causallib

Release 0.9.6

Causal Machine Learning for Healthcare and Life Sciences, IBM R

Oct 25, 2023

CONTENTS

CHAPTER

ONE

CAUSAL INFERENCE 360

A Python package for inferring causal effects from observational data.

1.1 Description

Causal inference analysis enables estimating the causal effect of an intervention on some outcome from real-world non-experimental observational data.

This package provides a suite of causal methods, under a unified scikit-learn-inspired API. It implements meta-algorithms that allow plugging in arbitrarily complex machine learning models. This modular approach supports highly-flexible causal modelling. The fit-and-predict-like API makes it possible to train on one set of examples and estimate an effect on the other (out-of-bag), which allows for a more "honest" effect estimation.

The package also includes an evaluation suite. Since most causal-models utilize machine learning models internally, we can diagnose poor-performing models by re-interpreting known ML evaluations from a causal perspective.

If you use the package, please consider citing Shimoni et al., 2019:

Borrowing Wager & Athey terminology of avoiding overfit.

1.2 Installation

```
pip install causallib
```

1.3 Usage

The package is imported using the name `causallib`. Each causal model requires an internal machine-learning model. `causallib` supports any model that has a sklearn-like fit-predict API (note some models might require a `predict_proba` implementation). For example:

```
from sklearn.linear_model import LogisticRegression
from causallib.estimation import IPW
from causallib.datasets import load_nhefs

data = load_nhefs()
ipw = IPW(LogisticRegression())
ipw.fit(data.X, data.a)
```

(continues on next page)

(continued from previous page)

```
potential_outcomes = ipw.estimate_population_outcome(data.X, data.a, data.y)
effect = ipw.estimate_effect(potential_outcomes[1], potential_outcomes[0])
```

Comprehensive Jupyter Notebooks examples can be found in the examples directory.

1.3.1 Community support

We use the Slack workspace at causallib.slack.com for informal communication. We encourage you to ask questions regarding causal-inference modelling or usage of causallib that don't necessarily merit opening an issue on Github.

Use this invite link to join causallib on Slack.

1.3.2 Approach to causal-inference

Some key points on how we address causal-inference estimation

1.3.2.1 1. Emphasis on potential outcome prediction

Causal effect may be the desired outcome. However, every effect is defined by two potential (counterfactual) outcomes. We adopt this two-step approach by separating the effect-estimating step from the potential-outcome-prediction step. A beneficial consequence to this approach is that it better supports multi-treatment problems where "effect" is not well-defined.

1.3.2.2 2. Stratified average treatment effect

The causal inference literature devotes special attention to the population on which the effect is estimated on. For example, ATE (average treatment effect on the entire sample), ATT (average treatment effect on the treated), etc. By allowing out-of-bag estimation, we leave this specification to the user. For example, ATE is achieved by `model.estimate_population_outcome(X, a)` and ATT is done by stratifying on the treated: `model.estimate_population_outcome(X.loc[a==1], a.loc[a==1])`

1.3.2.3 3. Families of causal inference models

We distinguish between two types of models:

- ***Weight models*** [weight the data to balance between the treatment and control groups,] and then estimates the potential outcome by using a weighted average of the observed outcome. Inverse Probability of Treatment Weighting (IPW or IPTW) is the most known example of such models.
- ***Direct outcome models*** [uses the covariates (features) and treatment assignment to build a] model that predicts the outcome directly. The model can then be used to predict the outcome under any assignment of treatment values, specifically the potential-outcome under assignment of all controls or all treated. These models are usually known as *Standardization* models, and it should be noted that, currently, they are the only ones able to generate *individual effect estimation* (otherwise known as CATE).

1.3.2.4 4. Confounders and DAGs

One of the most important steps in causal inference analysis is to have proper selection on both dimensions of the data to avoid introducing bias:

- On rows: thoughtfully choosing the right inclusionexclusion criteria for individuals in the data.
- On columns: thoughtfully choosing what covariates (features) act as confounders and should be included in the analysis.

This is a place where domain expert knowledge is required and cannot be fully and truly automated by algorithms. This package assumes that the data provided to the model fit the criteria. However, filtering can be applied in real-time using a scikit-learn pipeline estimator that chains preprocessing steps (that can filter rows and select columns) with a causal model at the end.

CHAPTER

TWO

EXAMPLES

Comprehensive Jupyter Notebooks examples can be found in the examples directory on GitHub.

CHAPTER

THREE

DOCUMENTATION

3.1 Package `causallib`

A package for estimating causal effect and counterfactual outcomes from observational data.

`casuallib` provide various causal inference methods with a distinct paradigm:

- Every causal model has some machine learning model at its core. This allows to mix & match causal models with powerful machine learning tools, simply by plugging them into the causal model.
- Inspired by the scikit-learn design, once trained, causal models can be applied onto out-of-bag samples.

