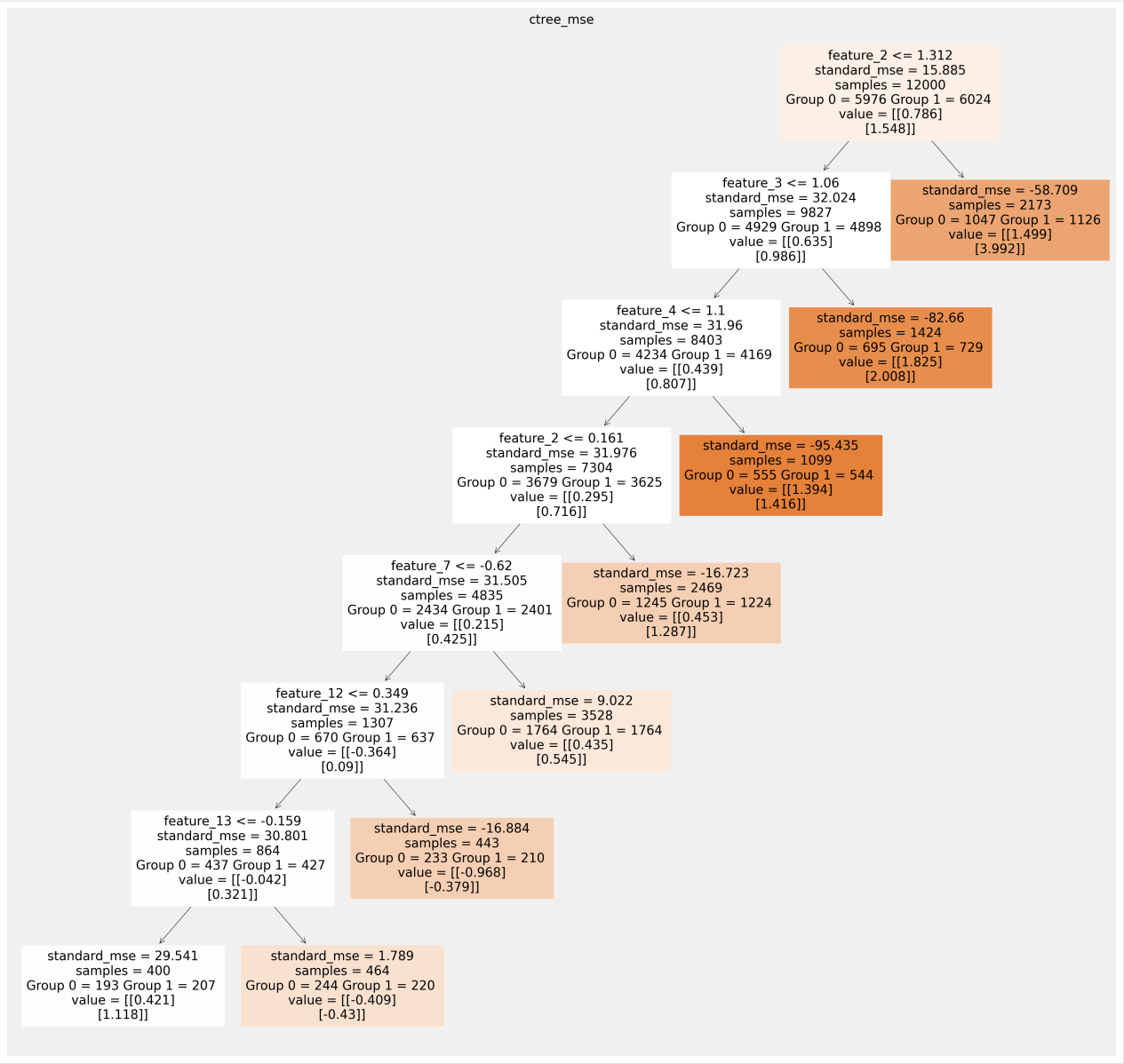
The cumulative difference between the mean outcomes of the treatment and control groups in each population

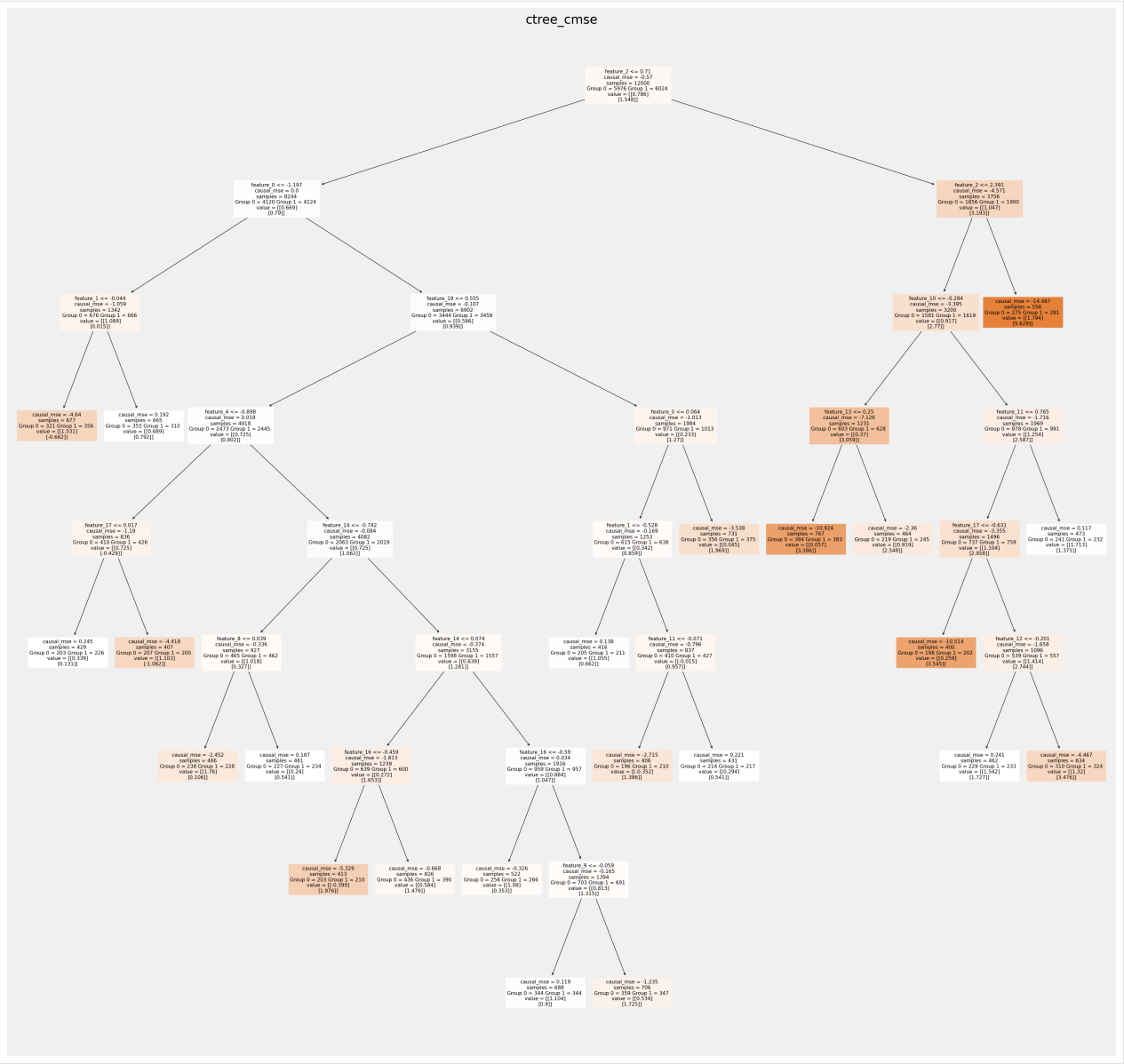
[16]:
```
plot_gain(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          n = n_test,
          figsize=(5,5)
         )
```

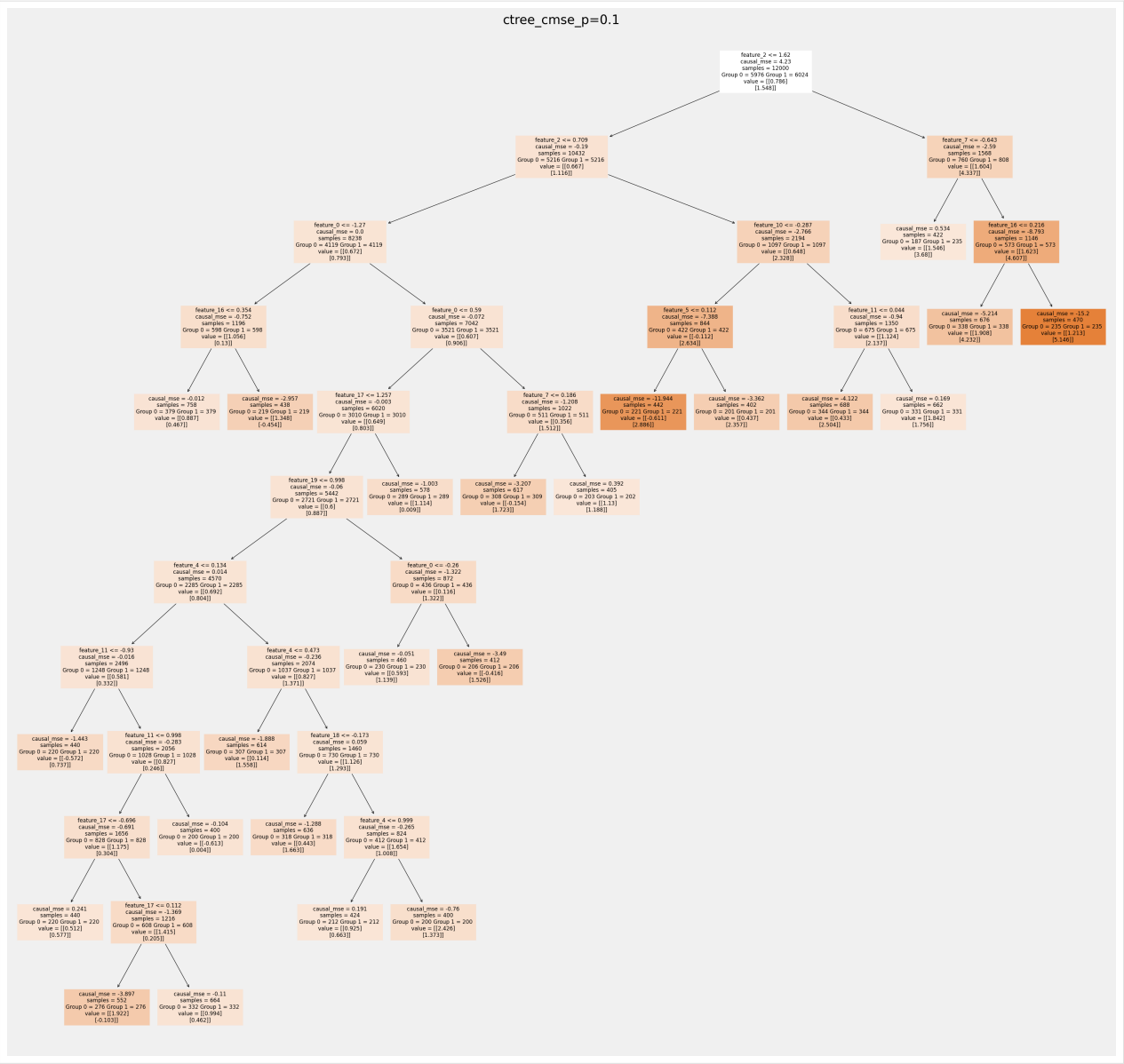


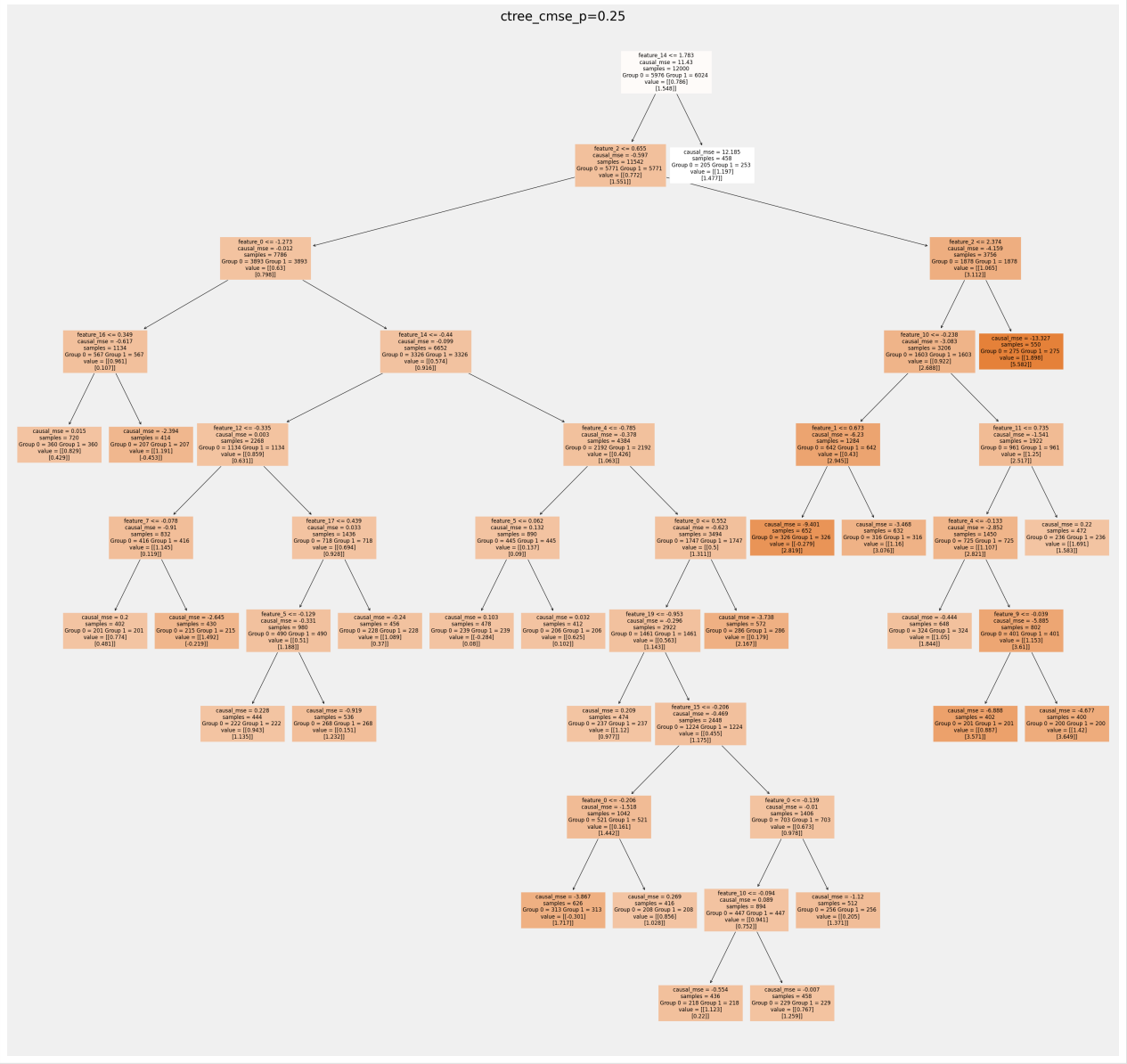
Plot trees with sklearn function and save as vector graphics

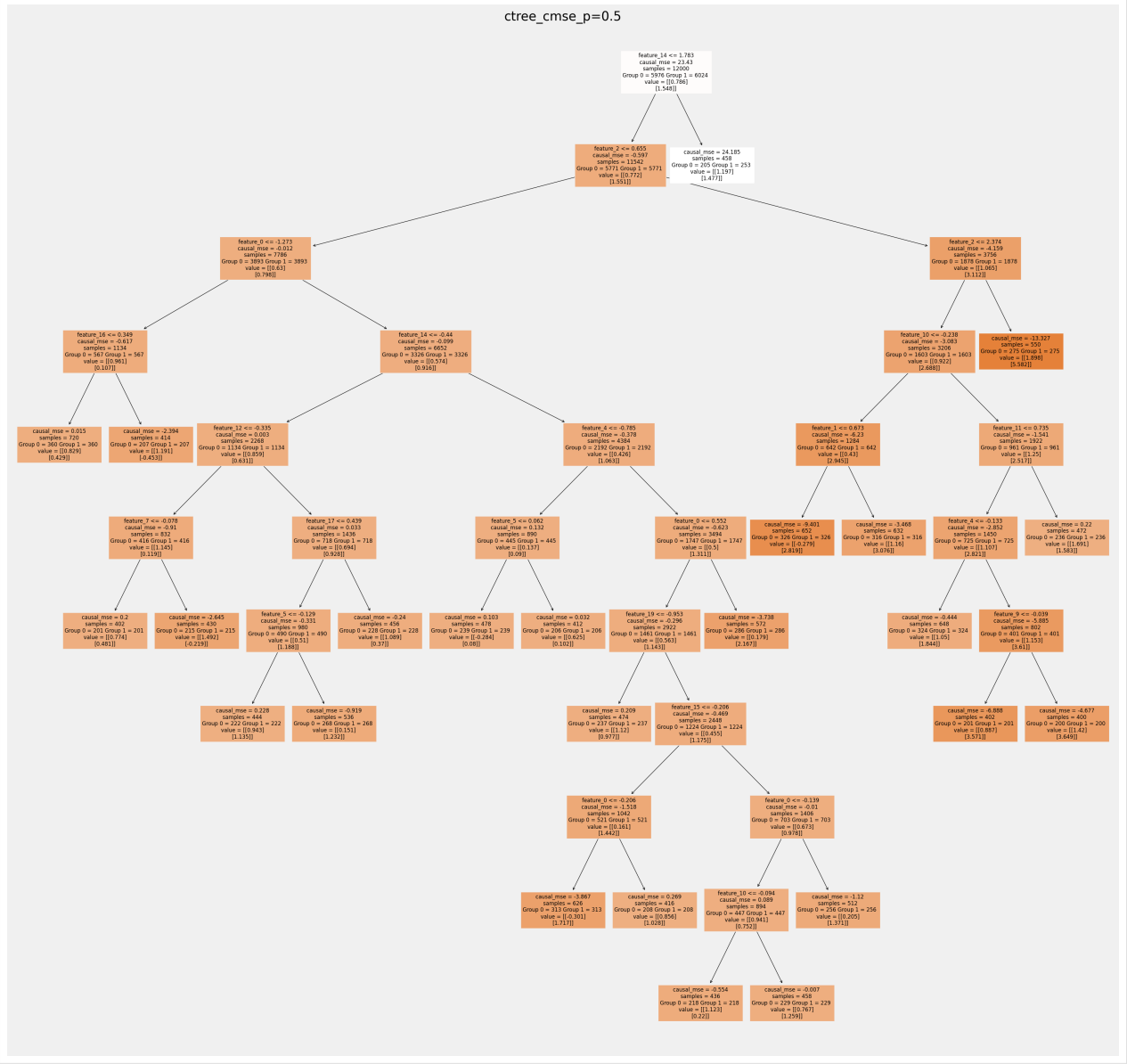
[17]:
```
for ctree_name, ctree_info in ctrees.items():
    plt.figure(figsize=(20,20))
    plot_causal_tree(ctree_info['model'],
                     feature_names = feature_names,
                     filled=True,
                     impurity=True,
                     proportion=False,
                )
    plt.title(ctree_name)
    plt.savefig(f'{ctree_name}.svg')
```

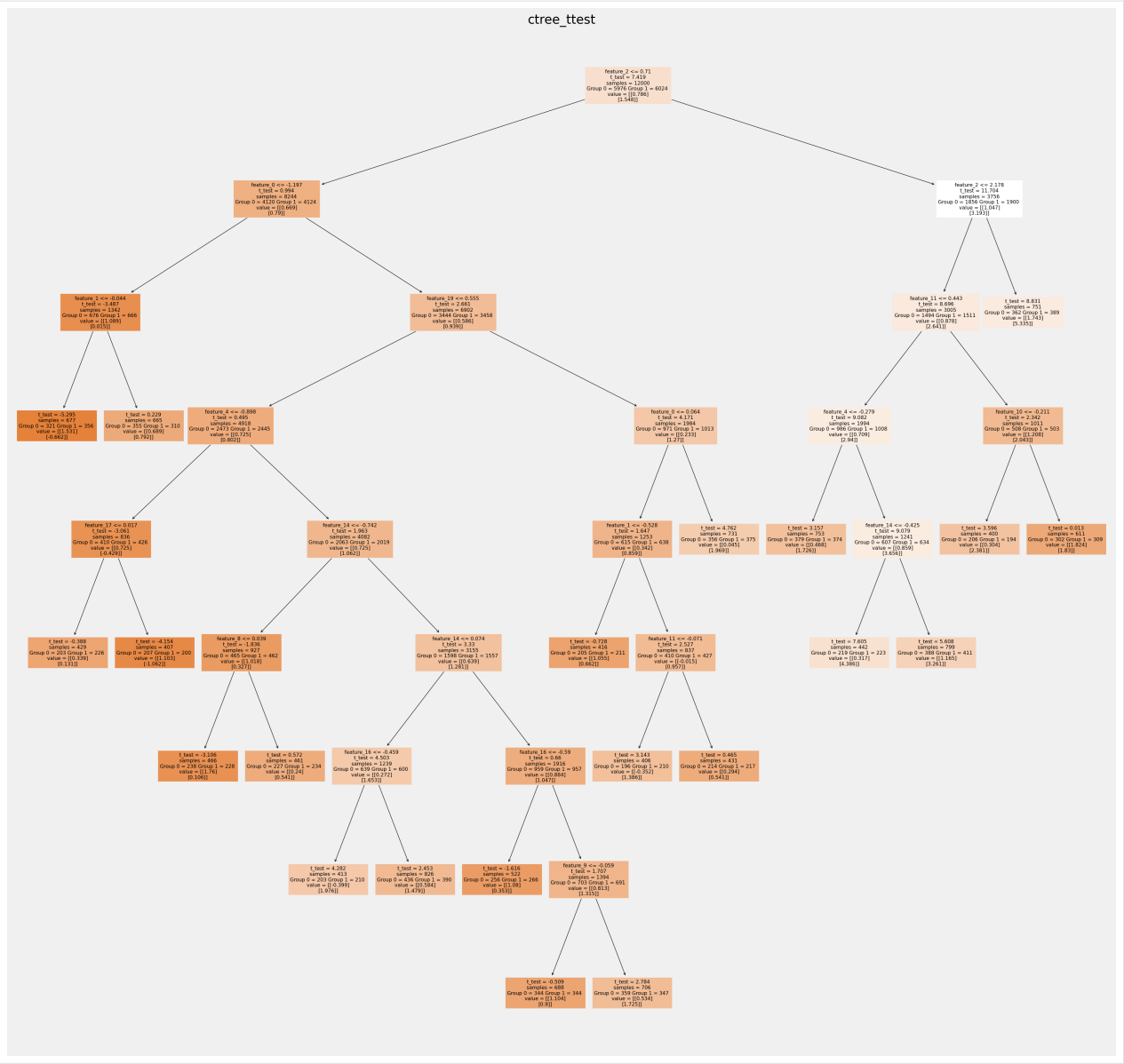

ctree_mse
feature_2 <= 1.312
standard_mse = 15.885
samples = 12000
Group 0 = 5976 Group 1 = 6024
value = [[0.786]
[1.548]]
feature_3 <= 1.06
standard_mse = 32.024
samples = 9827
Group 0 = 4929 Group 1 = 4898
value = [[0.635]
[0.986]]
standard_mse = -58.709
samples = 2173
Group 0 = 1047 Group 1 = 1126
value = [[1.499]
[3.992]]
feature_4 <= 1.1
standard_mse = 31.96
samples = 8403
Group 0 = 4234 Group 1 = 4169
value = [[0.439]
[0.807]]
standard_mse = -82.66
samples = 1424
Group 0 = 695 Group 1 = 729
value = [[1.825]
[2.008]]
feature_2 <= 0.161
standard_mse = 31.976
samples = 7304
Group 0 = 3679 Group 1 = 3625
value = [[0.295]
[0.716]]
standard_mse = -95.435
samples = 1099
Group 0 = 555 Group 1 = 544
value = [[1.394]
[1.416]]
feature_7 <= -0.62
standard_mse = 31.505
samples = 4835
Group 0 = 2434 Group 1 = 2401
value = [[0.215]
[0.425]]
standard_mse = -16.723
samples = 2469
Group 0 = 1245 Group 1 = 1224
value = [[0.453]
[1.287]]
feature_12 <= 0.349
standard_mse = 31.236
samples = 1307
Group 0 = 670 Group 1 = 637
value = [[-0.364]
[0.09]]
standard_mse = 9.022
samples = 3528
Group 0 = 1764 Group 1 = 1764
value = [[0.435]
[0.545]]
feature_13 <= -0.159
standard_mse = 30.801
samples = 864
Group 0 = 437 Group 1 = 427
value = [[-0.042]
[0.321]]
standard_mse = -16.884
samples = 443
Group 0 = 233 Group 1 = 210
value = [[-0.968]
[-0.379]]
standard_mse = 29.541
samples = 400
Group 0 = 193 Group 1 = 207
value = [[0.421]
[1.118]]
standard_mse = 1.789
samples = 464
Group 0 = 244 Group 1 = 220
value = [[-0.409]
[-0.43]]


ctree_cmse


ctree_cmse_p=0.1
feature_2 <= 1.62
causal_mse = 4.23
samples = 12000
Group 0 = 5976 Group 1 = 6024
value = [[0.786]
[1.548]]
feature_2 <= 0.709
causal_mse = -0.19
samples = 10432
Group 0 = 5216 Group 1 = 5216
value = [[0.667]
[1.116]]
feature_7 <= -0.643
causal_mse = -2.59
samples = 1568
Group 0 = 760 Group 1 = 808
value = [[1.604]
[4.337]]
feature_0 <= -1.27
causal_mse = 0.0
samples = 8238
Group 0 = 4119 Group 1 = 4119
value = [[0.672]
[0.793]]
feature_10 <= -0.287
causal_mse = -2.766
samples = 2194
Group 0 = 1097 Group 1 = 1097
value = [[0.648]
[2.328]]
causal_mse = 0.534
samples = 422
Group 0 = 187 Group 1 = 235
value = [[1.546]
[3.68]]
feature_16 <= 0.216
causal_mse = -8.793
samples = 1146
Group 0 = 573 Group 1 = 573
value = [[1.623]
[4.607]]
feature_16 <= 0.354
causal_mse = -0.752
samples = 1196
Group 0 = 598 Group 1 = 598
value = [[1.056]
[0.13]]
feature_0 <= 0.59
causal_mse = -0.072
samples = 7042
Group 0 = 3521 Group 1 = 3521
value = [[0.607]
[0.906]]
feature_5 <= 0.112
causal_mse = -7.388
samples = 844
Group 0 = 422 Group 1 = 422
value = [[-0.112]
[2.634]]
feature_11 <= 0.044
causal_mse = -0.94
samples = 1350
Group 0 = 675 Group 1 = 675
value = [[1.124]
[2.137]]
causal_mse = -5.214
samples = 676
Group 0 = 338 Group 1 = 338
value = [[1.908]
[4.232]]
causal_mse = -15.2
samples = 470
Group 0 = 235 Group 1 = 235
value = [[1.213]
[5.146]]
causal_mse = -0.012
samples = 758
Group 0 = 379 Group 1 = 379
value = [[0.887]
[0.467]]
causal_mse = -2.957
samples = 438
Group 0 = 219 Group 1 = 219
value = [[1.348]
[-0.454]]
feature_17 <= 1.257
causal_mse = -0.003
samples = 6020
Group 0 = 3010 Group 1 = 3010
value = [[0.649]
[0.803]]
feature_7 <= 0.186
causal_mse = -1.208
samples = 1022
Group 0 = 511 Group 1 = 511
value = [[0.356]
[1.512]]
causal_mse = -11.944
samples = 442
Group 0 = 221 Group 1 = 221
value = [[-0.611]
[2.886]]
causal_mse = -3.362
samples = 402
Group 0 = 201 Group 1 = 201
value = [[0.437]
[2.357]]
causal_mse = -4.122
samples = 688
Group 0 = 344 Group 1 = 344
value = [[0.433]
[2.504]]
causal_mse = 0.169
samples = 662
Group 0 = 331 Group 1 = 331
value = [[1.842]
[1.756]]
feature_19 <= 0.998
causal_mse = -0.06
samples = 5442
Group 0 = 2721 Group 1 = 2721
value = [[0.6]
[0.887]]
causal_mse = -1.003
samples = 578
Group 0 = 289 Group 1 = 289
value = [[1.114]
[0.009]]
causal_mse = -3.207
samples = 617
Group 0 = 308 Group 1 = 309
value = [[-0.154]
[1.723]]
causal_mse = 0.392
samples = 405
Group 0 = 203 Group 1 = 202
value = [[1.13]
[1.188]]
feature_4 <= 0.134
causal_mse = 0.014
samples = 4570
Group 0 = 2285 Group 1 = 2285
value = [[0.692]
[0.804]]
feature_0 <= -0.26
causal_mse = -1.322
samples = 872
Group 0 = 436 Group 1 = 436
value = [[0.116]
[1.322]]
feature_11 <= -0.93
causal_mse = -0.016
samples = 2496
Group 0 = 1248 Group 1 = 1248
value = [[0.581]
[0.332]]
feature_4 <= 0.473
causal_mse = -0.236
samples = 2074
Group 0 = 1037 Group 1 = 1037
value = [[0.827]
[1.371]]
causal_mse = -0.051
samples = 460
Group 0 = 230 Group 1 = 230
value = [[0.593]
[1.139]]
causal_mse = -3.49
samples = 412
Group 0 = 206 Group 1 = 206
value = [[-0.416]
[1.526]]
causal_mse = -1.443
samples = 440
Group 0 = 220 Group 1 = 220
value = [[-0.572]
[0.737]]
feature_11 <= 0.998
causal_mse = -0.283
samples = 2056
Group 0 = 1028 Group 1 = 1028
value = [[0.827]
[0.246]]
causal_mse = -1.888
samples = 614
Group 0 = 307 Group 1 = 307
value = [[0.114]
[1.556]]
feature_18 <= -0.173
causal_mse = 0.059
samples = 1460
Group 0 = 730 Group 1 = 730
value = [[1.126]
[1.293]]
feature_17 <= -0.696
causal_mse = -0.691
samples = 1656
Group 0 = 828 Group 1 = 828
value = [[1.175]
[0.304]]
causal_mse = -0.104
samples = 400
Group 0 = 200 Group 1 = 200
value = [[-0.613]
[0.004]]
causal_mse = -1.288
samples = 636
Group 0 = 318 Group 1 = 318
value = [[0.443]
[1.663]]
feature_4 <= 0.999
causal_mse = -0.265
samples = 824
Group 0 = 412 Group 1 = 412
value = [[1.654]
[1.008]]
causal_mse = 0.241
samples = 440
Group 0 = 220 Group 1 = 220
value = [[0.512]
[0.577]]
feature_17 <= 0.112
causal_mse = -1.369
samples = 1216
Group 0 = 608 Group 1 = 608
value = [[1.415]
[0.205]]
causal_mse = 0.191
samples = 424
Group 0 = 212 Group 1 = 212
value = [[0.925]
[0.663]]
causal_mse = -0.76
samples = 400
Group 0 = 200 Group 1 = 200
value = [[2.426]
[1.373]]
causal_mse = -3.897
samples = 552
Group 0 = 276 Group 1 = 276
value = [[1.922]
[-0.103]]
causal_mse = -0.11
samples = 664
Group 0 = 332 Group 1 = 332
value = [[0.994]
[0.462]]

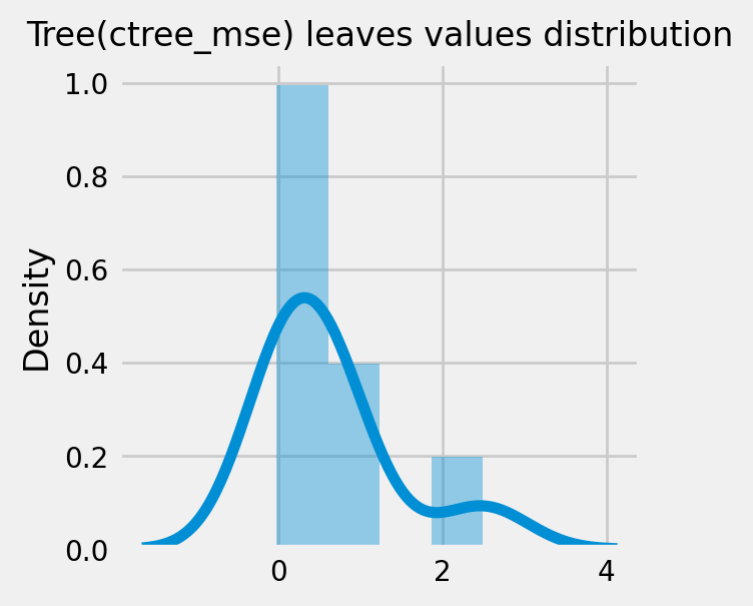

ctree_cmse_p=0.25

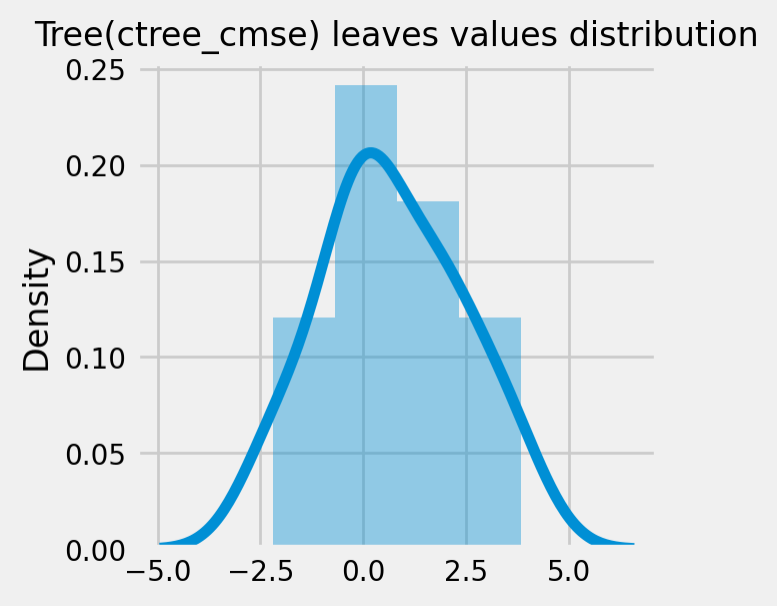

ctree_cmse_p=0.5
feature_14 <= 1.783
causal_mse = 23.43
samples = 12000
Group 0 = 5976 Group 1 = 6024
value = [[0.786]
[1.548]]
feature_2 <= 0.655
causal_mse = -0.597
samples = 11542
Group 0 = 5771 Group 1 = 5771
value = [[0.772]
[1.551]]
causal_mse = 24.185
samples = 458
Group 0 = 205 Group 1 = 253
value = [[1.197]
[1.477]]
feature_0 <= -1.273
causal_mse = -0.012
samples = 7786
Group 0 = 3893 Group 1 = 3893
value = [[0.63]
[0.798]]
feature_2 <= 2.374
causal_mse = -4.159
samples = 3756
Group 0 = 1878 Group 1 = 1878
value = [[1.065]
[3.112]]
feature_16 <= 0.349
causal_mse = -0.617
samples = 1134
Group 0 = 567 Group 1 = 567
value = [[0.961]
[0.107]]
feature_14 <= -0.44
causal_mse = -0.099
samples = 6652
Group 0 = 3326 Group 1 = 3326
value = [[0.574]
[0.916]]
feature_10 <= -0.238
causal_mse = -3.083
samples = 3206
Group 0 = 1603 Group 1 = 1603
value = [[0.922]
[2.688]]
causal_mse = -13.327
samples = 550
Group 0 = 275 Group 1 = 275
value = [[1.898]
[5.582]]
causal_mse = 0.015
samples = 720
Group 0 = 360 Group 1 = 360
value = [[0.829]
[0.429]]
causal_mse = -2.394
samples = 414
Group 0 = 207 Group 1 = 207
value = [[1.191]
[-0.453]]
feature_12 <= -0.335
causal_mse = 0.003
samples = 2268
Group 0 = 1134 Group 1 = 1134
value = [[0.859]
[0.631]]
feature_4 <= -0.785
causal_mse = -0.378
samples = 4384
Group 0 = 2192 Group 1 = 2192
value = [[0.426]
[1.063]]
feature_1 <= 0.673
causal_mse = -6.23
samples = 1284
Group 0 = 642 Group 1 = 642
value = [[0.43]
[2.945]]
feature_11 <= 0.735
causal_mse = -1.541
samples = 1922
Group 0 = 961 Group 1 = 961
value = [[1.25]
[2.517]]
feature_7 <= -0.078
causal_mse = -0.91
samples = 832
Group 0 = 416 Group 1 = 416
value = [[1.145]
[0.119]]
feature_17 <= 0.439
causal_mse = 0.053
samples = 1436
Group 0 = 718 Group 1 = 718
value = [[0.694]
[0.928]]
feature_5 <= 0.062
causal_mse = 0.132
samples = 890
Group 0 = 445 Group 1 = 445
value = [[0.137]
[0.09]]
feature_0 <= 0.552
causal_mse = -0.623
samples = 3494
Group 0 = 1747 Group 1 = 1747
value = [[0.5]
[1.311]]
causal_mse = -8.401
samples = 652
Group 0 = 326 Group 1 = 326
value = [[-0.279]
[2.819]]
causal_mse = -3.468
samples = 632
Group 0 = 316 Group 1 = 316
value = [[1.16]
[3.076]]
feature_4 <= -0.133
causal_mse = -2.852
samples = 1450
Group 0 = 725 Group 1 = 725
value = [[1.107]
[2.821]]
causal_mse = 0.22
samples = 472
Group 0 = 236 Group 1 = 236
value = [[1.691]
[1.583]]
causal_mse = 0.2
samples = 402
Group 0 = 201 Group 1 = 201
value = [[0.774]
[0.461]]
causal_mse = -2.645
samples = 430
Group 0 = 215 Group 1 = 215
value = [[1.492]
[-0.219]]
feature_5 <= -0.129
causal_mse = -0.331
samples = 980
Group 0 = 490 Group 1 = 490
value = [[0.51]
[1.188]]
causal_mse = -0.24
samples = 456
Group 0 = 228 Group 1 = 228
value = [[1.089]
[0.37]]
causal_mse = 0.103
samples = 478
Group 0 = 239 Group 1 = 239
value = [[-0.284]
[0.08]]
causal_mse = 0.032
samples = 412
Group 0 = 206 Group 1 = 206
value = [[0.625]
[0.102]]
feature_19 <= -0.953
causal_mse = -0.296
samples = 2922
Group 0 = 1461 Group 1 = 1461
value = [[0.563]
[1.143]]
causal_mse = -3.738
samples = 572
Group 0 = 286 Group 1 = 286
value = [[0.179]
[2.167]]
causal_mse = -0.444
samples = 648
Group 0 = 324 Group 1 = 324
value = [[1.05]
[1.844]]
feature_9 <= -0.039
causal_mse = -5.885
samples = 802
Group 0 = 401 Group 1 = 401
value = [[1.153]
[3.61]]
causal_mse = 0.228
samples = 444
Group 0 = 222 Group 1 = 222
value = [[0.943]
[1.135]]
causal_mse = -0.919
samples = 536
Group 0 = 268 Group 1 = 268
value = [[0.151]
[1.232]]
causal_mse = 0.209
samples = 474
Group 0 = 237 Group 1 = 237
value = [[1.12]
[0.977]]
feature_15 <= -0.206
causal_mse = -0.469
samples = 2448
Group 0 = 1224 Group 1 = 1224
value = [[0.455]
[1.175]]
causal_mse = -6.888
samples = 402
Group 0 = 201 Group 1 = 201
value = [[0.887]
[3.571]]
causal_mse = -4.677
samples = 400
Group 0 = 200 Group 1 = 200
value = [[1.42]
[3.649]]
feature_0 <= -0.206
causal_mse = -1.518
samples = 1042
Group 0 = 521 Group 1 = 521
value = [[0.161]
[1.442]]
feature_0 <= -0.139
causal_mse = -0.01
samples = 1406
Group 0 = 703 Group 1 = 703
value = [[0.673]
[0.978]]
causal_mse = -3.867
samples = 626
Group 0 = 313 Group 1 = 313
value = [[-0.301]
[1.717]]
causal_mse = 0.269
samples = 416
Group 0 = 208 Group 1 = 208
value = [[0.856]
[1.028]]
feature_10 <= -0.094
causal_mse = 0.089
samples = 894
Group 0 = 447 Group 1 = 447
value = [[0.941]
[0.752]]
causal_mse = -1.12
samples = 512
Group 0 = 256 Group 1 = 256
value = [[0.205]
[1.371]]
causal_mse = -0.554
samples = 436
Group 0 = 218 Group 1 = 218
value = [[1.123]
[0.22]]
causal_mse = -0.007
samples = 458
Group 0 = 229 Group 1 = 229
value = [[0.767]
[1.259]]

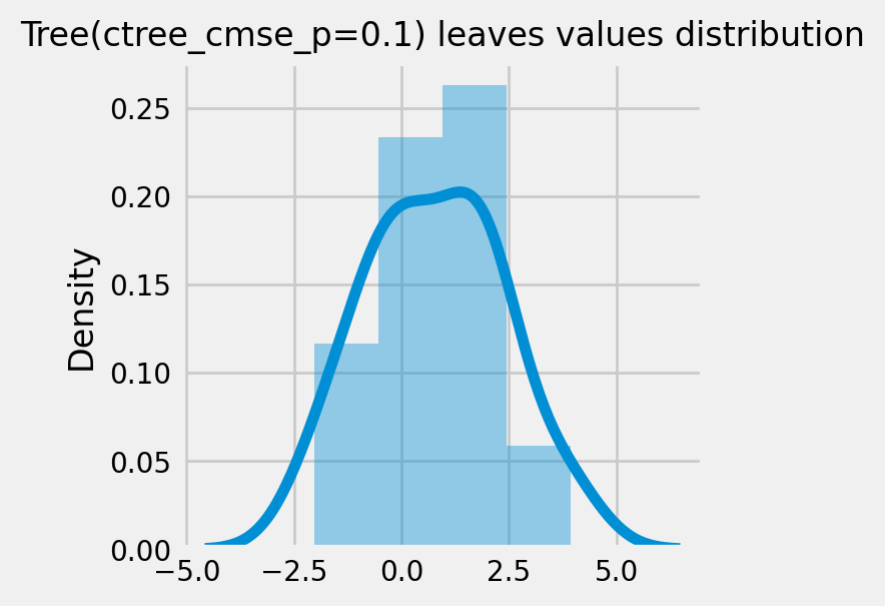


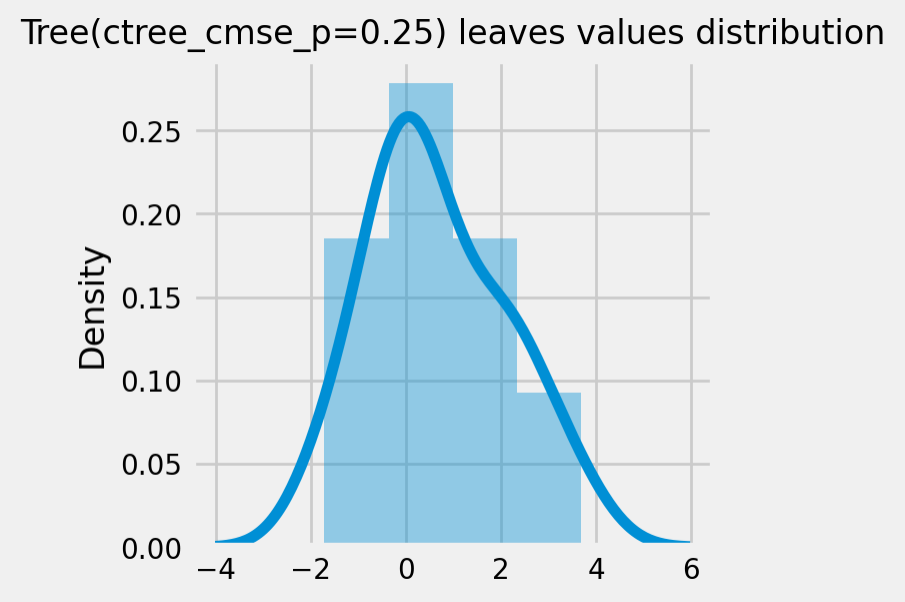
How values in leaves of the fitted trees differ from each other:

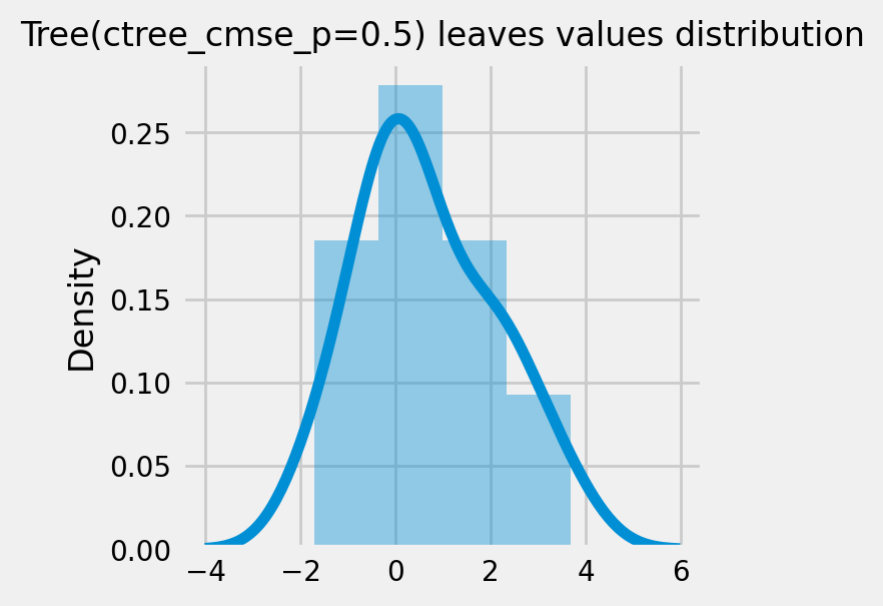
[18]:
```
for ctree_name, ctree_info in ctrees.items():
    plot_dist_tree_leaves_values(ctree_info['model'],
                                 figsize=(3,3),
                                 title=f'Tree({ctree_name}) leaves values distribution
↪')
```

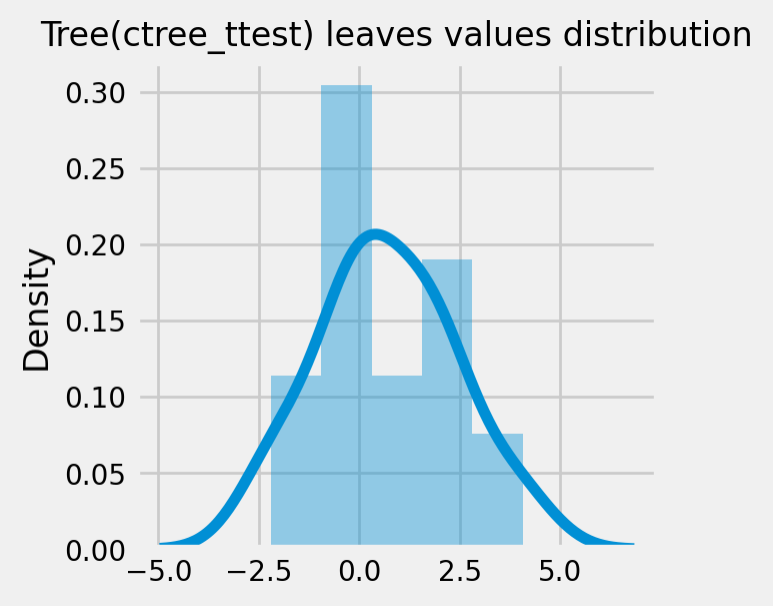

Tree(ctree_mse) leaves values distribution
Density
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4


Tree(ctree_cmse) leaves values distribution
Density
0.25
0.20
0.15
0.10
0.05
0.00
−5.0
−2.5
0.0
2.5
5.0


Tree(ctree_cmse_p=0.1) leaves values distribution
Density
0.25
0.20
0.15
0.10
0.05
0.00
−5.0
−2.5
0.0
2.5
5.0


Tree(ctree_cmse_p=0.25) leaves values distribution
Density
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4
6





5.17.2 CausalRandomForestRegressor

[19]:
```
cforests = {
    'cforest_mse': {
        'params':
        dict(criterion='standard_mse',
             control_name=0,
             min_impurity_decrease=0,
             min_samples_leaf=400,
             groups_penalty=0.,
             groups_cnt=True),
    },
```

(continues on next page)

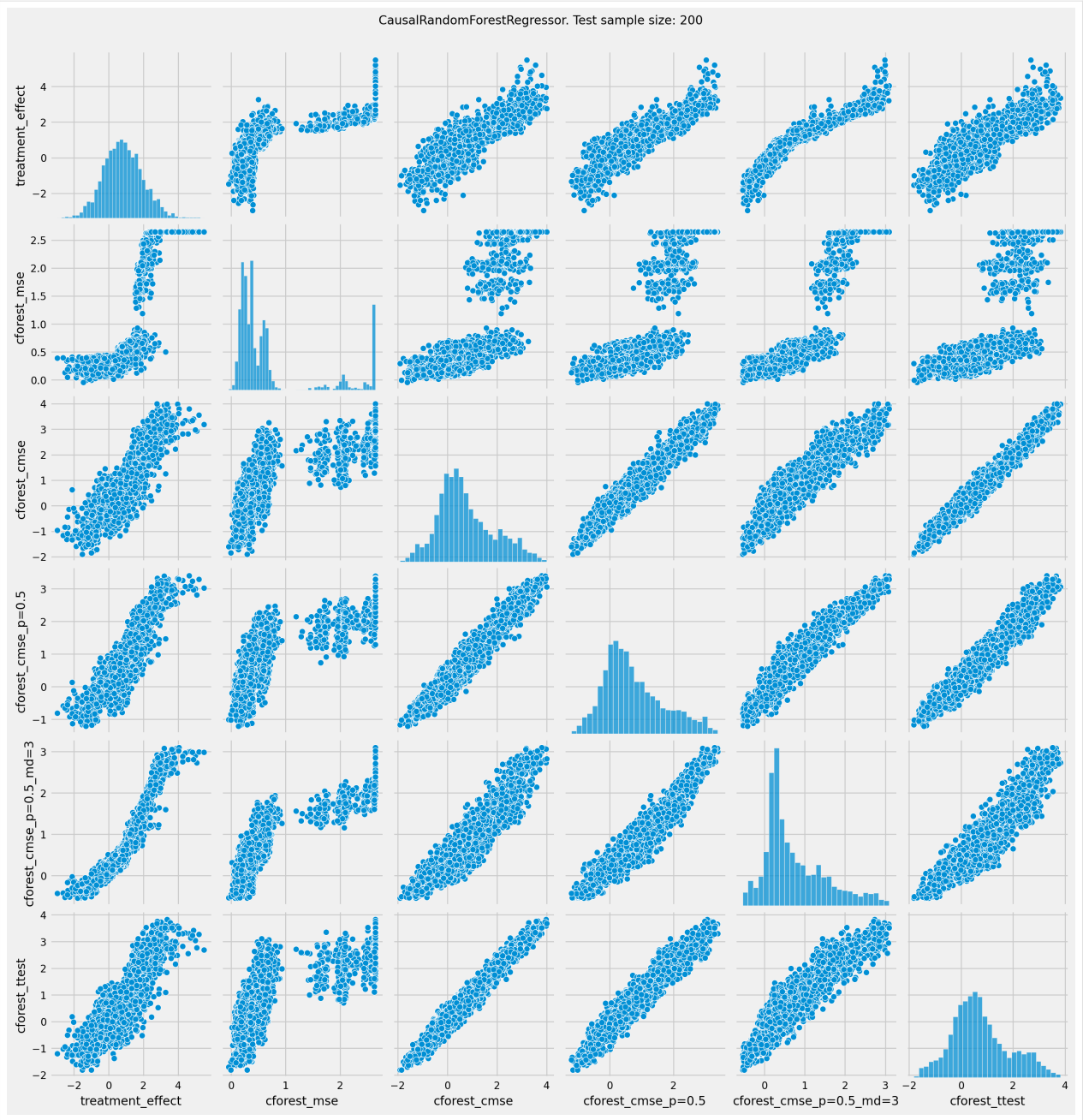
(continued from previous page)

```
    'cforest_cmse': {
        'params':
        dict(
            criterion='causal_mse',
            control_name=0,
            min_samples_leaf=400,
            groups_penalty=0.,
            groups_cnt=True
        ),
    },
    'cforest_cmse_p=0.5': {
        'params':
        dict(
            criterion='causal_mse',
            control_name=0,
            min_samples_leaf=400,
            groups_penalty=0.5,
            groups_cnt=True,
        ),
    },
    'cforest_cmse_p=0.5_md=3': {
        'params':
        dict(
            criterion='causal_mse',
            control_name=0,
            max_depth=3,
            min_samples_leaf=400,
            groups_penalty=0.5,
            groups_cnt=True,
        ),
    },
    'cforest_ttest': {
        'params':
        dict(criterion='t_test',
             control_name=0,
             min_samples_leaf=400,
             groups_penalty=0.,
             groups_cnt=True),
    },
}
```

[20]:
```
# Model treatment effect
for cforest_name, cforest_info in cforests.items():
    print(f"Fitting: {cforest_name}")
    cforest = CausalRandomForestRegressor(**cforest_info['params'])
    cforest.fit(X=df_train[feature_names].values,
                treatment=df_train['treatment'].values,
                y=df_train['outcome'].values)

    cforests[cforest_name].update({'model': cforest})
    df_result[cforest_name] = cforest.predict(df_test[feature_names].values)
```

```
Fitting: cforest_mse
Fitting: cforest_cmse
Fitting: cforest_cmse_p=0.5
Fitting: cforest_cmse_p=0.5_md=3
Fitting: cforest_ttest
```

[21]:
```
# See treatment effect estimation with CausalRandomForestRegressor vs true treatment effect

n_obs = 200

indxs = df_result.index.values
np.random.shuffle(indxs)
indxs = indxs[:n_obs]

plt.rcParams.update({'font.size': 10})
pairplot = sns.pairplot(df_result[['treatment_effect', *list(cforests)]])
pairplot.fig.suptitle(f"CausalRandomForestRegressor. Test sample size: {n_obs}" , y=1.02)
plt.show()
```



[22]:
```
df_qini = qini_score(df_result,
            outcome_col='outcome',
            treatment_col='is_treated',
            treatment_effect_col='treatment_effect')

df_qini.sort_values(ascending=False)
```

[22]:
```
cforest_cmse_p=0.5_md=3    0.379598
cforest_cmse_p=0.5         0.344870
cforest_ttest              0.329592
cforest_mse                0.326770
cforest_cmse               0.323620
```

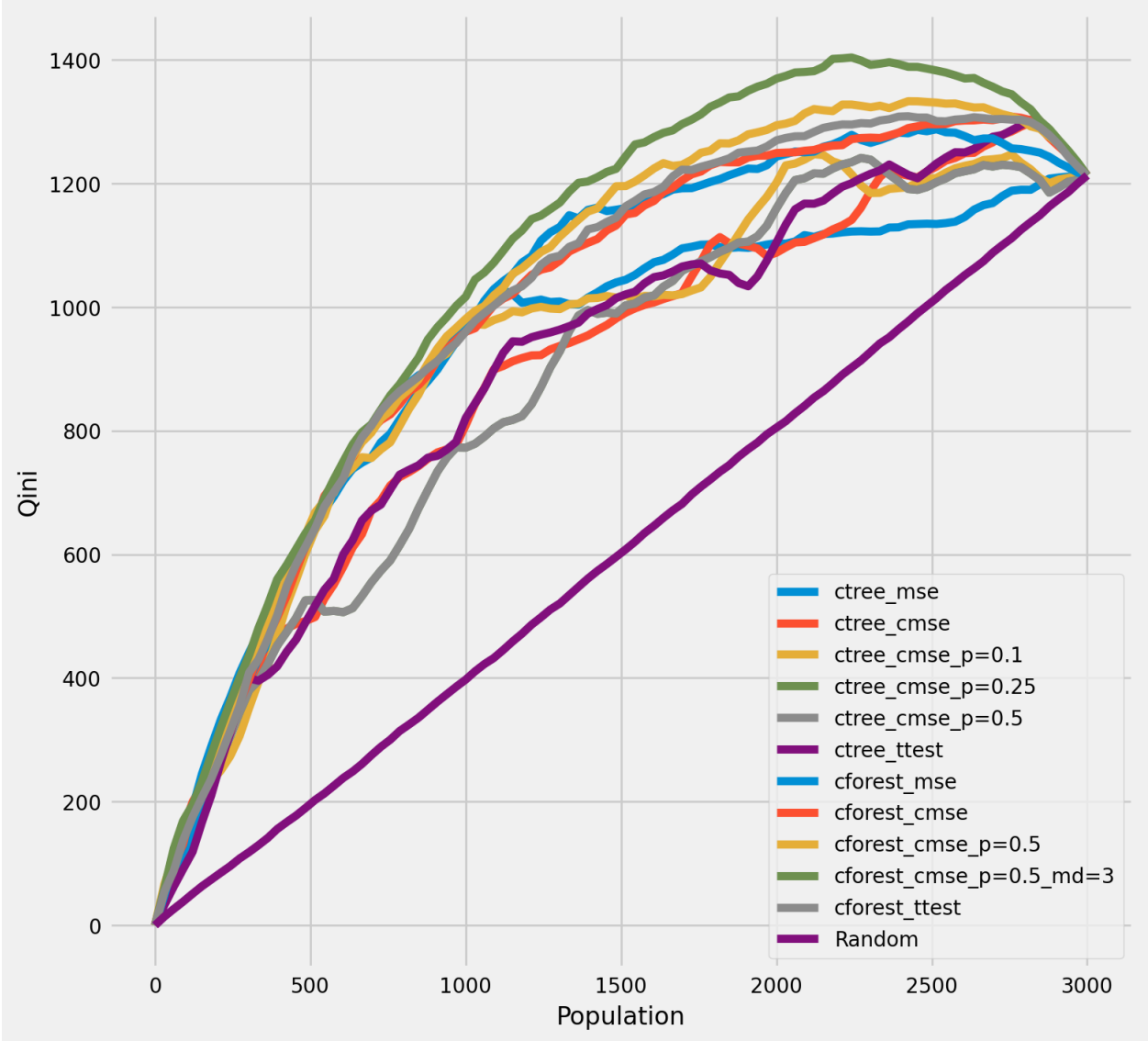
(continues on next page)

(continued from previous page)

```
ctree_cmse_p=0.1           0.273112
ctree_mse                  0.260141
ctree_ttest                0.247668
ctree_cmse                 0.242333
ctree_cmse_p=0.25          0.231232
ctree_cmse_p=0.5           0.231232
Random                     0.000000
dtype: float64
```

Qini chart

[23]:
```
plot_qini(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          treatment_effect_col='treatment_effect',
          figsize=(8,8)
         )
```



[24]:
```
df_qini = qini_score(df_result,
           outcome_col='outcome',
           treatment_col='is_treated',
           treatment_effect_col='treatment_effect')

df_qini.sort_values(ascending=False)
```

[24]:
```
cforest_cmse_p=0.5_md=3    0.379598
cforest_cmse_p=0.5         0.344870
cforest_ttest              0.329592
cforest_mse                0.326770
cforest_cmse               0.323620
ctree_cmse_p=0.1           0.273112
ctree_mse                  0.260141
ctree_ttest                0.247668
ctree_cmse                 0.242333
ctree_cmse_p=0.25          0.231232
ctree_cmse_p=0.5           0.231232
```

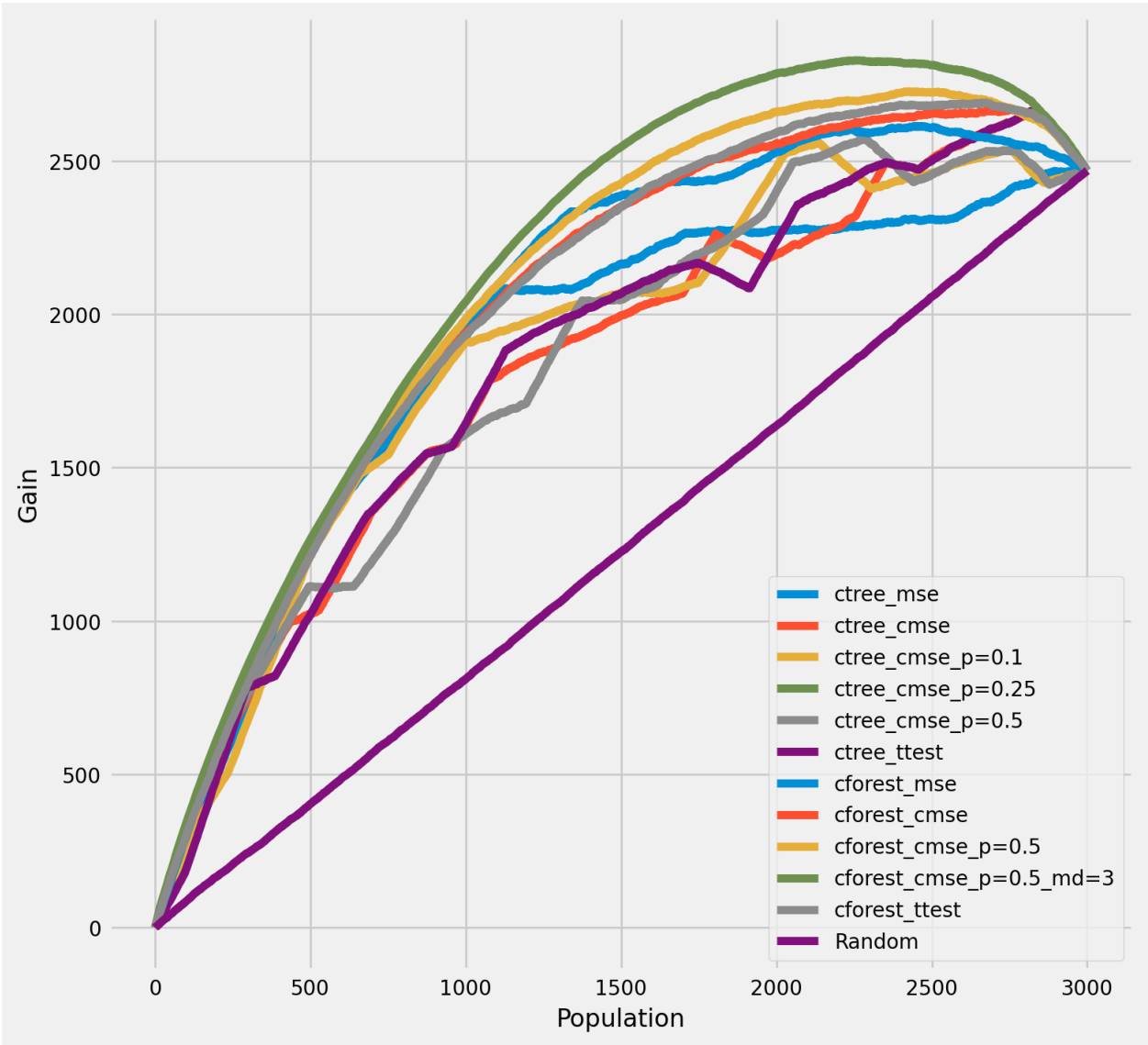
(continues on next page)

(continued from previous page)

```
Random                      0.000000
dtype: float64
```

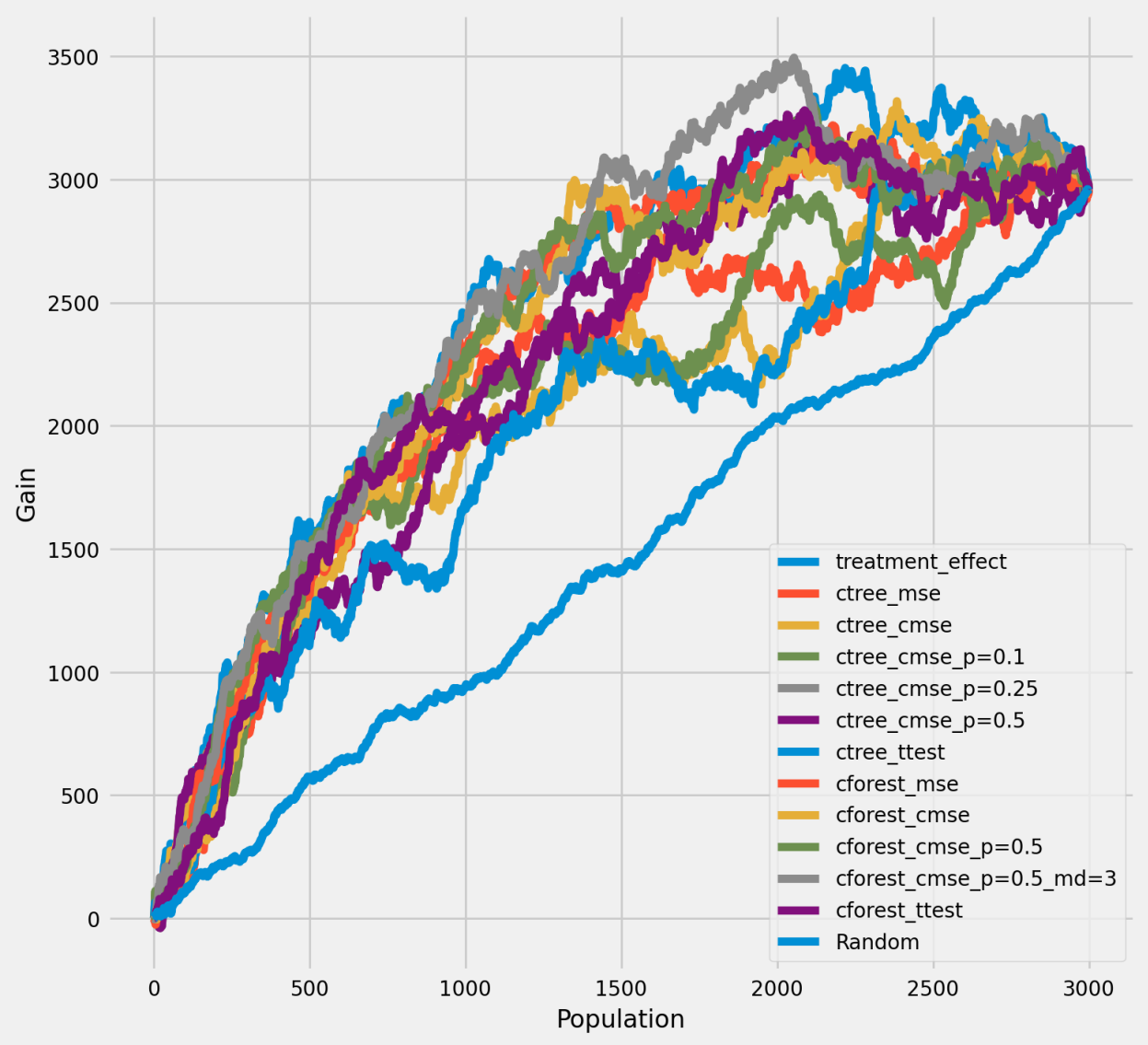
The cumulative gain of the true treatment effect in each population

[25]:
```
plot_gain(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          treatment_effect_col='treatment_effect',
          n = n_test
         )
```



The cumulative difference between the mean outcomes of the treatment and control groups in each population

[26]:
```
plot_gain(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          n = n_test
         )
```

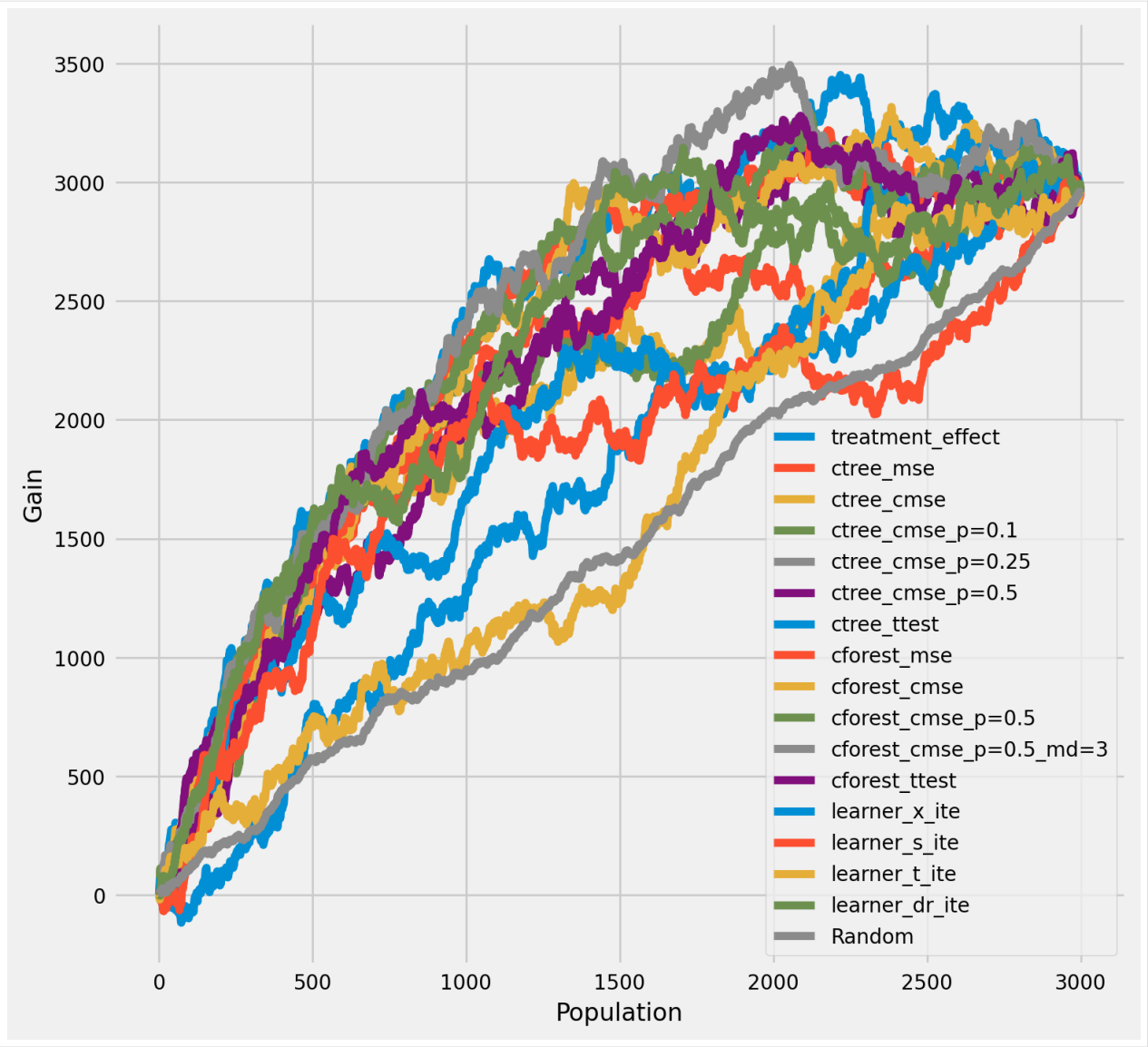


5.17.3 Meta-Learner Algorithms

[27]:
```
X_train = df_train[feature_names].values
X_test = df_test[feature_names].values

# learner = DecisionTreeRegressor
# treatment learner = LinearRegression()

learner_x = BaseXRegressor(learner=DecisionTreeRegressor(),
                           treatment_effect_learner=LinearRegression())
learner_s = BaseSRegressor(learner=DecisionTreeRegressor())
learner_t = BaseTRegressor(learner=DecisionTreeRegressor(),
                           treatment_learner=LinearRegression())
learner_dr = BaseDRRegressor(learner=DecisionTreeRegressor(),
                             treatment_effect_learner=LinearRegression())

learner_x.fit(X=X_train, treatment=df_train['treatment'].values, y=df_train['outcome
↪'].values)
learner_s.fit(X=X_train, treatment=df_train['treatment'].values, y=df_train['outcome
↪'].values)
learner_t.fit(X=X_train, treatment=df_train['treatment'].values, y=df_train['outcome
↪'].values)
learner_dr.fit(X=X_train, treatment=df_train['treatment'].values, y=df_train['outcome
↪'].values)

df_result['learner_x_ite'] = learner_x.predict(X_test)
df_result['learner_s_ite'] = learner_s.predict(X_test)
df_result['learner_t_ite'] = learner_t.predict(X_test)
df_result['learner_dr_ite'] = learner_dr.predict(X_test)
```

[28]:
```
cate_dr = learner_dr.predict(X)
cate_x = learner_x.predict(X)
cate_s = learner_s.predict(X)
cate_t = learner_t.predict(X)

cate_ctrees = [info['model'].predict(X) for _, info in ctrees.items()]
cate_cforests = [info['model'].predict(X) for _, info in cforests.items()]

model_cate = [
    *cate_ctrees,
    *cate_cforests,
    cate_x, cate_s, cate_t, cate_dr
]

model_names = [
    *list(ctrees), *list(cforests),
    'X Learner', 'S Learner', 'T Learner', 'DR Learner']
```

[29]:
```
plot_gain(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          n = n_test
         )
```



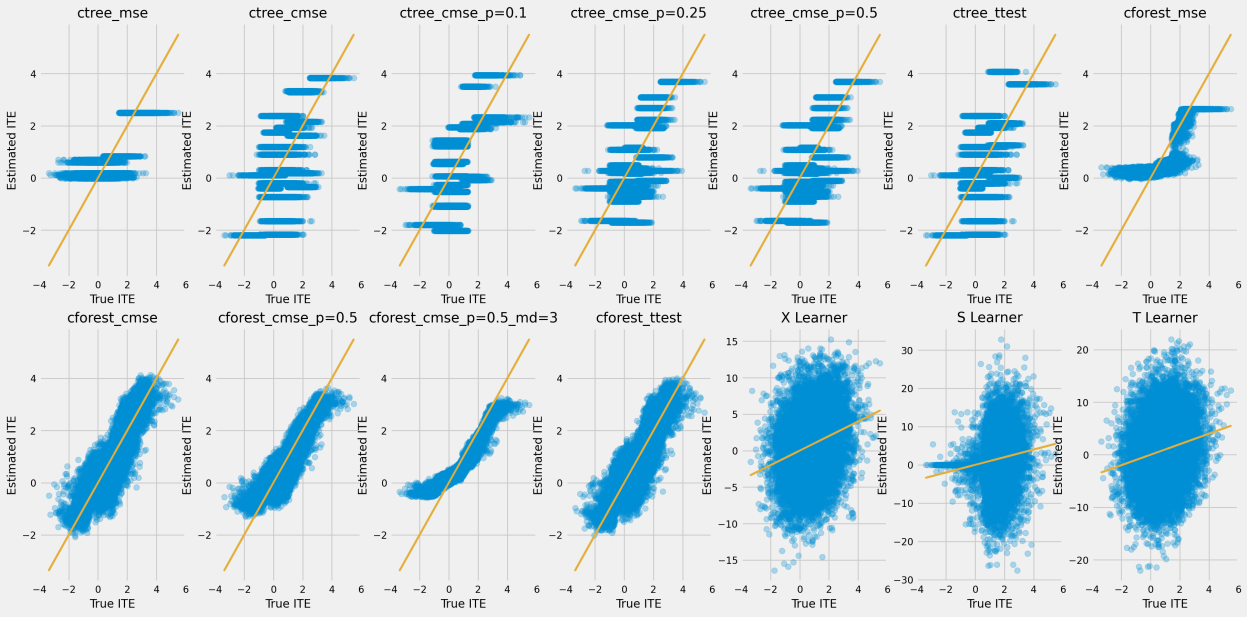
[30]:
```
rows = 2
cols = 7
row_idxs = np.arange(rows)
col_idxs = np.arange(cols)

ax_idxs = np.dstack(np.meshgrid(col_idxs, row_idxs)).reshape(-1, 2)
```

[31]:
```
fig, ax = plt.subplots(rows, cols, figsize=(20, 10))
plt.rcParams.update({'font.size': 10})

for ax_idx, cate, model_name in zip(ax_idxs, model_cate, model_names):
    col_id, row_id = ax_idx
    cur_ax = ax[row_id, col_id]
    cur_ax.scatter(tau, cate, alpha=0.3)
    cur_ax.plot(tau, tau, color='C2', linewidth=2)
    cur_ax.set_xlabel('True ITE')
    cur_ax.set_ylabel('Estimated ITE')
```

(continues on next page)

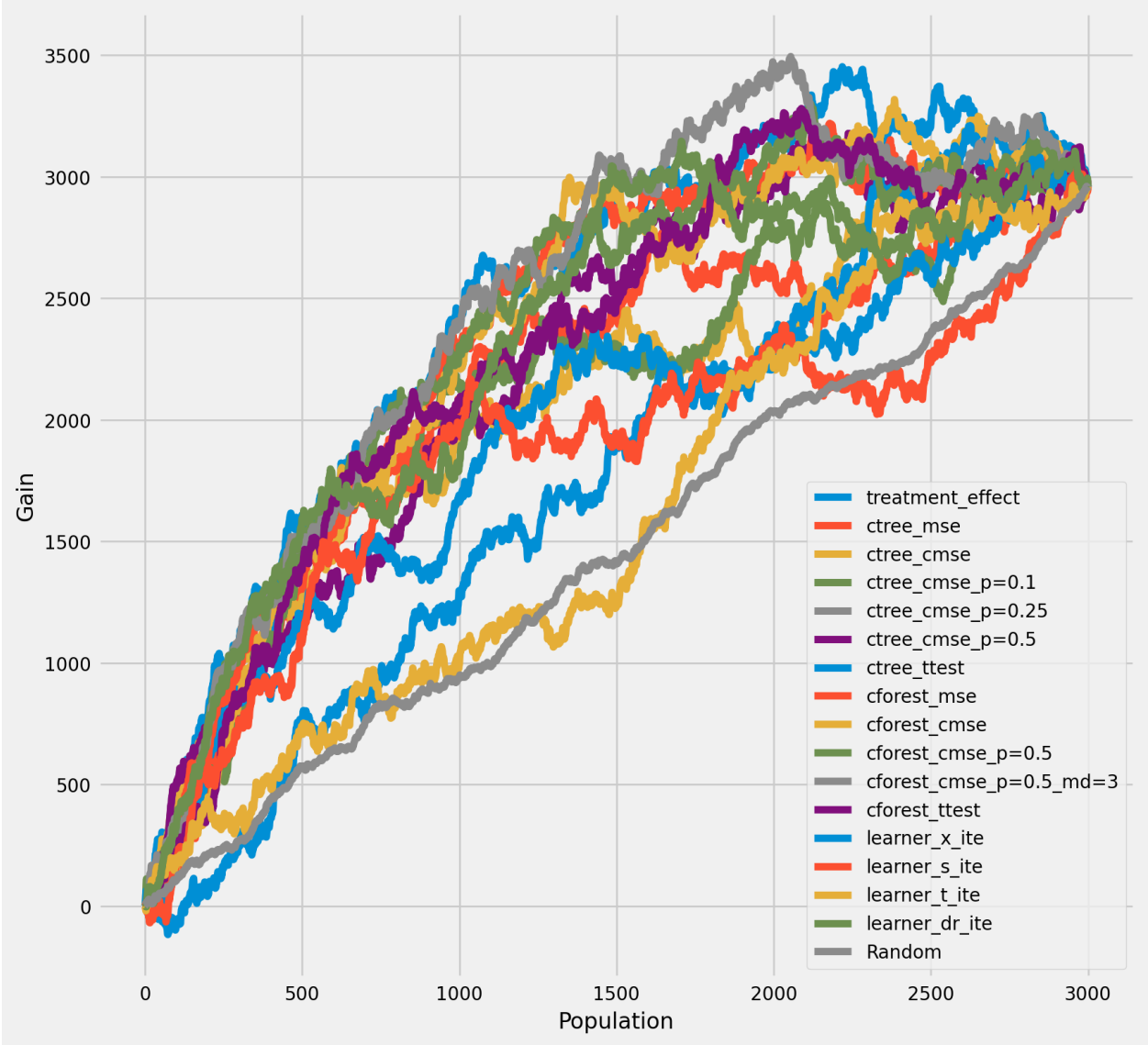
(continued from previous page)

```
    cur_ax.set_title(model_name)
    cur_ax.set_xlim((-4, 6))
```



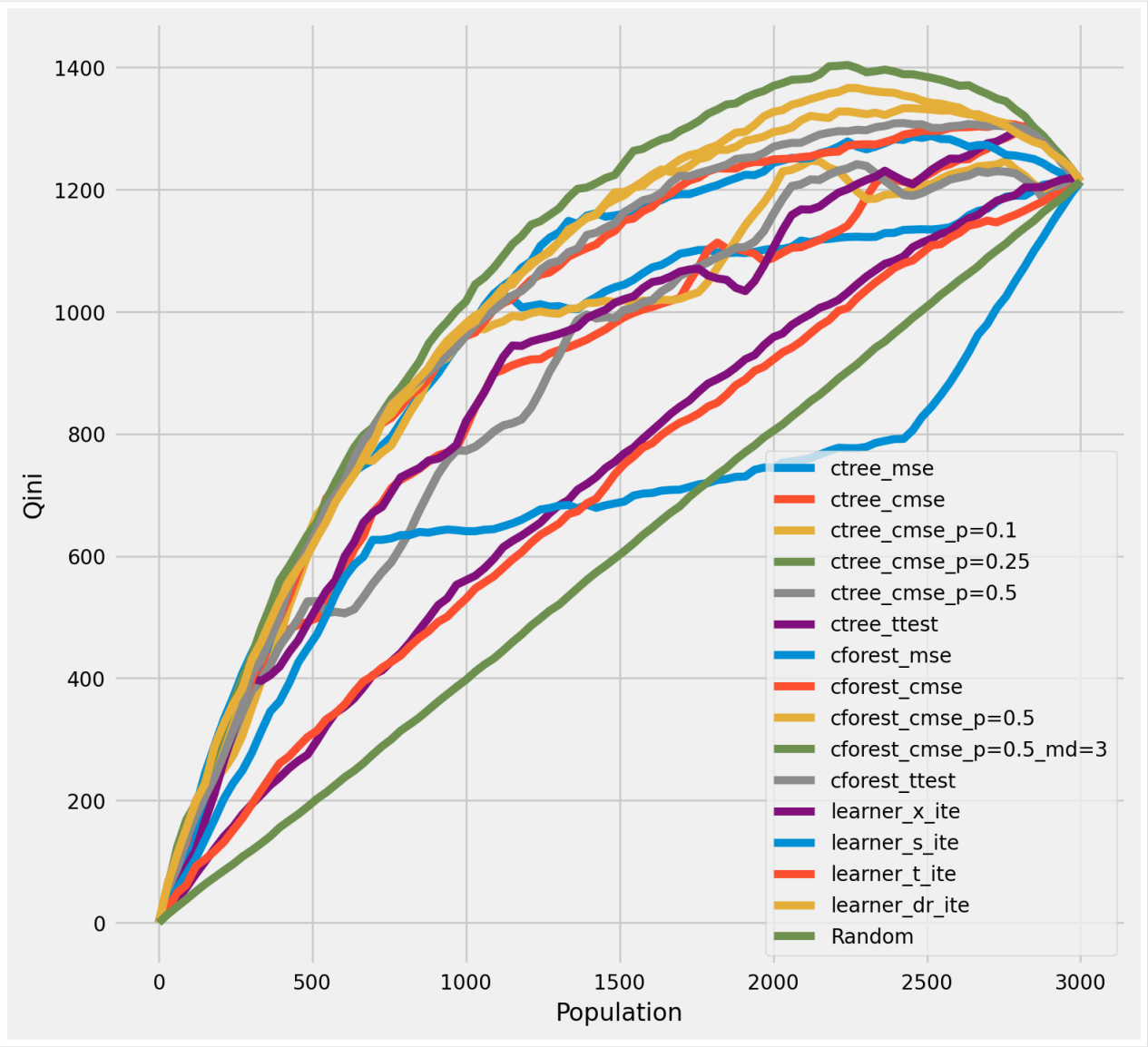
The cumulative difference between the mean outcomes of the treatment and control groups in each population

[32]:
```
plot_gain(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          n = n_test,
          figsize=(9, 9),
         )
```



Qini chart

[33]:
```
plot_qini(df_result,
          outcome_col='outcome',
          treatment_col='is_treated',
          treatment_effect_col='treatment_effect',
         )
```



[34]:
```
df_qini = qini_score(df_result,
           outcome_col='outcome',
           treatment_col='is_treated',
           treatment_effect_col='treatment_effect')
df_qini.sort_values(ascending=False)
```

[34]:
```
cforest_cmse_p=0.5_md=3    0.379598
learner_dr_ite             0.350053
cforest_cmse_p=0.5         0.344870
cforest_ttest              0.329592
cforest_mse                0.326770
cforest_cmse               0.323620
ctree_cmse_p=0.1           0.273112
ctree_mse                  0.260141
ctree_ttest                0.247668
ctree_cmse                 0.242333
ctree_cmse_p=0.25          0.231232
ctree_cmse_p=0.5           0.231232
```

(continues on next page)

(continued from previous page)

```
learner_x_ite              0.096785
learner_t_ite              0.082948
learner_s_ite              0.055251
Random                     0.000000
dtype: float64
```

5.17.4 Return outcomes along with estimated treatment effects

[35]:
```
ctree_outcomes = ctrees["ctree_mse"]["model"].predict(X_test, with_outcomes=True)
df_ctree_outcomes = pd.DataFrame(ctree_outcomes, columns=["Y0", "Y1", "ITE"])
df_ctree_outcomes.head()
```

[35]:

	Y0	Y1	ITE
0	0.434715	0.545400	0.110685
1	1.825205	2.008497	0.183293
2	0.453153	1.287316	0.834163
3	1.825205	2.008497	0.183293
4	1.825205	2.008497	0.183293

[36]:
```
cforest_outcomes = cforests["cforest_mse"]["model"].predict(X_test, with_
↪outcomes=True)
df_cforest_outcomes = pd.DataFrame(cforest_outcomes, columns=["Y0", "Y1", "ITE"])
df_cforest_outcomes.head()
```

[36]:

	Y0	Y1	ITE
0	0.390998	0.575902	0.184904
1	1.445800	1.852422	0.406622
2	0.385520	0.982415	0.596895
3	1.279227	1.731512	0.452285
4	1.279100	1.730838	0.451738

5.17.5 Bootstrap confidence intervals for individual treatment effects

[37]:
```
alpha=0.05
tree = CausalTreeRegressor(criterion='causal_mse', control_name=0, min_samples_
↪leaf=200, alpha=alpha)
```