`causallib` also provide performance evaluation scheme of the causal model by evaluating the machine learning core model in a causal inference context.

Accompanying datasets are also available, both real and simulated ones. The various modules and folders provide the specific usage for each part.

3.1.1 Structure

The package is comprised of several modules, each providing a different functionality that is related to the causal inference models.

3.1.1.1 `estimation`

This module includes the estimator classes, where multiple popular estimators are implemented. Specifically, This includes

- Inverse probability weighting (IPW).
- Standardization.
- 3 versions of doubly-robust methods.

Each of these methods receives one or more machine learning models that can be trained (fit), and then used to estimate (predict) the relevant outcome of interest.

3.1.1.2 evaluation

This module provides the classes to evaluate the performance of methods defined in the estimation module. Evaluations are tailored to the type of method that is used. For example, weight estimators such as IPW can be evaluated for how well they remove bias from the data, while outcome models can be evaluated for their precision.

3.1.1.3 preprocessing

This module provides several enhancements to the filters and transformers provided by scikit-learn. These can be used within a pipeline framework together with the models.

3.1.1.4 datasets

Several datasets are provided within the package in the `datasets` module:

- NHEFS study data on the effect of smoking cessation on weight gain. Adapted from Hernán and Robins' Causal Inference Book
- A handful of simulation sets from the 2016 Atlantic Causal Inference Conference (ACIC) data challenge.
- Simulation module allows creating simulated data based on a causal graph depicting the connection between covariates, treatment assignment and outcomes.

3.1.1.5 Additional folders

Several additional folders exist under the package and hold several internal utilities. They should only be used as part of development. This folders include `analysis`, `simulation`, `utils`, and `tests`.

3.1.2 Usage

The examples folder contains several notebooks exemplifying the use of the package.

3.1.3 Subpackages

3.1.3.1 causallib.analysis package

3.1.3.1.1 Module contents

3.1.3.2 Module causallib.contrib

This module currently includes additional causal methods contributed to the package by causal inference researchers other than `causallib`'s core developers.

The causal models in this module can be slightly more novel then in the ones in `estimation` module. However, they should largely adhere to `causallib` API (e.g., `IndividualOutcomeEstimator` or `WeightEstimator`). Since code here is more experimental, models might also require additional (and less trivial) package dependencies, or have less test coverage. Well-integrated models could be transferred into the main `estimation` module in the future.

3.1.3.2.1 Contributed Methods

Currently contributed methods are:

1. Adversarial Balancing: implementing the algorithm described in Adversarial Balancing for Causal Inference. .. code-block:: python

   from causallib.contrib.adversarial_balancing import AdversarialBalancing

2. Interpretable Subgroup Discovery in Treatment Effect Estimation: implementing the heterogeneous effect mixture model (HEMM) presented in Interpretable Subgroup Discovery in Treatment Effect Estimation with Application to Opioid Prescribing Guidelines .. code-block:: python

   from causallib.contrib.hemm import HEMM

3. Matching Estimation/Transform using `faiss`.

   Implemented a nearest neighbors search with API that matches `sklearn.NearestNeighbors` but is powered by faiss for GPU support and much faster search on CPU as well.

```
from causallib.contrib.faissknn import FaissNearestNeighbors
```

3.1.3.2.2 Dependencies

Each model might have slightly different requirements. Refer to the documentation of each model for the additional packages it requires.

Requirements for `contrib` models are concentrated in `contrib/requirements.txt` and can be automatically installed using the extra-requirements `contrib` flag: `shell script pip install causallib[contrib] -f https://download.pytorch.org/whl/torch_stable.html` The `-f` find-links option is required to install PyTorch dependency.

3.1.3.2.3 References

Ozery-Flato, M., Thodoroff, P., Ninio, M., Rosen-Zvi, M., & El-Hay, T. (2018). Adversarial balancing for causal inference. arXiv preprint arXiv:1810.07406.

Nagpal, C., Wei, D., Vinzamuri, B., Shekhar, M., Berger, S. E., Das, S., & Varshney, K. R. (2020, April). Interpretable subgroup discovery in treatment effect estimation with application to opioid prescribing guidelines. In Proceedings of the ACM Conference on Health, Inference, and Learning (pp. 19-29).