[38]:
```
# For time measurements
for n_jobs in (4, mp.cpu_count() - 1):
    for n_bootstraps in (10, 50, 100):
        print(f"n_jobs: {n_jobs} n_bootstraps: {n_bootstraps}" )
        tree.bootstrap_pool(
            X=X,
            treatment=w,
            y=y,
            n_bootstraps=n_bootstraps,
            bootstrap_size=10000,
            n_jobs=n_jobs,
            verbose=False
        )
```

```
n_jobs: 4 n_bootstraps: 10
```

```
100
↪%|████████████████████████████████████████████████████████████████████████████████
↪10/10 [00:00<00:00, 20.81it/s]
```

```
Function: bootstrap_pool Kwargs:{'n_bootstraps': 10, 'bootstrap_size': 10000, 'n_jobs
↪': 4, 'verbose': False} Elapsed time: 0.9135
n_jobs: 4 n_bootstraps: 50
```

```
100
↪%|████████████████████████████████████████████████████████████████████████████████
↪50/50 [00:02<00:00, 21.29it/s]
```

```
Function: bootstrap_pool Kwargs:{'n_bootstraps': 50, 'bootstrap_size': 10000, 'n_jobs
↪': 4, 'verbose': False} Elapsed time: 2.4252
n_jobs: 4 n_bootstraps: 100
```

```
100
↪%|████████████████████████████████████████████████████████████████████████████████
↪100/100 [00:04<00:00, 21.70it/s]
```

```
Function: bootstrap_pool Kwargs:{'n_bootstraps': 100, 'bootstrap_size': 10000, 'n_jobs
↪': 4, 'verbose': False} Elapsed time: 4.6932
n_jobs: 11 n_bootstraps: 10
```

```
100
↪%|████████████████████████████████████████████████████████████████████████████████
↪10/10 [00:00<00:00, 32.93it/s]
```

```
Function: bootstrap_pool Kwargs:{'n_bootstraps': 10, 'bootstrap_size': 10000, 'n_jobs
↪': 11, 'verbose': False} Elapsed time: 0.6439
n_jobs: 11 n_bootstraps: 50
```

```
100
↪%|████████████████████████████████████████████████████████████████████████████████
↪50/50 [00:01<00:00, 30.42it/s]
```

```
Function: bootstrap_pool Kwargs:{'n_bootstraps': 50, 'bootstrap_size': 10000, 'n_jobs
↪': 11, 'verbose': False} Elapsed time: 1.8508
n_jobs: 11 n_bootstraps: 100
```

```
100
↪%|████████████████████████████████████████████████████████████████████████████████
↪100/100 [00:02<00:00, 33.67it/s]
```

```
Function: bootstrap_pool Kwargs:{'n_bootstraps': 100, 'bootstrap_size': 10000, 'n_jobs
↪': 11, 'verbose': False} Elapsed time: 3.1553
```

[39]:
```
te, te_lower, te_upper = tree.fit_predict(
        X=df_train[feature_names].values,
        treatment=df_train["treatment"].values,
        y=df_train["outcome"].values,
        return_ci=True,
        n_bootstraps=500,
        bootstrap_size=5000,
```

(continues on next page)

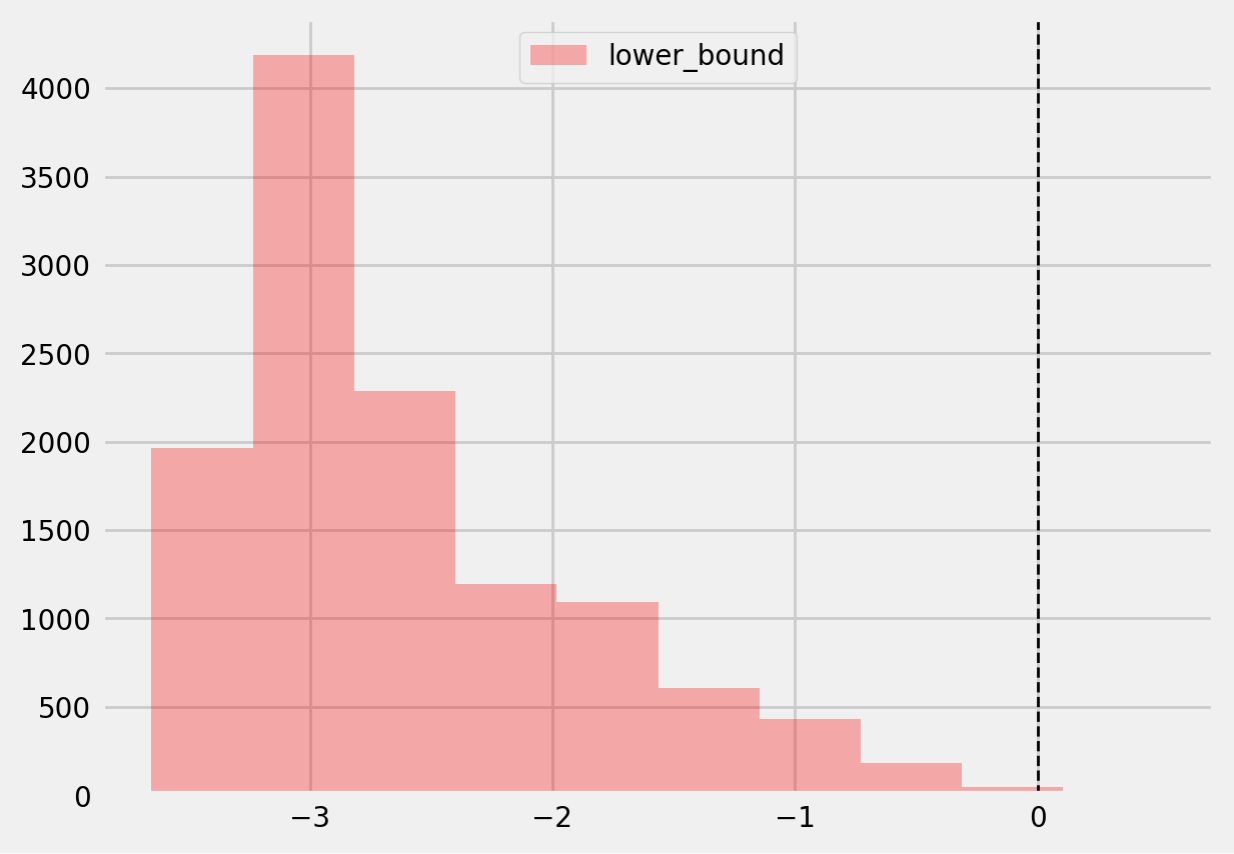
(continued from previous page)

```
        n_jobs=mp.cpu_count() - 1,
        verbose=False)
```

```
100
↪%|████████████████████████████████████████████████████████████████████████████████
↪500/500 [00:06<00:00, 81.19it/s]
```

```
Function: bootstrap_pool Kwargs:{'n_bootstraps': 500, 'bootstrap_size': 5000, 'n_jobs
↪': 11, 'verbose': False} Elapsed time: 6.4025
```

[40]:
```
plt.hist(te_lower, color='red', alpha=0.3, label='lower_bound')
plt.axvline(x = 0, color = 'black', linestyle='--', lw=1, label='')
plt.legend()
plt.show()
```



[41]:
```
# Significant estimates for negative and positive individual effects
# Default alpha = 0.05

bootstrap_neg = te[(te_lower < 0) & (te_upper < 0)]
bootstrap_pos = te[(te_lower > 0) & (te_upper > 0)]
print(bootstrap_neg.shape, bootstrap_pos.shape)
```

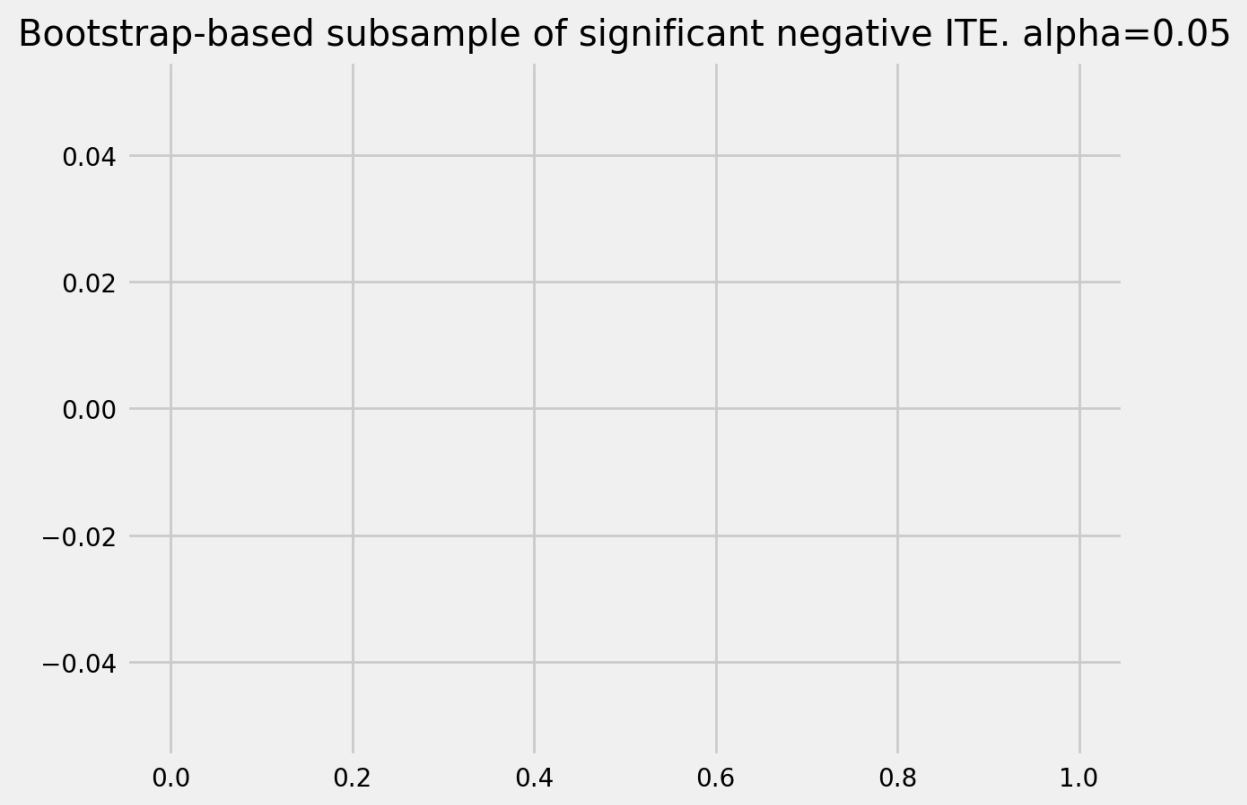
```
(0,) (7,)
```

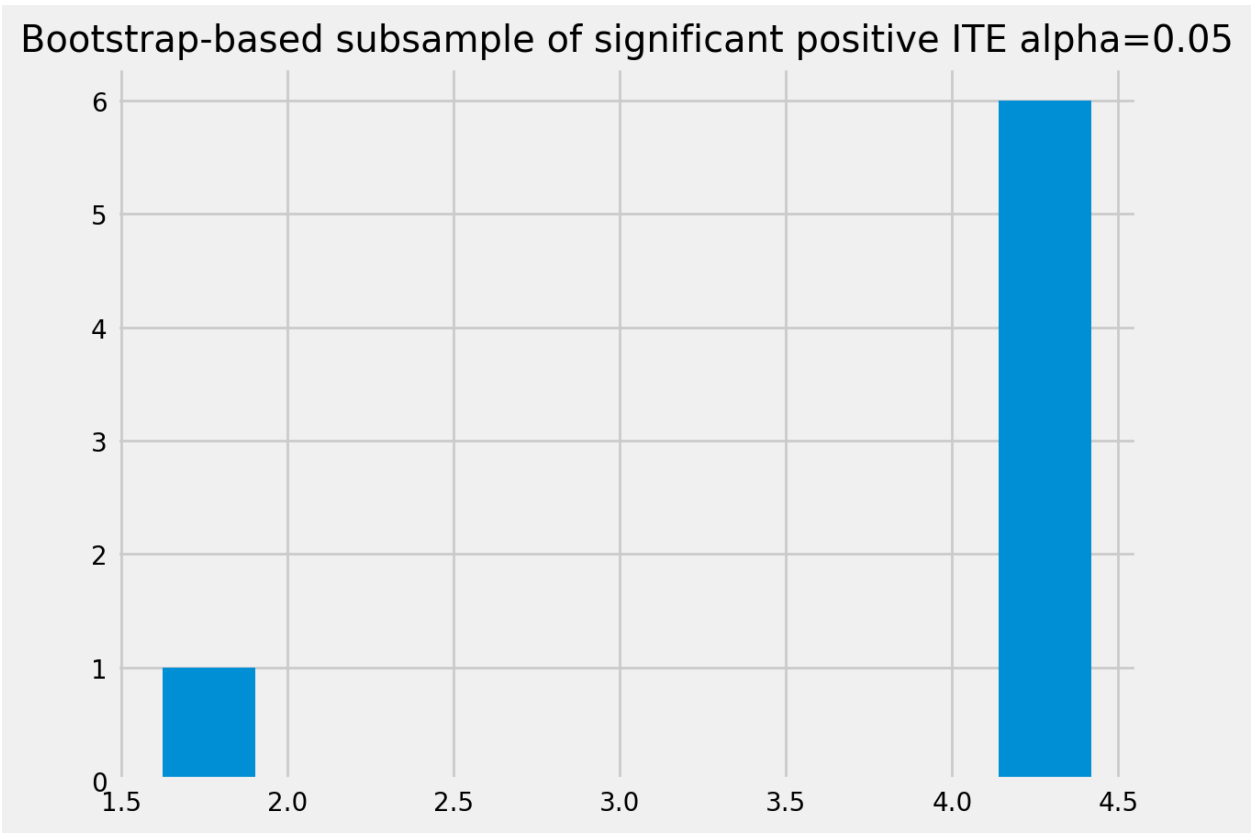
[42]:
```
plt.hist(bootstrap_neg)
plt.title(f'Bootstrap-based subsample of significant negative ITE. alpha={alpha}')
plt.show()
```

(continues on next page)

(continued from previous page)

```
plt.hist(bootstrap_pos)
plt.title(f'Bootstrap-based subsample of significant positive ITE alpha={alpha}')
plt.show()
```





5.17.6 Average treatment effect

[43]:
```
tree = CausalTreeRegressor(criterion='causal_mse', control_name=0, min_samples_
↪leaf=200, alpha=alpha)
te, te_lb, te_ub = tree.estimate_ate(X=X, treatment=w, y=y)
print('ATE:', te, 'Bounds:', (te_lb, te_ub ), 'alpha:', alpha)
```

```
ATE: 0.80899590297471 Bounds: (0.808752005241054, 0.8092398007083661) alpha: 0.05
```

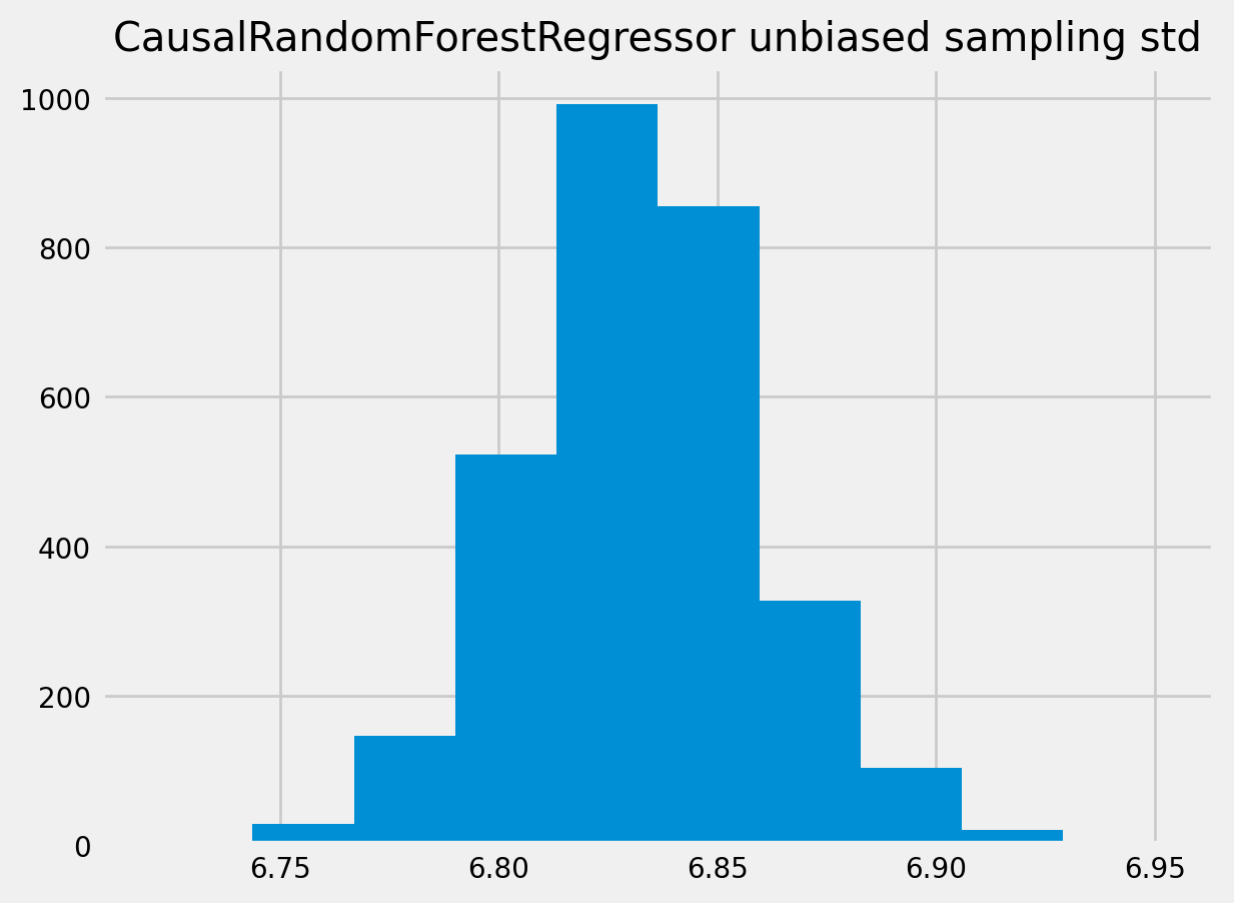
5.17.7 CausalRandomForestRegressor ITE std

[44]:
```
crforest = CausalRandomForestRegressor(criterion="causal_mse",  min_samples_leaf=200,
                                       control_name=0, n_estimators=50, n_jobs=mp.cpu_
↪count()-1)
crforest.fit(X=df_train[feature_names].values,
             treatment=df_train['treatment'].values,
             y=df_train['outcome'].values
             )
```

[44]:
```
CausalRandomForestRegressor(min_samples_leaf=200, n_estimators=50, n_jobs=11)
```

[45]:
```
crforest_te_pred = crforest.predict(df_test[feature_names])
crforest_test_var = crforest.calculate_error(X_train=df_train[feature_names].values,
                                             X_test=df_test[feature_names].values)
crforest_test_std = np.sqrt(crforest_test_var)
```

[46]:
```
plt.hist(crforest_test_std)
plt.title("CausalRandomForestRegressor unbiased sampling std")
plt.show()
```



[ ]:

[ ]:

5.18 Causal Trees/Forests Interpretation with Feature Importance and SHAP Values

[1]:
```
import pandas as pd
import numpy as np
import multiprocessing as mp
np.random.seed(42)

from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance
import shap

import causalml
from causalml.metrics import plot_gain, plot_qini, qini_score
```

(continues on next page)

(continued from previous page)

```
from causalml.dataset import synthetic_data
from causalml.inference.tree import plot_dist_tree_leaves_values, get_tree_leaves_mask
from causalml.inference.tree import CausalRandomForestRegressor, CausalTreeRegressor
from causalml.inference.tree.utils import timeit

import matplotlib.pyplot as plt
import seaborn as sns

%config InlineBackend.figure_format = 'retina'
```

```
Failed to import duecredit due to No module named 'duecredit'
```

[2]:
```
import importlib
for libname in ["causalml", "shap"]:
    print(f"{libname}: {importlib.metadata.version(libname)}")
```

```
causalml: 0.14.1
shap: 0.42.1
```

[3]:
```
# Simulate randomized trial: mode=2
y, X, w, tau, b, e = synthetic_data(mode=2, n=2000, p=10, sigma=3.0)

df = pd.DataFrame(X)
feature_names = [f'feature_{i}' for i in range(X.shape[1])]
df.columns = feature_names
df['outcome'] = y
df['treatment'] = w
df['treatment_effect'] = tau
```

[4]:
```
# Split data to training and testing samples for model validation
df_train, df_test = train_test_split(df, test_size=0.2, random_state=111)
n_train, n_test = df_train.shape[0], df_test.shape[0]

X_train, y_train = df_train[feature_names], df_train['outcome'].values
X_test, y_test = df_test[feature_names], df_test['outcome'].values
treatment_train, treatment_test = df_train['treatment'].values, df_test['treatment'].
↪values
effect_test = df_test['treatment_effect'].values

observation = X_test.loc[[0]]
```

5.18.1 CausalTreeRegressor

[5]:
```
ctree = CausalTreeRegressor()
ctree.fit(X=X_train.values, y=y_train, treatment=treatment_train)
```

[5]:
```
CausalTreeRegressor()
```

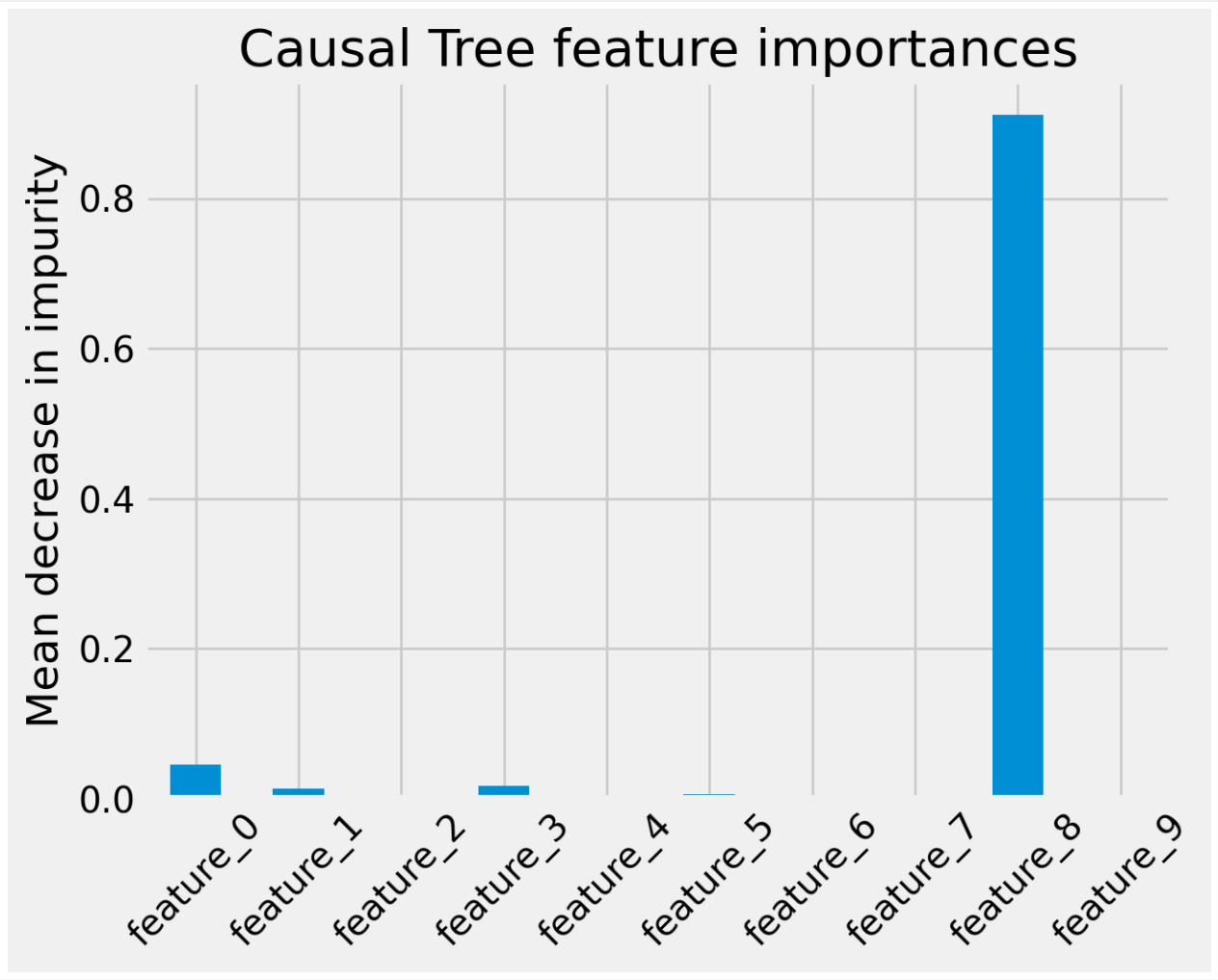
5.18.2 CausalRandomForestRegressor

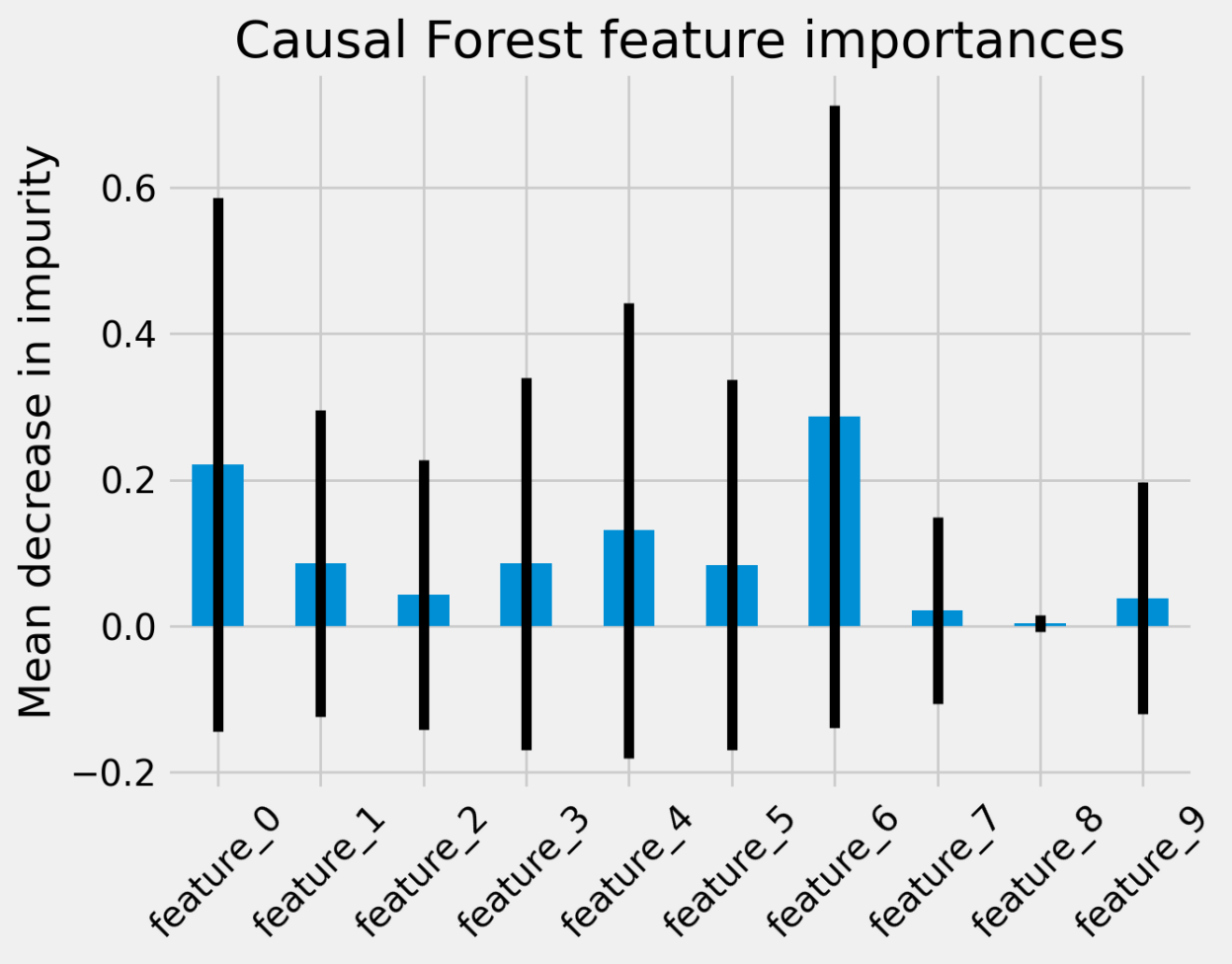
[6]:
```
crforest = CausalRandomForestRegressor(criterion="causal_mse",
                                       min_samples_leaf=200,
                                       control_name=0,
                                       n_estimators=50,
                                       n_jobs=mp.cpu_count() - 1)
crforest.fit(X=X_train, y=y_train, treatment=treatment_train)
```

[6]:
```
CausalRandomForestRegressor(min_samples_leaf=200, n_estimators=50, n_jobs=11)
```

1. Impurity-based feature importance

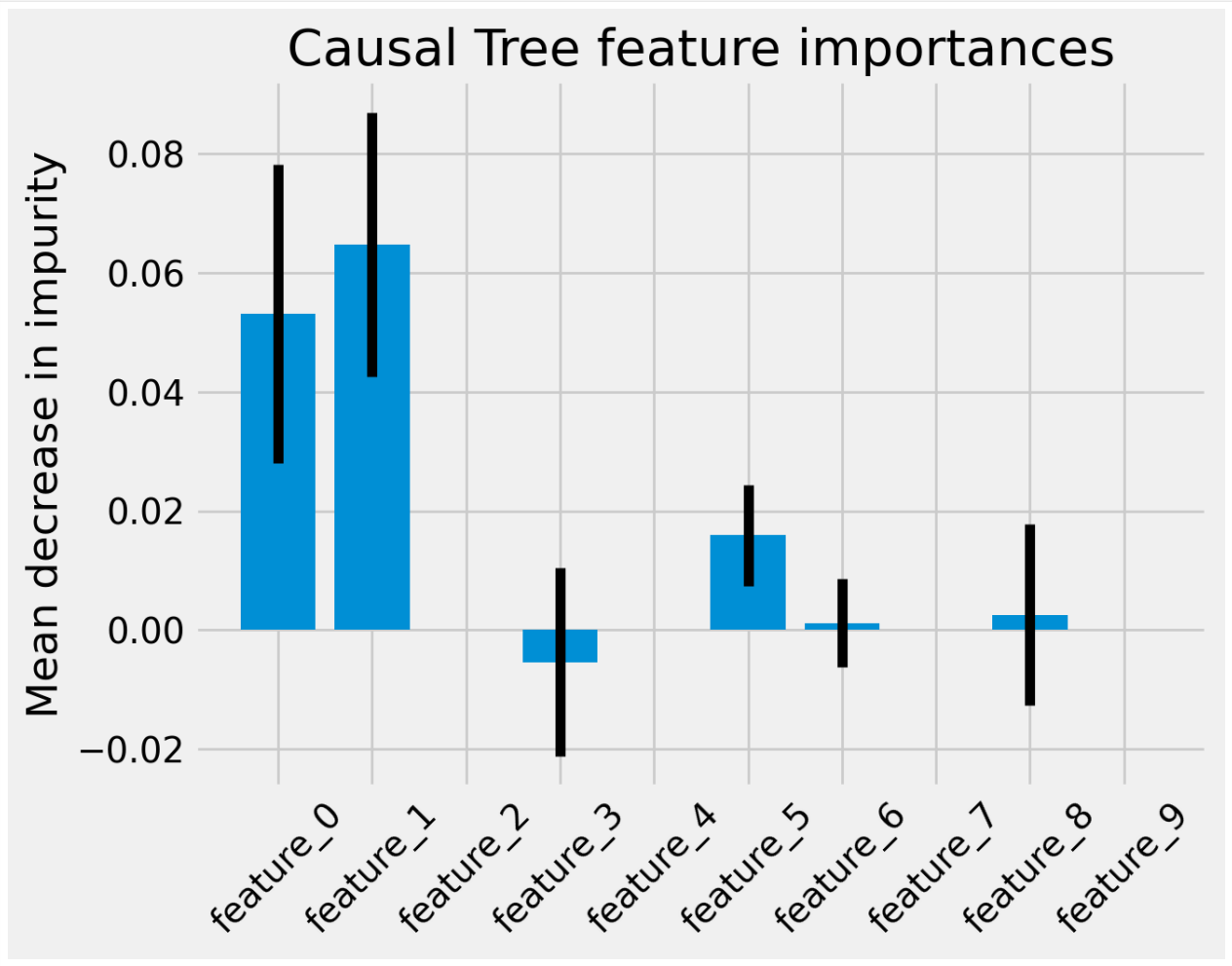
[7]:
```
df_importances = pd.DataFrame({'tree': ctree.feature_importances_,
                               'forest': crforest.feature_importances_,
                               'feature': feature_names
                              })
forest_std = np.std([tree.feature_importances_ for tree in crforest.estimators_],␣
↪axis=0)

fig, ax = plt.subplots()
df_importances['tree'].plot.bar(ax=ax)
ax.set_title("Causal Tree feature importances")
ax.set_ylabel("Mean decrease in impurity")
ax.set_xticklabels(feature_names, rotation=45)
plt.show()

fig, ax = plt.subplots()
df_importances['forest'].plot.bar(yerr=forest_std, ax=ax)
ax.set_title("Causal Forest feature importances")
ax.set_ylabel("Mean decrease in impurity")
ax.set_xticklabels(feature_names, rotation=45)
plt.show()
```

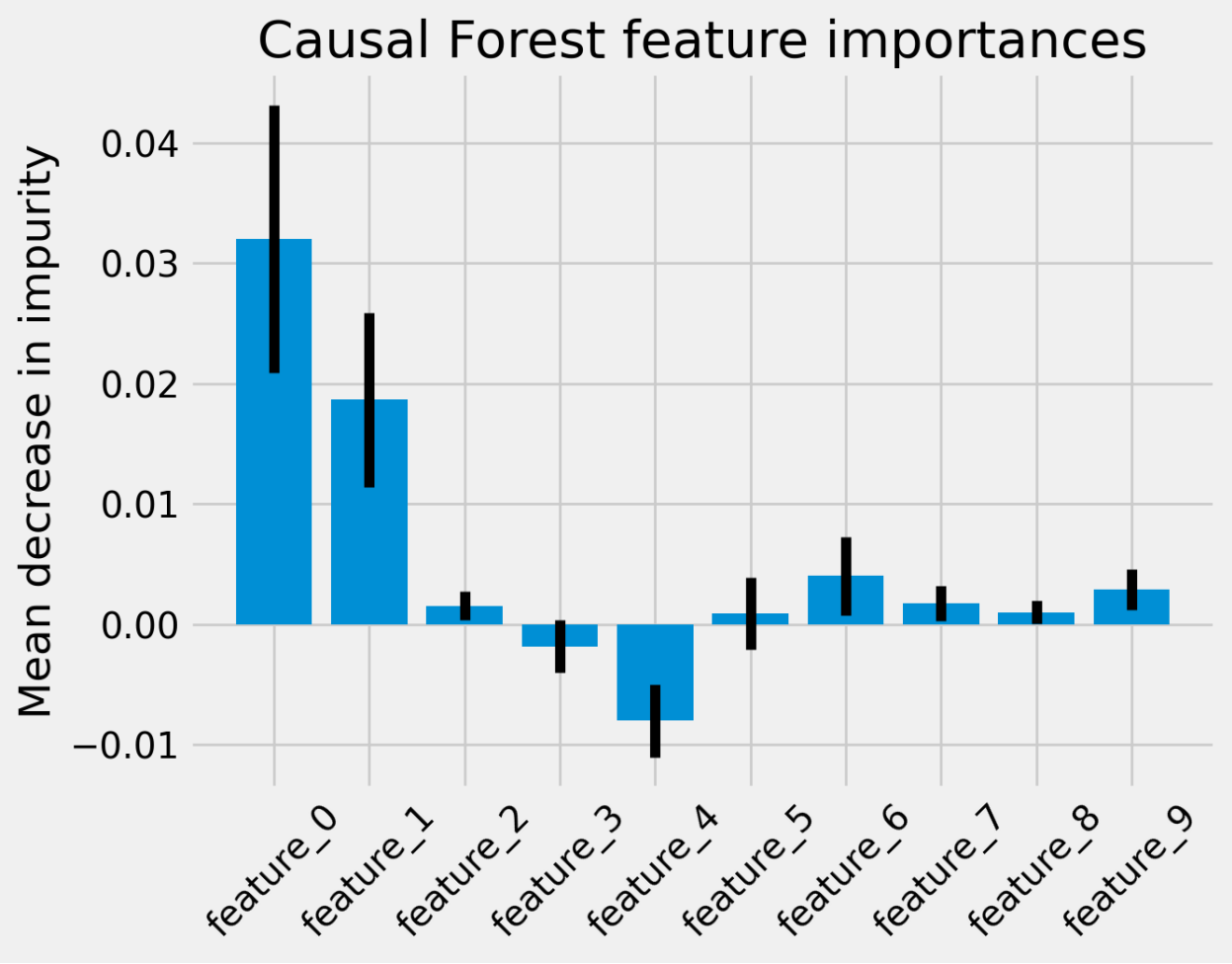

Causal Tree feature importances
Mean decrease in impurity
0.0
0.2
0.4
0.6
0.8
feature_0
feature_1
feature_2
feature_3
feature_4
feature_5
feature_6
feature_7
feature_8
feature_9



2. Permutation-based feature importance

[8]:
```
for name, model in zip(('Causal Tree', 'Causal Forest'), (ctree, crforest)):

    imp = permutation_importance(model, X_test, y_test,
                                 n_repeats=50,
                                 random_state=0)

    fig, ax = plt.subplots()
    ax.set_title(f"{name} feature importances")
    ax.set_ylabel("Mean decrease in impurity")
    plt.bar(feature_names, imp['importances_mean'], yerr=imp['importances_std'])
    ax.set_xticklabels(feature_names, rotation=45)
    plt.show()
```


Causal Tree feature importances
Mean decrease in impurity
0.08
0.06
0.04
0.02
0.00
−0.02
feature_0
feature_1
feature_2
feature_3
feature_4
feature_5
feature_6
feature_7
feature_8
feature_9



SHAP values

5.18.3 TreeExplainer

Details: https://shap.readthedocs.io/en/latest/generated/shap.TreeExplainer.html#shap.TreeExplainer

[10]:
```
tree_explainer = shap.TreeExplainer(ctree)
# Expected values for treatment=0 and treatment=1. i.e. Y|X,T=0 and Y|X,T=1
tree_explainer.expected_value
```

[10]:
```
array([0.93121212, 1.63459276])
```

[11]:
```
# Tree Explainer for treatment=0
shap.initjs()
shap_values = tree_explainer.shap_values(observation)
shap.force_plot(tree_explainer.expected_value[0],
                shap_values[0],
                observation)
```

```
<IPython.core.display.HTML object>
```

[11]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x7fb9f4b52640>
```

[12]:
```
# Tree Explainer for treatment=1
tree_explainer = shap.TreeExplainer(ctree)
shap.initjs()
shap_values = tree_explainer.shap_values(observation)
shap.force_plot(tree_explainer.expected_value[1],
                shap_values[1],
                observation)
```

```
<IPython.core.display.HTML object>
```

[12]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x7fb9f4b4d190>
```

[13]:
```
# Tree Explainer for treatment=0
cforest_explainer = shap.TreeExplainer(crforest)
shap.initjs()
shap_values = cforest_explainer.shap_values(observation)
shap.force_plot(cforest_explainer.expected_value[0],
                shap_values[0],
                observation)
```

```
<IPython.core.display.HTML object>
```

[13]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x7fb9f4b0f940>
```

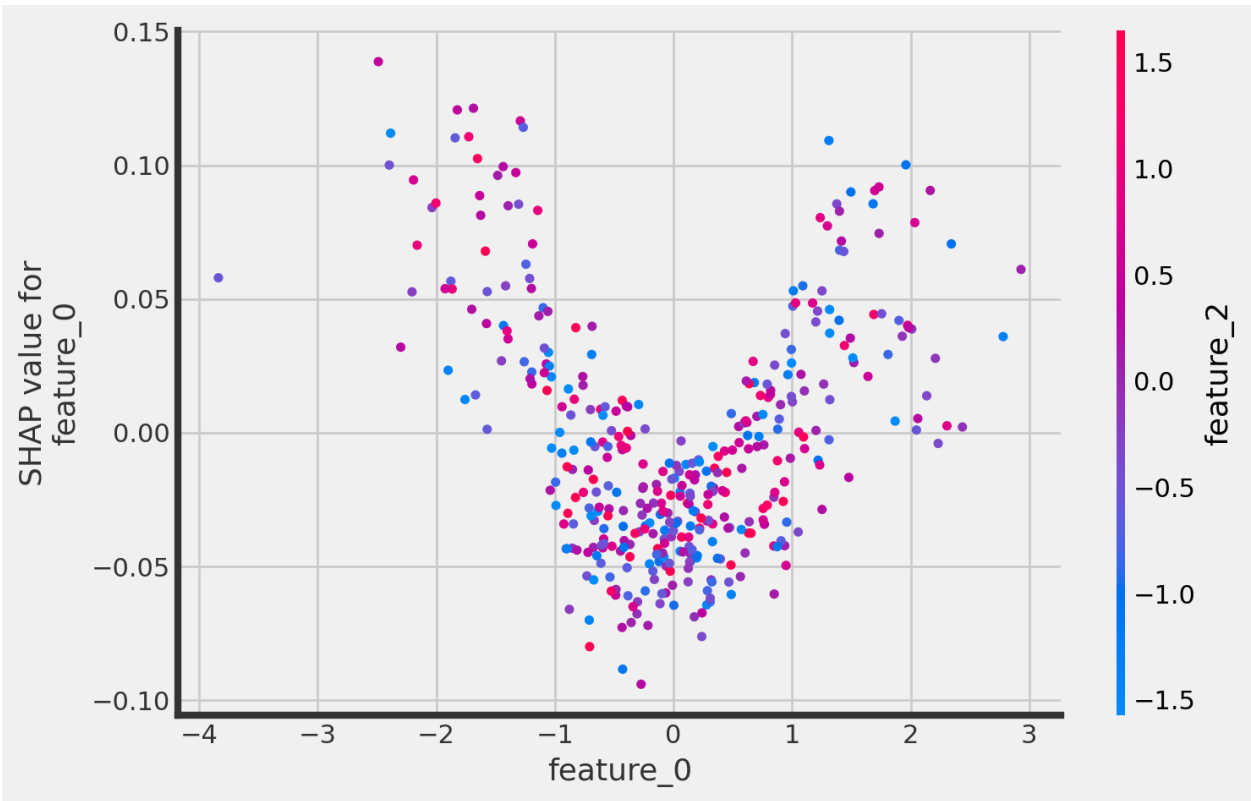
[14]:
```
# Tree Explainer for treatment=1
cforest_explainer = shap.TreeExplainer(crforest)
shap.initjs()
shap_values = cforest_explainer.shap_values(observation)
shap.force_plot(cforest_explainer.expected_value[1],
                shap_values[1],
                observation)
```

```
<IPython.core.display.HTML object>
```

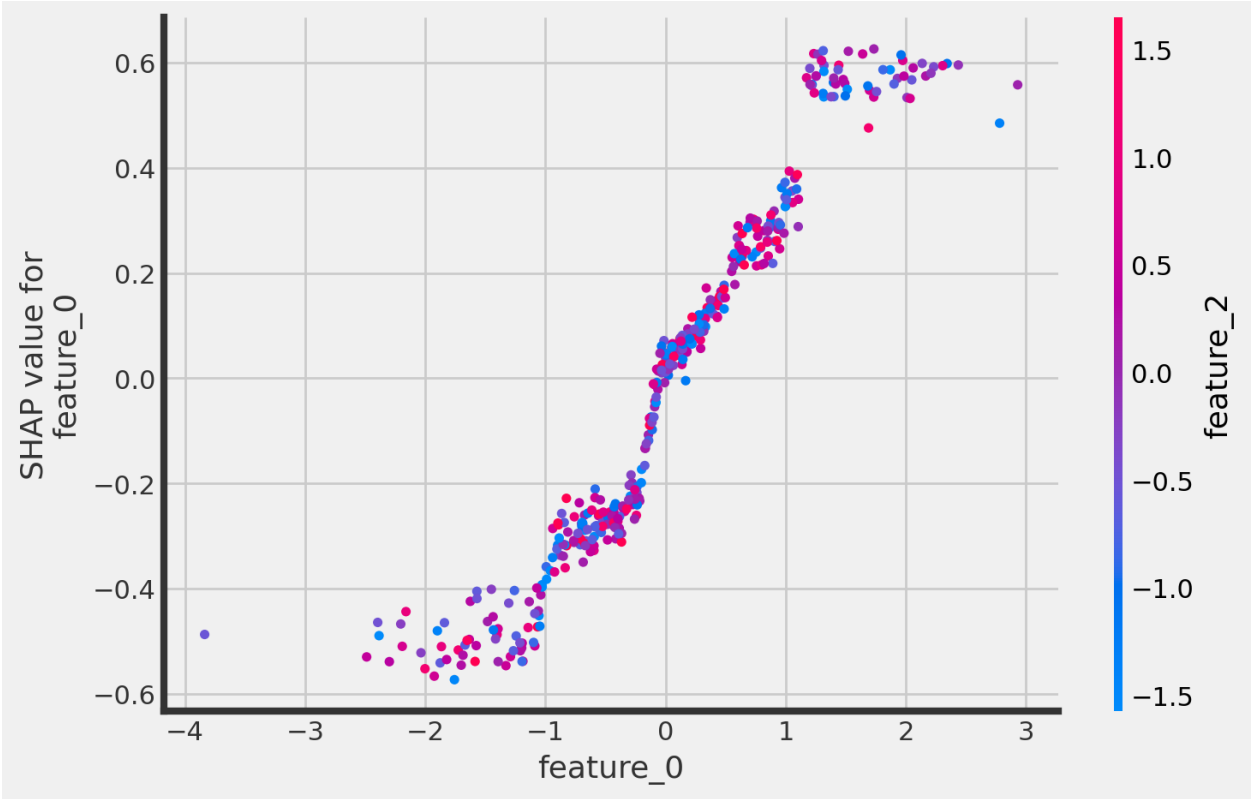
[14]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x7fb9f4b4d790>
```