3.1.3.2.4 Subpackages

causallib.contrib.adversarial_balancing package

Submodules

causallib.contrib.adversarial_balancing.adversarial_balancing module

class causallib.contrib.adversarial_balancing.adversarial_balancing.**AdversarialBalancing**(*learner*, *iterations=20*, *lr=0.5*, *decay=1*, *loss_type='01'*, *use_stabilized=Tru*, *verbose=False*, **args*, ***kwargs*)

Bases: *causallib.estimation.base_weight.WeightEstimator*, *causallib.estimation.base_estimator.PopulationOutcomeEstimator*

Adversarial Balancing finds sample weights such that the weighted population under any treatment A looks similar (distribution-wise) to the true population. Borrowing from GANs, the main idea is that, for each treatment A, the algorithm find weights such that a specified classifier cannot distinguish between the entire population and the weighted population under treatment *a*.

At each step we update the weights using the gradient of the exponential loss function, and re-train the classifier. For a given classifier family, an optimal solution are weights that maximize the minimal error of classifiers in this family.

For more details about the algorithm see: *Adversarial Balancing for Causal Inference* by Ozery-Flato and Thodoroff et al. https://arxiv.org/abs/1810.07406

Parameters

- **learner** – An initialized classifier object implementing fit and predict (scikit-learn compatible) Will be used to discriminate between the population under treatment a and the entire global population. A selection for each treatment value can be performed to choose the best classifier for that treatment group. It can be done by providing a scikit-learn initialized SearchCV model (either GridSearchCV or RandomizedSearchCV), or by providing a list of classifiers. If providing a list of classifiers, a selection will be done for each treatment value using cross-validation that will use the best-performing classifier among the list. see select_classifier module.
- **iterations** (*int*) – The number of iterations to adjust the weights of each sample
- **lr** (*float*) – Learning rate used to update the weights
- **decay** (*float*) – Parameter to decay the learning rate through the iterations
- **loss_type** (*str*) – Use '01' for zero-one loss, otherwise cross-entropy is used (and provided *learner* should also implement *predict_proba* methods).
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. Note: Adversarial balancing already has inherent component weighting treatment

  prevalence. Setting to False will "de-stabilize" the weights after they are calculated.
- **verbose** (*bool*) – Whether to print out statistics to console during training.

iterative_models_
np.ndarray of size(n_treatment_values, iterations) holding all the models created during training process.

iterative_normalizing_consts_
np.ndarray of size(n_treatment_values, iterations) holding all the normalizing constants calculated during training process.

discriminator_loss_
np.ndarray of size(n_treatment_values, iterations) holding the loss of the learner throughout the training process.

treatments_frequency_
if use_stabilized=True, the proportions of the treatment values.

compute_weight_matrix(*X*, *a*, *use_stabilized=None*, ***kwargs*)
Computes individual weight across all possible treatment values. f(Pr[A=a_j | X_i]) for all individual i and treatment j.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- ****kwargs** –

Returns

A matrix of size (num_subjects, num_treatments) with weight for every individual and every treatment.

Return type pd.DataFrame

compute_weights(*X*, *a*, *treatment_values=None*, *use_stabilized=None*, ***kwargs*)
Computes individual weight given the individual's treatment assignment. f(Pr[A=a_i | X_i]) for each individual i.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any | None*) – A desired value/s to extract weights to (i.e. weights to what treatment value should be calculated). If not specified, then the weights are chosen by the individual's actual treatment assignment.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- ****kwargs** –

Returns A vector of size (num_subjects,) with a weight for each individual

Return type pd.Series

estimate_population_outcome(*X*, *a*, *y*, *w=None*, *treatment_values=None*)

fit(*X*, *a*, *y=None*, *w_init=None*, ***select_kwargs*)
Trains an Adversarial Balancing model.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** – IGNORED.
- **w_init** (*pd.Series*) – Initial sample weights. If not provided, assumes uniform.
- **select_kwargs** – keywords argument to past into select_classifier. relevant only if model was initialized with list of classifiers in *learner*.

Returns AdversarialBalancing

causallib.contrib.adversarial_balancing.classifier_selection module

causallib.contrib.adversarial_balancing.classifier_selection.**select_classifier**(*model*, *X*, *A*, *n_splits=5*, *loss_type='01'*, *seed=None*)

Utility for selecting best classifier using cross-validation.

Parameters

- **model** – Either one of: scikit-learn classifier, scikit-learn SearchCV model (GridSearchCV, RandomizedSearchCV), list of classifiers.
- **X** (*np.ndarray*) – Covariate matrix size (num_samples, num_features)
- **A** (*np.ndarray*) – binary labels indicating the source and target populations (num_samples,)
- **n_splits** (*int*) – number of splits in cross-validation. relevant only if list of classifiers is passed.
- **loss_type** (*str*) – name of loss metric to select classifier by. Either '01' for zero-one loss, otherwise cross-entropy is used (and classifiers must implement predict_proba). relevant only if list of classifiers is passed.
- **seed** (*int*) – random seed for cross-validation split. relevant only if list of classifiers is passed.

Returns best performing classifier on validation set.