[15]:
```
for i in [0, 1]:
    print(f"If treatment = {i},i.e. Y_hat|X,T={i}")
    shap.dependence_plot("feature_0",
                         cforest_explainer.shap_values(X_test)[i],
                         X_test,
                         interaction_index="feature_2")
```

```
If treatment = 0,i.e. Y_hat|X,T=0
```

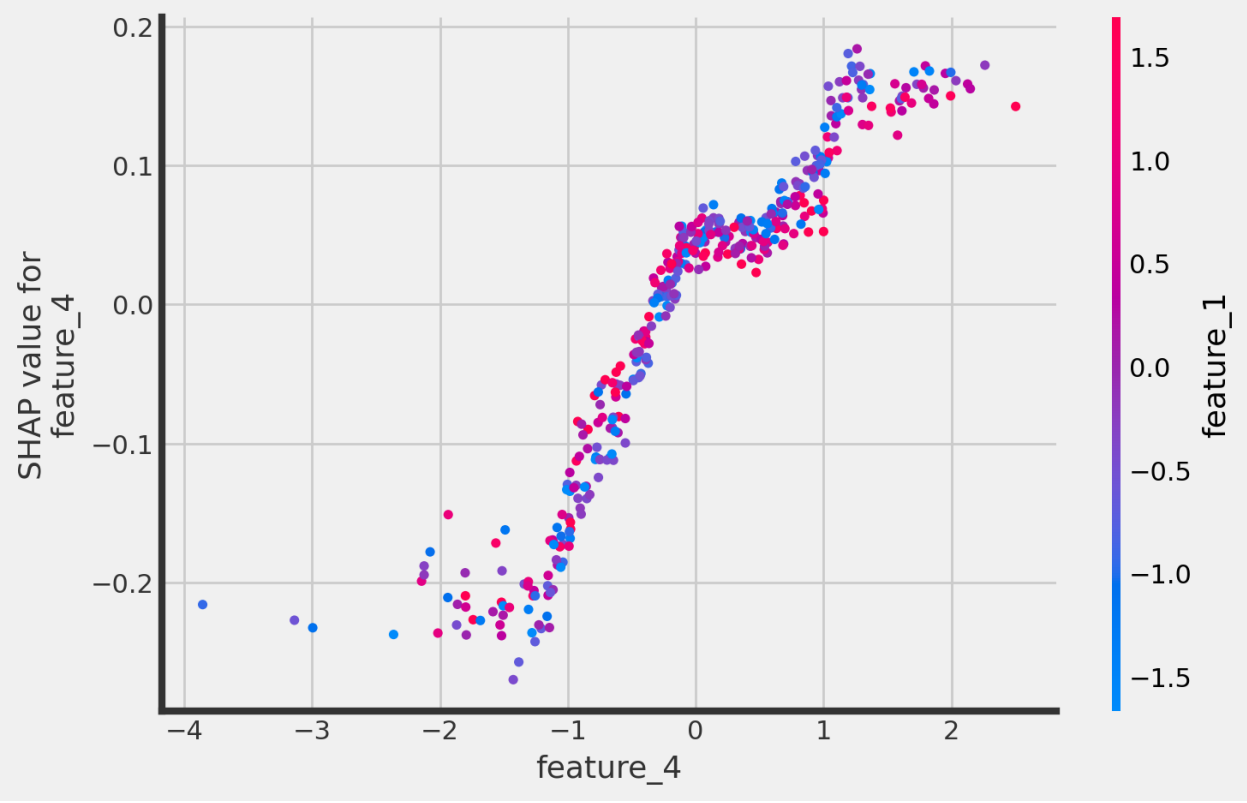


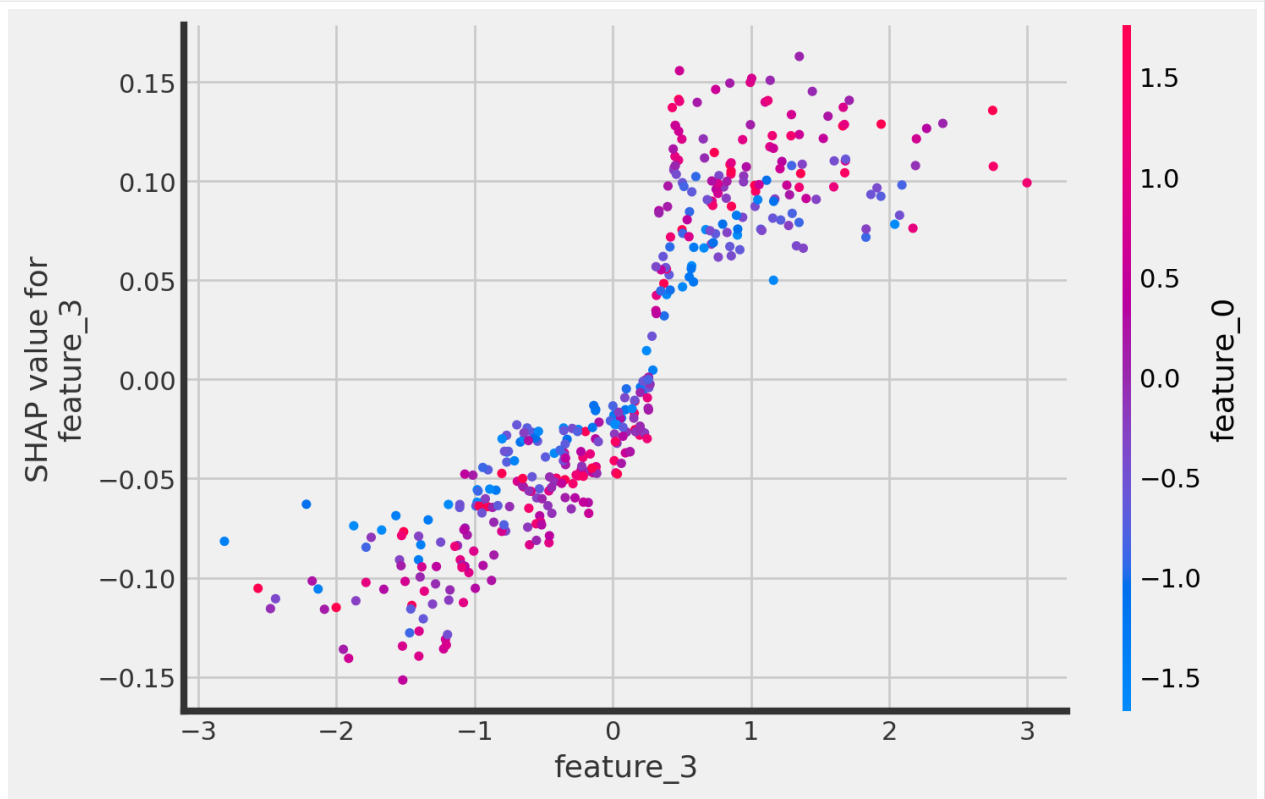

```
If treatment = 1,i.e. Y_hat|X,T=1
```

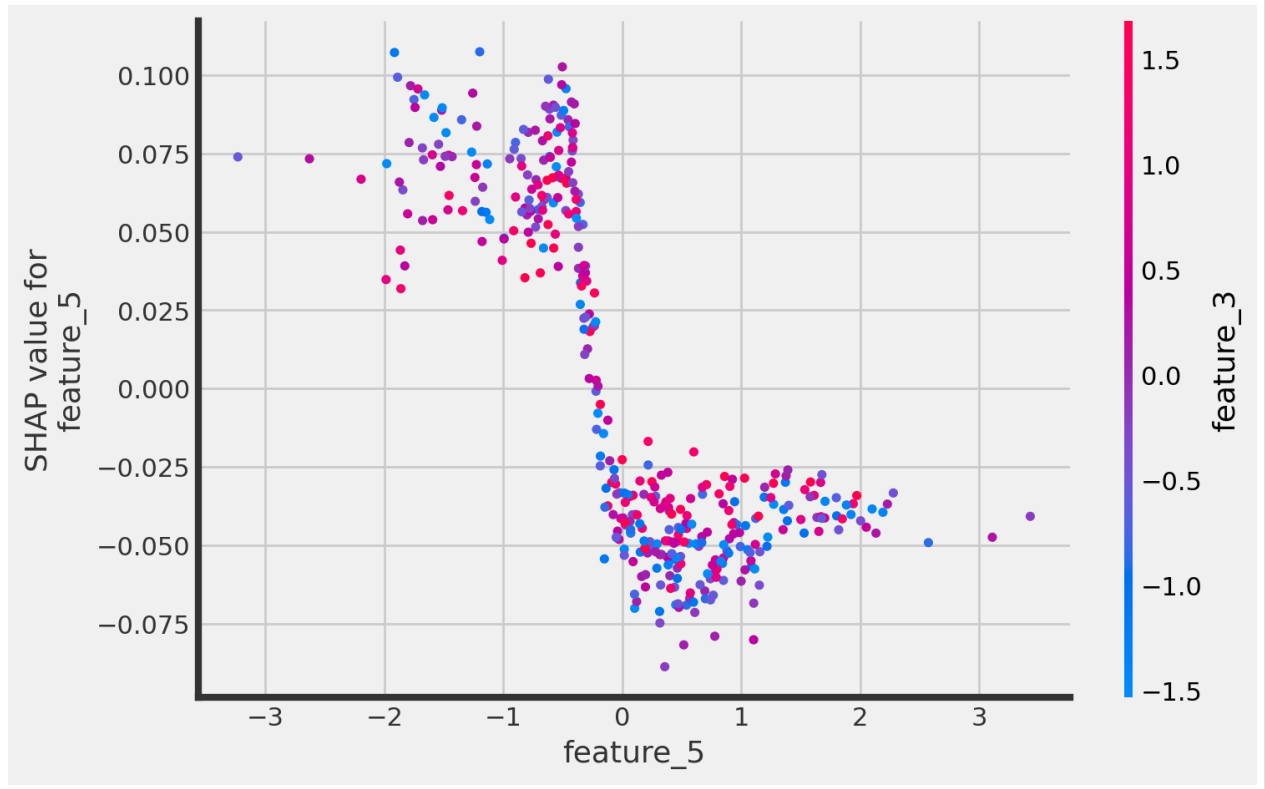


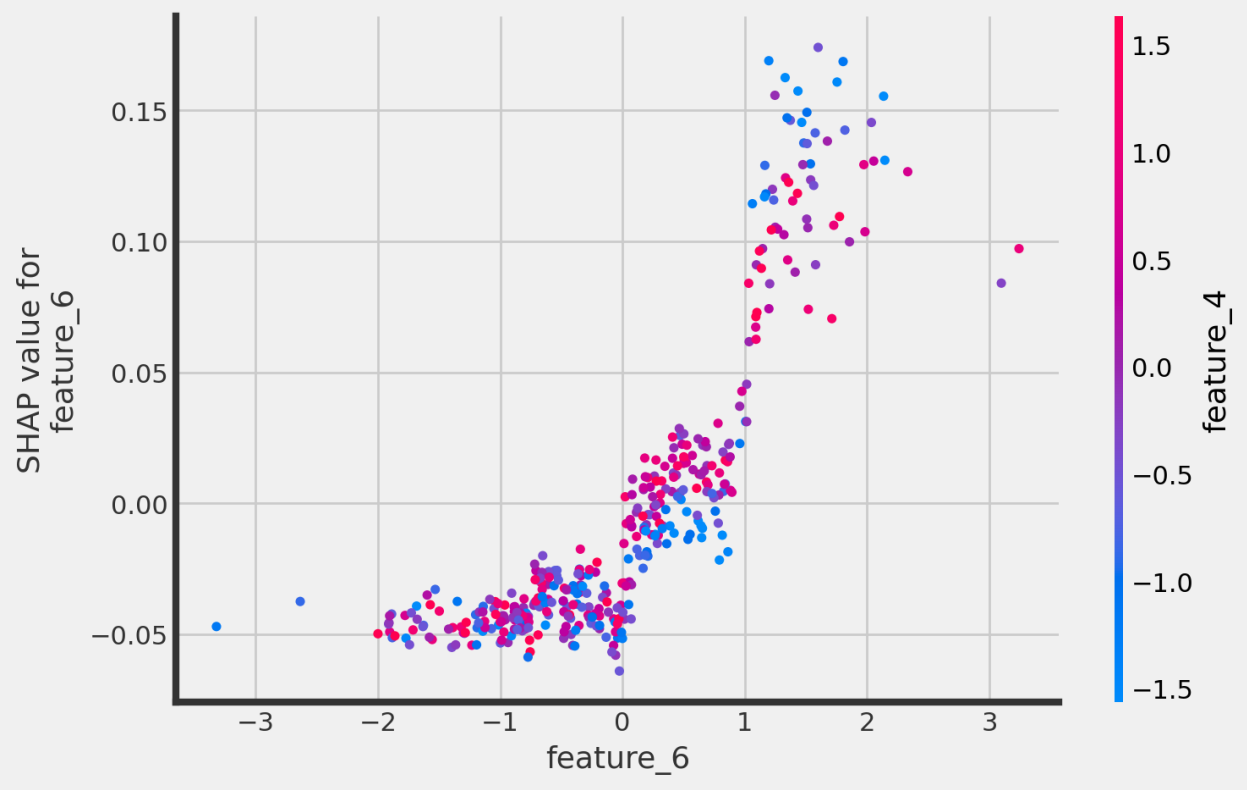
[16]:
```
# Sort the features indexes by their importance in the model
# (sum of SHAP value magnitudes over the validation dataset)


for i in [0, 1]:
    print(f"If treatment = {i},i.e. Y_hat|X,T={i}")
    shap_values = cforest_explainer.shap_values(X_test)[i]
    top_inds = np.argsort(-np.sum(np.abs(shap_values), 0))

    # Make SHAP plots of the three most important features
    for i in range(4):
        shap.dependence_plot(top_inds[i], shap_values, X_test)
```

```
If treatment = 0,i.e. Y_hat|X,T=0
```

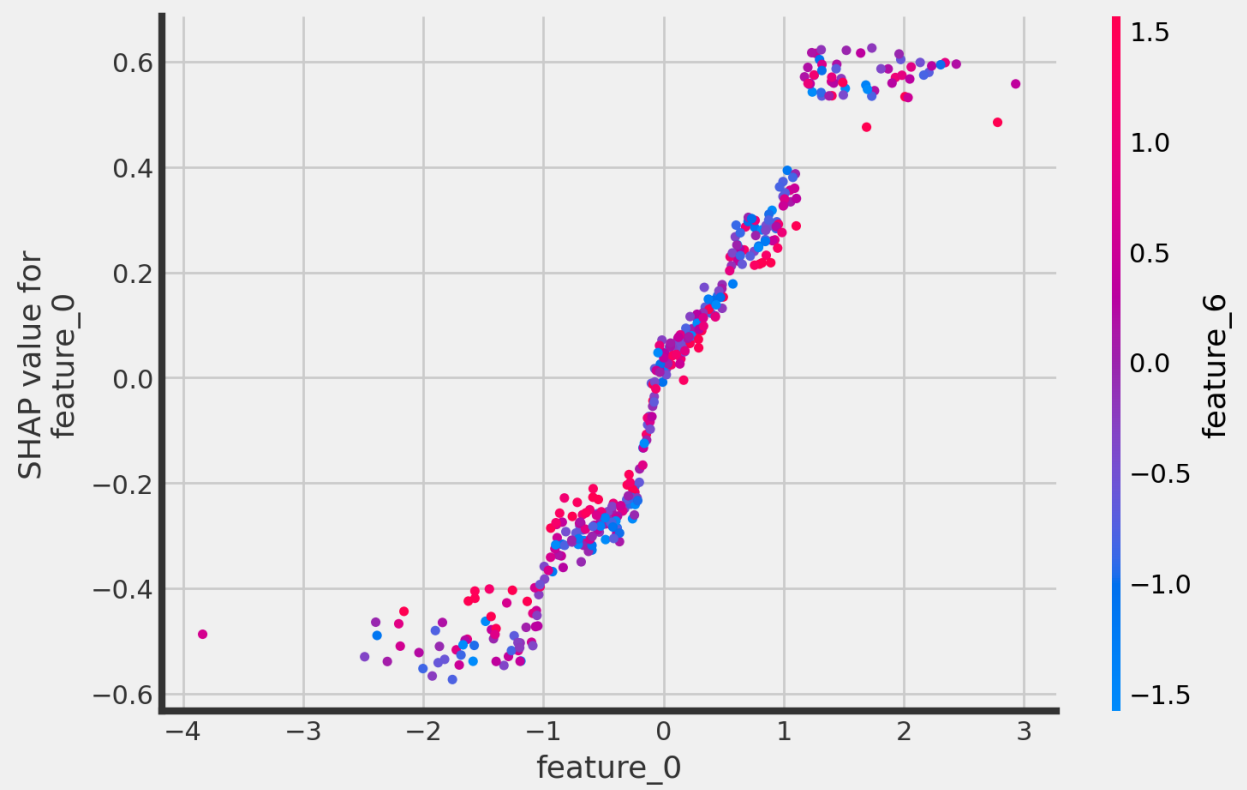


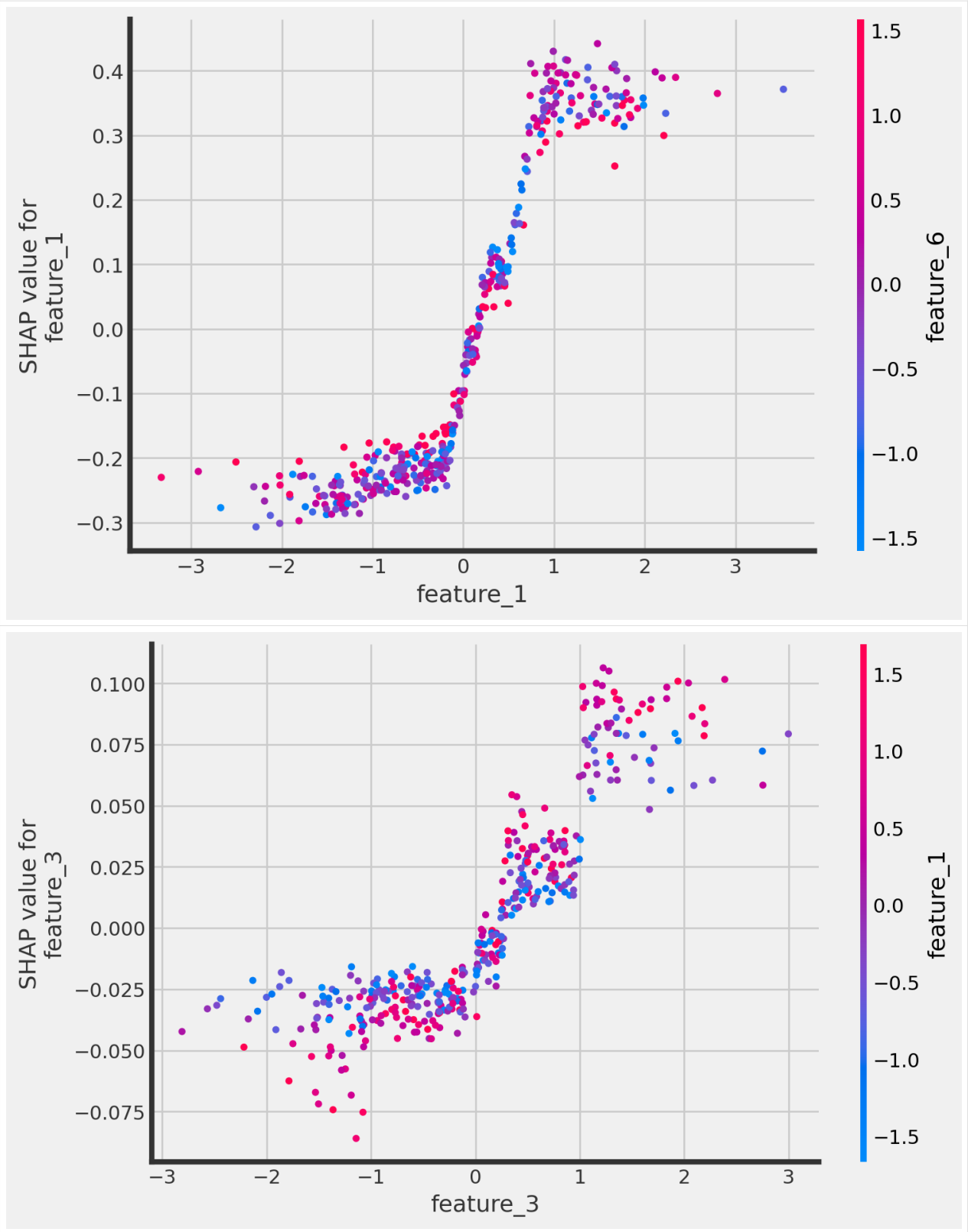

SHAP value for feature_3
feature_3
feature_0
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−3
−2
−1
0
1
2
3
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

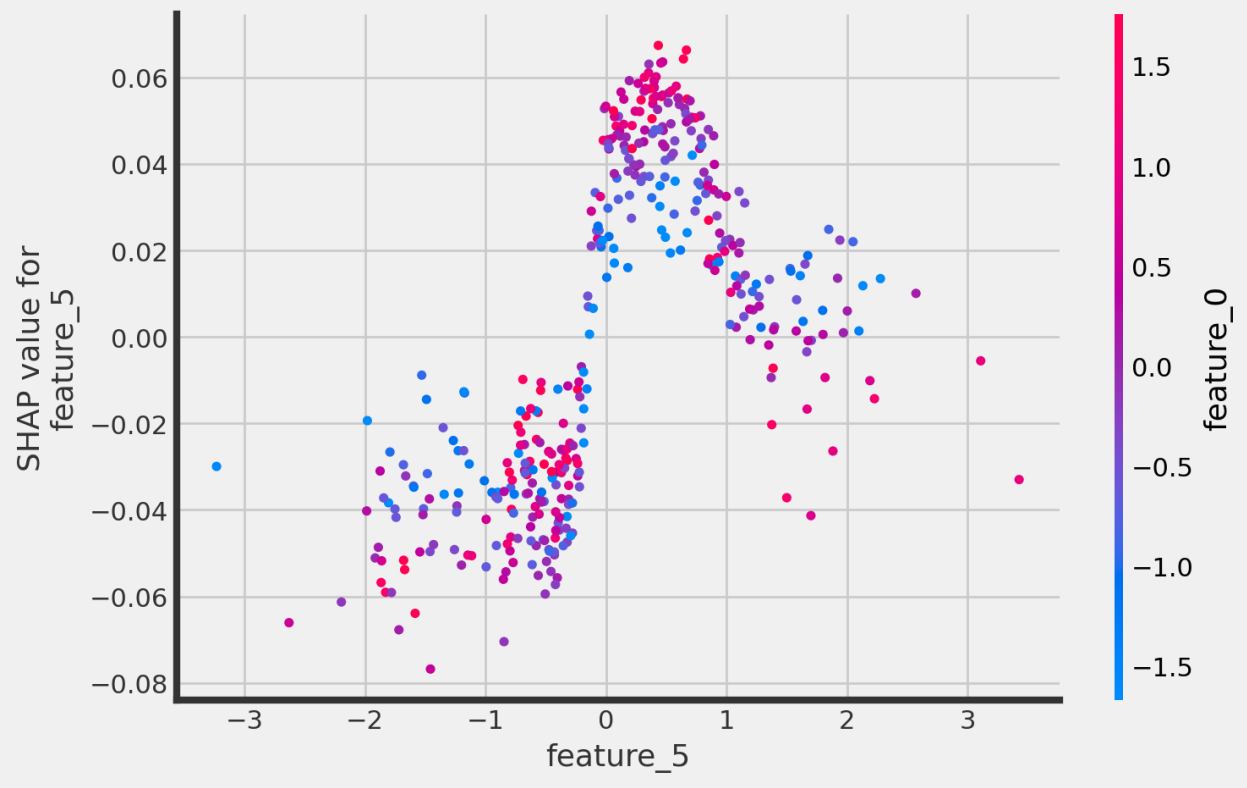

SHAP value for feature_5
feature_5
feature_3
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−3
−2
−1
0
1
2
3
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5




```
If treatment = 1,i.e. Y_hat|X,T=1
```




SHAP value for feature_1
feature_1
feature_6
SHAP value for feature_3
feature_3
feature_1



[ ]:

5.19 Logistic Regression Based Data Generation Function for Uplift Classification Problem

This Data Generation Function uses Logistic Regression as the underlying data generation model. This function enables better control of feature patterns: how feature is associated with outcome baseline and treatment effect. It enables 6 differernt patterns: Linear, Quadratic, Cubic, Relu, Sine, and Cosine.

This notebook shows how to use this data generation function to generate data, with a visualization of the feature patterns.

[1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

%matplotlib inline
```

5.19.1 Import Data Generation Function

[2]:
```
from causalml.dataset import make_uplift_classification_logistic
```

```
The sklearn.utils.testing module is  deprecated in version 0.22 and will be removed␣
↪in version 0.24. The corresponding classes / functions should instead be imported␣
↪from sklearn.utils. Anything that cannot be imported from sklearn.utils is now part␣
↪of the private API.
```

5.19.2 Generate Data

[47]:
```
df, feature_name = make_uplift_classification_logistic( n_samples=100000,
                                                         treatment_name=['control',
↪'treatment1', 'treatment2', 'treatment3'],
                                                         y_name='conversion',
                                                         n_classification_features=10,
                                                         n_classification_
↪informative=5,
                                                         n_classification_redundant=0,
                                                         n_classification_repeated=0,
                                                         n_uplift_dict={'treatment1':␣
↪2, 'treatment2': 2, 'treatment3': 3},
                                                         n_mix_informative_uplift_dict=
↪{'treatment1': 1, 'treatment2': 1, 'treatment3': 0},
                                                         delta_uplift_dict={'treatment1
↪': 0.05, 'treatment2': 0.02, 'treatment3': -0.05},
                                                        feature_association_list = [
↪'linear','quadratic','cubic','relu','sin','cos'],
                                                         random_select_association =␣
↪False,
                                                         random_seed=20200416

                                                       )
```

[48]:
```
df.head()
```

[48]:

	treatment_group_key	x1_informative	x1_informative_transformed	x2_informative	x2_informative_transformed	x3_informative	x3_informative_transformed	x4_informative	x4_informative_transformed
0	treatment1	-0.194205	-0.192043	1.791408	1.572609	0.678028	0.080696	-0.169306	-0.683035
1	treatment1	-0.898070	-0.894462	0.252125	-0.663393	-0.842844	-0.156004	-0.047769	-0.683035
2	treatment1	0.701002	0.701325	0.239320	-0.667867	1.700766	1.278676	-0.734568	-0.683035
3	control	-1.653684	-1.648524	-0.119123	-0.698492	-0.037645	-0.000355	0.687429	0.495943
4	treatment3	1.057909	1.057498	-2.019523	2.190564	-0.950180	-0.223370	-1.505741	-0.683035

(continues on next page)

(continued from previous page)

```
   x5_informative  …  conversion_prob  control_conversion_prob  \
0       -1.837155  …          0.126770                 0.076138
1       -0.251752  …          0.064278                 0.070799
2       -1.130113  …          0.018480                 0.014947
3       -1.427400  …          0.102799                 0.102799
4       -0.399457  …          0.012964                 0.106241

   control_true_effect  treatment1_conversion_prob  treatment1_true_effect  \
0                  0.0                    0.126770                0.050632
1                  0.0                    0.064278               -0.006522
2                  0.0                    0.018480                0.003534
3                  0.0                    0.101410               -0.001390
4                  0.0                    0.171309                0.065068

   treatment2_conversion_prob  treatment2_true_effect  \
0                    0.087545                0.011407
1                    0.101076                0.030277
2                    0.018055                0.003109
3                    0.040230               -0.062569
4                    0.114526                0.008285

   treatment3_conversion_prob  treatment3_true_effect  conversion
0                    0.029396               -0.046742           0
1                    0.050778               -0.020021           0
2                    0.019327                0.004380           0
3                    0.030753               -0.072046           0
4                    0.012964               -0.093277           0

[5 rows x 47 columns]
```

[49]:
```
feature_name
```

[49]:
```
['x1_informative',
 'x2_informative',
 'x3_informative',
 'x4_informative',
 'x5_informative',
 'x6_irrelevant',
 'x7_irrelevant',
 'x8_irrelevant',
 'x9_irrelevant',
 'x10_irrelevant',
 'x11_uplift',
 'x12_uplift',
 'x13_uplift',
 'x14_uplift',
 'x15_uplift',
 'x16_uplift',
 'x17_uplift',
 'x18_mix',
 'x19_mix']
```

5.19.3 Experiment Group Mean

[50]:
```
df.groupby(['treatment_group_key'])['conversion'].mean()
```

[50]:
```
treatment_group_key
control       0.09896
treatment1    0.15088
treatment2    0.12042
treatment3    0.04972
Name: conversion, dtype: float64
```

5.19.4 Visualize Feature Pattern

[51]:
```
# Extract control and treatment1 for illustration
treatment_group_keys = ['control','treatment1']
y_name='conversion'
df1 = df[df['treatment_group_key'].isin(treatment_group_keys)].reset_index(drop=True)
df1.groupby(['treatment_group_key'])['conversion'].mean()
```

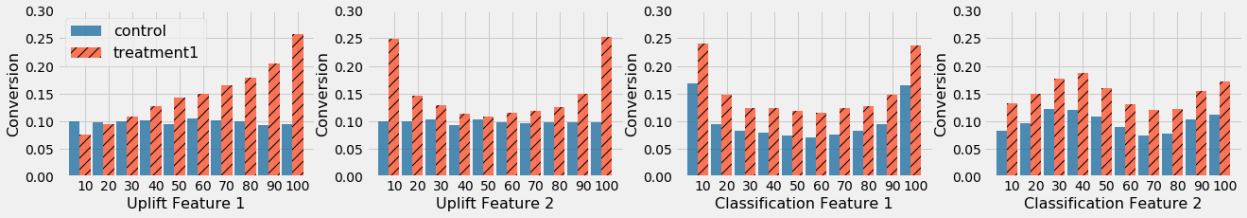
[51]:
```
treatment_group_key
control       0.09896
treatment1    0.15088
Name: conversion, dtype: float64
```

[53]:
```
color_dict = {'control':'#2471a3','treatment1':'#FF5733','treatment2':'#5D6D7E'
             ,'treatment3':'#34495E','treatment4':'#283747'}

hatch_dict = {'control':'','treatment1':'//'}

x_name_plot = ['x11_uplift', 'x12_uplift', 'x2_informative', 'x5_informative']

x_new_name_plot = ['Uplift Feature 1', 'Uplift Feature 2', 'Classification Feature 1',
↪'Classification Feature 2']
opacity = 0.8

plt.figure(figsize=(20, 3))
subplot_list = [141,142,143,144]
counter = 0
bar_width = 0.9/len(treatment_group_keys)
for x_name_i in x_name_plot:
    bins = np.percentile(df1[x_name_i].values, np.linspace(0, 100, 11))[:-1]
    df1['x_bin'] = np.digitize(df1[x_name_i].values, bins)
    df_gb = df1.groupby(['treatment_group_key','x_bin'],as_index=False)[y_name].mean()
    plt.subplot(subplot_list[counter])
    for ti in range(len(treatment_group_keys)):
        x_index = [ti * bar_width - len(treatment_group_keys)/2*bar_width + xi for xi␣
↪in range(10)]
        plt.bar(x_index,
                df_gb[df_gb['treatment_group_key']==treatment_group_keys[ti]][y_name].
↪values,
                bar_width,
                alpha=opacity,
                color=color_dict[treatment_group_keys[ti]],
                hatch = hatch_dict[treatment_group_keys[ti]],
                label=treatment_group_keys[ti]
               )
```

(continues on next page)

(continued from previous page)

```
    plt.xticks(range(10), [int(xi+10) for xi in np.linspace(0, 100, 11)[:-1]])
    plt.xlabel(x_new_name_plot[counter],fontsize=16)
    plt.ylabel('Conversion',fontsize=16)
    #plt.title(x_name_i)
    if counter == 0:
        plt.legend(treatment_group_keys, loc=2,fontsize=16)
    plt.ylim([0.,0.3])
    counter+=1
```



In the figure above, Uplift Feature 1 has a linear pattern on treatment effect, Uplift Feature 2 has a quadratic pattern on treatment effect, Classification Feature 1 has a quadratic pattern on baseline for both treatment and control, and Classification Feature 2 has a Sine pattern on baseline for both treatment and control.

[ ]:

5.20 Qini curves with multiple costly treatment arms

This notebook shows approaches to evaluating multi-armed CATE estimators from `causalML` with the Multi-Armed Qini metric available in the `maq` package (available at https://github.com/grf-labs/maq).

This metric is a generalization of the familiar *Qini curve* to settings where we have multiple treatment arms available, and the cost of assigning treatment can vary by both unit and treatment arm according to some known cost structure. At a high level, this metric essentially allows you to quantify the value of targeting with more treatment arms by undertaking a cost-benefit exercise that uses your CATE estimates to assign the arm to the unit that is most cost-beneficial at various budget constraints.

This notebook gives a brief overview of the statistical setup and a walkthrough with a simple simulated example.

To use this functionality, you first have to install the `maq` Python package from GitHub. The latest source release can be installed with:

```
pip install "git+https://github.com/grf-labs/maq.git#egg=maq&subdirectory=python-
↪package"
```

[2]:
```
# Treatment effect estimators (R-learner with Causal ML + XGBoost)
from causalml.inference.meta import BaseRRegressor
from xgboost import XGBRFRegressor

# Generalized Qini curves
from maq import MAQ, get_ipw_scores

import numpy as np
np.random.seed(42)
```

5.20.1 Statistical setup

Let $k = 1, \dots K$ denote one of K mutually exclusive and costly treatment arms and $k = 0$ a (costless) control arm. Let $Y_i(k)$ denote the potential outcome in the k-th arm for unit i, and X_i a set of observable characteristics.

Let the function $\hat{\tau}(\cdot)$ be an estimate of the conditional average treatment effect (CATE) obtained from a training set, where the k-th element estimates

$$\tau_k(X_i) = E[Y_i(k) - Y(0) \mid X_i].$$

The Qini curve $Q(B)$ quantifies the value of assigning treatment in accordance with our estimated function $\hat{\tau}(\cdot)$ over different values of a budget constraint B. With a **single treatment arm** $K = 1$ we can formalize this as

$$Q(B) = E[\pi_B(X_i)\,(Y_i(1) - Y_i(0))] = E[\pi_B(X_i)\tau(X_i)],$$

where $\pi_B(X_i) \in \{0, 1\}$ is the *policy* that assigns treatment (=1) to those units *predicted* by $\hat{\tau}(\cdot)$ to benefit the most such that on average we incur a cost of at most B. If we let $C(\cdot)$ denote our known cost function (e.g. $C(X_i) = 4.2$ means assigning the i-th unit the treatment costs 4.2 on some chosen cost denomination), then π_B is going to look like

$$\pi_B = argmax_\pi \left\{E[\pi_B(X_i)\hat{\tau}(X_i)] : E[\pi_B(X_i)C(X_i)] \leq B\right\}$$

While slightly daunting written down formally, it turns out expressing π_B is quite simple: it essentially reduces to a thresholding rule: for a given B, treat the units where the predicted cost-to-benefit ratio $\frac{\hat{\tau}(X_i)}{C(X_i)}$ is above a cutoff. The Qini curve can be used to quantify the value, as measured by the expected gain over assigning each unit the control arm when using the estimated function $\hat{\tau}(\cdot)$ with cost structure $C(\cdot)$ to allocate treatment, as we vary the available budget B.

With **multiple treatment arms** $K > 1$, our object of interest, the Qini curve, is the same, we just need to add an inner product $\langle, \rangle$ to the notation

$$Q(B) = E[\langle \pi_B(X_i),\ \tau(X_i) \rangle],$$

to denote that $\pi_B(X_i)$ now is a K-length selection vector with 1 in the k-th entry if we predict that it is optimal to assign the i-th unit that arm at the given budget constraint. Similarly to above, π_B takes the following form

$$\pi_B = argmax_\pi \left\{E[\langle \pi_B(X_i),\ \hat{\tau}(X_i) \rangle] : E[\langle \pi_B(X_i),\ C(X_i) \rangle] \leq B\right\}.$$

Expressing π_B is more complicated now, as for each budget constraint B, π_B has to make decisions of the form "should I assign the i-th unit an initial arm, or if the j-th unit had already been assigned an arm: should I upgrade this person to a costlier but more effective arm?". It turns out that it is possible to express π_B as a thresholding rule (for details we refer to this paper), yielding tractable ways to construct Qini curves for multi-armed treatment rules.

5.20.2 Example

Fitting a CATE function on a training set

Generate some simple (synthetic) data with $K = 2$ treatment arms, where the second arm is more effective on average.

[3]:
```
n = 20000
p = 5

# Draw a treatment assignment from {0, 1, 2} uniformly
W = np.random.choice([0, 1, 2], n)
# Generate p observable characteristics where some are related to the CATE
X = np.random.rand(n, p)
Y = X[:, 1] + X[:, 2]*(W == 1) + 1.5*X[:, 3]*(W == 2) + np.random.randn(n)
```

(continues on next page)

(continued from previous page)

```
# (in this example, the true arm 2 CATE is 1.5*X[:, 3])

# Generate a train/test split
n_train = 15000
ix = np.random.permutation(n)
train, test = ix[:n_train], ix[n_train:]
```

Obtain $\hat{\tau}(\cdot)$ by fitting a CATE estimator on the training set (using an *R-learner*, for example).

[4]:
```
# Use known propensities (1/3)
W_hat = np.repeat(1 / 3, n_train)
propensities = {0: W_hat, 1: W_hat, 2: W_hat}

tau_function = BaseRRegressor(XGBRFRegressor(), random_state=42)
tau_function.fit(X[train, :], W[train], Y[train], propensities)
```

Estimating Q(B) on a test set

At a high level, there are two tasks associated with estimating a Qini curve $Q(B)$. The first one is estimating the underlying policy π_B, and the second is estimating the value of this policy.

As mentioned in the previous section, with multiple costly treatment arms, the policy π_B is more complicated to compute than in the single-armed case, since given a treatment effect function (obtained from some training set) and a cost structure, we need to figure out which arm to assign to which unit at every budget constraint. The maq package performs this *first* step with an algorithm that gives the path of multi-armed policies π_B.

For the *second* step of estimating the value of this policy, we need to construct a matrix of suitable evaluation *scores* (that we denote by Γ) that have the property that when averaged they act as treatment effect estimates.

If we know the treatment randomization probabilities $P[W_i = k \mid X_i]$, it turns out that constructing these scores is easy: we can simply use inverse-propensity weighting (IPW). With K treatment arms, the scores for the k-th arm then takes the following form

$$\Gamma_{i,k} = \frac{1(W_i = k)Y_i}{P[W_i = k \mid X_i]} - \frac{1(W_i = 0)Y_i}{P[W_i = 0 \mid X_i]},$$

where W_i and Y_i are the treatment assignment and observed outcome for test set unit i. An estimate of the ATE for the k-th arm is given by the average of these scores: $\frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \Gamma_{i,k}$. An IPW-based estimate of $Q(B)$ is going to be an average of these scores that "matches" the underlying policy prescription π_B.

the `maq` package has a simple convenience utility `get_ipw_scores` that can be used to construct these via IPW (which by default assumes the arms have uniform assignment probabilities $\frac{1}{K+1}$).

Note: if the randomization probabilities are not known (as could be the case in an observational setting), then a more robust alternative to form the scores via plugging in estimates of the propensities into the expression above, is to use augmented inverse-propensity weighting (AIPW), yielding a doubly robust estimate of the Qini curve. This approach is not covered here, for details we refer to the paper.

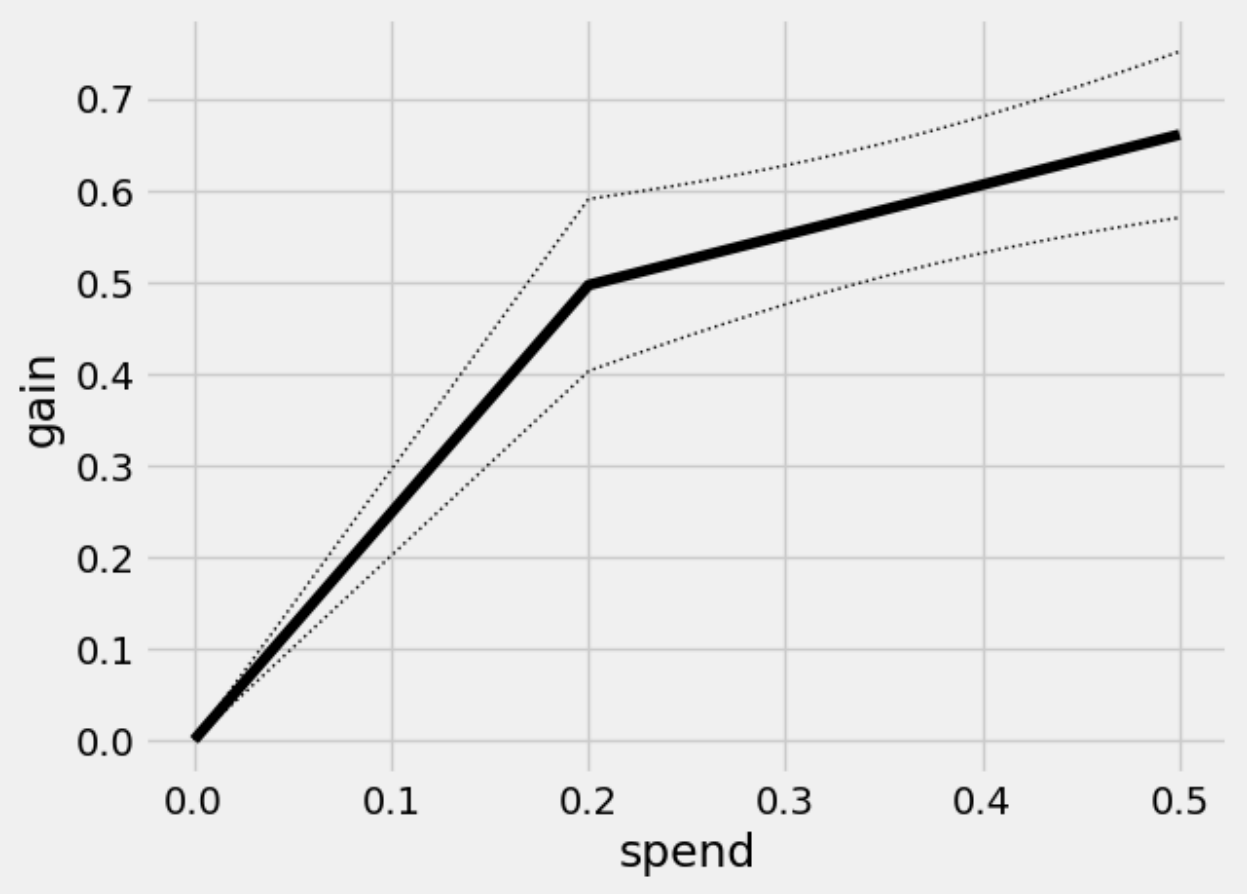
[5]:
```
# Construct an n_test*K matrix of evaluation scores
IPW_scores = get_ipw_scores(Y[test], W[test])

# Predict CATEs on the test set
tau_hat = tau_function.predict(X[test, :])

# Specify our cost structure,
# assume the cost of assigning each unit the first arm is 0.2
```

(continues on next page)

(continued from previous page)

```
# and the cost of assigning each unit the second more effective arm is 0.5
# (these can also be array-valued if costs vary by unit)
cost = [0.2, 0.5]
```

[6]:
```
# Start by fitting a baseline Qini curve that only considers the average treatment␣
↪effects and costs
qini_baseline = MAQ(target_with_covariates=False, n_bootstrap=200)
qini_baseline.fit(tau_hat, cost, IPW_scores)

qini_baseline.plot()
```



This curve has a kink at $B = 0.2$: the first segment traces out the ATE of the lower cost arm, and the second segment the ATE of the higher cost but on average more effective arm. Points on this curve represents the average benefit per unit when targeting an arbitrary group of units.

For example, at an average spend of 0.2 our gain (along with standard errors) is equal to the arm 1 ATE of

[7]:
```
qini_baseline.average_gain(0.2)
```

[7]:
```
(0.49665725358631685, 0.047913821952266705)
```

Next, we fit a Qini curve for arm 1

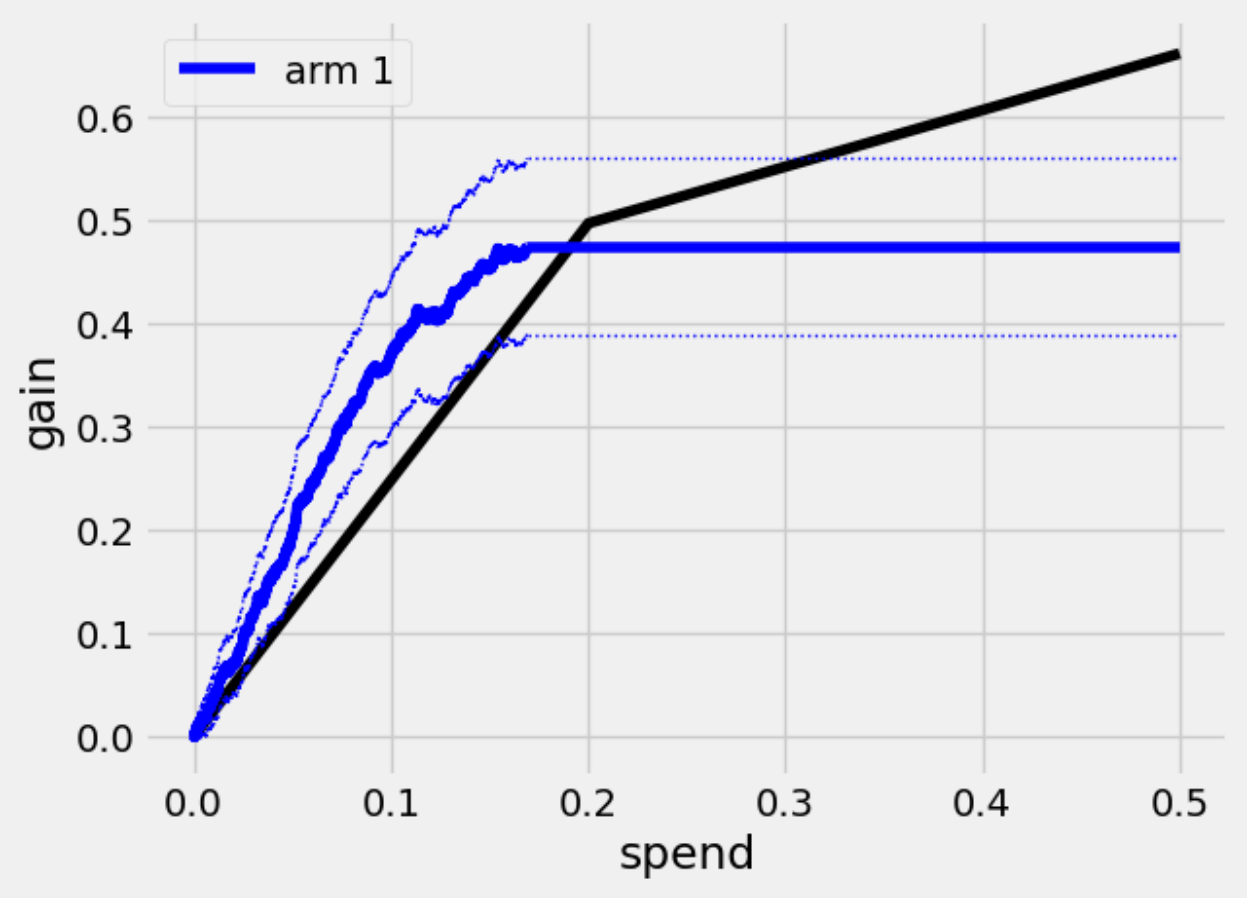
[8]:
```
tau_hat_1 = tau_hat[:, 0]
cost_1 = cost[0]
IPW_scores_1 = IPW_scores[:, 0]
```

(continues on next page)

(continued from previous page)

```
qini_1 = MAQ(n_bootstrap=200).fit(tau_hat_1, cost_1, IPW_scores_1)
```

[9]:
```
qini_baseline.plot(show_ci=False)
# Plot curve with 95% confidence bars
qini_1.plot(color="blue", label="arm 1")
```



The Qini curve for this arm plateaus at a spend of around

[10]:
```
print(qini_1.path_spend_[-1])
```

```
0.168840000000009
```

which means that at this spend level we have given treatment to all units predicted to benefit from treatment (that is, `tau_hat_1` is > 0). We can read off estimates and std errors from the curve, for example at a spend of $B = 0.1$ per unit, the estimated average treatment effect per unit is

[11]:
```
qini_1.average_gain(0.1)
```

[11]:
```
(0.36935574662263526, 0.037401976389534526)
```

(these standard errors are conditional on the estimated function $\hat{\tau}(\cdot)$ and quantify test set uncertainty in estimating the Qini curve).

We can assess the value of targeting with arm 1 at various spend levels by estimating the vertical difference between the blue and black line. Let's call the Qini curve for arm 1 $Q_1(B)$ and the Qini curve for the baseline policy $\overline{Q}(B)$. At

$B = 0.1$, an estimate of $Q_1(0.1) - \overline{Q}(0.1)$ is

[12]:
```
est, std_err = qini_1.difference_gain(qini_baseline, 0.1)
est, std_err
```

[12]:
```
(0.12102711982945671, 0.024068445449392815)
```

That is, at a budget of 0.1 per unit a 95% confidence interval for the increase in gain when targeting with the given arm 1 CATE function over random targeting is

[13]:
```
[est - 1.96*std_err, est + 1.96*std_err]
```

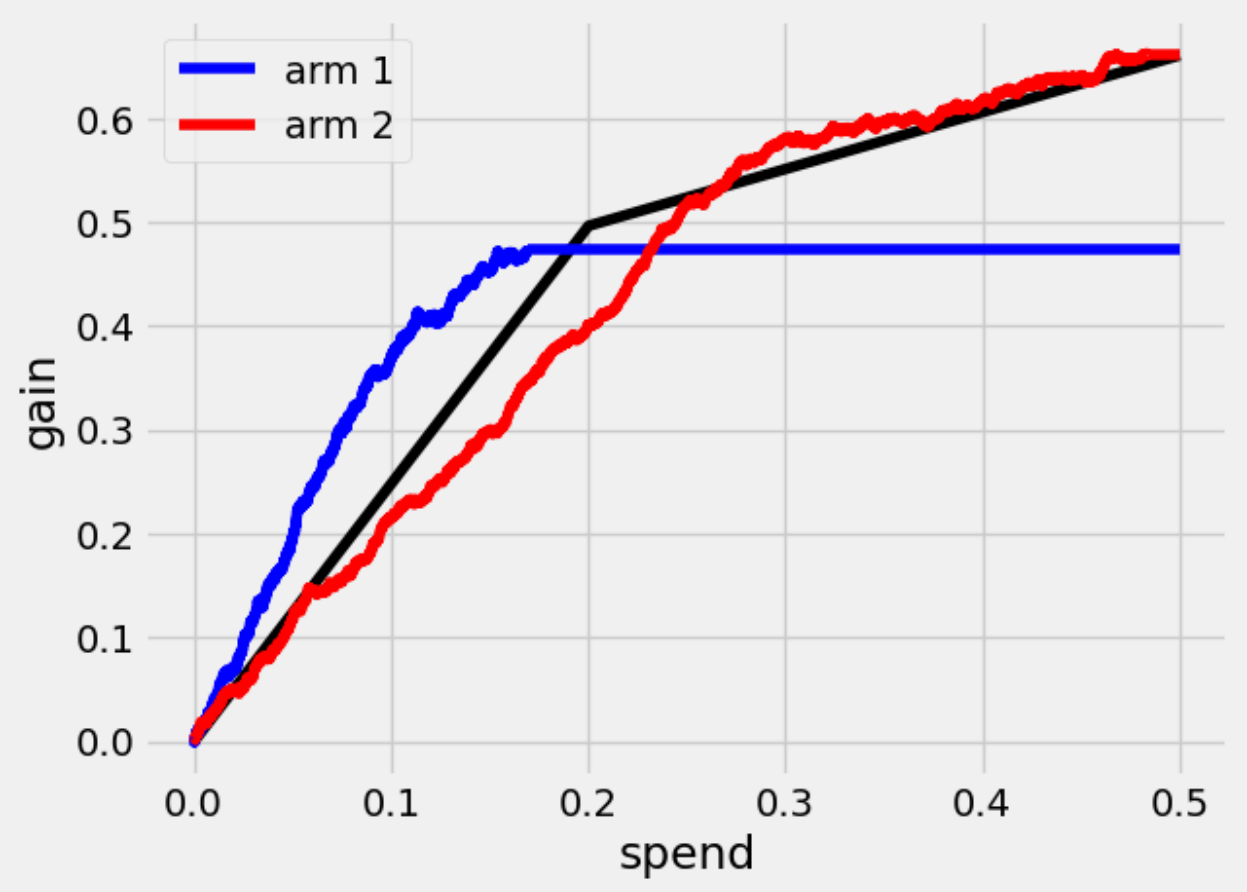
[13]:
```
[0.0738529667486468, 0.16820127291026662]
```

(points on aribtrary curves can be compared with the `difference_gain` method, yielding paired standard errors that account for the correlation between Qini curves fit on the same test data).

Similarily, we can estimate a Qini curve $Q_2(B)$ for the second costlier arm

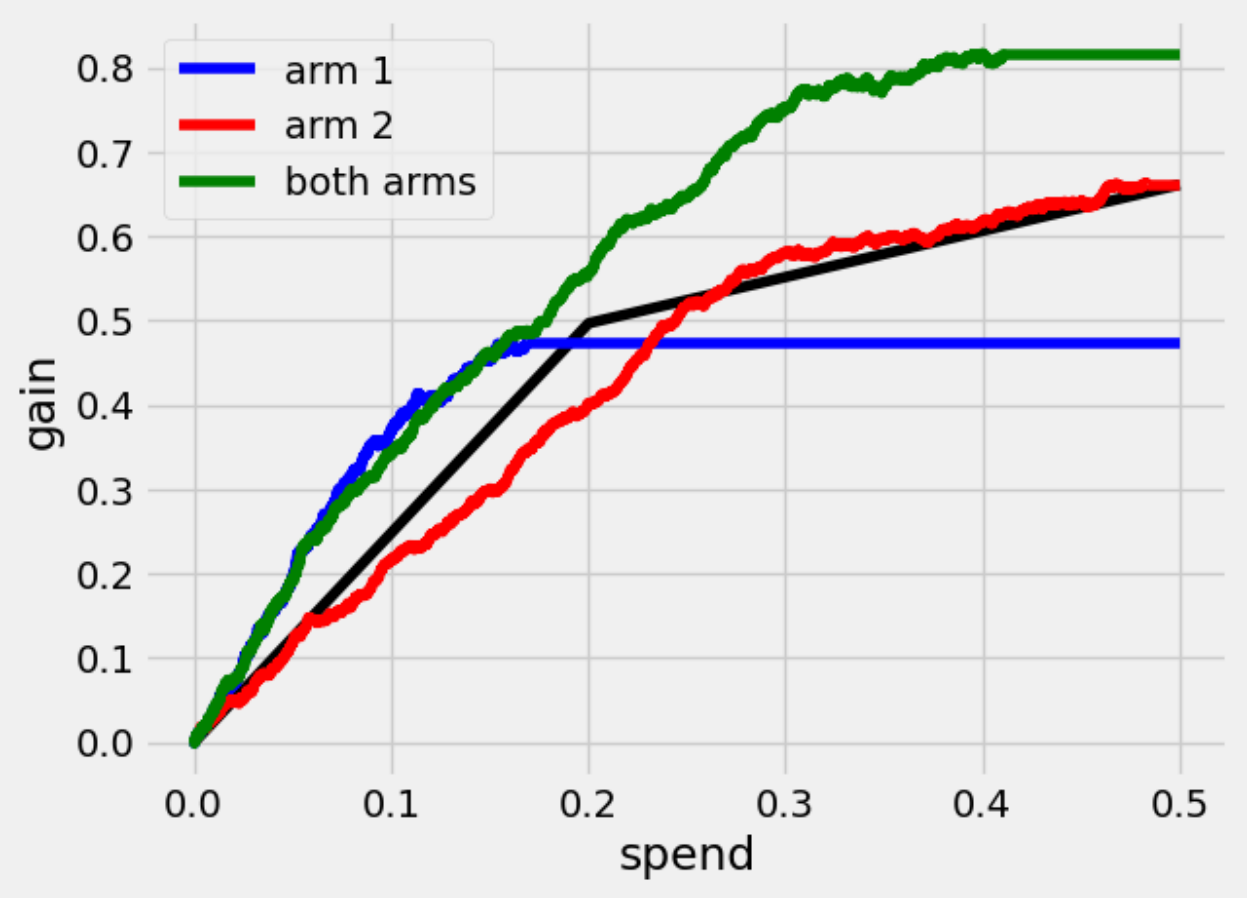
[14]:
```
tau_hat_2 = tau_hat[:, 1]
cost_2 = cost[1]
IPW_scores_2 = IPW_scores[:, 1]

qini_2 = MAQ(n_bootstrap=200).fit(tau_hat_2, cost_2, IPW_scores_2)
```

[15]:
```
qini_baseline.plot(show_ci=False) # Leave out CIs for legibility
qini_1.plot(color="blue", label="arm 1", show_ci=False)
qini_2.plot(color="red", label="arm 2", show_ci=False)
```



Finally, we can see what a Qini curve $Q(B)$ using both arms looks like.

[16]:
```
qini_ma = MAQ(n_bootstrap=200).fit(tau_hat, cost, IPW_scores)

qini_baseline.plot(show_ci=False)
qini_1.plot(color="blue", label="arm 1", show_ci=False)
qini_2.plot(color="red", label="arm 2", show_ci=False)
qini_ma.plot(color="green", label="both arms", show_ci=False)
```



Qini curves for single-armed treatment rules allow for assessing the value of targeting with a specific arm or targeting function. The generalization of the Qini to multiple treatment arms allows us to also assess the value of targeting with a combination of arms.

At $B = 0.3$, the estimated increase in gain when targeting with both arms over using only the second arm, $Q(0.3) - Q_2(0.3)$, is

[17]:
```
qini_ma.difference_gain(qini_2, 0.3)
```

[17]:
```
(0.17003364056661086, 0.036733311977033105)
```

In this example, a multi-armed policy achieves a larger gain by assigning the treatment that is most cost-beneficial to each test set unit. The underlying policy π_B looks like

[18]:
```
qini_ma.predict(0.3)
```

[18]:
```
array([[0., 1.],
       [0., 1.],
       [1., 0.],
```

(continues on next page)

(continued from previous page)

```
       …,
       [1., 0.],
       [0., 1.],
       [0., 1.]])
```

where rows correspond to $\pi_B(X_i)$, where the k-th column contains a 1 if it is optimal to assign this arm to the i-th unit at the given spend (and all entries 0 if the control arm is assigned). An alternative representation of the policy is to take values in the treatment arm set $\{0, 1, 2\}$

[19]:
```
qini_ma.predict(0.3, prediction_type="vector")
```

[19]:
```
array([2, 2, 1, …, 1, 2, 2])
```

In addition to comparing points on different Qini curves, we can also compare across a range of spend levels by estimating an area between two curves up to a maximum $\overline{B}$. An estimate and standard error of the area between the green and red curves up to $\overline{B} = 0.5$, the integral $\int_0^{0.5}(Q(B) - Q_2(B))dB$, is

[20]:
```
qini_ma.integrated_difference(qini_2, 0.5)
```

[20]:
```
(0.12548196750671803, 0.02945344768121884)
```

CHAPTER

SIX

METHODOLOGY

In this section we dive more deeply into the algorithms implemented in CausalML. To provide a basis for the discussion, we review some of the frameworks and definitions used in the literature.

We use the Neyman-Rubin potential outcomes framework and assume Y represents the outcome, W represents the treatment assignment, and X_i the observed covariates.

6.1 Supported Algorithms

CausalML 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
  - *Uplift Random Forests* on delta-delta-p ($\Delta\Delta P$) criterion (only for binary trees and two-class problems)
  - *Uplift Random Forests* on IDDP (only for binary trees and two-class problems)
  - *Interaction Tree* (only for binary trees and two-class problems)
  - *Causal Inference Tree* (only for binary trees and two-class problems)
- **Meta-learner algorithms**
  - *S-Learner*
  - *T-Learner*
  - *X-Learner*
  - *R-Learner*
  - *Doubly Robust (DR) learner*
- **Instrumental variables algorithms**
  - *2-Stage Least Squares (2SLS)*
  - *Doubly Robust Instrumental Variable (DRIV) learner*
- **Neural network based algorithms**
  - CEVAE
  - DragonNet
- **Treatment optimization algorithms**

- *Counterfactual Unit Selection*
- *Counterfactual Value Estimator*

6.2 Decision Guide

See image in: https://github.com/uber/causalml/issues/677#issuecomment-1712088558

6.3 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) [16].

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 [14].

6.3.1 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 W:

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

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 [30].

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].

6.3.2 T-Learner

T-learner [16] consists of two stages as follows:

Stage 1

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

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

using machine learning models.

Stage 2

Define the CATE estimate as:

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

6.3.3 X-Learner

X-learner [16] is an extension of T-learner, and consists of three stages as follows:

Stage 1

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

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

using machine learning models.

Stage 2

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

$$D_i^1 = Y_i^1 - \hat{\mu}_0(X_i^1)$$
$$D_i^0 = \hat{\mu}_1(X_i^0) - Y_i^0$$

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 $g(x)$.

6.3.4 R-Learner

R-learner [19] 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, $\hat{L}_n(\tau(x))$:

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

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

6.3.5 Doubly Robust (DR) learner

DR-learner [15] estimates the CATE via cross-fitting a doubly-robust score function in two stages as follows. We start by randomly split the data $\{Y, X, W\}$ into 3 partitions $\{Y^i, X^i, W^i\}, i = \{1, 2, 3\}$.

Stage 1

Fit a propensity score model $\hat{e}(x)$ with machine learning using $\{X^1, W^1\}$, and fit outcome regression models $\hat{m}_0(x)$ and $\hat{m}_1(x)$ for treated and untreated users with machine learning using $\{Y^2, X^2, W^2\}$.

Stage 2

Use machine learning to fit the CATE model, $\hat{\tau}(X)$ from the pseudo-outcome

$$\phi = \frac{W - \hat{e}(X)}{\hat{e}(X)(1 - \hat{e}(X))} (Y - \hat{m}_W(X)) + \hat{m}_1(X) - \hat{m}_0(X)$$

with $\{Y^3, X^3, W^3\}$

Stage 3

Repeat Stage 1 and Stage 2 again twice. First use $\{Y^2, X^2, W^2\}$, $\{Y^3, X^3, W^3\}$, and $\{Y^1, X^1, W^1\}$ for the propensity score model, the outcome models, and the CATE model. Then use $\{Y^3, X^3, W^3\}$, $\{Y^2, X^2, W^2\}$, and $\{Y^1, X^1, W^1\}$ for the propensity score model, the outcome models, and the CATE model. The final CATE model is the average of the 3 CATE models.

6.3.6 Doubly Robust Instrumental Variable (DRIV) learner

We combine the idea from DR-learner [15] with the doubly robust score function for LATE described in [10] to estimate the conditional LATE. Towards that end, we start by randomly split the data $\{Y, X, W, Z\}$ into 3 partitions $\{Y^i, X^i, W^i, Z^i\}, i = \{1, 2, 3\}$.

Stage 1

Fit propensity score models $\hat{e}_0(x)$ and $\hat{e}_1(x)$ for assigned and unassigned users using $\{X^1, W^1, Z^1\}$, and fit outcome regression models $\hat{m}_0(x)$ and $\hat{m}_1(x)$ for assigned and unassigned users with machine learning using $\{Y^2, X^2, Z^2\}$. Assignment probabiliy, p_Z, can either be user provided or come from a simple model, since in most use cases assignment is random by design.

Stage 2

Use machine learning to fit the conditional *LATE* model, $\hat{\tau}(X)$ by minimizing the following loss function

$$L(\hat{\tau}(X)) = \hat{E}\left[\left(\hat{m}_1(X) - \hat{m}_0(X) + \frac{Z(Y - \hat{m}_1(X))}{p_Z} - \frac{(1 - Z)(Y - \hat{m}_0(X))}{1 - p_Z} - \left(\hat{e}_1(X) - \hat{e}_0(X) + \frac{Z(W - \hat{e}_1(X))}{p_Z} - \frac{(1 - Z)(W - \hat{e}_0(X))}{1 - p_Z}\right)\hat{\tau}(X)\right)^2\right]$$

with $\{Y^3, X^3, W^3\}$

Stage 3

Similar to the DR-Learner Repeat Stage 1 and Stage 2 again twice with different permutations of partitions for estimation. The final conditional LATE model is the average of the 3 conditional LATE models.

6.4 Tree-Based Algorithms

6.4.1 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. [22] proposed three different ways to quantify the gain in divergence as the result of splitting [11]:

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

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.

6.4.2 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 [11]

6.4.3 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.

6.4.4 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.

6.4.5 DDP

Another Uplift Tree algorithm that is implemented is the delta-delta-p ($\Delta\Delta P$) approach by [9], where the sample splitting criterion is defined as follows:

$$\Delta\Delta P = |(P^T(y|a_0) - P^C(y|a_0) - (P^T(y|a_1) - P^C(y|a_1)))|$$

where a_0 and a_1 are the outcomes of a Split A, y is the selected class, and P^T and P^C are the response rates of treatment and control group, respectively. In other words, we first calculate the difference in the response rate in each branch (ΔP_{left} and ΔP_{right}), and subsequently, calculate their differences ($\Delta\Delta P = |\Delta P_{left} - \Delta P_{right}|$).

6.4.6 IDDP

Build upon the $\Delta\Delta P$ approach, the IDDP approach by [23] is implemented, where the sample splitting criterion is defined as follows:

$$IDDP = \frac{\Delta\Delta P^*}{I(\phi, \phi_l, \phi_r)}$$

where $\Delta\Delta P^*$ is defined as $\Delta\Delta P - |E[Y(1) - Y(0)]|X \epsilon \phi|$ and $I(\phi, \phi_l, \phi_r)$ is defined as:

$$I(\phi, \phi_l, \phi_r) = H(\frac{n_t(\phi)}{n(\phi)}, \frac{n_c(\phi)}{n(\phi)}) * 2\frac{1 + \Delta\Delta P^*}{3} + \frac{n_t(\phi)}{n(\phi)} H(\frac{n_t(\phi_l)}{n(\phi)}, \frac{n_t(\phi_r)}{n(\phi)}) + \frac{n_c(\phi)}{n(\phi)} * H(\frac{n_c(\phi_l)}{n(\phi)}, \frac{n_c(\phi_r)}{n(\phi)}) + \frac{1}{2}$$

where the entropy H is defined as $H(p, q) = (-p * log_2(p)) + (-q * log_2(q))$ and where ϕ is a subset of the feature space associated with the current decision node, and ϕ_l and ϕ_r are the left and right child nodes, respectively. $n_t(\phi)$ is the number of treatment samples, $n_c(\phi)$ the number of control samples, and $n(\phi)$ the number of all samples in the current (parent) node.

6.4.7 IT

Further, the package implements the Interaction Tree (IT) proposed by [26], where the sample splitting criterion maximizes the G statistic among all permissible splits:

$$G(s^*) = maxG(s)$$

where $G(s) = t^2(s)$ and $t(s)$ is defined as:

$$t(s) = \frac{(y_1^L - y_0^L) - (y_1^R - y_0^R)}{\sigma * (1/n_1 + 1/n_2 + 1/n_3 + 1/n_4)}$$

where $\sigma = \sum_{i=4}^{4} w_i s_i^2$ is a pooled estimator of the constant variance, and $w_i = (n_i - 1)/\sum_{j=1}^{4}(n_j - 1)$. Further, y_1^L, s_1^2, and n_1 are the the sample mean, the sample variance, and the sample size for the treatment group in the left child node ,respectively. Similar notation applies to the other quantities.

Note that this implementation deviates from the original implementation in that (1) the pruning techniques and (2) the validation method for determining the best tree size are different.

6.4.8 CIT

Also, the package implements the Causal Inference Tree (CIT) by [25], where the sample splitting criterion calculates the likelihood ratio test statistic:

$$LRT(s) = -n_{\tau L}/2 * ln(n_{\tau L} SSE_{\tau L}) - n_{\tau R}/2 * ln(n_{\tau R} SSE_{\tau R}) + n_{\tau L1} lnn_{\tau L1} + n_{\tau L0} lnn_{\tau L0} + n_{\tau R1} lnn_{\tau R1} + n_{\tau R0} lnn_{\tau R0}$$

where n_τ, $n_{\tau 0}$, and $n_{\tau 1}$ are the total number of observations in node τ, the number of observations in node τ that are assigned to the control group, and the number of observations in node τ that are assigned to the treatment group, respectively. SSE_τ is defined as:

$$SSE_\tau = \sum_{i \epsilon \tau : t_i = 1} (y_i - \hat{y_{t1}})^2 + \sum_{i \epsilon \tau : t_i = 0} (y_i - \hat{y_{t0}})^2$$

and $\hat{y_{t0}}$ and $\hat{y_{t1}}$ are the sample average responses of the control and treatment groups in node τ, respectively.

Note that this implementation deviates from the original implementation in that (1) the pruning techniques and (2) the validation method for determining the best tree size are different.

6.4.9 CTS

The final Uplift Tree algorithm that is implemented is the Contextual Treatment Selection (CTS) approach by [28], where the sample splitting criterion is defined as follows:

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

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.

6.5 Value optimization methods

The package supports methods for assigning treatment groups when treatments are costly. To understand the problem, it is helpful to divide populations into the following four categories:

- **Compliers**. Those who will have a favourable outcome if and only if they are treated.
- **Always-takers**. Those who will have a favourable outcome whether or not they are treated.
- **Never-takers**. Those who will never have a favourable outcome whether or not they are treated.
- **Defiers**. Those who will have a favourable outcome if and only if they are not treated.

For a more detailed discussion see e.g. [2].

6.5.1 Counterfactual Unit Selection

[18] propose a method for selecting units for treatments using counterfactual logic. Suppose the following benefits for selecting units belonging to the different categories above:

- Compliers: β
- Always-takers: γ
- Never-takers: θ
- Defiers: δ

If X denotes the set of individual's features, the unit selection problem can be formulated as follows:

$$argmax_X \beta P(\text{complier} \mid X) + \gamma P(\text{always-taker} \mid X) + \theta P(\text{never-taker} \mid X) + \delta P(\text{defier} \mid X)$$

The problem can be reformulated using counterfactual logic. Suppose $W = w$ indicates that an individual is treated and $W = w'$ indicates he or she is untreated. Similarly, let $F = f$ denote a favourable outcome for the individual and $F = f'$ an unfavourable outcome. Then the optimization problem becomes:

$$argmax_X \beta P(f_w, f'_{w'} \mid X) + \gamma P(f_w, f_{w'} \mid X) + \theta P(f'_w, f'_{w'} \mid X) + \delta P(f_{w'}, f'_w \mid X)$$

Note that the above simply follows from the definitions of the relevant users segments. [18] then use counterfactual logic ([21]) to solve the above optimization problem under certain conditions.

N.B. The current implementation in the package is highly experimental.

6.5.2 Counterfactual Value Estimator

The counterfactual value estimation method implemented in the package predicts the outcome for a unit under different treatment conditions using a standard machine learning model. The expected value of assigning a unit into a particular treatment is then given by

$$\mathbb{E}[(v - cc_w)Y_w - ic_w]$$

where Y_w is the probability of a favourable event (such as conversion) under a given treatment w, v is the value of the favourable event, cc_w is the cost of the treatment triggered in case of a favourable event, and ic_w is the cost associated with the treatment whether or not the outcome is favourable. This method builds upon the ideas discussed in [29].

6.6 Probabilities of causation

A cause is said to be *necessary* for an outcome if the outcome would not have occurred in the absence of the cause. A cause is said to be *sufficient* for an outcome if the outcome would have occurred in the presence of the cause. A cause is said to be *necessary and sufficient* if both of the above two conditions hold. [27] show that we can calculate bounds for the probability that a cause is of each of the above three types.

To understand how the bounds for the probabilities of causation are calculated, we need special notation to represent counterfactual quantities. Let y_t represent the proposition "y would occur if the treatment group was set to 'treatment'", y'_c represent the proposition "y would not occur if the treatment group was set to 'control'", and similarly for the remaining two combinations of the (by assumption) binary outcome and treatment variables.

Then the probability that the treatment is *sufficient* for y to occur can be defined as

$$PS = P(y_t \mid c, y')$$

This is the probability that the y would occur if the treatment was set to t when in fact the treatment was set to control and the outcome did not occur.

The probability that the treatment is *necessary* for y to occur can be defined as

$$PN = P(y'_c \mid t, y)$$

This is the probability that y would not occur if the treatment was set to control, while in actuality both y occurs and the treatment takes place.

Finally, the probability that the treatment is both necessary and sufficient is defined as

$$PNS = P(y_t, y'_c)$$

and states that y would occur if the treatment took place; and y would not occur if the treatment did not take place. PNS is related with PN and PS as follows:

$$PNS = P(t, y)PN + P(c, y')PS$$

In bounding the above three quantities, we utilize observational data in addition to experimental data. The observational data is characterized in terms of the joint probabilities:

$$P_{TY} = P(t, y), P(c, y), P(t, y'), P(c, y')$$

Given this, [27] use the program developed in [8] to obtain sharp bounds of the above three quantities. The main idea in this program is to turn the bounding task into a linear programming problem (for a modern implementation of their approach see here).

Using the linear programming approach and given certain constraints together with observational data, [27] find that the shar lower bound for PNS is given by

$$max\{0, P(y_t) - P(y_c), P(y) - P(y_c), P(y_t) - P(y)\}$$

and the sharp upper bound is given by

$$min\{P(y_t), P(y_c'), P(t, y) + P(c, y'), P(y_t) - P(y_c) + P(t, y') + P(c, y)\}$$

They use a similar routine to find the bounds for PS and PN. The *get_pns_bounds()* function calculates the bounds for each of the three probabilities of causation using the results in [27].

6.7 Selected traditional methods

The package supports selected traditional causal inference methods. These are usually used to conduct causal inference with observational (non-experimental) data. In these types of studies, the observed difference between the treatment and the control is in general not equal to the difference between "potential outcomes" $\mathbb{E}[Y(1) - Y(0)]$. Thus, the methods below try to deal with this problem in different ways.

6.7.1 Matching

The general idea in matching is to find treated and non-treated units that are as similar as possible in terms of their relevant characteristics. As such, matching methods can be seen as part of the family of causal inference approaches that try to mimic randomized controlled trials.

While there are a number of different ways to match treated and non-treated units, the most common method is to use the propensity score:

$$e_i(X_i) = P(W_i = 1 \mid X_i)$$

Treated and non-treated units are then matched in terms of $e(X)$ using some criterion of distance, such as $k : 1$ nearest neighbours. Because matching is usually between the treated population and the control, this method estimates the average treatment effect on the treated (ATT):

$$\mathbb{E}[Y(1) \mid W = 1] - \mathbb{E}[Y(0) \mid W = 1]$$

See [24] for a discussion of the strengths and weaknesses of the different matching methods.

6.7.2 Inverse probability of treatment weighting

The inverse probability of treatment weighting (IPTW) approach uses the propensity score e to weigh the treated and non-treated populations by the inverse of the probability of the actual treatment W. For a binary treatment $W \in \{1, 0\}$:

$$\frac{W}{e} + \frac{1 - W}{1 - e}$$

In this way, the IPTW approach can be seen as creating an artificial population in which the treated and non-treated units are similar in terms of their observed features X.

One of the possible benefits of IPTW compared to matching is that less data may be discarded due to lack of overlap between treated and non-treated units. A known problem with the approach is that extreme propensity scores can generate highly variable estimators. Different methods have been proposed for trimming and normalizing the IPT weights ([13]). An overview of the IPTW approach can be found in [7].

6.7.3 2-Stage Least Squares (2SLS)

One of the basic requirements for identifying the treatment effect of W on Y is that W is orthogonal to the potential outcome of Y, conditional on the covariates X. This may be violated if both W and Y are affected by an unobserved variable, the error term after removing the true effect of W from Y, that is not in X. In this case, the instrumental variables approach attempts to estimate the effect of W on Y with the help of a third variable Z that is correlated with W but is uncorrelated with the error term. In other words, the instrument Z is only related with Y through the directed path that goes through W. If these conditions are satisfied, in the case without covariates, the effect of W on Y can be estimated using the sample analog of:

$$\frac{Cov(Y_i, Z_i)}{Cov(W_i, Z_i)}$$

The most common method for instrumental variables estimation is the two-stage least squares (2SLS). In this approach, the cause variable W is first regressed on the instrument Z. Then, in the second stage, the outcome of interest Y is regressed on the predicted value from the first-stage model. Intuitively, the effect of W on Y is estimated by using only the proportion of variation in W due to variation in Z. Specifically, assume that we have the linear model

$$Y = W\alpha + X\beta + u = \Xi\gamma + u$$

Here for convenience we let $\Xi = [W, X]$ and $\gamma = [\alpha', \beta']'$. Assume that we have instrumental variables Z whose number of columns is at least the number of columns of W, let $\Omega = [Z, X]$, 2SLS estimator is as follows

$$\hat{\gamma}_{2SLS} = \left[\Xi'\Omega(\Omega'\Omega)^{-1}\Omega'\Xi\right]^{-1}\left[\Xi'\Omega'(\Omega'\Omega)^{-1}\Omega'Y\right].$$

See [3] for a detailed discussion of the method.

6.7.4 LATE

In many situations the treatment W may depend on subject's own choice and cannot be administered directly in an experimental setting. However one can randomly assign users into treatment/control groups so that users in the treatment group can be nudged to take the treatment. This is the case of noncompliance, where users may fail to comply with their assignment status, Z, as to whether to take treatment or not. Similar to the section of Value optimization methods, in general there are 3 types of users in this situation,

- **Compliers** Those who will take the treatment if and only if they are assigned to the treatment group.
- **Always-Taker** Those who will take the treatment regardless which group they are assigned to.
- **Never-Taker** Those who wil not take the treatment regardless which group they are assigned to.

However one assumes that there is no Defier for identification purposes, i.e. those who will only take the treatment if they are assigned to the control group.

In this case one can measure the treatment effect of Compliers,

$$\hat{\tau}_{Complier} = \frac{E[Y|Z=1] - E[Y|Z=0]}{E[W|Z=1] - E[W|Z=0]}$$

This is Local Average Treatment Effect (LATE). The estimator is also equivalent to 2SLS if we take the assignment status, Z, as an instrument.

6.8 Targeted maximum likelihood estimation (TMLE) for ATE

Targeted maximum likelihood estimation (TMLE) [17] provides a doubly robust semiparametric method that "targets" directly on the average treatment effect with the aid from machine learning algorithms. Compared to other methods including outcome regression and inverse probability of treatment weighting, TMLE usually gives better performance especially when dealing with skewed treatment and outliers.

Given binary treatment W, covariates X, and outcome Y, the TMLE for ATE is performed in the following steps

Step 1

Use cross fit to estimate the propensity score $\hat{e}(x)$, the predicted outcome for treated $\hat{m}_1(x)$, and predicted outcome for control $\hat{m}_0(x)$ with machine learning.

Step 2

Scale Y into $\tilde{Y} = \frac{Y - \min Y}{\max Y - \min Y}$ so that $\tilde{Y} \in [0, 1]$. Use the same scale function to transform $\hat{m}_i(x)$ into $\tilde{m}_i(x)$, $i = 0, 1$. Clip the scaled functions so that their values stay in the unit interval.

Step 3

Let $Q = \log(\tilde{m}_W(X)/(1 - \tilde{m}_W(X)))$. Maximize the following pseudo log-likelihood function

$$\max_{h_0, h_1} -\frac{1}{N} \sum_i \left[\tilde{Y}_i \log \left(1 + \exp(-Q_i - h_0 \frac{1 - W}{1 - \hat{e}(X_i)} - h_1 \frac{W}{\hat{e}(X_i)} \right) \right.$$
$$\left. + (1 - \tilde{Y}_i) \log \left(1 + \exp(Q_i + h_0 \frac{1 - W}{1 - \hat{e}(X_i)} + h_1 \frac{W}{\hat{e}(X_i)} \right) \right]$$

Step 4

Let

$$\tilde{Q}_0 = \frac{1}{1 + \exp\left(-Q - h_0 \frac{1}{1 - \hat{e}(X)}\right)},$$
$$\tilde{Q}_1 = \frac{1}{1 + \exp\left(-Q - h_1 \frac{1}{\hat{e}(X)}\right)}.$$

The ATE estimate is the sample average of the differences of $\tilde{Q}_1$ and $\tilde{Q}_0$ after rescale to the original range.

CHAPTER

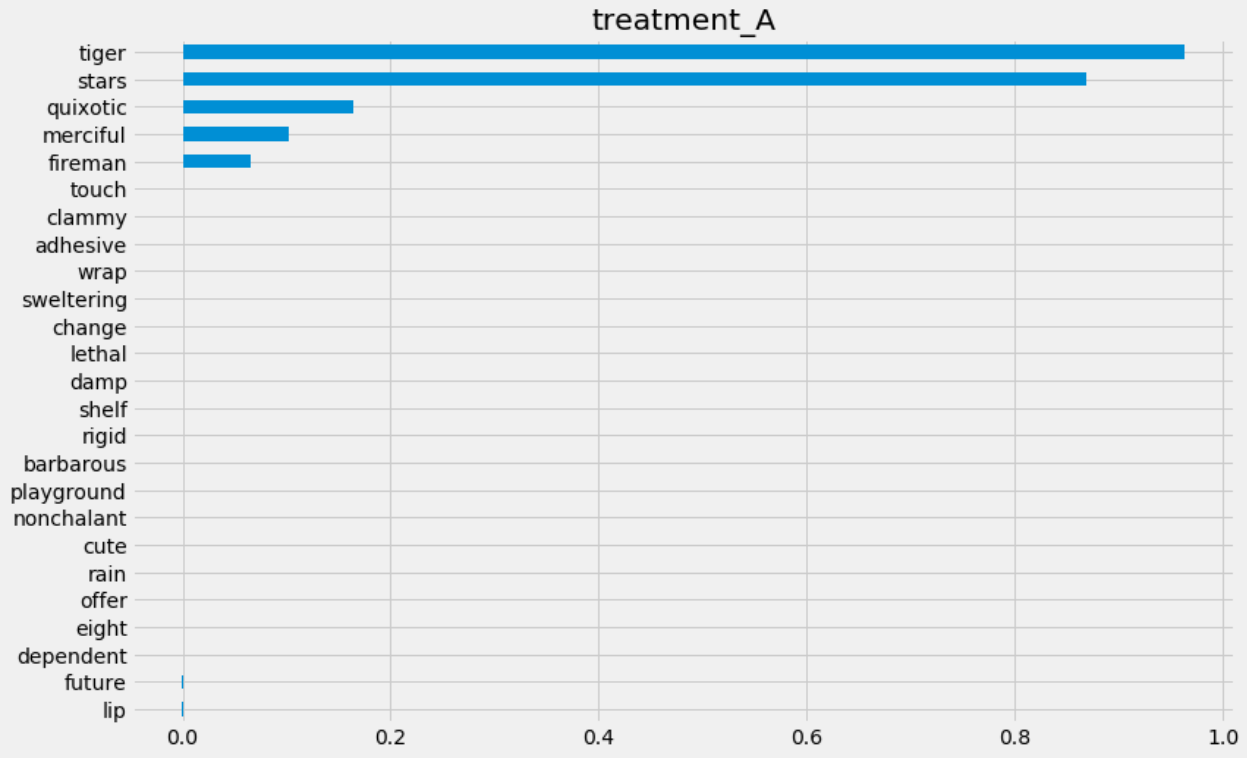
SEVEN

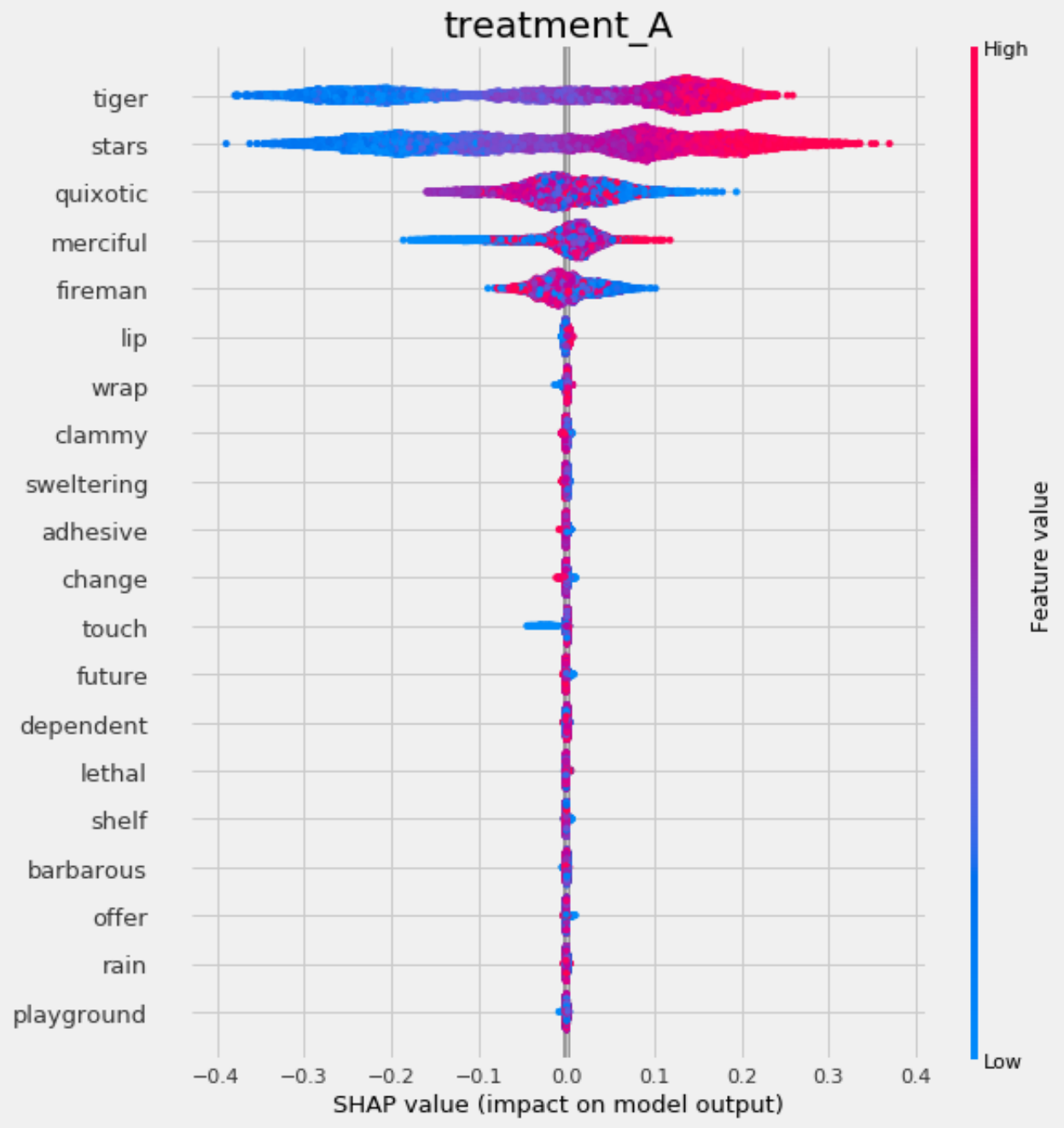
INTERPRETABLE CAUSAL ML

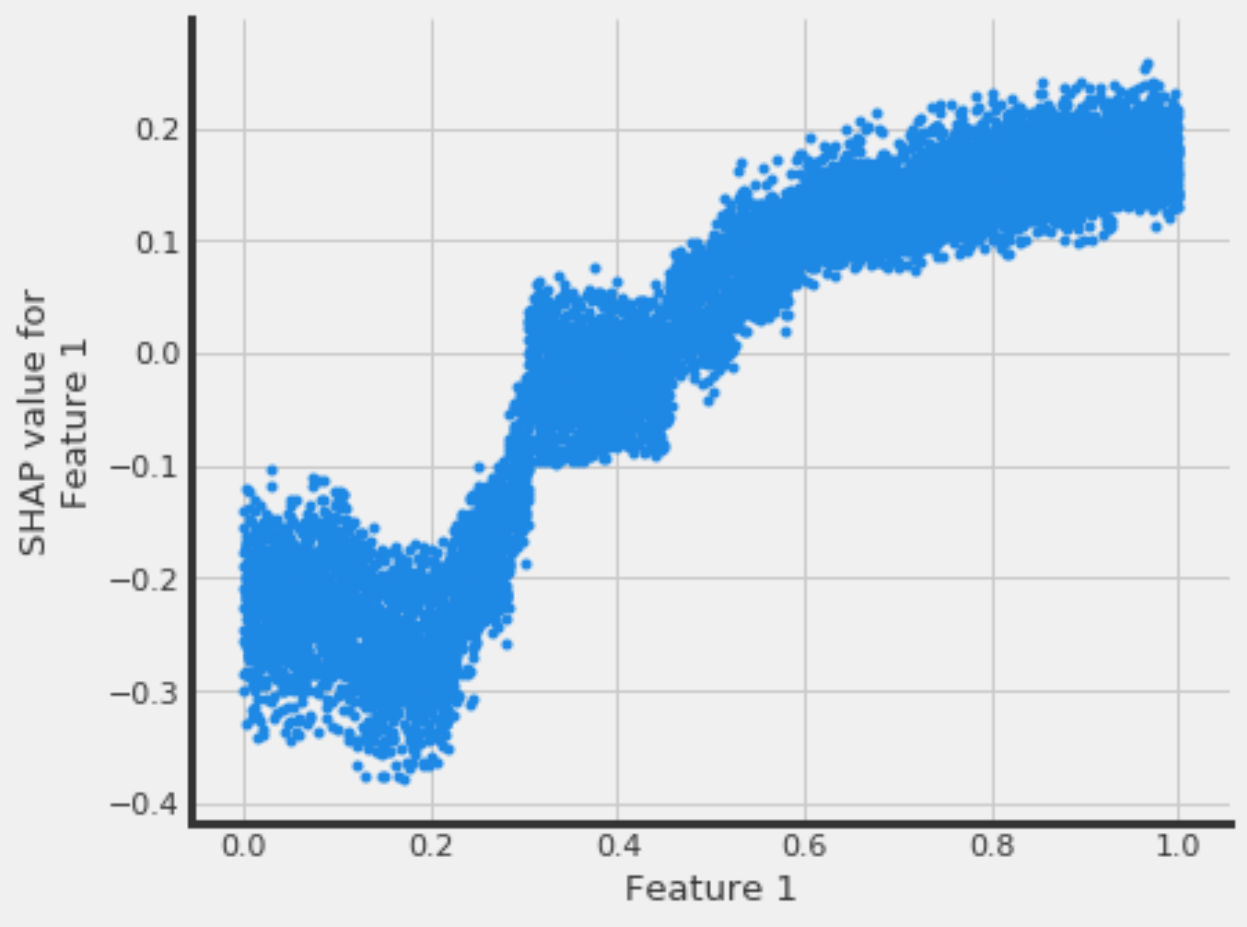
Causal ML provides methods to interpret the treatment effect models trained, where we provide more sample code in feature_interpretations_example.ipynb notebook.

7.1 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(X=X, shap_dict=shap_slearner)

# 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')
```