Return type classifier

Module contents

Module causallib.contrib.hemm

Implementation of the heterogeneous effect mixture model (HEMM) presented in the *Interpretable Subgroup Discovery in Treatment Effect Estimation with Application to Opioid Prescribing Guidelines* paper.

HEMM is used for discovering subgroups with enhanced and diminished treatment effects in a potential outcomes causal inference framework, using sparsity to enhance interpretability. The HEMM's outcome model is extended to include neural networks to better adjust for confounding and develop a joint inference procedure for the overall graphical model and neural networks. The model has two parts:

1. The subgroup discovery component.
2. The outcome prediction from the subgroup assignment and the interaction with confounders through an MLP.

The model can be initialized with any of the following outcome models:

- **Balanced Net**: A torch.model class that is used as a component of the HEMM module to determine the outcome as a function of confounders. The balanced net consists of two different neural networks for the two potential outcomes (under treatment and under control).
- **MLP model**: An MLP with an ELU activation. This allows for a single neural network to have two heads, one for each of the potential outcomes.
- **Linear model**: Linear model with two separate linear functions of the input covariates.

The balanced net outcome model relies on utility functions that are to be used with the balanced net outcome model based on *Estimating individual treatment effect: generalization bounds and algorithms*, Shalit et al., ICML (2017). The utility functions mainly consist of IPM metrics to calculate the imbalance between the control and treated population.

Submodules

causallib.contrib.hemm.gen_synthetic_data module

causallib.contrib.hemm.hemm module

causallib.contrib.hemm.hemm_api module

causallib.contrib.hemm.hemm_metrics module

causallib.contrib.hemm.hemm_outcome_models module

causallib.contrib.hemm.hemm_utilities module

causallib.contrib.hemm.load_ihdp_data module

Module contents

causallib.contrib.shared_sparsity_selection package

Submodules

causallib.contrib.shared_sparsity_selection.shared_sparsity_selection module

class causallib.contrib.shared_sparsity_selection.shared_sparsity_selection.**SharedSparsityConfounderSel**

Bases: causallib.preprocessing.confounder_selection._BaseConfounderSelection

Class to select confounders by first applying shared sparsity method. Method by Greenewald, Katz-Rogozhnikov, and Shanmugam: https://arxiv.org/abs/2011.01979

Constructor for SharedSparsityConfounderSelection

Parameters

- **mcp_lambda** (*str | float*) – Parameter (>= 0) to control shape of MCP regularizer. The bigger the value the stronger the regularization. "auto" will auto-select good regularization value.
- **mcp_alpha** (*float*) – Associated lambda parameter (>= 0) to control shape of MCP regularizer. The smaller the value the stronger the regularization.
- **step** (*float*) – Step size for proximal gradient, equivalent of learning rate.
- **max_iter** (*int*) – Maximum number of iterations of MCP proximal gradient.
- **tol** (*float*) – Stopping criterion for MCP. If the normalized value of proximal gradient is less than tol then the algorithm is assumed to have converged.
- **threshold** (*float*) – Only if the importance of a confounder exceeds threshold for all values of treatments, then the confounder is retained by transform() call.
- **importance_getter** – IGNORED.
- **covariates** (*list | np.ndarray*) – Specifying a subset of columns to perform selection on. Columns in *X* but not in *covariates* will be included after *transform* no matter the selection. Can be either a list of column names, or an array of boolean indicators length of *X*, or anything compatible with pandas *loc* function for columns. if *None* then all columns are participating in the selection process. This is similar to using sklearn's *ColumnTransformer* or *make_column_selector*.

fit(*X*, **args*, ***kwargs*)

Module contents

3.1.3.2.5 Submodules

causallib.contrib.faissknn module

3.1.3.2.6 Module contents

3.1.3.3 Module causallib.datasets

This module contains an example dataset, and a simulator to create a dataset.

3.1.3.3.1 Datasets

Currently one dataset is included. This is the National Health and Nutrition Examination Survey (NNHEFS) dataset. The dataset was adapted from the data available at https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/.

It can be loaded using

```
from causallib.datasets.data_loader import load_nhefs
data = load_nhefs()
covariates = data.X,
treatment_assignment = data.a,
observed_outcome = data.y
```

This loads an object in which `data.X`, `data.a`, and `data.y` respectively hold the features for each individual, whether they stopped-smoking, and their observed difference in weight between 1971 and 1983.

3.1.3.3.2 Simulator

This module implements a simulator and some related functions (e.g. creating random graph topologies)

CausalSimulator is based on an explicit graphical model connecting the feature data with several special nodes for treatment assignmenr, outcome, and censoring. CausalSimulator can generate the feature data randomly, or it can use a given dataset. The approach without input data is exhibited below, and the approach based on existing data is exemplified in the notebook ``*CasualSimulator_example.ipynb*`<CasualSimulator_example.ipynb>`_

With no given data

To initialize the simulator you need to state all the arguments regarding the graph's structure and variable related information

```
import numpy as np
from causallib.datasets import CausalSimulator
topology = np.zeros((4, 4), dtype=np.bool)  # topology[i,j] iff node j is a parent of␣
↪node i
topology[1, 0] = topology[2, 0] = topology[2, 1] = topology[3, 1] = topology[3, 2] = True
var_types = ["hidden", "covariate", "treatment", "outcome"]
link_types = ['linear', 'linear', 'linear', 'linear']
```

(continues on next page)

(continued from previous page)

```
prob_categories = [[0.25, 0.25, 0.5], None, [0.5, 0.5], None]
treatment_methods = "gaussian"
snr = 0.9
treatment_importance = 0.8
effect_sizes = None
outcome_types = "binary"

sim = CausalSimulator(topology=topology, prob_categories=prob_categories,
                      link_types=link_types, snr=snr, var_types=var_types,
                      treatment_importances=treatment_importance,
                      outcome_types=outcome_types,
                      treatment_methods=treatment_methods,
                      effect_sizes=effect_sizes)
X, prop, (y0, y1) = sim.generate_data(num_samples=100)
```

```
digraph CausalGraph {
hidden -> covariate
hidden -> treatment
covariate -> treatment
covariate -> outcome
treatment -> outcome
}
```

- This creates a graph `topology` of 4 variables, as depicted in the graph above: 1 hidden var (i.e. latent covariate), 1 regular covariate, 1 treatment variable and 1 outcome.
- `link_types` determines that all variables will have linear dependencies on their predecessors.
- `var_types`, together with `prov_categories` define:
  - Variable 0 (hidden) is categorical with categories distributed by the multinomial distribution `[0.25, 0.25, 0.5]`.
  - Variable 1 (covariate) is continuous (since its corresponding prob_category is None).
  - Variable 2 (treatment) is categorical and treatment assignment is equal between the treatment groups.
- `treatment_methods` means that treatment will be assigned by percentiles using a Gaussian distribution.
- All variables have signal to noise ratio of *signal* / (*signal*+*noise*) = 0.9.
- `treatment_importance = 0.8` indicates that the outcome will be affected 80% by treatment and 20% by all other predecessors.
- Effect size won't be manipulated into a specific desired value (since it is None).
- Outcome will be binary.

The data that is generated contains:

- `X` contains all the data generated (including latent variables, treatment assignments and outcome)
- `prop` contains the propensities
- `y0` and `y1` hold the counterfactual outcomes without and with treatment, respectively.

Additional examples

A more elaborate example that includes using existing data is available in the example notebook.

3.1.3.3.3 License

Datasets are provided under Community Data License Agreement (CDLA). The ACIC16 dataset is provided under CDLA-sharing license. The NHEFS dataset is provided under CDLA-permissive license. Please see the full corresponding license within each directory.

We thank the authors for sharing their data within this package.

3.1.3.3.4 Submodules

causallib.datasets.data_loader module

causallib.datasets.data_loader.**load_acic16**(*instance=1*, *raw=False*)

Loads single dataset from the 2016 Atlantic Causal Inference Conference data challenge.

The dataset is based on real covariates but synthetically simulates the treatment assignment and potential outcomes. It therefore also contains sufficient ground truth to evaluate the effect estimation of causal models. The competition introduced 7700 simulated files (100 instances for each of the 77 data-generating-processes). We provide a smaller sample of one instance from 10 DGPs. For the full dataset, see the link below to the competition site.

If used for academic purposes, please consider citing the competition organizers: Vincent Dorie, Jennifer Hill, Uri Shalit, Marc Scott, and Dan Cervone. "Automated versus do-it-yourself methods for causal inference: Lessons learned from a data analysis competition." Statistical Science 34, no. 1 (2019): 43-68.

Parameters

- **instance** (*int*) – number between 1-10 (inclusive), dataset to load.
- **raw** (*bool*) – Whether to apply contrast ("dummify") on non-numeric columns If True, returns a (pd.DataFrame, pd.DataFrame) tuple (one for covariates and the second with treatment assignment, noisy potential outcomes and true potential outcomes).

Returns

dictionary-like object

attributes are: *X* (covariates), *a* (treatment assignment), *y* (outcome),

***po* (ground truth potential outcomes: *po[0]* potential outcome for controls and *po[1]* potential outcome for treated),**

descriptors (feature description).