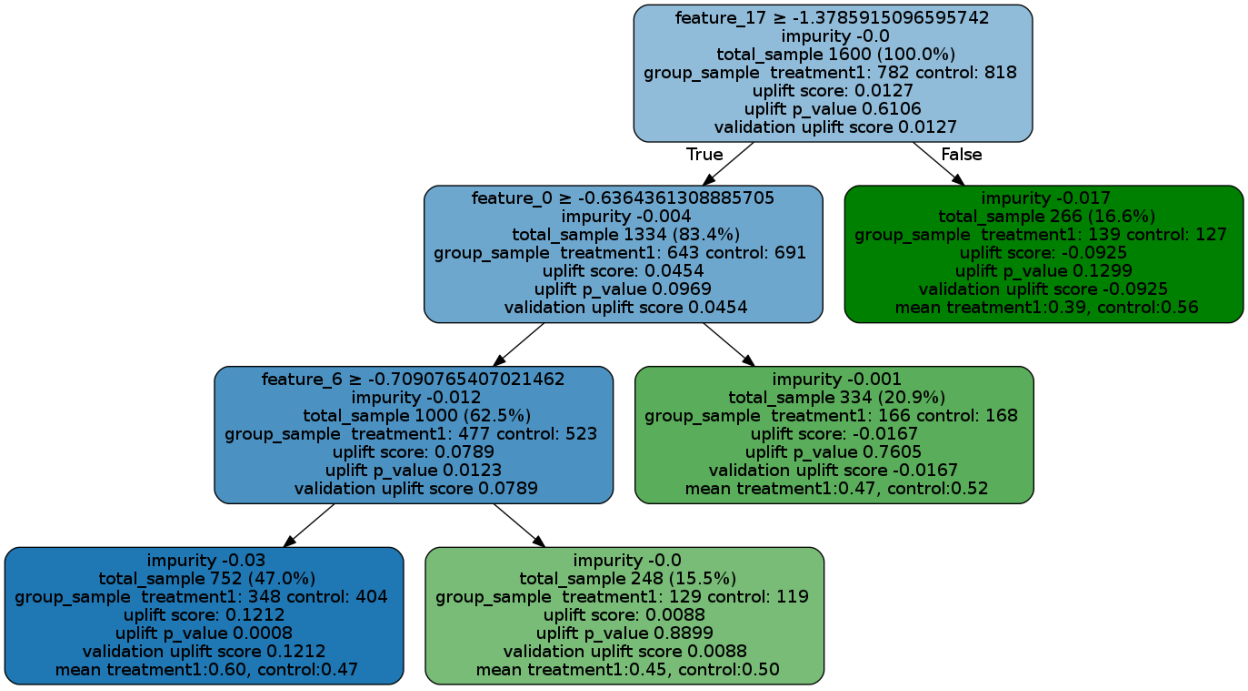

treatment_A
tiger
stars
quixotic
merciful
fireman
touch
clammy
adhesive
wrap
sweltering
change
lethal
damp
shelf
rigid
barbarous
playground
nonchalant
cute
rain
offer
eight
dependent
future
lip
0.0
0.2
0.4
0.6
0.8
1.0


treatment_A
tiger
stars
quixotic
merciful
fireman
lip
wrap
clammy
sweltering
adhesive
change
touch
future
dependent
lethal
shelf
barbarous
offer
rain
playground
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
SHAP value (impact on model output)
High
Low
Feature value



7.2 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
from causalml.dataset import make_uplift_classification

df, x_names = make_uplift_classification()
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[x_names].values,
                 treatment=df['treatment_group_key'].values,
                 y=df['conversion'].values)

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

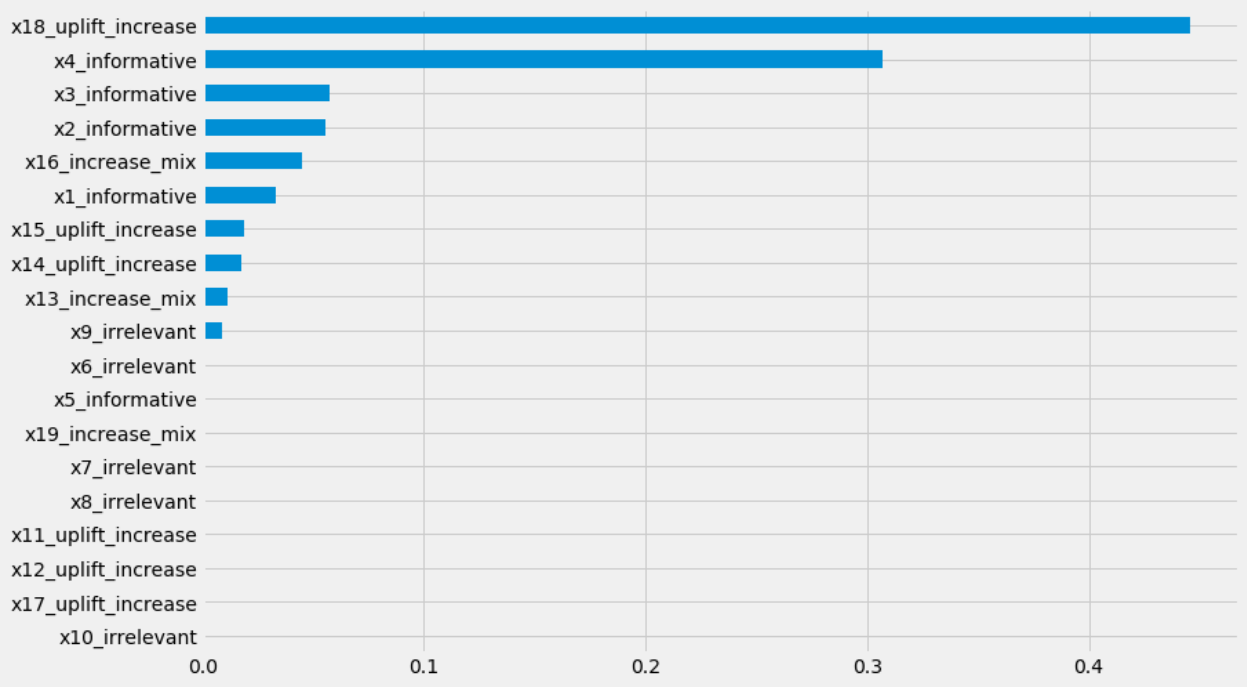


Please see below for how to read the plot, and uplift_tree_visualization.ipynb example notebook is provided in the repo.

- 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.

7.3 Uplift Tree Feature Importances

```
pd.Series(uplift_model.feature_importances_, index=x_names).sort_values().plot(kind=
↪'barh', figsize=(12,8))
```


x18_uplift_increase
x4_informative
x3_informative
x2_informative
x16_increase_mix
x1_informative
x15_uplift_increase
x14_uplift_increase
x13_increase_mix
x9_irrelevant
x6_irrelevant
x5_informative
x19_increase_mix
x7_irrelevant
x8_irrelevant
x11_uplift_increase
x12_uplift_increase
x17_uplift_increase
x10_irrelevant
0.0
0.1
0.2
0.3
0.4

CHAPTER

EIGHT

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.

8.1 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.

8.1.1 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.

8.1.2 User Level/Segment Level/Cohort Level Consistency

We can also evaluate user-level/segment level/cohort level estimation consistency by conducting T-test.

8.1.3 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.

8.2 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 [19]:

8.2.1 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:

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

With :

$$trim_\eta(x) = \max(\eta, \min(x, 1-\eta))$$

The outcome variable is:

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

8.2.2 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

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

8.2.3 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:

$$P(W_i = 1|X_i) = \frac{1}{1+\exp X_{i2}+X_{i3}}$$

The outcome variable is:

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

8.2.4 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:

$$P(W_i = 1|X_i) = \frac{1}{1+\exp -X_{i1}+\exp -X_{i2}}$$

The outcome variable is:

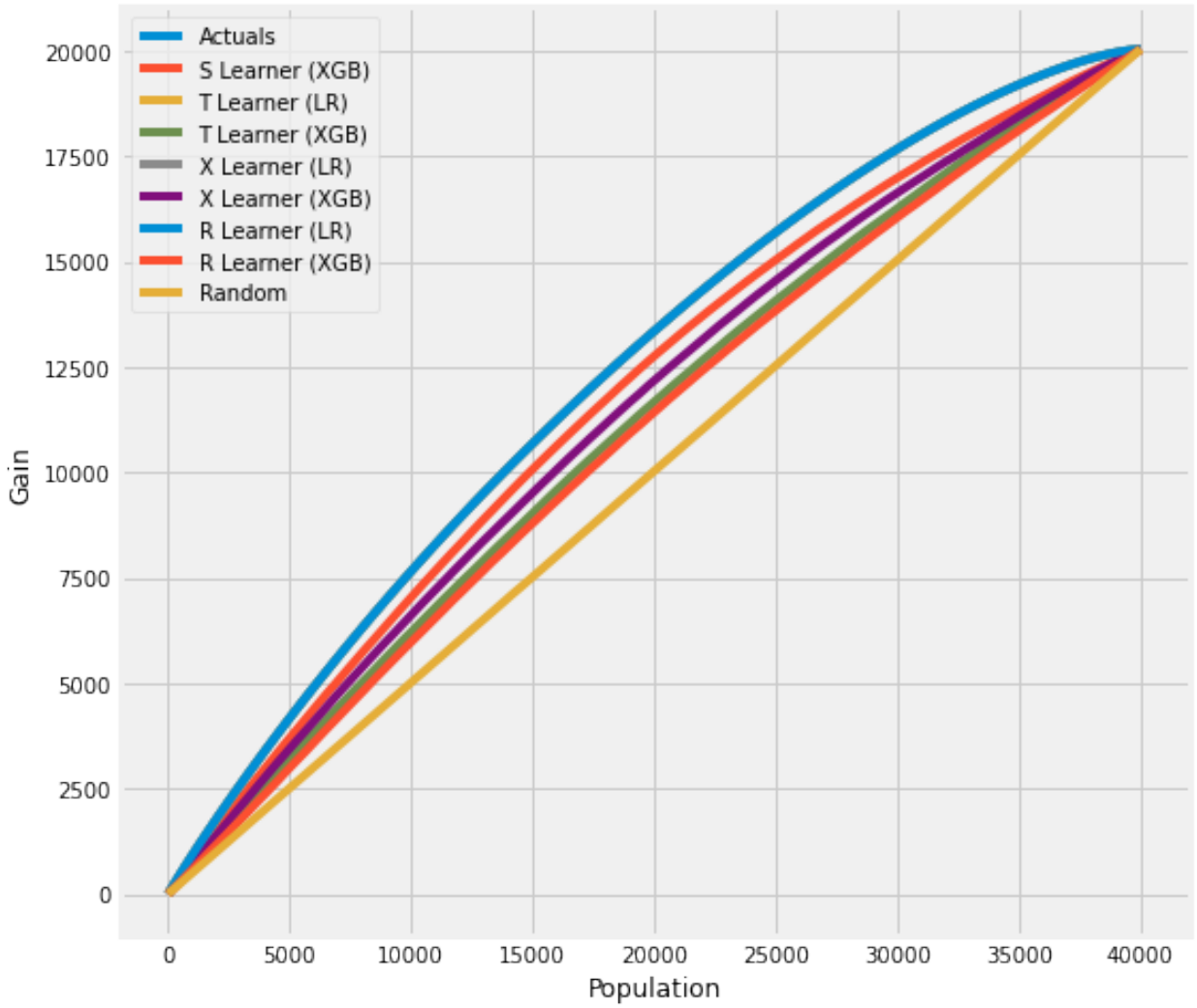
$$y_i = \frac{1}{2}\big(max(X_{i1}+X_{i2}+X_{i3},0)+max(X_{i4}+X_{i5},0)\big)+(W_i-0.5)(max(X_{i1}+X_{i2}+X_{i3},0)-max(X_{i4},X_{i5},0))$$

8.3 Validation with Uplift Curve (AUUC)

We can validate the estimation by evaluating and comparing the uplift gains with AUUC (Area Under Uplift Curve), it calculates cumulative gains. Please find more details in meta_learners_with_synthetic_data.ipynb example notebook.

```
from causalml.dataset import *
from causalml.metrics import *
# Single simulation
train_preds, valid_preds = get_synthetic_preds_holdout(simulate_nuisance_and_easy_
↪treatment,
                                                       n=50000,
                                                       valid_size=0.2)
# Cumulative Gain AUUC values for a Single Simulation of Validaiton Data
get_synthetic_auuc(valid_preds)
```

	Learner	cum_gain_auuc
0	Actuals	4.942619e+06
6	R Learner (LR)	4.941699e+06
2	T Learner (LR)	4.941643e+06
4	X Learner (LR)	4.941643e+06
1	S Learner (XGB)	4.723843e+06
5	X Learner (XGB)	4.580028e+06
3	T Learner (XGB)	4.446320e+06
7	R Learner (XGB)	4.364945e+06
8	Random	4.010939e+06



For data with skewed treatment, it is sometimes advantageous to use *Targeted maximum likelihood estimation (TMLE) for ATE* to generate the AUUC curve for validation, as TMLE provides a more accurate estimation of ATE. Please find validation_with_tmle.ipynb example notebook for details.

8.4 Validation with Sensitivity Analysis

Sensitivity analysis aim to check the robustness of the unconfoundeness assumption. If there is hidden bias (unobserved confounders), it detemineds how severe whould have to be to change conclusion by examine the average treatment effect estimation.

We implemented the following methods to conduct sensitivity analysis:

8.4.1 Placebo Treatment

Replace treatment with a random variable.

8.4.2 Irrelevant Additional Confounder

Add a random common cause variable.

8.4.3 Subset validation

Remove a random subset of the data.

8.4.4 Random Replace

Random replace a covariate with an irrelevant variable.

8.4.5 Selection Bias

Blackwell(2013) <https://www.mattblackwell.org/files/papers/sens.pdf> introduced an approach to sensitivity analysis for causal effects that directly models confounding or selection bias.

One Sided Confounding Function: here as the name implies, this function can detect sensitivity to one-sided selection bias, but it would fail to detect other deviations from ignobility. That is, it can only determine the bias resulting from the treatment group being on average better off or the control group being on average better off.

Alignment Confounding Function: this type of bias is likely to occur when units select into treatment and control based on their predicted treatment effects

The sensitivity analysis is rigid in this way because the confounding function is not identified from the data, so that the causal model in the last section is only identified conditional on a specific choice of that function. The goal of the sensitivity analysis is not to choose the "correct" confounding function, since we have no way of evaluating this correctness. By its very nature, unmeasured confounding is unmeasured. Rather, the goal is to identify plausible deviations from ignobility and test sensitivity to those deviations. The main harm that results from the incorrect specification of the confounding function is that hidden biases remain hidden.

CHAPTER

NINE

CAUSALML PACKAGE

9.1 Submodules

9.2 causalml.inference.tree module

class causalml.inference.tree.**CausalRandomForestRegressor**(*n_estimators: int = 100*, ***, *control_name: int | str = 0*, *criterion: str = 'causal_mse'*, *alpha: float = 0.05*, *max_depth: int | None = None*, *min_samples_split: int = 60*, *min_samples_leaf: int = 100*, *min_weight_fraction_leaf: float = 0.0*, *max_features: int | float | str = 1.0*, *max_leaf_nodes: int | None = None*, *min_impurity_decrease: float = -inf*, *bootstrap: bool = True*, *oob_score: bool = False*, *n_jobs: int | None = None*, *random_state: int | None = None*, *verbose: int = 0*, *warm_start: bool = False*, *ccp_alpha: float = 0.0*, *groups_penalty: float = 0.5*, *max_samples: int | None = None*, *groups_cnt: bool = True*)

Bases: ForestRegressor

calculate_error(*X_train: ndarray*, *X_test: ndarray*, *inbag: ndarray | None = None*, *calibrate: bool = True*, *memory_constrained: bool = False*, *memory_limit: int | None = None*) → ndarray

Calculate error bars from scikit-learn RandomForest estimators Source: https://github.com/scikit-learn-contrib/forest-confidence-interval

Parameters

- **X_train** – (np.ndarray), training subsample of feature matrix, (n_train_sample, n_features)
- **X_test** – (np.ndarray), test subsample of feature matrix, (n_train_sample, n_features)

- **inbag** – (ndarray, optional), The inbag matrix that fit the data. If set to *None* (default) it will be inferred from the forest. However, this only works for trees for which bootstrapping was set to *True*. That is, if sampling was done with replacement. Otherwise, users need to provide their own inbag matrix.
- **calibrate** – (boolean, optional) Whether to apply calibration to mitigate Monte Carlo noise. Some variance estimates may be negative due to Monte Carlo effects if the number of trees in the forest is too small. To use calibration, Default: True
- **memory_constrained** – (boolean, optional) Whether or not there is a restriction on memory. If False, it is assumed that a ndarray of shape (n_train_sample,n_test_sample) fits in main memory. Setting to True can actually provide a speedup if memory_limit is tuned to the optimal range.
- **memory_limit** – (int, optional) An upper bound for how much memory the intermediate matrices will take up in Megabytes. This must be provided if memory_constrained=True.

Returns
(np.ndarray), An array with the unbiased sampling variance for a RandomForest object.

fit (*X: ndarray*, *treatment: ndarray*, *y: ndarray*)

Fit Causal RandomForest :param X: (np.ndarray), feature matrix :param treatment: (np.ndarray), treatment vector :param y: (np.ndarray), outcome vector

Returns
self

predict (*X: ndarray*, *with_outcomes: bool = False*) → ndarray

Predict individual treatment effects

Parameters

- **X** (*np.matrix*) – a feature matrix
- **with_outcomes** (*bool*) – include outcomes Y_hat(X|T=0), Y_hat(X|T=1) along with individual treatment effect

Returns

individual treatment effect (ITE), dim=nx1
or ITE with outcomes [Y_hat(X|T=0), Y_hat(X|T=1), ITE], dim=nx3

Return type
(np.matrix)

class causalml.inference.tree.**CausalTreeRegressor**(*, *criterion: str = 'causal_mse'*, *splitter: str = 'best'*, *alpha: float = 0.05*, *control_name: int | str = 0*, *max_depth: int | None = None*, *min_samples_split: int | float = 60*, *min_weight_fraction_leaf: float = 0.0*, *max_features: int | float | str | None = None*, *max_leaf_nodes: int | None = None*, *min_impurity_decrease: float = -inf*, *ccp_alpha: float = 0.0*, *groups_penalty: float = 0.5*, *min_samples_leaf: int = 100*, *random_state: int | None = None*, *groups_cnt: bool = False*, *groups_cnt_mode: str = 'nodes'*)

Bases: RegressorMixin, BaseCausalDecisionTree

A Causal Tree regressor class. The Causal Tree is a decision tree regressor with a split criteria for treatment effects. Details are available at Athey and Imbens (2015).

bootstrap(*X: ndarray*, *treatment: ndarray*, *y: ndarray*, *sample_size: int*, *seed: int*) → ndarray

Runs a single bootstrap.

Fits on bootstrapped sample, then predicts on whole population.

Parameters

- **X** (*np.ndarray*) – a feature matrix
- **treatment** (*np.ndarray*) – a treatment vector
- **y** (*np.ndarray*) – an outcome vector
- **sample_size** (*int*) – bootstrap sample size
- **seed** – (int): bootstrap seed

Returns
bootstrap predictions

Return type
(np.ndarray)

bootstrap_pool(***kw*)

estimate_ate(*X: ndarray*, *treatment: ndarray*, *y: ndarray*) → tuple

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

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

fit(*X: ndarray*, *y: ndarray*, *treatment: ndarray | None = None*, *sample_weight: ndarray | None = None*, *check_input=False*)

Fit CausalTreeRegressor :param X: : (np.ndarray), feature matrix :param y: : (np.ndarray), outcome vector :param treatment: : (np.ndarray), treatment vector :param sample_weight: (np.ndarray), sample_weight :param check_input: (bool)

Returns
self

fit_predict(*X: ndarray*, *treatment: ndarray*, *y: ndarray*, *return_ci: bool = False*, *n_bootstraps: int = 1000*, *bootstrap_size: int = 10000*, *n_jobs: int = 1*, *verbose: bool = False*) → tuple

Fit the Causal Tree model and predict treatment effects.

Parameters

- **X** (*np.matrix*) – a feature matrix
- **treatment** (*np.array*) – a treatment vector
- **y** (*np.array*) – 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
- **n_jobs** (*int*) – the number of jobs for bootstrap
- **verbose** (*str*) – whether to output progress logs

Returns

- te (numpy.ndarray): Predictions of treatment effects.
- te_lower (numpy.ndarray, optional): lower bounds of treatment effects
- te_upper (numpy.ndarray, optional): upper bounds of treatment effects

Return type
(tuple)

predict (*X: ndarray*, *with_outcomes: bool = False*, *check_input=True*) → ndarray

Predict individual treatment effects

Parameters

- **X** (*np.matrix*) – a feature matrix
- **with_outcomes** (*bool*) – include outcomes Y_hat(X|T=0), Y_hat(X|T=1) along with individual treatment effect
- **check_input** (*bool*) – Allow to bypass several input checking.

Returns

individual treatment effect (ITE), dim=nx1
or ITE with outcomes [Y_hat(X|T=0), Y_hat(X|T=1), ITE], dim=nx3

Return type
(np.matrix)

class causalml.inference.tree.**DecisionTree**(*classes_*, *col=-1*, *value=None*, *trueBranch=None*, *falseBranch=None*, *results=None*, *summary=None*, *maxDiffTreatment=None*, *maxDiffSign=1.0*, *nodeSummary=None*, *backupResults=None*, *bestTreatment=None*, *upliftScore=None*, *matchScore=None*)

Bases: object

Tree Node Class

Tree node class to contain all the statistics of the tree node.

Parameters

- **classes** (*list of str*) – A list of the control and treatment group names.
- **col** (*int, optional (default = -1)*) – The column index for splitting the tree node to children nodes.
- **value** (*float, optional (default = None)*) – The value of the feature column to split the tree node to children nodes.
- **trueBranch** (*object of DecisionTree*) – The true branch tree node (feature > value).
- **falseBranch** (*object of DecisionTree*) – The false branch tree node (feature > value).
- **results** (*list of float*) – The classification probability P(Y=1|T) for each of the control and treatment groups in the tree node.
- **summary** (*list of list*) – Summary statistics of the tree nodes, including impurity, sample size, uplift score, etc.

- **maxDiffTreatment** (*int*) – The treatment index generating the maximum difference between the treatment and control groups.
- **maxDiffSign** (*float*) – The sign of the maximum difference (1. or -1.).
- **nodeSummary** (*list of list*) – Summary statistics of the tree nodes [P(Y=1|T), N(T)], where y_mean stands for the target metric mean and n is the sample size.
- **backupResults** (*list of float*) – The positive probabilities in each of the control and treatment groups in the parent node. The parent node information is served as a backup for the children node, in case no valid statistics can be calculated from the children node, the parent node information will be used in certain cases.
- **bestTreatment** (*int*) – The treatment index providing the best uplift (treatment effect).
- **upliftScore** (*list*) – The uplift score of this node: [max_Diff, p_value], where max_Diff stands for the maximum treatment effect, and p_value stands for the p_value of the treatment effect.
- **matchScore** (*float*) – The uplift score by filling a trained tree with validation dataset or testing dataset.

class causalml.inference.tree.**UpliftRandomForestClassifier**(*control_name*, *n_estimators=10*, *max_features=10*, *random_state=None*, *max_depth=5*, *min_samples_leaf=100*, *min_samples_treatment=10*, *n_reg=10*, *early_stopping_eval_diff_scale=1*, *evaluationFunction='KL'*, *normalization=True*, *honesty=False*, *estimation_sample_size=0.5*, *n_jobs=-1*, *joblib_prefer: unicode = 'threads'*)

Bases: object

Uplift Random Forest for Classification Task.

Parameters

- **n_estimators** (*integer, optional (default=10)*) – The number of trees in the uplift random forest.
- **evaluationFunction** (*string*) – Choose from one of the models: 'KL', 'ED', 'Chi', 'CTS', 'DDP', 'IT', 'CIT', 'IDDP'.
- **max_features** (*int, optional (default=10)*) – The number of features to consider when looking for the best split.
- **random_state** (*int, RandomState instance or None (default=None)*) – A random seed or *np.random.RandomState* to control randomness in building the trees and forest.
- **max_depth** (*int, optional (default=5)*) – The maximum depth of the tree.
- **min_samples_leaf** (*int, optional (default=100)*) – The minimum number of samples required to be split at a leaf node.

- **min_samples_treatment** (*int, optional (default=10)*) – The minimum number of samples required of the experiment group to be split at a leaf node.
- **n_reg** (*int, optional (default=10)*) – The regularization parameter defined in Rzepakowski et al. 2012, the weight (in terms of sample size) of the parent node influence on the child node, only effective for 'KL', 'ED', 'Chi', 'CTS' methods.
- **early_stopping_eval_diff_scale** (*float, optional (default=1)*) – If train and valid uplift score diff bigger than min(train_uplift_score,valid_uplift_score)/early_stopping_eval_diff_scale, stop.
- **control_name** (*string*) – The name of the control group (other experiment groups will be regarded as treatment groups)
- **normalization** (*boolean, optional (default=True)*) – The normalization factor defined in Rzepakowski et al. 2012, correcting for tests with large number of splits and imbalanced treatment and control splits
- **honesty** (*bool (default=False)*) – True if the honest approach based on "Athey, S., & Imbens, G. (2016). Recursive partitioning for heterogeneous causal effects." shall be used.
- **estimation_sample_size** (*float (default=0.5)*) – Sample size for estimating the CATE score in the leaves if honesty == True.
- **n_jobs** (*int, optional (default=-1)*) – The parallelization parameter to define how many parallel jobs need to be created. This is passed on to joblib library for parallelizing uplift-tree creation and prediction.
- **joblib_prefer** (*str, optional (default="threads")*) – The preferred backend for joblib (passed as *prefer* to joblib.Parallel). See the joblib documentation for valid values.
- **Outputs** –
- **----------** –
- **df_res** (*pandas dataframe*) – A user-level results dataframe containing the estimated individual treatment effect.

static bootstrap(*X*, *treatment*, *y*, *X_val*, *treatment_val*, *y_val*, *tree*)

fit(*X*, *treatment*, *y*, *X_val=None*, *treatment_val=None*, *y_val=None*)

Fit the UpliftRandomForestClassifier.

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **treatment** (*array-like, shape = [num_samples]*) – An array containing the treatment group for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.
- **X_val** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to valid the uplift model.
- **treatment_val** (*array-like, shape = [num_samples]*) – An array containing the validation treatment group for each unit.

- **y_val** (*array-like, shape = [num_samples]*) – An array containing the validation outcome of interest for each unit.

predict(*X*, *full_output=False*)

Returns the recommended treatment group and predicted optimal probability conditional on using the recommended treatment group.

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **full_output** (*bool, optional (default=False)*) – Whether the UpliftTree algorithm returns upliftScores, pred_nodes alongside the recommended treatment group and p_hat in the treatment group.

Returns

- **y_pred_list** (*ndarray, shape = (num_samples, num_treatments])*) – An ndarray containing the predicted treatment effect of each treatment group for each sample
- **df_res** (*DataFrame, shape = [num_samples, (num_treatments * 2 + 3)]*) – If *full_output* is *True*, a DataFrame containing the predicted outcome of each treatment and control group, the treatment effect of each treatment group, the treatment group with the highest treatment effect, and the maximum treatment effect for each sample.

class causalml.inference.tree.**UpliftTreeClassifier**(*control_name*, *max_features=None*, *max_depth=3*, *min_samples_leaf=100*, *min_samples_treatment=10*, *n_reg=100*, *early_stopping_eval_diff_scale=1*, *evaluationFunction='KL'*, *normalization=True*, *honesty=False*, *estimation_sample_size=0.5*, *random_state=None*)

Bases: object

Uplift Tree Classifier for Classification Task.

A uplift tree classifier estimates the individual treatment effect by modifying the loss function in the classification trees.

The uplift tree classifier is used in uplift random forest to construct the trees in the forest.

Parameters

- **evaluationFunction** (*string*) – Choose from one of the models: 'KL', 'ED', 'Chi', 'CTS', 'DDP', 'IT', 'CIT', 'IDDP'.
- **max_features** (*int, optional (default=None)*) – The number of features to consider when looking for the best split.
- **max_depth** (*int, optional (default=3)*) – The maximum depth of the tree.
- **min_samples_leaf** (*int, optional (default=100)*) – The minimum number of samples required to be split at a leaf node.
- **min_samples_treatment** (*int, optional (default=10)*) – The minimum number of samples required of the experiment group to be split at a leaf node.
- **n_reg** (*int, optional (default=100)*) – The regularization parameter defined in Rzepakowski et al. 2012, the weight (in terms of sample size) of the parent node influence on the child node, only effective for 'KL', 'ED', 'Chi', 'CTS' methods.

- **early_stopping_eval_diff_scale** (*float, optional (default=1)*) – If train and valid uplift score diff bigger than min(train_uplift_score,valid_uplift_score)/early_stopping_eval_diff_scale, stop.
- **control_name** (*string*) – The name of the control group (other experiment groups will be regarded as treatment groups).
- **normalization** (*boolean, optional (default=True)*) – The normalization factor defined in Rzepakowski et al. 2012, correcting for tests with large number of splits and imbalanced treatment and control splits.
- **honesty** (*bool (default=False)*) – True if the honest approach based on "Athey, S., & Imbens, G. (2016). Recursive partitioning for heterogeneous causal effects." shall be used. If 'IDDP' is used as evaluation function, this parameter is automatically set to true.
- **estimation_sample_size** (*float (default=0.5)*) – Sample size for estimating the CATE score in the leaves if honesty == True.
- **random_state** (*int, RandomState instance or None (default=None)*) – A random seed or *np.random.RandomState* to control randomness in building a tree.

static arr_evaluate_CIT (*cur_node_summary_p*, *cur_node_summary_n*, *left_node_summary_p*, *left_node_summary_n*, *right_node_summary_p*, *right_node_summary_n*)

Calculate likelihood ratio test statistic as split evaluation criterion for a given node

NOTE: n_class should be 2.

Parameters

- **cur_node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the current node, i.e. [P(Y=1|T=i)…]
- **cur_node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]
- **left_node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the left node, i.e. [P(Y=1|T=i)…]
- **left_node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the left node, i.e. [N(T=i)…]
- **right_node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the right node, i.e. [P(Y=1|T=i)…]
- **right_node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the right node, i.e. [N(T=i)…]

Returns
lrt

Return type
Likelihood ratio test statistic

static arr_evaluate_CTS (*node_summary_p*, *node_summary_n*)

Calculate CTS (conditional treatment selection) as split evaluation criterion for a given node.

Parameters

- **node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the current node, i.e. [P(Y=1|T=i)…]
- **node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]

Returns
d_res

Return type
CTS score

static arr_evaluate_Chi (*node_summary_p*, *node_summary_n*)

Calculate Chi-Square statistic as split evaluation criterion for a given node.

Parameters

- **node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the current node, i.e. [P(Y=1|T=i)…]
- **node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]

Returns
d_res

Return type
Chi-Square

static arr_evaluate_DDP (*node_summary_p*, *node_summary_n*)

Calculate Delta P as split evaluation criterion for a given node.

Parameters

- **node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the current node, i.e. [P(Y=1|T=i)…]
- **node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]

Returns
d_res

Return type
Delta P

static arr_evaluate_ED (*node_summary_p*, *node_summary_n*)

Calculate Euclidean Distance as split evaluation criterion for a given node.

Parameters

- **node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the current node, i.e. [P(Y=1|T=i)…]
- **node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]

Returns
d_res

Return type
Euclidean Distance

static arr_evaluate_IDDP(*node_summary_p*, *node_summary_n*)

Calculate Delta P as split evaluation criterion for a given node.

Parameters

- **node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the current node, i.e. [P(Y=1|T=i)…]
- **node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]

Returns
d_res

Return type
Delta P

static arr_evaluate_IT(*left_node_summary_p*, *left_node_summary_n*, *right_node_summary_p*, *right_node_summary_n*)

Calculate Squared T-Statistic as split evaluation criterion for a given node

NOTE: n_class should be 2.

Parameters

- **left_node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the left node, i.e. [P(Y=1|T=i)…]
- **left_node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the left node, i.e. [N(T=i)…]
- **right_node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the right node, i.e. [P(Y=1|T=i)…]
- **right_node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the right node, i.e. [N(T=i)…]

Returns
g_s

Return type
Squared T-Statistic

static arr_evaluate_KL(*node_summary_p*, *node_summary_n*)

Calculate KL Divergence as split evaluation criterion for a given node. Modified to accept new node summary format.

Parameters

- **node_summary_p** (*array of shape [n_class]*) – Has type numpy.double. The positive probabilities of each of the control and treament groups of the current node, i.e. [P(Y=1|T=i)…]
- **node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]

Returns
d_res

Return type
KL Divergence

arr_normI (*cur_node_summary_n*, *left_node_summary_n*, *alpha: float = 0.9*, *currentDivergence: float = 0.0*) → float

Normalization factor.

Parameters

- **cur_node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the current node, i.e. [N(T=i)…]
- **left_node_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. The counts of each of the control and treament groups of the left node, i.e. [N(T=i)…]
- **alpha** (*float*) – The weight used to balance different normalization parts.

Returns
norm_res – Normalization factor.

Return type
float

static classify (*observations*, *tree*, *dataMissing=False*)

Classifies (prediction) the observations according to the tree.

Parameters

- **observations** (*list of list*) – The internal data format for the training data (combining X, Y, treatment).
- **dataMissing** (*boolean, optional (default = False)*) – An indicator for if data are missing or not.

Returns
The results in the leaf node.

Return type
tree.results, tree.upliftScore

static divideSet (*X*, *treatment_idx*, *y*, *column*, *value*)

Tree node split.

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group index for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.
- **column** (*int*) – The column used to split the data.
- **value** (*float or int*) – The value in the column for splitting the data.

Returns
(X_l, X_r, treatment_l, treatment_r, y_l, y_r) – The covariates, treatments and outcomes of left node and the right node.

Return type
list of ndarray

static divideSet_len(*X*, *treatment_idx*, *y*, *column*, *value*)

Tree node split.

Modified from dividedSet(), but return the len(X_l) and len(X_r) instead of the split X_l and X_r, to avoid some overhead, intended to be used for finding the split. After finding the best splits, can split to find the X_l and X_r.

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group index for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.
- **column** (*int*) – The column used to split the data.
- **value** (*float or int*) – The value in the column for splitting the data.

Returns
(len_X_l, len_X_r, treatment_l, treatment_r, y_l, y_r) – The covariates nrows, treatments and outcomes of left node and the right node.

Return type
list of ndarray

static evaluate_CIT(*currentNodeSummary*, *leftNodeSummary*, *rightNodeSummary*, *y_l*, *y_r*, *w_l*, *w_r*, *y*, *w*)

Calculate likelihood ratio test statistic as split evaluation criterion for a given node :param currentNodeSummary: The parent node summary statistics :type currentNodeSummary: list of lists :param leftNodeSummary: The left node summary statistics. :type leftNodeSummary: list of lists :param rightNodeSummary: The right node summary statistics. :type rightNodeSummary: list of lists :param y_l: An array containing the outcome of interest for each unit in the left node :type y_l: array-like, shape = [num_samples] :param y_r: An array containing the outcome of interest for each unit in the right node :type y_r: array-like, shape = [num_samples] :param w_l: An array containing the treatment for each unit in the left node :type w_l: array-like, shape = [num_samples] :param w_r: An array containing the treatment for each unit in the right node :type w_r: array-like, shape = [num_samples] :param y: An array containing the outcome of interest for each unit :type y: array-like, shape = [num_samples] :param w: An array containing the treatment for each unit :type w: array-like, shape = [num_samples]

Returns
lrt

Return type
Likelihood ratio test statistic

static evaluate_CTS(*nodeSummary*)

Calculate CTS (conditional treatment selection) as split evaluation criterion for a given node.

Parameters
nodeSummary (*list of list*) – The tree node summary statistics, [P(Y=1|T), N(T)], produced by tree_node_summary() method.

Returns
d_res

Return type
CTS score

static evaluate_Chi(*nodeSummary*)

Calculate Chi-Square statistic as split evaluation criterion for a given node.

Parameters
nodeSummary (*dictionary*) – The tree node summary statistics, produced by tree_node_summary() method.

Returns
d_res

Return type
Chi-Square

static evaluate_DDP(*nodeSummary*)

Calculate Delta P as split evaluation criterion for a given node.

Parameters
nodeSummary (*list of list*) – The tree node summary statistics, [P(Y=1|T), N(T)], produced by tree_node_summary() method.

Returns
d_res

Return type
Delta P

static evaluate_ED(*nodeSummary*)

Calculate Euclidean Distance as split evaluation criterion for a given node.

Parameters
nodeSummary (*dictionary*) – The tree node summary statistics, produced by tree_node_summary() method.

Returns
d_res

Return type
Euclidean Distance

static evaluate_IDDP(*nodeSummary*)

Calculate Delta P as split evaluation criterion for a given node.

Parameters

- **nodeSummary** (*dictionary*) – The tree node summary statistics, produced by tree_node_summary() method.
- **control_name** (*string*) – The control group name.

Returns
d_res

Return type
Delta P

static evaluate_IT(*leftNodeSummary*, *rightNodeSummary*, *w_l*, *w_r*)

Calculate Squared T-Statistic as split evaluation criterion for a given node

Parameters

- **leftNodeSummary** (*list of list*) – The left node summary statistics.
- **rightNodeSummary** (*list of list*) – The right node summary statistics.
- **w_l** (*array-like, shape = [num_samples]*) – An array containing the treatment for each unit in the left node
- **w_r** (*array-like, shape = [num_samples]*) – An array containing the treatment for each unit in the right node

Returns
g_s

Return type
Squared T-Statistic

static evaluate_KL(*nodeSummary*)

Calculate KL Divergence as split evaluation criterion for a given node.

Parameters
nodeSummary (*list of list*) – The tree node summary statistics, [P(Y=1|T), N(T)], produced by tree_node_summary() method.

Returns
d_res

Return type
KL Divergence

fill(*X*, *treatment*, *y*)

Fill the data into an existing tree. This is a higher-level function to transform the original data inputs into lower level data inputs (list of list and tree).

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **treatment** (*array-like, shape = [num_samples]*) – An array containing the treatment group for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.

Returns
self

Return type
object

fillTree(*X*, *treatment_idx*, *y*, *tree*)

Fill the data into an existing tree. This is a lower-level function to execute on the tree filling task.

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group index for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.

- **tree** (*object*) – object of DecisionTree class

Returns
self

Return type
object

fit (*X*, *treatment*, *y*, *X_val=None*, *treatment_val=None*, *y_val=None*)

Fit the uplift model.

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **treatment** (*array-like, shape = [num_samples]*) – An array containing the treatment group for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.

Returns
self

Return type
object

group_uniqueCounts (*treatment_idx*, *y*)

Count sample size by experiment group.

Parameters

- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group index for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.

Returns
results – The negative and positive outcome sample sizes for each of the control and treatment groups.

Return type
list of list

growDecisionTreeFrom (*X*, *treatment_idx*, *y*, *X_val*, *treatment_val_idx*, *y_val*, *early_stopping_eval_diff_scale=1*, *max_depth=10*, *min_samples_leaf=100*, *depth=1*, *min_samples_treatment=10*, *n_reg=100*, *parentNodeSummary_p=None*)

Train the uplift decision tree.

Parameters

- **X** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.
- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group idx for each unit. The dtype should be numpy.int8.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.

- **X_val** (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to valid the uplift model.
- **treatment_val_idx** (*array-like, shape = [num_samples]*) – An array containing the validation treatment group idx for each unit.
- **y_val** (*array-like, shape = [num_samples]*) – An array containing the validation outcome of interest for each unit.
- **max_depth** (*int, optional (default=10)*) – The maximum depth of the tree.
- **min_samples_leaf** (*int, optional (default=100)*) – The minimum number of samples required to be split at a leaf node.
- **depth** (*int, optional (default = 1)*) – The current depth.
- **min_samples_treatment** (*int, optional (default=10)*) – The minimum number of samples required of the experiment group to be split at a leaf node.
- **n_reg** (*int, optional (default=10)*) – The regularization parameter defined in Rzepakowski et al. 2012, the weight (in terms of sample size) of the parent node influence on the child node, only effective for 'KL', 'ED', 'Chi', 'CTS' methods.
- **parentNodeSummary_p** (*array-like, shape [n_class]*) – Node summary probability statistics of the parent tree node.

Return type
object of DecisionTree class

honestApproach (*X_est*, *T_est*, *Y_est*)

Apply the honest approach based on "Athey, S., & Imbens, G. (2016). Recursive partitioning for heterogeneous causal effects." :param X_est: An ndarray of the covariates used to calculate the unbiased estimates in the leafs of the decision tree. :type X_est: ndarray, shape = [num_samples, num_features] :param T_est: An array containing the treatment group for each unit. :type T_est: array-like, shape = [num_samples] :param Y_est: An array containing the outcome of interest for each unit. :type Y_est: array-like, shape = [num_samples]

normI (*n_c: int*, *n_c_left: int*, *n_t: list*, *n_t_left: list*, *alpha: float = 0.9*, *currentDivergence: float = 0.0*) → float

Normalization factor.

Parameters

- **currentNodeSummary** (*list of list*) – The summary statistics of the current tree node, [P(Y=1|T), N(T)].
- **leftNodeSummary** (*list of list*) – The summary statistics of the left tree node, [P(Y=1|T), N(T)].
- **alpha** (*float*) – The weight used to balance different normalization parts.

Returns
norm_res – Normalization factor.

Return type
float

predict (*X*)

Returns the recommended treatment group and predicted optimal probability conditional on using the recommended treatment group.

Parameters
X (*ndarray, shape = [num_samples, num_features]*) – An ndarray of the covariates used to train the uplift model.

Returns
pred – An ndarray of predicted treatment effects across treatments.

Return type
ndarray, shape = [num_samples, num_treatments]

prune(*X*, *treatment*, *y*, *minGain=0.0001*, *rule='maxAbsDiff'*)

Prune the uplift model. :param X: An ndarray of the covariates used to train the uplift model. :type X: ndarray, shape = [num_samples, num_features] :param treatment: An array containing the treatment group for each unit. :type treatment: array-like, shape = [num_samples] :param y: An array containing the outcome of interest for each unit. :type y: array-like, shape = [num_samples] :param minGain: The minimum gain required to make a tree node split. The children

tree branches are trimmed if the actual split gain is less than the minimum gain.

Parameters
rule(*string, optional (default = 'maxAbsDiff')*) – The prune rules. Supported values are 'maxAbsDiff' for optimizing the maximum absolute difference, and 'bestUplift' for optimizing the node-size weighted treatment effect.

Returns
self

Return type
object

pruneTree(*X*, *treatment_idx*, *y*, *tree*, *rule='maxAbsDiff'*, *minGain=0.0*, *n_reg=0*, *parentNodeSummary=None*)

Prune one single tree node in the uplift model. :param X: An ndarray of the covariates used to train the uplift model. :type X: ndarray, shape = [num_samples, num_features] :param treatment_idx: An array containing the treatment group index for each unit. :type treatment_idx: array-like, shape = [num_samples] :param y: An array containing the outcome of interest for each unit. :type y: array-like, shape = [num_samples] :param rule: The prune rules. Supported values are 'maxAbsDiff' for optimizing the maximum absolute difference, and

'bestUplift' for optimizing the node-size weighted treatment effect.

Parameters

- **minGain**(*float, optional (default = 0.)*) – The minimum gain required to make a tree node split. The children tree branches are trimmed if the actual split gain is less than the minimum gain.
- **n_reg**(*int, optional (default=0)*) – The regularization parameter defined in Rzepakowski et al. 2012, the weight (in terms of sample size) of the parent node influence on the child node, only effective for 'KL', 'ED', 'Chi', 'CTS' methods.
- **parentNodeSummary**(*list of list, optional (default = None)*) – Node summary statistics, [P(Y=1|T), N(T)] of the parent tree node.

Returns
self

Return type
object

tree_node_summary(*treatment_idx*, *y*, *min_samples_treatment=10*, *n_reg=100*, *parentNodeSummary=None*)

Tree node summary statistics.

Parameters

- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group index for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.
- **min_samples_treatment** (*int, optional (default=10)*) – The minimum number of samples required of the experiment group t be split at a leaf node.
- **n_reg** (*int, optional (default=10)*) – The regularization parameter defined in Rzepakowski et al. 2012, the weight (in terms of sample size) of the parent node influence on the child node, only effective for 'KL', 'ED', 'Chi', 'CTS' methods.
- **parentNodeSummary** (*list of list*) – The positive probabilities and sample sizes of each of the control and treatment groups in the parent node.

Returns

nodeSummary – The positive probabilities and sample sizes of each of the control and treatment groups in the current node.

Return type

list of list

static tree_node_summary_from_counts(*group_count_arr*, *out_summary_p*, *out_summary_n*, *parentNodeSummary_p*, *has_parent_summary*, *min_samples_treatment=10*, *n_reg=100*)

Tree node summary statistics.

Modified from tree_node_summary_to_arr, to use different format for the summary and to calculate based on already calculated group counts. Instead of [[P(Y=1|T=0), N(T=0)], [P(Y=1|T=1), N(T=1)], …], use two arrays [N(T=i)…] and [P(Y=1|T=i)…].

Parameters

- **group_count_arr** (*array of shape [2*n_class]*) – Has type numpy.int32. The grounp counts, where entry 2*i is N(Y=0, T=i), and entry 2*i+1 is N(Y=1, T=i).
- **out_summary_p** (*array of shape [n_class]*) – Has type numpy.double. To be filled with the positive probabilities of each of the control and treament groups of the current node.
- **out_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. To be filled with the counts of each of the control and treament groups of the current node.
- **parentNodeSummary_p** (*array of shape [n_class]*) – The positive probabilities of each of the control and treatment groups in the parent node.
- **has_parent_summary** (*bool as int*) – If True (non-zero), then parentNodeSummary_p is a valid parent node summary probabilities. If False (0), assume no parent node summary and parentNodeSummary_p is not touched.
- **min_samples_treatment** (*int, optional (default=10)*) – The minimum number of samples required of the experiment group t be split at a leaf node.
- **n_reg** (*int, optional (default=10)*) – The regularization parameter defined in Rzepakowski et al. 2012, the weight (in terms of sample size) of the parent node influence on the child node, only effective for 'KL', 'ED', 'Chi', 'CTS' methods.

Return type

No return values, but will modify out_summary_p and out_summary_n.

static tree_node_summary_to_arr(*treatment_idx*, *y*, *out_summary_p*, *out_summary_n*, *buf_count_arr*, *parentNodeSummary_p*, *has_parent_summary*, *min_samples_treatment=10*, *n_reg=100*)

Tree node summary statistics. Modified from tree_node_summary, to use different format for the summary. Instead of [[P(Y=1|T=0), N(T=0)], [P(Y=1|T=1), N(T=1)], …], use two arrays [N(T=i)…] and [P(Y=1|T=i)…].

Parameters

- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group index for each unit. Has type numpy.int8.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit. Has type numpy.int8.
- **out_summary_p** (*array of shape [n_class]*) – Has type numpy.double. To be filled with the positive probabilities of each of the control and treament groups of the current node.
- **out_summary_n** (*array of shape [n_class]*) – Has type numpy.int32. To be filled with the counts of each of the control and treament groups of the current node.
- **buf_count_arr** (*array of shape [2*n_class]*) – Has type numpy.int32. To be use as temporary buffer for group_uniqueCounts_to_arr.
- **parentNodeSummary_p** (*array of shape [n_class]*) – The positive probabilities of each of the control and treatment groups in the parent node.
- **has_parent_summary** (*bool as int*) – If True (non-zero), then parentNodeSummary_p is a valid parent node summary probabilities. If False (0), assume no parent node summary and parentNodeSummary_p is not touched.
- **min_samples_treatment** (*int, optional (default=10)*) – The minimum number of samples required of the experiment group t be split at a leaf node.
- **n_reg** (*int, optional (default=10)*) – The regularization parameter defined in Rzepakowski et al. 2012, the weight (in terms of sample size) of the parent node influence on the child node, only effective for 'KL', 'ED', 'Chi', 'CTS' methods.

Return type
No return values, but will modify out_summary_p and out_summary_n.

uplift_classification_results(*treatment_idx*, *y*)

Classification probability for each treatment in the tree node.

Parameters

- **treatment_idx** (*array-like, shape = [num_samples]*) – An array containing the treatment group index for each unit.
- **y** (*array-like, shape = [num_samples]*) – An array containing the outcome of interest for each unit.

Returns
res – The positive probabilities P(Y = 1) of each of the control and treatment groups

Return type
list of list

causalml.inference.tree.**cat_continuous**(*x*, *granularity='Medium'*)

Categorize (bin) continuous variable based on percentile.

Parameters

- **x** (*list*) – Feature values.
- **granularity** (*string, optional, (default = 'Medium')*) – Control the granularity of the bins, optional values are: 'High', 'Medium', 'Low'.

Returns
res – List of percentile bins for the feature value.

Return type
list

causalml.inference.tree.**cat_group**(*dfx*, *kpix*, *n_group=10*)

Category Reduction for Categorical Variables

Parameters

- **dfx** (*dataframe*) – The inputs data dataframe.
- **kpix** (*string*) – The column of the feature.
- **n_group** (*int, optional (default = 10)*) – The number of top category values to be remained, other category values will be put into "Other".

Return type
The transformed categorical feature value list.

causalml.inference.tree.**cat_transform**(*dfx*, *kpix*, *kpi1*)

Encoding string features.

Parameters

- **dfx** (*dataframe*) – The inputs data dataframe.
- **kpix** (*string*) – The column of the feature.
- **kpi1** (*list*) – The list of feature names.

Returns

- **dfx** (*DataFrame*) – The updated dataframe containing the encoded data.
- **kpi1** (*list*) – The updated feature names containing the new dummy feature names.

causalml.inference.tree.**cv_fold_index**(*n*, *i*, *k*, *random_seed=2018*)

Encoding string features.

Parameters

- **dfx** (*dataframe*) – The inputs data dataframe.
- **kpix** (*string*) – The column of the feature.
- **kpi1** (*list*) – The list of feature names.

Returns

- **dfx** (*DataFrame*) – The updated dataframe containing the encoded data.
- **kpi1** (*list*) – The updated feature names containing the new dummy feature names.

causalml.inference.tree.**get_tree_leaves_mask**(*tree*) → ndarray

Get mask array for tree leaves :param tree: CausalTreeRegressor

Tree object

Returns: np.ndarray
Mask array

causalml.inference.tree.**kpi_transform**(*dfx*, *kpi_combo*, *kpi_combo_new*)

Feature transformation from continuous feature to binned features for a list of features

Parameters

- **dfx** (*DataFrame*) – DataFrame containing the features.
- **kpi_combo** (*list of string*) – List of feature names to be transformed
- **kpi_combo_new** (*list of string*) – List of new feature names to be assigned to the transformed features.

Returns
dfx – Updated DataFrame containing the new features.

Return type
DataFrame

causalml.inference.tree.**plot_dist_tree_leaves_values**(*tree: CausalTreeRegressor*, *title: str = 'Leaves values distribution'*, *figsize: tuple = (5, 5)*, *fontsize: int = 12*) → None

Create distplot for tree leaves values :param tree: (CausalTreeRegressor), Tree object :param title: (str), plot title :param figsize: (tuple), figure size :param fontsize: (int), title font size

Returns: None

causalml.inference.tree.**uplift_tree_plot**(*decisionTree*, *x_names*)

Convert the tree to dot graph for plots.

Parameters

- **decisionTree** (*object*) – object of DecisionTree class
- **x_names** (*list*) – List of feature names

Return type
Dot class representing the tree graph.

causalml.inference.tree.**uplift_tree_string**(*decisionTree*, *x_names*)

Convert the tree to string for print.

Parameters

- **decisionTree** (*object*) – object of DecisionTree class
- **x_names** (*list*) – List of feature names

Return type
A string representation of the tree.

9.3 causalml.inference.meta module

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

Bases: BaseLearner

A parent class for DR-learner regressor classes.

A DR-learner estimates treatment effects with machine learning models.

Details of DR-learner are available at Kennedy (2020).

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

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
- **seed** (*int*) – random seed for cross-fitting
- **pretrain** (*bool*) – whether a model has been fit, default False.

Returns

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

fit(*X*, *treatment*, *y*, *p=None*, *seed=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.
- **seed** (*int*) – random seed for cross-fitting

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

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
- **seed** (*int*) – random seed for cross-fitting

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)

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
- **verbose** (*bool, optional*) – whether to output progress logs

Returns
Predictions of treatment effects.

Return type
(numpy.ndarray)

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

Bases: *BaseDRLearner*

A parent class for DR-learner regressor classes.

class causalml.inference.meta.**BaseRClassifier**(*outcome_learner=None*, *effect_learner=None*, *propensity_learner=LogisticRegressionCV(Cs=array([1.00230524, 2.15608891, 4.63802765, 9.97700064]), cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True), l1_ratios=array([0.001, 0.33366667, 0.66633333, 0.999]), penalty='elasticnet', random_state=42, solver='saga')*, *ate_alpha=0.05*, *control_name=0*, *n_fold=5*, *random_state=None*)

Bases: *BaseRLearner*

A parent class for R-learner classifier classes.

fit(*X*, *treatment*, *y*, *p=None*, *sample_weight=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.
- **sample_weight** (*np.array or pd.Series, optional*) – an array of sample weights indicating the weight of each observation for *effect_learner*. If None, it assumes equal weight.
- **verbose** (*bool, optional*) – whether to output progress logs

predict(*X*, *p=None*)

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*, *propensity_learner=LogisticRegressionCV(Cs=array([1.00230524, 2.15608891, 4.63802765, 9.97700064]), cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True), l1_ratios=array([0.001, 0.33366667, 0.66633333, 0.999]), penalty='elasticnet', random_state=42, solver='saga')*, *ate_alpha=0.05*, *control_name=0*, *n_fold=5*, *random_state=None*, *cv_n_jobs=-1*)

Bases: BaseLearner

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).

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

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*) – only needed when pretrain=False, a treatment vector
- **y** (*np.array or pd.Series*) – only needed when pretrain=False, 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.
- **sample_weight** (*np.array or pd.Series, optional*) – an array of sample weights indicating the weight of each observation for *effect_learner*. If None, it assumes equal weight.
- **bootstrap_ci** (*bool*) – whether run bootstrap for confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap
- **pretrain** (*bool*) – whether a model has been fit, default False.

Returns

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

fit (*X*, *treatment*, *y*, *p=None*, *sample_weight=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.
- **sample_weight** (*np.array or pd.Series, optional*) – an array of sample weights indicating the weight of each observation for *effect_learner*. If None, it assumes equal weight.
- **verbose** (*bool, optional*) – whether to output progress logs

fit_predict (*X*, *treatment*, *y*, *p=None*, *sample_weight=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.

- **sample_weight** (*np.array or pd.Series, optional*) – an array of sample weights indicating the weight of each observation for *effect_learner*. If None, it assumes equal weight.
- **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)

predict(*X*, *p=None*)

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*, *propensity_learner=LogisticRegressionCV(Cs=array([1.00230524, 2.15608891, 4.63802765, 9.97700064]), cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True), l1_ratios=array([0.001, 0.33366667, 0.66633333, 0.999]), penalty='elasticnet', random_state=42, solver='saga')*, *ate_alpha=0.05*, *control_name=0*, *n_fold=5*, *random_state=None*)

Bases: *BaseRLearner*

A parent class for R-learner regressor classes.

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

Bases: *BaseSLearner*

A parent class for S-learner classifier classes.

predict(*X*, *treatment=None*, *y=None*, *p=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.**BaseSLearner**(*learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: BaseLearner

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).

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

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
- **pretrain** (*bool*) – whether a model has been fit, default False.

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

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

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*, *p=None*, *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)

predict(*X*, *treatment=None*, *y=None*, *p=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: *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: *BaseTLearner*

A parent class for T-learner classifier classes.

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
- **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: BaseLearner

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).

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

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. pretrain (bool): whether a model has been fit, default False.

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

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

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

- **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: *BaseTLearner*

A parent class for T-learner regressor classes.

class causalml.inference.meta.**BaseXClassifier**(*outcome_learner=None*, *effect_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: *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: BaseLearner

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).

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

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
- **pretrain** (*bool*) – whether a model has been fit, default False.

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)

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: *BaseXLearner*

A parent class for X-learner regressor classes.

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

Bases: *BaseSRegressor*

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

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: *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*, *treatment*, *y*, *p*, *segment=None*, *return_ci=False*)

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*) – 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)
- **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.**XGBDRRegressor**(*ate_alpha=0.05*, *control_name=0*, **args*, ***kwargs*)

Bases: *BaseDRRegressor*

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

Bases: *BaseRRegressor*

fit(*X*, *treatment*, *y*, *p=None*, *sample_weight=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.
- **sample_weight** (*np.array or pd.Series, optional*) – an array of sample weights indicating the weight of each observation for *effect_learner*. If None, it assumes equal weight.
- **verbose** (*bool, optional*) – whether to output progress logs

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

Bases: *BaseTRegressor*

9.4 causalml.inference.iv module

class causalml.inference.iv.**BaseDRIVLearner**(*learner=None*, *control_outcome_learner=None*, *treatment_outcome_learner=None*, *treatment_effect_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: object

A parent class for DRIV-learner regressor classes.

A DRIV-learner estimates endogenous treatment effects for compliers with machine learning models.

Details of DR-learner are available at Kennedy (2020). The DR moment condition for LATE comes from Chernozhukov et al (2018).

bootstrap(*X*, *assignment*, *treatment*, *y*, *p*, *pZ*, *size=10000*, *seed=None*)

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

estimate_ate(*X*, *assignment*, *treatment*, *y*, *p=None*, *pZ=None*, *bootstrap_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*, *seed=None*, *calibrate=True*)

Estimate the Average Treatment Effect (ATE) for compliers.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **assignment** (*np.array or pd.Series*) – an assignment vector. The assignment is the instrumental variable that does not depend on unknown confounders. The assignment status influences treatment in a monotonic way, i.e. one can only be more likely to take the treatment if assigned.
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*2-tuple of np.ndarray or pd.Series or dict, optional*) – The first (second) element corresponds to unassigned (assigned) units. Each is 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.
- **pZ** (*np.array or pd.Series, optional*) – an array of assignment probability of float (0,1); if None will run ElasticNetPropensityModel() to generate the assignment probability score.
- **bootstrap_ci** (*bool*) – whether run bootstrap for confidence intervals
- **n_bootstraps** (*int*) – number of bootstrap iterations
- **bootstrap_size** (*int*) – number of samples per bootstrap
- **seed** (*int*) – random seed for cross-fitting

Returns

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

fit (*X*, *assignment*, *treatment*, *y*, *p=None*, *pZ=None*, *seed=None*, *calibrate=True*)

Fit the inference model.

Parameters

- **X** (*np.matrix or np.array or pd.Dataframe*) – a feature matrix
- **assignment** (*np.array or pd.Series*) – a (0,1)-valued assignment vector. The assignment is the instrumental variable that does not depend on unknown confounders. The assignment status influences treatment in a monotonic way, i.e. one can only be more likely to take the treatment if assigned.
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*2-tuple of np.ndarray or pd.Series or dict, optional*) – The first (second) element corresponds to unassigned (assigned) units. Each is 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.
- **pZ** (*np.array or pd.Series, optional*) – an array of assignment probability of float (0,1); if None will run ElasticNetPropensityModel() to generate the assignment probability score.
- **seed** (*int*) – random seed for cross-fitting

fit_predict (*X*, *assignment*, *treatment*, *y*, *p=None*, *pZ=None*, *return_ci=False*, *n_bootstraps=1000*, *bootstrap_size=10000*, *return_components=False*, *verbose=True*, *seed=None*, *calibrate=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
- **assignment** (*np.array or pd.Series*) – a (0,1)-valued assignment vector. The assignment is the instrumental variable that does not depend on unknown confounders. The assignment status influences treatment in a monotonic way, i.e. one can only be more likely to take the treatment if assigned.
- **treatment** (*np.array or pd.Series*) – a treatment vector
- **y** (*np.array or pd.Series*) – an outcome vector
- **p** (*2-tuple of np.ndarray or pd.Series or dict, optional*) – The first (second) element corresponds to unassigned (assigned) units. Each is 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.
- **pZ** (*np.array or pd.Series, optional*) – an array of assignment probability of float (0,1); if None will run ElasticNetPropensityModel() to generate the assignment probability score.
- **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
- **seed** (*int*) – random seed for cross-fitting

Returns

Predictions of treatment effects for compliers, , i.e. those individuals
who take the treatment only if they are assigned. 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
- **verbose** (*bool, optional*) – whether to output progress logs

Returns

Predictions of treatment effects for compliers, i.e. those individuals who take the treatment only if they are assigned.

Return type
(numpy.ndarray)

class causalml.inference.iv.**BaseDRIVRegressor**(*learner=None*, *control_outcome_learner=None*, *treatment_outcome_learner=None*, *treatment_effect_learner=None*, *ate_alpha=0.05*, *control_name=0*)

Bases: *BaseDRIVLearner*

A parent class for DRIV-learner regressor classes.

class causalml.inference.iv.**IVRegressor**

Bases: object

A wrapper class that uses IV2SLS from statsmodel

A linear 2SLS model that estimates the average treatment effect with endogenous treatment variable.

fit(*X*, *treatment*, *y*, *w*)

Fits the 2SLS 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
- **w** (*np.array or pd.Series*) – an instrument vector

predict()

Returns the average treatment effect and its estimated standard error

Returns
average treatment effect (float): standard error of the estimation

Return type
(float)

class causalml.inference.iv.**XGBDRIVRegressor**(*ate_alpha=0.05*, *control_name=0*, **args*, ***kwargs*)

Bases: *BaseDRIVRegressor*

9.5 causalml.inference.nn module

class causalml.inference.nn.**CEVAE**(*outcome_dist='studentt'*, *latent_dim=20*, *hidden_dim=200*, *num_epochs=50*, *num_layers=3*, *batch_size=100*, *learning_rate=0.001*, *learning_rate_decay=0.1*, *num_samples=1000*, *weight_decay=0.0001*)

Bases: object

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

Fits CEVAE.

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*, *p=None*)

Fits the CEVAE model and then predicts.

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

Returns
Predictions of treatment effects.

Return type
(np.ndarray)

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

Calls predict on fitted DragonNet.

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

Returns
Predictions of treatment effects.

Return type
(np.ndarray)

9.6 causalml.inference.tf module

9.7 causalml.optimize module

class causalml.optimize.**CounterfactualUnitSelector**(*learner*, *nevertaker_payoff*, *alwaystaker_payoff*, *complier_payoff*, *defier_payoff*, *organic_conversion=None*)

Bases: object

A highly experimental implementation of the counterfactual unit selection model proposed by Li and Pearl (2019).

Parameters

- **learner** (*object*) – The base learner used to estimate the segment probabilities.
- **nevertaker_payoff** (*float*) – The payoff from targeting a never-taker
- **alwaystaker_payoff** (*float*) – The payoff from targeting an always-taker
- **complier_payoff** (*float*) – The payoff from targeting a complier
- **defier_payoff** (*float*) – The payoff from targeting a defier
- **organic_conversion** (*float, optional (default=None)*) – The organic conversion rate in the population without an intervention. If None, the organic conversion rate is obtained from tne control group.

  NB: The organic conversion in the control group is not always the same as the organic conversion rate without treatment.
- **data** (*DataFrame*) – A pandas DataFrame containing the features, treatment assignment indicator and the outcome of interest.
- **treatment** (*string*) – A string corresponding to the name of the treatment column. The assumed coding in the column is 1 for treatment and 0 for control.
- **outcome** (*string*) – A string corresponding to the name of the outcome column. The assumed coding in the column is 1 for conversion and 0 for no conversion.

References

Li, Ang, and Judea Pearl. 2019. "Unit Selection Based on Counterfactual Logic." https://ftp.cs.ucla.edu/pub/stat_ser/r488.pdf.

fit(*data*, *treatment*, *outcome*)

Fits the class.

predict(*data*, *treatment*, *outcome*)

Predicts an individual-level payoff. If gain equality is satisfied, uses the exact function; if not, uses the midpoint between bounds.

class causalml.optimize.**CounterfactualValueEstimator**(*treatment*, *control_name*, *treatment_names*, *y_proba*, *cate*, *value*, *conversion_cost*, *impression_cost*, **args*, ***kwargs*)

Bases: object

Parameters

- **treatment** (*array, shape = (num_samples,)*) – An array of treatment group indicator values.
- **control_name** (*string*) – The name of the control condition as a string. Must be contained in the treatment array.
- **treatment_names** (*list, length = cate.shape[1]*) – A list of treatment group names. NB: The order of the items in the list must correspond to the order in which the conditional average treatment effect estimates are in cate_array.

- **y_proba** (*array, shape = (num_samples,)*) – The predicted probability of conversion using the Y ~ X model across the total sample.
- **cate** (*array, shape = (num_samples, len(set(treatment)))*) – Conditional average treatment effect estimations from any model.
- **value** (*array, shape = (num_samples,)*) – Value of converting each unit.
- **conversion_cost** (*shape = (num_samples, len(set(treatment)))*) – The cost of a treatment that is triggered if a unit converts after having been in the treatment, such as a promotion code.
- **impression_cost** (*shape = (num_samples, len(set(treatment)))*) – The cost of a treatment that is the same for each unit whether or not they convert, such as a cost associated with a promotion channel.

Notes

Because we get the conditional average treatment effects from cate-learners relative to the control condition, we subtract the cate for the unit in their actual treatment group from y_proba for that unit, in order to recover the control outcome. We then add the cates to the control outcome to obtain y_proba under each condition. These outcomes are counterfactual because just one of them is actually observed.

predict_best()

Predict the best treatment group based on the highest counterfactual value for a treatment.

predict_counterfactuals()

Predict the counterfactual values for each treatment group.

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

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).

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

Fit the treatment assignment policy learner.

Parameters

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

predict_proba(*X*)

Predict treatment assignment score that optimizes the outcome.

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

Returns
predictions of treatment assignment score.

Return type
(numpy.ndarray)

causalml.optimize.**get_actual_value**(*treatment*, *observed_outcome*, *conversion_value*, *conditions*, *conversion_cost*, *impression_cost*)

Set the conversion and impression costs based on a dict of parameters.

Calculate the actual value of targeting a user with the actual treatment group using the above parameters.

9.7.1 Params

treatment
[array, shape = (num_samples,)] Treatment array.

observed_outcome
[array, shape = (num_samples,)] Observed outcome array, aka y.

conversion_value
[array, shape = (num_samples,)] The value of converting a given user.

conditions
[list, len = len(set(treatment))] List of treatment conditions.

conversion_cost
[array, shape = (num_samples, num_treatment)] Array of conversion costs for each unit in each treatment.

impression_cost
[array, shape = (num_samples, num_treatment)] Array of impression costs for each unit in each treatment.

returns

- **actual_value** (*array, shape = (num_samples,)*) – Array of actual values of havng a user in their actual treatment group.
- **conversion_value** (*array, shape = (num_samples,)*) – Array of payoffs from converting a user.

causalml.optimize.**get_pns_bounds**(*data_exp*, *data_obs*, *T*, *Y*, *type='PNS'*)

Parameters

- **data_exp** (*DataFrame*) – Data from an experiment.
- **data_obs** (*DataFrame*) – Data from an observational study
- **T** (*str*) – Name of the binary treatment indicator
- **y** (*str*) – Name of the binary outcome indicator
- **type** (*str*) –

  Type of probability of causation desired. Acceptable args are:

  - PNS: Probability of necessary and sufficient causation
  - PS: Probability of sufficient causation
  - PN: Probability of necessary causation

Notes

Based on Equation (24) in Tian and Pearl (2000).

To capture the counterfactual notation, we use 1 and 0 to indicate the actual and counterfactual values of a variable, respectively, and we use do to indicate the effect of an intervention.

The experimental and observational data are either assumed to come to the same population, or from random samples of the population. If the data are from a sample, the bounds may be incorrectly calculated because the relevant quantities in the Tian-Pearl equations are defined e.g. as $P(Y|do(T))$, not $P(Y|do(T), S)$ where S corresponds to sample selection. Bareinboim and Pearl (2016) discuss conditions under which $P(Y|do(T))$ can be recovered from $P(Y|do(T), S)$.

causalml.optimize.**get_treatment_costs**(*treatment*, *control_name*, *cc_dict*, *ic_dict*)

Set the conversion and impression costs based on a dict of parameters.

Calculate the actual cost of targeting a user with the actual treatment group using the above parameters.

9.7.2 Params

treatment
[array, shape = (num_samples,)] Treatment array.

control_name, str
Control group name as string.

cc_dict
[dict] Dict containing the conversion cost for each treatment.

ic_dict
Dict containing the impression cost for each treatment.

returns

- **conversion_cost** (*ndarray, shape = (num_samples, num_treatments)*) – An array of conversion costs for each treatment.
- **impression_cost** (*ndarray, shape = (num_samples, num_treatments)*) – An array of impression costs for each treatment.

- **conditions** (*list, len = len(set(treatment))*) – A list of experimental conditions.

causalml.optimize.**get_uplift_best**(*cate*, *conditions*)

Takes the CATE prediction from a learner, adds the control outcome array and finds the name of the argmax conditon.

9.7.3 Params

cate

[array, shape = (num_samples,)] The conditional average treatment effect prediction.

conditions : list, len = len(set(treatment))

returns

uplift_recomm_name – The experimental group recommended by the learner.

rtype

array, shape = (num_samples,)

9.8 causalml.dataset module

causalml.dataset.**bar_plot_summary**(*synthetic_summary*, *k*, *drop_learners=[]*, *drop_cols=[]*, *sort_cols=['MSE', 'Abs % Error of ATE']*)

Generates a bar plot comparing learner performance.

Parameters

- **synthetic_summary** (*pd.DataFrame*) – summary generated by get_synthetic_summary()
- **k** (*int*) – number of simulations (used only for plot title text)
- **drop_learners** (*list, optional*) – list of learners (str) to omit when plotting
- **drop_cols** (*list, optional*) – list of metrics (str) to omit when plotting
- **sort_cols** (*list, optional*) – list of metrics (str) to sort on when plotting

causalml.dataset.**bar_plot_summary_holdout**(*train_summary*, *validation_summary*, *k*, *drop_learners=[]*, *drop_cols=[]*)

Generates a bar plot comparing learner performance by training and validation

Parameters

- **train_summary** (*pd.DataFrame*) – summary for training synthetic data generated by get_synthetic_summary_holdout()
- **validation_summary** (*pd.DataFrame*) – summary for validation synthetic data generated by get_synthetic_summary_holdout()
- **k** (*int*) – number of simulations (used only for plot title text)
- **drop_learners** (*list, optional*) – list of learners (str) to omit when plotting
- **drop_cols** (*list, optional*) – list of metrics (str) to omit when plotting

causalml.dataset.**distr_plot_single_sim**(*synthetic_preds*, *kind='kde'*, *drop_learners=[]*, *bins=50*, *histtype='step'*, *alpha=1*, *linewidth=1*, *bw_method=1*)

Plots the distribution of each learner's predictions (for a single simulation). Kernel Density Estimation (kde) and actual histogram plots supported.

Parameters

- **synthetic_preds** (*dict*) – dictionary of predictions generated by get_synthetic_preds()
- **kind** (*str, optional*) – 'kde' or 'hist'
- **drop_learners** (*list, optional*) – list of learners (str) to omit when plotting
- **bins** (*int, optional*) – number of bins to plot if kind set to 'hist'
- **histtype** (*str, optional*) – histogram type if kind set to 'hist'
- **alpha** (*float, optional*) – alpha (transparency) for plotting
- **linewidth** (*int, optional*) – line width for plotting
- **bw_method** (*float, optional*) – parameter for kde

causalml.dataset.**get_synthetic_auuc**(*synthetic_preds*, *drop_learners=[]*, *outcome_col='y'*, *treatment_col='w'*, *treatment_effect_col='tau'*, *plot=True*)

Get auuc values for cumulative gains of model estimates in quantiles.

For details, reference get_cumgain() and plot_gain() :param synthetic_preds: dictionary of predictions generated by get_synthetic_preds() :type synthetic_preds: dict :param or get_synthetic_preds_holdout(): :param outcome_col: the column name for the actual outcome :type outcome_col: str, optional :param treatment_col: the column name for the treatment indicator (0 or 1) :type treatment_col: str, optional :param treatment_effect_col: the column name for the true treatment effect :type treatment_effect_col: str, optional :param plot: plot the cumulative gain chart or not :type plot: boolean,optional

Returns
auuc values by learner for cumulative gains of model estimates

Return type
(pandas.DataFrame)

causalml.dataset.**get_synthetic_preds**(*synthetic_data_func*, *n=1000*, *estimators={}*)

Generate predictions for synthetic data using specified function (single simulation)

Parameters

- **synthetic_data_func** (*function*) – synthetic data generation function
- **n** (*int, optional*) – number of samples
- **estimators** (*dict of object*) – dict of names and objects of treatment effect estimators

Returns
dict of the actual and estimates of treatment effects

Return type
(dict)

causalml.dataset.**get_synthetic_preds_holdout**(*synthetic_data_func*, *n=1000*, *valid_size=0.2*, *estimators={}*)

Generate predictions for synthetic data using specified function (single simulation) for train and holdout

Parameters

- **synthetic_data_func** (*function*) – synthetic data generation function
- **n** (*int, optional*) – number of samples
- **valid_size** (*float,optional*) – validaiton/hold out data size
- **estimators** (*dict of object*) – dict of names and objects of treatment effect estimators

Returns

synthetic training and validation data dictionaries:

- preds_dict_train (dict): synthetic training data dictionary
- preds_dict_valid (dict): synthetic validation data dictionary

Return type
(tuple)

causalml.dataset.**get_synthetic_summary**(*synthetic_data_func*, *n=1000*, *k=1*, *estimators={}*)

Generate a summary for predictions on synthetic data using specified function

Parameters

- **synthetic_data_func** (*function*) – synthetic data generation function
- **n** (*int, optional*) – number of samples per simulation
- **k** (*int, optional*) – number of simulations

causalml.dataset.**get_synthetic_summary_holdout**(*synthetic_data_func*, *n=1000*, *valid_size=0.2*, *k=1*)

Generate a summary for predictions on synthetic data for train and holdout using specified function

Parameters

- **synthetic_data_func** (*function*) – synthetic data generation function
- **n** (*int, optional*) – number of samples per simulation
- **valid_size** (*float,optional*) – validation/hold out data size
- **k** (*int, optional*) – number of simulations

Returns

summary evaluation metrics of predictions for train and validation:

- summary_train (pandas.DataFrame): training data evaluation summary
- summary_train (pandas.DataFrame): validation data evaluation summary

Return type
(tuple)

causalml.dataset.**make_uplift_classification**(*n_samples=1000*, *treatment_name=['control', 'treatment1', 'treatment2', 'treatment3']*, *y_name='conversion'*, *n_classification_features=10*, *n_classification_informative=5*, *n_classification_redundant=0*, *n_classification_repeated=0*, *n_uplift_increase_dict={'treatment1': 2, 'treatment2': 2, 'treatment3': 2}*, *n_uplift_decrease_dict={'treatment1': 0, 'treatment2': 0, 'treatment3': 0}*, *delta_uplift_increase_dict={'treatment1': 0.02, 'treatment2': 0.05, 'treatment3': 0.1}*, *delta_uplift_decrease_dict={'treatment1': 0.0, 'treatment2': 0.0, 'treatment3': 0.0}*, *n_uplift_increase_mix_informative_dict={'treatment1': 1, 'treatment2': 1, 'treatment3': 1}*, *n_uplift_decrease_mix_informative_dict={'treatment1': 0, 'treatment2': 0, 'treatment3': 0}*, *positive_class_proportion=0.5*, *random_seed=20190101*)

Generate a synthetic dataset for classification uplift modeling problem.

Parameters

- **n_samples** (*int, optional (default=1000)*) – The number of samples to be generated for each treatment group.
- **treatment_name** (*list, optional (default = ['control', 'treatment1','treatment2','treatment3'])*) – The list of treatment names.
- **y_name** (*string, optional (default = 'conversion')*) – The name of the outcome variable to be used as a column in the output dataframe.
- **n_classification_features** (*int, optional (default = 10)*) – Total number of features for base classification
- **n_classification_informative** (*int, optional (default = 5)*) – Total number of informative features for base classification
- **n_classification_redundant** (*int, optional (default = 0)*) – Total number of redundant features for base classification
- **n_classification_repeated** (*int, optional (default = 0)*) – Total number of repeated features for base classification
- **n_uplift_increase_dict** (*dictionary, optional (default: {'treatment1': 2, 'treatment2': 2, 'treatment3': 2})*) – Number of features for generating positive treatment effects for corresponding treatment group. Dictionary of {treatment_key: number_of_features_for_increase_uplift}.
- **n_uplift_decrease_dict** (*dictionary, optional (default: {'treatment1': 0, 'treatment2': 0, 'treatment3': 0})*) – Number of features for generating negative treatment effects for corresponding treatment group. Dictionary of {treatment_key: number_of_features_for_increase_uplift}.
- **delta_uplift_increase_dict** (*dictionary, optional (default: {'treatment1': .02, 'treatment2': .05, 'treatment3': .1})*) – Positive treatment effect created by the positive uplift features on the base classification label. Dictionary of {treatment_key: increase_delta}.

- **delta_uplift_decrease_dict** (*dictionary, optional (default: {'treatment1': 0., 'treatment2': 0., 'treatment3': 0.})*) – Negative treatment effect created by the negative uplift features on the base classification label. Dictionary of {treatment_key: increase_delta}.
- **n_uplift_increase_mix_informative_dict** (*dictionary, optional*) – Number of positive mix features for each treatment. The positive mix feature is defined as a linear combination of a randomly selected informative classification feature and a randomly selected positive uplift feature. The linear combination is made by two coefficients sampled from a uniform distribution between -1 and 1. default: {'treatment1': 1, 'treatment2': 1, 'treatment3': 1}
- **n_uplift_decrease_mix_informative_dict** (*dictionary, optional*) – Number of negative mix features for each treatment. The negative mix feature is defined as a linear combination of a randomly selected informative classification feature and a randomly selected negative uplift feature. The linear combination is made by two coefficients sampled from a uniform distribution between -1 and 1. default: {'treatment1': 0, 'treatment2': 0, 'treatment3': 0}
- **positive_class_proportion** (*float, optional (default = 0.5)*) – The proportion of positive label (1) in the control group.
- **random_seed** (*int, optional (default = 20190101)*) – The random seed to be used in the data generation process.

Returns

- **df_res** (*DataFrame*) – A data frame containing the treatment label, features, and outcome variable.
- **x_name** (*list*) – The list of feature names generated.

Notes

The algorithm for generating the base classification dataset is adapted from the make_classification method in the sklearn package, that uses the algorithm in Guyon [1] designed to generate the "Madelon" dataset.

References

causalml.dataset.**make_uplift_classification_logistic**(*n_samples=10000*, *treatment_name=['control', 'treatment1', 'treatment2', 'treatment3']*, *y_name='conversion'*, *n_classification_features=10*, *n_classification_informative=5*, *n_classification_redundant=0*, *n_classification_repeated=0*, *n_uplift_dict={'treatment1': 2, 'treatment2': 2, 'treatment3': 3}*, *n_mix_informative_uplift_dict={'treatment1': 1, 'treatment2': 1, 'treatment3': 0}*, *delta_uplift_dict={'treatment1': 0.02, 'treatment2': 0.05, 'treatment3': -0.05}*, *positive_class_proportion=0.1*, *random_seed=20200101*, *feature_association_list=['linear', 'quadratic', 'cubic', 'relu', 'sin', 'cos']*, *random_select_association=True*, *error_std=0.05*)

Generate a synthetic dataset for classification uplift modeling problem.

Parameters

- **n_samples** (*int, optional (default=1000)*) – The number of samples to be generated for each treatment group.
- **treatment_name** (*list, optional (default = ['control', 'treatment1','treatment2','treatment3'])*) – The list of treatment names. The first element must be 'control' as control group, and the rest are treated as treatment groups.
- **y_name** (*string, optional (default = 'conversion')*) – The name of the outcome variable to be used as a column in the output dataframe.
- **n_classification_features** (*int, optional (default = 10)*) – Total number of features for base classification
- **n_classification_informative** (*int, optional (default = 5)*) – Total number of informative features for base classification
- **n_classification_redundant** (*int, optional (default = 0)*) – Total number of redundant features for base classification
- **n_classification_repeated** (*int, optional (default = 0)*) – Total number of repeated features for base classification
- **n_uplift_dict** (*dictionary, optional (default: {'treatment1': 2, 'treatment2': 2, 'treatment3': 3})*) – Number of features for generating heterogeneous treatment effects for corresponding treatment group. Dictionary of {treatment_key: number_of_features_for_uplift}.
- **n_mix_informative_uplift_dict** (*dictionary, optional (default: {'treatment1': 1, 'treatment2': 1, 'treatment3': 1})*) – Number of mix features for each treatment. The mix feature is defined as a linear combination of a randomly selected informative classification feature and a randomly selected uplift feature. The mixture is made by a weighted sum (p*feature1 + (1-p)*feature2), where the weight p is drawn from a uniform distribution between 0 and 1.

- **delta_uplift_dict** (*dictionary, optional (default: {'treatment1': .02, 'treatment2': .05, 'treatment3': -.05})*) – Treatment effect (delta), can be positive or negative. Dictionary of {treatment_key: delta}.
- **positive_class_proportion** (*float, optional (default = 0.1)*) – The proportion of positive label (1) in the control group, or the mean of outcome variable for control group.
- **random_seed** (*int, optional (default = 20200101)*) – The random seed to be used in the data generation process.
- **feature_association_list** (*list, optional (default = ['linear', 'quadratic','cubic','relu','sin','cos'])*) – List of uplift feature association patterns to the treatment effect. For example, if the feature pattern is 'quadratic', then the treatment effect will increase or decrease quadratically with the feature. The values in the list must be one of ('linear','quadratic','cubic','relu','sin','cos'). However, the same value can appear multiple times in the list.
- **random_select_association** (*boolean, optional (default = True)*) – How the feature patterns are selected from the feature_association_list to be applied in the data generation process. If random_select_association = True, then for every uplift feature, a random feature association pattern is selected from the list. If random_select_association = False, then the feature association pattern is selected from the list in turns to be applied to each feature one by one.
- **error_std** (*float, optional (default = 0.05)*) – Standard deviation to be used in the error term of the logistic regression. The error is drawn from a normal distribution with mean 0 and standard deviation specified in this argument.

Returns

- **df1** (*DataFrame*) – A data frame containing the treatment label, features, and outcome variable.
- **x_name** (*list*) – The list of feature names generated.

causalml.dataset.**scatter_plot_single_sim**(*synthetic_preds*)

Creates a grid of scatter plots comparing each learner's predictions with the truth (for a single simulation).