Return type Bunch

See also:

- Publication
- Official competition site
- Official github with data generating code

causallib.datasets.data_loader.**load_data_file**(*file_name*, *data_dir_name*, *sep=','*)

causallib.datasets.data_loader.**load_nhefs**(*raw=False*, *restrict=True*, *augment=True*, *onehot=True*)

Loads the NHEFS smoking-cessation and weight-loss dataset.

Data was gathered during an observational study conducted by the NHANS during the 1970's and 1980'. It follows a cohort a people whom some decided to quite smoking and some decided to persist, and record the gain in weight for each individual to try estimate the causal contribution of smoking cessation on weight gain.

This dataset is used throughout Hernán and Robins' Causal Inference Book. https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/

If used for academic purposes, please consider citing the book: Hernán MA, Robins JM (2020). Causal Inference: What If. Boca Raton: Chapman & Hall/CRC.

Parameters

- **raw** (*bool*) – Whether to return the entire DataFrame and descriptors or not. If False, only confounders are used for the data. If True, returns a (pd.DataFrame, pd.Series) tuple (data and description).
- **restrict** (*bool*) – Whether to apply exclusion criteria on missing data or not. Note: if False - data will have censored (NaN) outcomes.
- **augment** (*bool*) – Whether to add augmented (squared) features If False, only original data returned. If True, squares continuous valued columns ['age', 'wt71', 'smokeintensity', 'smokeyrs'] and joins to data frame with suffix '^2'
- **onehot** (*bool*) – Whether to one-hot encode categorical data. If False, categorical data ["active", "education", "exercise"], will be returned in individual columns with categorical values. If True, extra columns with the categorical value one-hot encoded.

Returns

dictionary-like object

attributes are: *X* (covariates), *a* (treatment assignment) *y* (outcome), *descriptors* (feature description)

Return type Bunch

causallib.datasets.data_loader.**load_nhefs_survival**(*augment=True*, *onehot=True*)

Loads and pre-processes the NHEFS smoking-cessation dataset.

Data was gathered in an observational study conducted by the NHANS during the 1970's and 1980'. It follows a cohort a people whom some decided to quite smoking and some decided to persist, and record the death events within 10 years of follow-up.

This dataset is used throughout Hernán and Robins' Causal Inference Book. https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/ If used for academic purposes, please consider citing the book: Hernán MA, Robins JM (2020). Causal Inference: What If. Boca Raton: Chapman & Hall/CRC.

Parameters

- **augment** (*bool*) – Whether to add augmented (squared) features If False, only original data returned. If True, squares continuous valued columns ['age', 'wt71', 'smokeintensity', 'smokeyrs'] and joins to data frame with suffix '^2'
- **onehot** (*bool*) – Whether to one-hot encode categorical data. If False, categorical data ["active", "education", "exercise"], will be returned in individual columns with categorical values. If True, extra columns with the categorical value one-hot encoded.

Returns Baseline covariate matrix of size (num_subjects, num_features). a (pd.Series): Treatment assignment of size (num_subjects,). Quit smoking vs. non-quit. t (pd.Series): Followup duration, size (num_subjects,). y (pd.Series): Observed outcome (1) or right censoring event (0), size (num_subjects,).

Return type X (pd.DataFrame)

3.1.3.3.5 Module contents

3.1.3.4 Module `causallib.estimation`

This module allows estimating counterfactual outcomes and effect of treatment using a variety of common causal inference methods, as detailed below. Each of these methods can use an underlying machine learning model of choice. These models must have an interface similar to the one defined by scikit-learn. Namely, they must have `fit()` and `predict()` functions implemented, and `predict_proba()` implemented for models that predict categorical outcomes.

Additional methods will be added incrementally.

3.1.3.4.1 Available Methods

The methods that are currently available are:

1. Inverse probability weighting (with minimal value cutoff): `causallib.estimation.IPW`
2. Standardization
    1. As a single model depending on treatment: `causallib.estimation.Standardization`
    2. Stratified by treatment value (similar to pooled regression): `causallib.estimation.StratifiedStandardization`
3. Doubly robust methods, as explained here
    1. Using the weighting as an additional feature: `causallib.estimation.DoublyRobustIpFeature`
    2. Using the weighting for training the standardization model: `causallib.estimation.DoublyRobustJoffe`
    3. Using the original formula for doubly robust estimation: `causallib.estimation.DoublyRobustVanilla`

Example: Inverse Probability Weighting (IPW)

An IPW model can be run, for example, using

```
from sklearn.linear_model import LogisticRegression
from causallib.estimation import IPW
from causallib.datasets.data_loader import fetch_smoking_weight

model = LogisticRegression()
ipw = IPW(learner=model)
data = fetch_smoking_weight()
ipw.fit(data.X, data.a)
ipw.estimate_population_outcome(data.X, data.a, data.y)
```

Note that `model` can be replaced by any machine learning model as explained above.