Parameters

synthetic_preds (*dict*) – dictionary of predictions generated by get_synthetic_preds() or get_synthetic_preds_holdout()

causalml.dataset.**scatter_plot_summary**(*synthetic_summary*, *k*, *drop_learners=[]*, *drop_cols=[]*)

Generates a scatter plot comparing learner performance. Each learner's performance is plotted as a point in the (Abs % Error of ATE, MSE) space.

Parameters

- **synthetic_summary** (*pd.DataFrame*) – summary generated by get_synthetic_summary()
- **k** (*int*) – number of simulations (used only for plot title text)
- **drop_learners** (*list, optional*) – list of learners (str) to omit when plotting
- **drop_cols** (*list, optional*) – list of metrics (str) to omit when plotting

causalml.dataset.**scatter_plot_summary_holdout**(*train_summary*, *validation_summary*, *k*, *label=['Train', 'Validation']*, *drop_learners=[]*, *drop_cols=[]*)

Generates a scatter plot comparing learner performance by training and validation.

Parameters

- **train_summary** (*pd.DataFrame*) – summary for training synthetic data generated by get_synthetic_summary_holdout()
- **validation_summary** (*pd.DataFrame*) – summary for validation synthetic data generated by get_synthetic_summary_holdout()
- **label** (*string, optional*) – legend label for plot
- **k** (*int*) – number of simulations (used only for plot title text)
- **drop_learners** (*list, optional*) – list of learners (str) to omit when plotting
- **drop_cols** (*list, optional*) – list of metrics (str) to omit when plotting

causalml.dataset.**simulate_easy_propensity_difficult_baseline**(*n=1000*, *p=5*, *sigma=1.0*, *adj=0.0*)

Synthetic data with easy propensity and a difficult baseline

From Setup C in Nie X. and Wager S. (2018) 'Quasi-Oracle Estimation of Heterogeneous Treatment Effects'

Parameters

- **n** (*int, optional*) – number of observations
- **p** (*int optional*) – number of covariates (>=3)
- **sigma** (*float*) – standard deviation of the error term
- **adj** (*float*) – no effect. added for consistency

Returns

Synthetically generated samples with the following outputs:

- y ((n,)-array): outcome variable.
- X ((n,p)-ndarray): independent variables.
- w ((n,)-array): treatment flag with value 0 or 1.
- tau ((n,)-array): individual treatment effect.
- b ((n,)-array): expected outcome.
- e ((n,)-array): propensity of receiving treatment.

Return type

(tuple)

causalml.dataset.**simulate_hidden_confounder**(*n=10000*, *p=5*, *sigma=1.0*, *adj=0.0*)

Synthetic dataset with a hidden confounder biasing treatment.

From Louizos et al. (2018) "Causal Effect Inference with Deep Latent-Variable Models"

Parameters

- **n** (*int, optional*) – number of observations
- **p** (*int optional*) – number of covariates (>=3)
- **sigma** (*float*) – standard deviation of the error term
- **adj** (*float*) – no effect. added for consistency

Returns

Synthetically generated samples with the following outputs:

- y ((n,)-array): outcome variable.
- X ((n,p)-ndarray): independent variables.
- w ((n,)-array): treatment flag with value 0 or 1.
- tau ((n,)-array): individual treatment effect.
- b ((n,)-array): expected outcome.
- e ((n,)-array): propensity of receiving treatment.

Return type
(tuple)

causalml.dataset.**simulate_nuisance_and_easy_treatment**(*n=1000*, *p=5*, *sigma=1.0*, *adj=0.0*)

Synthetic data with a difficult nuisance components and an easy treatment effect
From Setup A in Nie X. and Wager S. (2018) 'Quasi-Oracle Estimation of Heterogeneous Treatment Effects'

Parameters

- **n** (*int, optional*) – number of observations
- **p** (*int optional*) – number of covariates (>=5)
- **sigma** (*float*) – standard deviation of the error term
- **adj** (*float*) – adjustment term for the distribution of propensity, e. Higher values shift the distribution to 0.

Returns

Synthetically generated samples with the following outputs:

- y ((n,)-array): outcome variable.
- X ((n,p)-ndarray): independent variables.
- w ((n,)-array): treatment flag with value 0 or 1.
- tau ((n,)-array): individual treatment effect.
- b ((n,)-array): expected outcome.
- e ((n,)-array): propensity of receiving treatment.

Return type
(tuple)

causalml.dataset.**simulate_randomized_trial**(*n=1000*, *p=5*, *sigma=1.0*, *adj=0.0*)

Synthetic data of a randomized trial
From Setup B in Nie X. and Wager S. (2018) 'Quasi-Oracle Estimation of Heterogeneous Treatment Effects'

Parameters

- **n** (*int, optional*) – number of observations
- **p** (*int optional*) – number of covariates (>=5)
- **sigma** (*float*) – standard deviation of the error term
- **adj** (*float*) – no effect. added for consistency

Returns

Synthetically generated samples with the following outputs:

- y ((n,)-array): outcome variable.
- X ((n,p)-ndarray): independent variables.
- w ((n,)-array): treatment flag with value 0 or 1.
- tau ((n,)-array): individual treatment effect.
- b ((n,)-array): expected outcome.
- e ((n,)-array): propensity of receiving treatment.

Return type
(tuple)

causalml.dataset.**simulate_unrelated_treatment_control**(*n=1000*, *p=5*, *sigma=1.0*, *adj=0.0*)

Synthetic data with unrelated treatment and control groups.
From Setup D in Nie X. and Wager S. (2018) 'Quasi-Oracle Estimation of Heterogeneous Treatment Effects'

Parameters

- **n** (*int, optional*) – number of observations
- **p** (*int optional*) – number of covariates (>=3)
- **sigma** (*float*) – standard deviation of the error term
- **adj** (*float*) – adjustment term for the distribution of propensity, e. Higher values shift the distribution to 0.

Returns

Synthetically generated samples with the following outputs:

- y ((n,)-array): outcome variable.
- X ((n,p)-ndarray): independent variables.
- w ((n,)-array): treatment flag with value 0 or 1.
- tau ((n,)-array): individual treatment effect.
- b ((n,)-array): expected outcome.
- e ((n,)-array): propensity of receiving treatment.

Return type
(tuple)

causalml.dataset.**synthetic_data**(*mode=1*, *n=1000*, *p=5*, *sigma=1.0*, *adj=0.0*)

Synthetic data in Nie X. and Wager S. (2018) 'Quasi-Oracle Estimation of Heterogeneous Treatment Effects' :param mode: mode of the simulation: 1 for difficult nuisance components and an easy treatment effect. 2 for a randomized trial. 3 for an easy propensity and a difficult baseline. 4 for unrelated treatment and control groups. 5 for a hidden confounder biasing treatment. :type mode: int, optional :param n: number of observations :type n: int, optional :param p: number of covariates (>=5) :type p: int optional :param sigma: standard deviation of the error term :type sigma: float :param adj: adjustment term for the distribution of propensity, e. Higher values shift the distribution to 0.

It does not apply to mode == 2 or 3.

Returns

Synthetically generated samples with the following outputs:

- y ((n,)-array): outcome variable.
- X ((n,p)-ndarray): independent variables.
- w ((n,)-array): treatment flag with value 0 or 1.
- tau ((n,)-array): individual treatment effect.
- b ((n,)-array): expected outcome.
- e ((n,)-array): propensity of receiving treatment.

Return type
(tuple)

9.9 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*, *n_jobs=-1*)

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

n_jobs

The number of parallel jobs to run for neighbors search. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors

Type
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*, *with_std=True*, *with_counts=True*)

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
- **with_std** (*bool*) – whether to output std together with mean values as in <mean> (<std>) format
- **with_counts** (*bool*) – whether to include a row counting the total number of samples

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)

9.10 causalml.propensity module

class causalml.propensity.**ElasticNetPropensityModel**(*clip_bounds=(0.001, 0.999)*, ***model_kwargs*)

Bases: *LogisticRegressionPropensityModel*

class causalml.propensity.**GradientBoostedPropensityModel**(*early_stop=False*, *clip_bounds=(0.001, 0.999)*, ***model_kwargs*)

Bases: *PropensityModel*

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

fit(*X*, *y*, *early_stopping_rounds=10*, *stop_val_size=0.2*)

Fit a propensity model.

Parameters

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

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.**LogisticRegressionPropensityModel**(*clip_bounds=(0.001, 0.999)*, ***model_kwargs*)

Bases: *PropensityModel*

Propensity regression model based on the LogisticRegression algorithm.

class causalml.propensity.**PropensityModel**(*clip_bounds=(0.001, 0.999)*, ***model_kwargs*)

Bases: object

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 (PropensityModel): a trained PropensityModel object

9.11 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)

static 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: *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: *Sensitivity*

Adds an irrelevant random covariate to the dataframe.

sensitivity_estimate()

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

Bases: *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: *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()

static 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

static 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: *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,), default=None*) – Sample weights.
- **multioutput** (*{'raw_values', 'uniform_average'} or array-like of shape (n_outputs,), default='uniform_average'*) – 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,), default=None*) – Sample weights.
- **multioutput** (*{'raw_values', 'uniform_average', 'variance_weighted'}, array-like of shape (n_outputs,) or None, default='uniform_average'*) – 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 case, it corresponds to an array of shape *(n_samples,)*. Both probability estimates and non-thresholded decision values can be provided. The probability estimates correspond to the **probability of the class with the greater label**, i.e. *estimator.classes_[1]* and thus *estimator.predict_proba(X, y)[:, 1]*. The decision values corresponds to the output of *estimator.decision_function(X, y)*. See more information in the User guide;
  - In the multiclass case, it corresponds to an array of shape *(n_samples, n_classes)* of probability estimates provided by the *predict_proba* method. The probability estimates **must** sum to 1 across the possible classes. In addition, 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`. See more information in the User guide;
  - In the multilabel case, it corresponds to an array of shape *(n_samples, n_classes)*. Probability estimates are provided by the *predict_proba* method and the non-thresholded decision values by the *decision_function* method. The probability estimates correspond to the **probability of the class with the greater label for each output** of the classifier. See more information in the User guide.
- **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'*) – Only used for multiclass targets. Determines the type of configuration to use. The default value raises an error, so either `'ovr'` or `'ovo'` must be passed explicitly.

  'ovr':
  Stands for One-vs-rest. 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':
  Stands for One-vs-one. 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*) – Only used for multiclass targets. 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.

RocCurveDisplay.from_estimator
Plot Receiver Operating Characteristic (ROC) curve given an estimator and some data.

RocCurveDisplay.from_predictions
Plot Receiver Operating Characteristic (ROC) curve given the true and predicted values.

[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.

Examples

Binary case:

```
>>> from sklearn.datasets import load_breast_cancer
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.metrics import roc_auc_score
>>> X, y = load_breast_cancer(return_X_y=True)
>>> clf = LogisticRegression(solver="liblinear", random_state=0).fit(X, y)
>>> roc_auc_score(y, clf.predict_proba(X)[:, 1])
0.99...
>>> roc_auc_score(y, clf.decision_function(X))
0.99...
```

Multiclass case:

```
>>> from sklearn.datasets import load_iris
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(solver="liblinear").fit(X, y)
>>> roc_auc_score(y, clf.predict_proba(X), multi_class='ovr')
0.99...
```

Multilabel case:

```
>>> import numpy as np
>>> from sklearn.datasets import make_multilabel_classification
>>> from sklearn.multioutput import MultiOutputClassifier
>>> X, y = make_multilabel_classification(random_state=0)
>>> clf = MultiOutputClassifier(clf).fit(X, y)
>>> # get a list of n_output containing probability arrays of shape
>>> # (n_samples, n_classes)
>>> y_pred = clf.predict_proba(X)
>>> # extract the positive columns for each output
>>> y_pred = np.transpose([pred[:, 1] for pred in y_pred])
>>> roc_auc_score(y, y_pred, average=None)
array([0.82..., 0.86..., 0.94..., 0.85... , 0.94...])
>>> from sklearn.linear_model import RidgeClassifierCV
>>> clf = RidgeClassifierCV().fit(X, y)
>>> roc_auc_score(y, clf.decision_function(X), average=None)
array([0.81..., 0.84... , 0.93..., 0.87..., 0.94...])
```

`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

Returns
sMAPE

Return type
e (numpy.float64)

9.12 causalml.feature_selection module

class causalml.feature_selection.**FilterSelect**

Bases: object

A class for feature importance methods.

filter_D(*data*, *features*, *y_name*, *n_bins=10*, *method='KL'*, *control_group='control'*, *experiment_group_column='treatment_group_key'*, *null_impute=None*)

Rank features based on the chosen divergence measure.

Parameters

- **data** (*pd.Dataframe*) – DataFrame containing outcome, features, and experiment group
- **treatment_indicator** (*string*) – the column name for binary indicator of treatment (1) or control (0)
- **features** (*list of string*) – list of feature names, that are columns in the data DataFrame
- **y_name** (*string*) – name of the outcome variable
- **method** (*string, optional, default = 'KL'*) – taking one of the following values {'F', 'LR', 'KL', 'ED', 'Chi'} The feature selection method to be used to rank the features. 'F' for F-test 'LR' for likelihood ratio test 'KL', 'ED', 'Chi' for bin-based uplift filter methods, KL divergence, Euclidean distance, Chi-Square respectively
- **experiment_group_column** (*string, optional, default = 'treatment_group_key'*) – the experiment column name in the DataFrame, which contains the treatment and control assignment label
- **control_group** (*string, optional, default = 'control'*) – name for control group, value in the experiment group column
- **n_bins** (*int, optional, default = 10*) – number of bins to be used for bin-based uplift filter methods
- **null_impute** (*str, optional, default=None*) – impute np.nan present in the data taking on of the followin strategy values {'mean', 'median', 'most_frequent', None}. If Value is None and null is present then exception will be raised

Returns

pd.DataFrame
a data frame containing the feature importance statistics

Return type
all_result

filter_F(*data*, *treatment_indicator*, *features*, *y_name*, *order=1*)

Rank features based on the F-statistics of the interaction.

Parameters

- **data** (*pd.Dataframe*) – DataFrame containing outcome, features, and experiment group
- **treatment_indicator** (*string*) – the column name for binary indicator of treatment (1) or control (0)

- **features** (*list of string*) – list of feature names, that are columns in the data DataFrame
- **y_name** (*string*) – name of the outcome variable
- **order** (*int*) – the order of feature to be evaluated with the treatment effect, order takes 3 values: 1,2,3. order = 1 corresponds to linear importance of the feature, order=2 corresponds to quadratic and linear importance of the feature,
- **forms.** (*order= 3 will calculate feature importance up to cubic*) –

Returns

pd.DataFrame
a data frame containing the feature importance statistics

Return type
all_result

filter_LR(*data*, *treatment_indicator*, *features*, *y_name*, *order=1*, *disp=True*)

Rank features based on the LRT-statistics of the interaction.

Parameters

- **data** (*pd.Dataframe*) – DataFrame containing outcome, features, and experiment group
- **treatment_indicator** (*string*) – the column name for binary indicator of treatment (1) or control (0)
- **feature_name** (*string*) – feature name, as one column in the data DataFrame
- **y_name** (*string*) – name of the outcome variable
- **order** (*int*) – the order of feature to be evaluated with the treatment effect, order takes 3 values: 1,2,3. order = 1 corresponds to linear importance of the feature, order=2 corresponds to quadratic and linear importance of the feature,
- **forms.** (*order= 3 will calculate feature importance up to cubic*) –

Returns

pd.DataFrame
a data frame containing the feature importance statistics

Return type
all_result

get_importance(*data*, *features*, *y_name*, *method*, *experiment_group_column='treatment_group_key'*, *control_group='control'*, *treatment_group='treatment'*, *n_bins=5*, *null_impute=None*, *order=1*, *disp=False*)

Rank features based on the chosen statistic of the interaction.

Parameters

- **data** (*pd.Dataframe*) – DataFrame containing outcome, features, and experiment group
- **features** (*list of string*) – list of feature names, that are columns in the data DataFrame
- **y_name** (*string*) – name of the outcome variable

- **method** (*string, optional, default = 'KL'*) – taking one of the following values {'F', 'LR', 'KL', 'ED', 'Chi'} The feature selection method to be used to rank the features. 'F' for F-test 'LR' for likelihood ratio test 'KL', 'ED', 'Chi' for bin-based uplift filter methods, KL divergence, Euclidean distance, Chi-Square respectively
- **experiment_group_column** (*string*) – the experiment column name in the DataFrame, which contains the treatment and control assignment label
- **control_group** (*string*) – name for control group, value in the experiment group column
- **treatment_group** (*string*) – name for treatment group, value in the experiment group column
- **n_bins** (*int, optional*) – number of bins to be used for bin-based uplift filter methods
- **null_impute** (*str, optional, default=None*) – impute np.nan present in the data taking on of the following strategy values {'mean', 'median', 'most_frequent', None}. If value is None and null is present then exception will be raised
- **order** (*int*) – the order of feature to be evaluated with the treatment effect for F filter and LR filter, order takes 3 values: 1,2,3. order = 1 corresponds to linear importance of the feature, order=2 corresponds to quadratic and linear importance of the feature,
- **forms.** (*order= 3 will calculate feature importance up to cubic*) –
- **disp** (*bool*) – Set to True to print convergence messages for Logistic regression convergence in LR method.

Returns

pd.DataFrame
a data frame with following columns: ['method', 'feature', 'rank', 'score', 'p_value', 'misc']

Return type
all_result

9.13 causalml.features module

class causalml.features.**LabelEncoder**(*min_obs=10*)

Bases: BaseEstimator

Label Encoder that groups infrequent values into one label.

Code from https://github.com/jeongyoonlee/Kaggler/blob/master/kaggler/preprocessing/data.py

min_obs

minimum number of observation to assign a label.

Type
int

label_encoders

label encoders for columns

Type
list of dict

label_maxes

maximum of labels for columns

Type
list of int

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

fit_transform(*X*, *y=None*)

Encode categorical columns into label encoded columns

Parameters
X (*pandas.DataFrame*) – categorical columns to encode

Returns
label encoded columns

Return type
X (pandas.DataFrame)

transform(*X*)

Encode categorical columns into label encoded columns

Parameters
X (*pandas.DataFrame*) – categorical columns to encode

Returns
label encoded columns

Return type
X (pandas.DataFrame)

class causalml.features.**OneHotEncoder**(*min_obs=10*)

Bases: BaseEstimator

One-Hot-Encoder that groups infrequent values into one dummy variable.

Code from https://github.com/jeongyoonlee/Kaggler/blob/master/kaggler/preprocessing/data.py

min_obs

minimum number of observation to create a dummy variable

Type
int

label_encoders

label encoders and their maximums for columns

Type
list of (dict, int)

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

fit_transform(*X*, *y=None*)

Encode categorical columns into sparse matrix with one-hot-encoding.

Parameters
X (*pandas.DataFrame*) – categorical columns to encode

Returns
sparse matrix encoding categorical variables into dummy variables

transform(*X*)

Encode categorical columns into sparse matrix with one-hot-encoding.

Parameters
X (*pandas.DataFrame*) – categorical columns to encode

Returns

sparse matrix encoding categorical
variables into dummy variables

Return type
X_new (scipy.sparse.coo_matrix)

causalml.features.**load_data**(*data*, *features*, *transformations={}*)

Load data and set the feature matrix and label vector.

Parameters

- **data** (*pandas.DataFrame*) – total input data
- **features** (*list of str*) – column names to be used in the inference model
- **transformation** (*dict of (str, func)*) – transformations to be applied to features

Returns
a feature matrix

Return type
X (numpy.matrix)

9.14 Module contents

CHAPTER

TEN

REFERENCES

10.1 Open Source Software Projects

10.1.1 Python Packages

- DoWhy: a package for causal inference based on causal graphs.
- CausalLift: a package for uplift modeling based on T-learner [16].
- PyLift: a package for uplift modeling based on the transformed outcome method in [4].
- EconML: a package for treatment effect estimation with orthogonal random forest [20], DeepIV [12] and other ML methods.

10.1.2 R Packages

- uplift: a package for treatment effect estimation with ML.
- grf: a package for forest-based honest estimation from [5].

10.2 Papers

CHAPTER

ELEVEN

CHANGELOG

11.1 0.15.1 (Apr 2024)

- This release fixes the build failure on macOS and a few bugs in `UpliftTreeClassifier`.
- We have two new contributors, @lee-junseok and @IanDelbridge. Thanks for your contributions!

11.1.1 Updates

- Relax `pandas` version requirement by @jeongyoonlee in https://github.com/uber/causalml/pull/743
- Remove undefined variables in `match.__main__()` by @jeongyoonlee in https://github.com/uber/causalml/pull/749
- Fix `distr_plot_single_sim()` by @jeongyoonlee in https://github.com/uber/causalml/pull/750
- Add `with_std`, `with_counts` to `create_table_one` by @lee-junseok in https://github.com/uber/causalml/pull/748
- fix stratified sampling call by @IanDelbridge in https://github.com/uber/causalml/pull/756
- 20240207 honest leaf size by @IanDelbridge in https://github.com/uber/causalml/pull/753
- 757: add `return_ci=True` in sensitivity by @lee-junseok in https://github.com/uber/causalml/pull/758
- Update sensitivity tests with more meta-learners by @jeongyoonlee in https://github.com/uber/causalml/pull/759
- manually specify `multiprocessing` use `fork` in `setup.py` by @IanDelbridge in https://github.com/uber/causalml/pull/754

11.1.2 New contributors

- @lee-junseok made their first contribution in https://github.com/uber/causalml/pull/748
- @IanDelbridge made their first contribution in https://github.com/uber/causalml/pull/756

11.2 0.15.0 (Feb 2024)

- In this release, we revamped documentation, cleaned up dependencies, and improved installation - in addition to the long list of bug fixes.
- We have three new contributors, @peterloleungyau, @SuperBo, and @ZiJiaW, who submitted their first PRs to CausalML. @erikcs also contributed to @ras44's PR #729 to add the wrapper for his MAQ implementation to CausalML. Thanks for your contributions!

11.2.1 Updates

- Update python-publish.yml by @jeongyoonlee in https://github.com/uber/causalml/pull/673
- Add build.[os, tools.python] to .readthedocs.yml by @jeongyoonlee in https://github.com/uber/causalml/pull/676
- Update notebook example with causal trees interpretation by @alexander-pv in https://github.com/uber/causalml/pull/683
- Remove the numpy and pandas version restriction in pyproject.toml by @jeongyoonlee in https://github.com/uber/causalml/pull/681
- Add governance documents by @jeongyoonlee in https://github.com/uber/causalml/pull/688
- Update GOVERNANCE.md by @ras44 in https://github.com/uber/causalml/pull/691
- Dev/governance docs to snake-case by @ras44 in https://github.com/uber/causalml/pull/693
- Reduce sklearn dependency in causalml by @alexander-pv in https://github.com/uber/causalml/pull/686
- Update MAINTAINERS.md by @jeongyoonlee in https://github.com/uber/causalml/pull/696
- Modified to speed up UpliftTreeClassifier.growDecisionTreeFrom. by @peterloleungyau in https://github.com/uber/causalml/pull/695
- Update README.md by @ras44 in https://github.com/uber/causalml/pull/698
- Add notebook examples to docs by @jeongyoonlee in https://github.com/uber/causalml/pull/697
- resolves change requests in #166 by @ras44 in https://github.com/uber/causalml/pull/701
- Fix the readthedocs build error by @jeongyoonlee in https://github.com/uber/causalml/pull/702
- Replace Stack and PriorityHeap with cpp stack/heap methods in trees by @SuperBo in https://github.com/uber/causalml/pull/700
- Hotfix for #701 by @jeongyoonlee in https://github.com/uber/causalml/pull/705
- Dev/699 win build fix by @ras44 in https://github.com/uber/causalml/pull/710
- expose n_jobs for rlearner by @ZiJiaW in https://github.com/uber/causalml/pull/714
- minimal fix to resolve #707 by @ras44 in https://github.com/uber/causalml/pull/720
- Add Python 3.10, 3.11, 3.12 to the testing by @cclauss in https://github.com/uber/causalml/pull/454
- Remove Python 3.12 from the build tests in python-test.yaml by @jeongyoonlee in https://github.com/uber/causalml/pull/726
- fix plot_std_diffs, add bal_tol, condense to one plot by @ras44 in https://github.com/uber/causalml/pull/723
- Dev/677 documentation by @ras44 in https://github.com/uber/causalml/pull/725
- documentation updates by @ras44 in https://github.com/uber/causalml/pull/728
- resolves #730, docs clean conda install by @ras44 in https://github.com/uber/causalml/pull/731

- minimal wrapper of MAQ #662 by @ras44 in https://github.com/uber/causalml/pull/729
- Temporary fix for causal trees missing values support #733 by @alexander-pv in https://github.com/uber/causalml/pull/734
- resolves #639, credit due to Dong Liu by @ras44 in https://github.com/uber/causalml/pull/722

11.2.2 New contributors

- @peterloleungyau made their first contribution in https://github.com/uber/causalml/pull/695
- @SuperBo made their first contribution in https://github.com/uber/causalml/pull/700
- @ZiJiaW made their first contribution in https://github.com/uber/causalml/pull/714

11.3 0.14.1 (Aug 2023)

- This release mainly addressed installation issues and updated documentation accordingly.
- We have 4 new contributors. @bsaunders27, @xhulianoThe1, @zpppy, and @bsaunders23. Thanks for your contributions!

11.3.1 Updates

- Update the python-publish workflow file to fix the package publish Gi… by @jeongyoonlee in https://github.com/uber/causalml/pull/633
- Update Cython dependency by @alexander-pv in https://github.com/uber/causalml/pull/640
- Fix for builds on Mac M1 infrastructure by @bsaunders27 in https://github.com/uber/causalml/pull/641
- code cleanups by @xhulianoThe1 in https://github.com/uber/causalml/pull/634
- support valid error early stopping by @zpppy in https://github.com/uber/causalml/pull/614
- fix: update to `envs/` conda build for precompiled M1 installs by @bsaunders27 in https://github.com/uber/causalml/pull/646
- Installation updates to README and .github/workflows by @ras44 in https://github.com/uber/causalml/pull/637
- fix: simulate_randomized_trial by @bsaunders23 in https://github.com/uber/causalml/pull/656
- issue 252 by @vincewu51 in https://github.com/uber/causalml/pull/660
- ras44/651 graph viz, resolves #651 by @ras44 in https://github.com/uber/causalml/pull/661
- linted with black by @ras44 in https://github.com/uber/causalml/pull/663
- Fix issue 650 by @vincewu51 in https://github.com/uber/causalml/pull/659
- Install graphviz in the workflow builds by @jeongyoonlee in https://github.com/uber/causalml/pull/668
- Update docs/installation.rst by @jeongyoonlee in https://github.com/uber/causalml/pull/667
- Schedule monthly PyPI install tests by @jeongyoonlee in https://github.com/uber/causalml/pull/670

11.3.2 New contributors

- @bsaunders27 made their first contribution in https://github.com/uber/causalml/pull/641
- @xhulianoThe1 made their first contribution in https://github.com/uber/causalml/pull/634
- @zpppy made their first contribution in https://github.com/uber/causalml/pull/614
- @bsaunders23 made their first contribution in https://github.com/uber/causalml/pull/656

11.4 0.14.0 (July 2023)

- CausalML surpassed 2MM downloads on PyPI and 4,100 stars on GitHub. Thanks for choosing CausalML and supporting us on GitHub.
- We have 7 new contributors: @darthtrevino, @ras44, @AbhishekVermaDH, @joel-mcmurry, @AlxClt, @kklein, and @volico. Thanks for your contributions!

11.4.1 Updates

- Fix the readthedocs build failure by @jeongyoonlee in https://github.com/uber/causalml/pull/545
- Add `pyproject.toml` with basic build dependencies for PEP518 compliance by @darthtrevino in https://github.com/uber/causalml/pull/553
- bump `numpy` from 1.20.3 to 1.23.2 in `environment-py38.yml` #338 by @ras44 in https://github.com/uber/causalml/pull/550
- CausalTree split criterions fix and fit optimization by @alexander-pv in https://github.com/uber/causalml/pull/557
- fixing math notations for proper rendering by @AbhishekVermaDH in https://github.com/uber/causalml/pull/558
- Update `methodology.rst` by @joel-mcmurry in https://github.com/uber/causalml/pull/568
- Causal trees bootstrapping and `max_leaf_nodes` fixes with minor update by @alexander-pv in https://github.com/uber/causalml/pull/583
- Fix #596 by @AlxClt in https://github.com/uber/causalml/pull/597
- Add `**kwargs` to `Explainer.plot_shap_values()` by @jeongyoonlee in https://github.com/uber/causalml/pull/603
- Make the Adam optimization optional and learning rate/epochs configurable in DragonNet by @jeongyoonlee in https://github.com/uber/causalml/pull/604
- Fix bug in variance calculation in drivlearner. by @huigangchen in https://github.com/uber/causalml/pull/606
- Bug Fix in Dragonnet: Adam parameter name lr depreciation by @huigangchen in https://github.com/uber/causalml/pull/617
- Fix AttributeError in builds with `numpy>=1.24` and `pandas>=2.0` by @jeongyoonlee in https://github.com/uber/causalml/pull/631
- Pass on `**kwargs` in `plot_shap_values` of base meta leaner by @kklein in https://github.com/uber/causalml/pull/627
- Bump `scipy` from 1.4.1 to 1.10.0 by @dependabot in https://github.com/uber/causalml/pull/629
- Feature/ttest criterion by @volico in https://github.com/uber/causalml/pull/570

- Added Interaction Tree (IT), Causal Inference Tree (CIT), and Invariant DDP (IDDP) by @jroessler in https://github.com/uber/causalml/pull/562
- Causal trees option to return counterfactual outcomes by @alexander-pv in https://github.com/uber/causalml/pull/623

11.4.2 New contributors

- @darthtrevino made their first contribution in https://github.com/uber/causalml/pull/553
- @ras44 made their first contribution in https://github.com/uber/causalml/pull/550
- @AbhishekVermaDH made their first contribution in https://github.com/uber/causalml/pull/558
- @joel-mcmurry made their first contribution in https://github.com/uber/causalml/pull/568
- @AlxClt made their first contribution in https://github.com/uber/causalml/pull/597
- @kklein made their first contribution in https://github.com/uber/causalml/pull/627
- @volico made their first contribution in https://github.com/uber/causalml/pull/570

11.5 0.13.0 (Sep 2022)

- CausalML surpassed 1MM downloads on PyPI and 3,200 stars on GitHub. Thanks for choosing CausalML and supporting us on GitHub.
- We have 7 new contributors @saiwing-yeung, @lixuan12315, @aldenrogers, @vincewu51, @AlkanSte, @enzoliao, and @alexander-pv. Thanks for your contributions!
- @alexander-pv revamped *CausalTreeRegressor* and added *CausalRandomForestRegressor* with more seamless integration with *scikit-learn*'s Cython tree module. He also added integration with *shap* for causal tree/ random forest interpretation. Please check out the example notebook.
- We dropped the support for Python 3.6 and removed its test workflow.

11.5.1 Updates

- Fix typo `(% -> $)` by @saiwing-yeung in https://github.com/uber/causalml/pull/488
- Add function for calculating PNS bounds by @t-tte in https://github.com/uber/causalml/pull/482
- Fix hard coding bug by @t-tte in https://github.com/uber/causalml/pull/492
- Update README of `conda` install and instruction of maintain in `conda-forge` by @ppstacy in https://github.com/uber/causalml/pull/485
- Update `examples.rst` by @lixuan12315 in https://github.com/uber/causalml/pull/496
- Fix incorrect `effect_learner_objective` in `XGBRRegressor` by @jeongyoonlee in https://github.com/uber/causalml/pull/504
- Fix Filter F doesn't work with latest `statsmodels`' F test f-value format by @paullo0106 in https://github.com/uber/causalml/pull/505
- Exclude tests in `setup.py` by @aldenrogers in https://github.com/uber/causalml/pull/508
- Enabling higher orders feature importance for F filter and LR filter by @zhenyuz0500 in https://github.com/uber/causalml/pull/509

- Ate pretrain 0506 by @vincewu51 in https://github.com/uber/causalml/pull/511
- Update `methodology.rst` by @AlkanSte in https://github.com/uber/causalml/pull/518
- Fix the bug of incorrect result in qini for multiple models by @enzoliao in https://github.com/uber/causalml/pull/520
- Test `get_qini()` by @enzoliao in https://github.com/uber/causalml/pull/523
- Fixed typo in `uplift_trees_with_synthetic_data.ipynb` by @jroessler in https://github.com/uber/causalml/pull/531
- Remove Python 3.6 test from workflows by @jeongyoonlee in https://github.com/uber/causalml/pull/535
- Causal trees update by @alexander-pv in https://github.com/uber/causalml/pull/522
- Causal trees interpretation example by @alexander-pv in https://github.com/uber/causalml/pull/536

11.6 0.12.3 (Feb 2022)

This patch is to release a version without the constraint for Shap to be abled to use for Conda.

11.6.1 Updates

- #483 by @ppstacy: Modify the requirement version of Shap

11.7 0.12.2 (Feb 2022)

This patch includes three updates by @tonkolviktor and @heiderich as follows. We also start using black, a Python formatter. Please check out the updated contribution guideline to learn how to use it.

11.7.1 Updates

- #473 by @tonkolviktor: Open up the scipy dependency version
- #476 by @heiderich: Use preferred backend for joblib instead of hard-coding it
- #477 by @heiderich: Allow parallel prediction for UpliftRandomForestClassifier and make the joblib's preferred backend configurable

11.8 0.12.1 (Feb 2022)

This patch includes two bug fixes for UpliftRandomForestClassifier as follows:

11.8.1 Updates

- #462 by @paullo0106: Use the correct treatment_idx for fillTree() when applying validation data set
- #468 by @jeongyoonlee: Switch the joblib backend for UpliftRandomForestClassifier to threading to avoid memory copy across trees

11.9 0.12.0 (Jan 2022)

- CausalML surpassed 637K downloads on PyPI and 2,500 stars on Github!
- We have 4 new community contributors, Luis (@lgmoneda), Ravi (@raviksharma), Louis (@LouisHernandez17) and JackRab (@JackRab). Thanks for the contribution!
- We refactored and speeded up UpliftTreeClassifier/UpliftRandomForestClassifier by 5x with Cython (#422 #440 by @jeongyoonlee)
- We revamped our API documentation, it now includes the latest methodology, references, installation, notebook examples, and graphs! (#413 by @huigangchen @t-tte @zhenyuz0500 @jeongyoonlee @paullo0106)
- Our team gave talks at 2021 Conference on Digital Experimentation @ MIT (CODE@MIT), Causal Data Science Meeting 2021, and KDD 2021 Tutorials on CausalML introduction and applications. Please take a look if you missed them! Full list of publications and talks can be found here.

11.9.1 Updates

- Update documentation on Instrument Variable methods @huigangchen (#447)
- Add benchmark simulation studies example notebook by @t-tte (#443)
- Add sample_weight support for R-learner by @paullo0106 (#425)
- Fix incorrect binning of numeric features in UpliftTreeClassifier by @jeongyoonlee (#420)
- Update papers, talks, and publication info to README and refs.bib by @zhenyuz0500 (#410 #414 #433)
- Add instruction for contributing.md doc by @jeongyoonlee (#408)
- Fix incorrect feature importance calculation logic by @paullo0106 (#406)
- Add parallel jobs support for NearestNeighbors search with n_jobs parameter by @paullo0106 (#389)
- Fix bug in simulate_randomized_trial by @jroessler (#385)
- Add GA pytest workflow by @ppstacy (#380)

11.10 0.11.0 (2021-07-28)

- CausalML surpassed 2K stars!
- We have 3 new community contributors, Jannik (@jroessler), Mohamed (@ibraaaa), and Leo (@lleiou). Thanks for the contribution!

11.10.1 Major Updates

- Make tensorflow dependency optional and add python 3.9 support by @jeongyoonlee (#343)
- Add delta-delta-p (ddp) tree inference approach by @jroessler (#327)
- Add conda env files for Python 3.6, 3.7, and 3.8 by @jeongyoonlee (#324)

11.10.2 Minor Updates

- Fix inconsistent feature importance calculation in uplift tree by @paullo0106 (#372)
- Fix filter method failure with NaNs in the data issue by @manojbalaji1 (#367)
- Add automatic package publish by @jeongyoonlee (#354)
- Fix typo in unit_selection optimization by @jeongyoonlee (#347)
- Fix docs build failure by @jeongyoonlee (#335)
- Convert pandas inputs to numpy in S/T/R Learners by @jeongyoonlee (#333)
- Require scikit-learn as a dependency of setup.py by @ibraaaa (#325)
- Fix AttributeError when passing in Outcome and Effect learner to R-Learner by @paullo0106 (#320)
- Fix error when there is no positive class for KL Divergence filter by @lleiou (#311)
- Add versions to cython and numpy in setup.py for requirements.txt accordingly by @maccam912 (#306)

11.11 0.10.0 (2021-02-18)

- CausalML surpassed 235,000 downloads!
- We have 5 new community contributors, Suraj (@surajiyer), Harsh (@HarshCasper), Manoj (@manojbalaji1), Matthew (@maccam912) and Václav (@vaclavbelak). Thanks for the contribution!

11.11.1 Major Updates

- Add Policy learner, DR learner, DRIV learner by @huigangchen (#292)
- Add wrapper for CEVAE, a deep latent-variable and variational autoencoder based model by @ppstacy(#276)

11.11.2 Minor Updates

- Add propensity_learner to R-learner by @jeongyoonlee (#297)
- Add BaseLearner class for other meta-learners to inherit from without duplicated code by @jeongyoonlee (#295)
- Fix installation issue for Shap>=0.38.1 by @paullo0106 (#287)
- Fix import error for sklearn>= 0.24 by @jeongyoonlee (#283)
- Fix KeyError issue in Filter method for certain dataset by @surajiyer (#281)
- Fix inconsistent cumlift score calculation of multiple models by @vaclavbelak (#273)
- Fix duplicate values handling in feature selection method by @manojbalaji1 (#271)

- Fix the color spectrum of SHAP summary plot for feature interpretations of meta-learners by @paullo0106 (#269)
- Add IIA and value optimization related documentation by @t-tte (#264)
- Fix StratifiedKFold arguments for propensity score estimation by @paullo0106 (#262)
- Refactor the code with string format argument and is to compare object types, and change methods not using bound instance to static methods by @harshcasper (#256, #260)

11.12 0.9.0 (2020-10-23)

- CausalML won the 1st prize at the poster session in UberML'20
- DoWhy integrated CausalML starting v0.4 (release note)
- CausalML team welcomes new project leadership, Mert Bay
- We have 4 new community contributors, Mario Wijaya (@mwijaya3), Harry Zhao (@deeplaunch), Christophe (@ccrndn) and Georg Walther (@waltherg). Thanks for the contribution!

11.12.1 Major Updates

- Add feature importance and its visualization to UpliftDecisionTrees and UpliftRF by @yungmsh (#220)
- Add feature selection example with Filter methods by @paullo0106 (#223)

11.12.2 Minor Updates

- Implement propensity model abstraction for common interface by @waltherg (#223)
- Fix bug in BaseSClassifier and BaseXClassifier by @yungmsh and @ppstacy (#217), (#218)
- Fix parentNodeSummary for UpliftDecisionTrees by @paullo0106 (#238)
- Add pd.Series for propensity score condition check by @paullo0106 (#242)
- Fix the uplift random forest prediction output by @ppstacy (#236)
- Add functions and methods to init for optimization module by @mwijaya3 (#228)
- Install GitHub Stale App to close inactive issues automatically @jeongyoonlee (#237)
- Update documentation by @deeplaunch, @ccrndn, @ppstacy(#214, #231, #232)

11.13 0.8.0 (2020-07-17)

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

11.13.1 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 (#199, #212)
- Add the *iv* estimator submodule and add 2SLS model to it by @huigangchen (#201)

11.13.2 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)

11.14 0.7.1 (2020-05-07)

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

11.14.1 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)

11.14.2 Minor Updates

- Fix typos and update documents by @paullo0106, @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)

11.15 0.7.0 (2020-02-28)

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

11.15.1 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

11.15.2 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

11.16 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

11.17 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 @paullo0106
- 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

11.18 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 *PermutationImportance* by @yungmsh
- Add bootstrap confidence intervals for the average treatment effect estimates of meta learners by @ppstacy

11.19 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

11.20 0.2.0 (2019-08-12)

- Add *optimize.PolicyLearner* based on Athey and Wager 2017 [6]
- Add the *CausalTreeRegressor* estimator based on Athey and Imbens 2016 [4] (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()*

11.21 0.1.0 (unreleased)

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

CHAPTER

TWELVE

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] Joshua D Angrist and Jörn-Steffen Pischke. *Mostly harmless econometrics: An empiricist's companion*. Princeton university press, 2008.

[3] Joshua D. Angrist and Alan B. Krueger. Instrumental variables and the search for identification: from supply and demand to natural experiments. *Journal of Economic Perspectives*, 15(4):69–85, December 2001. URL: https://www.aeaweb.org/articles?id=10.1257/jep.15.4.69, doi:10.1257/jep.15.4.69.

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

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

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

[7] Peter C. Austin and Elizabeth A. Stuart. Moving towards best practice when using inverse probability of treatment weighting (iptw) using the propensity score to estimate causal treatment effects in observational studies. *Statistics in Medicine*, 34(28):3661–3679, 2015. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.6607, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.6607, doi:https://doi.org/10.1002/sim.6607.

[8] Alexander Abraham Balke. *Probabilistic counterfactuals: semantics, computation, and applications*. University of California, Los Angeles, 1995.

[9] Hansotia Behram and Rukstales Brad. Incremental value modeling. *Journal of Interactive Marketing*, 16:35–46, 2002.

[10] Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 01 2018. URL: https://doi.org/10.1111/ectj.12097, arXiv:https://academic.oup.com/ectj/article-pdf/21/1/C1/27684918/ectj00c1.pdf, doi:10.1111/ectj.12097.

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

[12] 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.

[13] Keisuke Hirano, Guido W. Imbens, and Geert Ridder. Efficient estimation of average treatment effects using the estimated propensity score. *Econometrica*, 71(4):1161–1189, 2003. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0262.00442, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1468-0262.00442, doi:https://doi.org/10.1111/1468-0262.00442.

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

[15] Edward H. Kennedy. Optimal doubly robust estimation of heterogeneous causal effects. 2020. arXiv:2004.14497.

[16] 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.

[17] Mark Laan and Sherri Rose. *Targeted Learning: Causal Inference for Observational and Experimental Data*. Springer-Verlag New York, 01 2011. ISBN 978-1-4419-9781-4. doi:10.1007/978-1-4419-9782-1.

[18] Ang Li and Judea Pearl. Unit selection based on counterfactual logic. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 1793–1799. International Joint Conferences on Artificial Intelligence Organization, 7 2019. URL: https://doi.org/10.24963/ijcai.2019/248, doi:10.24963/ijcai.2019/248.

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

[20] 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.

[21] Judea Pearl. *Causality*. Cambridge university press, 2009.

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

[23] Jannik Rößler, Richard Guse, and Detlef Schoder. The best of two worlds: using recent advances from uplift modeling and heterogeneous treatment effects to optimize targeting policies. *International Conference on Information Systems*, 2022.

[24] Elizabeth A Stuart. Matching methods for causal inference: a review and a look forward. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 25(1):1, 2010.

[25] Xiaogang Su, Joseph Kang, Juanjuan Fan, Richard A Levine, and Xin Yan. Facilitating score and causal inference trees for large observational studies. *Journal of Machine Learning Research*, 13:2955, 2012.

[26] Xiaogang Su, Chih-Ling Tsai, Hansheng Wang, David M Nickerson, and Bogong Li. Subgroup analysis via recursive partitioning. *Journal of Machine Learning Research*, 2009.

[27] Jin Tian and Judea Pearl. Probabilities of causation: bounds and identification. *Annals of Mathematics and Artificial Intelligence*, 28(1):287–313, 2000.

[28] 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.

[29] Zhenyu Zhao and Totte Harinen. Uplift modeling for multiple treatments with cost optimization. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 422–431. IEEE, 2019.

[30] 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

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

X