3.1.3.4.2 Submodules

causallib.estimation.base_estimator module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Apr 16, 2018

A module defining the various hierarchy of causal models interface. Causal models have two main tasks - predicting counterfactual outcomes and predicting effect based on these estimated outcomes. On top of it there are two resolutions we can work on: the individual level (i.e. outcome and effect for each individual in the dataset) and population level (i.e. some aggregation on the sample level). This module defines it all with: * EffectEstimator - can estimate both individual and population level effect * PopulationOutcomeEstimator - estimates aggregated outcomes on different sub-groups in the dataset. * IndividualOutcomeEstimator - estimates individual level outcomes.

class causallib.estimation.base_estimator.**EffectEstimator**

Bases: sklearn.base.BaseEstimator

Class-based interface for estimating either individual-level or sample-level effect.

CALCULATE_EFFECT = {'diff': <function EffectEstimator.<lambda>>, 'or': <function EffectEstimator.<lambda>>, 'ratio': <function EffectEstimator.<lambda>>}

estimate_effect(*outcome_1*, *outcome_2*, *effect_types='diff'*)

Estimates an effect given two potential outcomes.

Parameters

- **outcome_1** (*pd.Series | float*) – A potential outcome.
- **outcome_2** (*pd.Series | float*) – A potential outcome.
- **effect_types** (*list[str] | str*) – Any iterable of strings from the set of EffectEstimator.CALCULATE_EFFECT keys

Returns

A Series if population effect (input is scalar) with index are the effect types and values are the corresponding computed effect. A DataFrame if individual effect (input is a vector) where columns are effects types and rows are effect in each individual. Always: Value type is same is outcome_1 and outcome_2 type.

Return type pd.Series | pd.DataFrame

Examples

```
>>> from causallib.estimation.base_estimator import EffectEstimator
>>> effect_estimator = EffectEstimator()
>>> effect_estimator.estimate_effect(0.3, 0.6)
>>> {"diff": -0.3,     # 0.3 - 0.6
     "ratio": 0.5,     # 0.3 / 0.6
     "or": 0.2857}     # Odds-Ratio(0.3, 0.6)
```

class causallib.estimation.base_estimator.**IndividualOutcomeEstimator**(*learner*, *predict_proba=False*, **args*, ***kwargs*)

Bases: *causallib.estimation.base_estimator.PopulationOutcomeEstimator*, *causallib.estimation.base_estimator.EffectEstimator*

Interface for estimating individual-level outcome for different treatment values.

Parameters

- **learner** – Initialized sklearn model.
- **predict_proba** (*bool*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications.

estimate_effect(*outcome1*, *outcome2*, *agg='population'*, *effect_types='diff'*)

Estimates an effect given two potential outcomes.

Parameters

- **outcome1** (*pd.Series*) – A potential outcome.
- **outcome2** (*pd.Series*) – A potential outcome.
- **agg** (*str*) – Either "population" or "individual" - whether to calculate individual effect or population effect.
- **effect_types** (*list[str] | str*) – Any iterable of strings from the set of EffectEstimator.CALCULATE_EFFECT keys

Returns

A Series if population effect (input is scalar) with index are the effect types and values are the corresponding computed effect. A DataFrame if individual effect (input is a vector) where columns are effects types and rows are effect in each individual. Always: Value type is the same as outcome_1 and outcome_2 type.

Return type pd.Series | pd.DataFrame

abstract estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)

Estimates individual outcome under different treatment values (interventions)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.

- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None - parameter is ignored and behaviour is as specified when initializing the IndividualOutcomeEstimator.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

estimate_population_outcome(*X*, *a*, *y=None*, *treatment_values=None*, *agg_func='mean'*)

Implements aggregation of individual outcome into population (sample) outcome.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series | None*) – Observed outcome of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to stratify upon before aggregating individual into population outcome. If not supplied, calculates for all available treatment values.
- **agg_func** (*str*) – Type of aggregation function (e.g. "mean" or "median").

Returns

Series which index are treatment values, and the values are numbers - the aggregated outcome for the strata of people whose assigned treatment is the key.

Return type pd.Series

evaluate_fit(*X*, *y*, *a=None*)

abstract fit(*X*, *a*, *y*, *sample_weight=None*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

class causallib.estimation.base_estimator.**PopulationOutcomeEstimator**

Bases: *causallib.estimation.base_estimator.EffectEstimator*

Interface for estimating aggregated outcome over different subgroups in the dataset.

abstract estimate_population_outcome(*X*, *a*, *y*, *treatment_values=None*)

causallib.estimation.base_weight module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Apr 25, 2018

class causallib.estimation.base_weight.**PropensityEstimator**(*learner*, *use_stabilized=False*, **args*, ***kwargs*)

Bases: *causallib.estimation.base_weight.WeightEstimator*

Interface for causal estimators balancing datasets through propensity (i.e. treatment probability) estimation (e.g. inverse probability weighting).

Parameters

- **learner** – Initialized sklearn model.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

abstract compute_propensity(*X*, *a*, *treatment_values=None*, ***kwargs*)

abstract compute_propensity_matrix(*X*, *a*, ***kwargs*)

class causallib.estimation.base_weight.**WeightEstimator**(*learner*, *use_stabilized=False*, **args*, ***kwargs*)

Bases: object

Interface for causal estimators balancing datasets through weighting.

Parameters

- **learner** – Initialized sklearn model.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

abstract compute_weight_matrix(*X*, *a*, *use_stabilized=None*, ***kwargs*)

Computes individual weight across all possible treatment values. f(Pr[A=a_j | X_i]) for all individual i and treatment j.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- ****kwargs** –

Returns

A matrix of size (num_subjects, num_treatments) with weight for every individual and every treatment.

Return type pd.DataFrame

abstract compute_weights(*X*, *a*, *treatment_values=None*, *use_stabilized=None*, ***kwargs*)
Computes individual weight given the individual's treatment assignment. f(Pr[A=a_i | X_i]) for each individual i.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any | None*) – A desired value/s to extract weights to (i.e. weights to what treatment value should be calculated). If not specified, then the weights are chosen by the individual's actual treatment assignment.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- ****kwargs** –

Returns A vector of size (num_subjects,) with a weight for each individual

Return type pd.Series

evaluate_balancing(*X*, *a*, *y*, *w*)

abstract fit(*X*, *a*, *y=None*)
Trains a model to predict treatment assignment given the covariates: Pr[A|X].

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** – IGNORED.

Returns A causal weight model with an inner learner fitted.

Return type *WeightEstimator*

causallib.estimation.doubly_robust module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Apr 25, 2018

A module implementing several doubly-robust methods. These methods utilize causal standardization model and causal weight models and combine them to hopefully achieve a better model. The exact way to combine differs from different models and is described in the class docstring.

class causallib.estimation.doubly_robust.**AIPW**(*outcome_model*, *weight_model*, *outcome_covariates=None*, *weight_covariates=None*, *overlap_weighting=False*)

Bases: causallib.estimation.doubly_robust.BaseDoublyRobust

Calculates a doubly-robust estimate of the treatment effect by performing potential-outcome prediction (*outcome_model*) and then correcting its prediction-residuals using re-weighting from a treatment model (*weight_model*, like IPW).

It has two flavors, which slightly change the weighting of the outcome model in the correction term. Let e(X) be the estimated propensity score and m(X, A) is the estimated effect by an estimator, then the individual estimates are:

m(X,1) + A*(Y-m(X,1))/e(X), and
m(X,0) + (1-A)*(Y-m(X,0))/(1-e(X))

Which are basically add IP-weighted residuals from the observed predictions. As described in Kang and Schafer (2007) section 3.1 and Robins, Rotnitzky, and Zhao (1994).

The additional flavor when *overlap_weighting=True* is from Glynn and Quinn (2010), adds weighting by the propensity-of-the-other-class to the outcome model, so extreme example (with poor covariate overlap) will contribute less to the correction (i.e. rely less on their prediction value that might extrapolate too much). This is a similar notion used in Overlap Weights model (hence the argument name)

A * [Y - (1-e(X))m(X,1)]/e(X) + (1-A) * m(X,1), and
(1-A) * [Y - e(X)m(X,0)]/(1-e(X)) + A * m(X,0)

Parameters

- **outcome_model** (IndividualOutcomeEstimator) – A causal model that estimate on individuals level (e.g. Standardization).
- **weight_model** (WeightEstimator | PropensityEstimator) – A causal model for weighting individuals (e.g. IPW). If *overlap_weighting=True* then must be a *PropensityEstimator* model.
- **outcome_covariates** (*array*) – Covariates to use for outcome model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **weight_covariates** (*array*) – Covariates to use for weight model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **overlap_weighting** (*bool*) – Whether to tweak the outcome-model correction-term to rely less on data-points with poor covariate overlap (extreme propensity). if *True*, requires *weight_model* to be an instance of *PropensityEstimator*.

References

- Kang and Schafer, 2007, (https://dx.doi.org/10.1214/07-STS227)
- Kang and Schafer attribute the original method to Cassel, Särndal and Wretman.
- Glynn and Quinn, 2010, https://doi.org/10.1093/pan/mpp036
- Robins, Rotnitzky, and Zhao, 1994, https://doi.org/10.1080/01621459.1994.10476818

estimate_effect(*outcome1*, *outcome2*, *agg='population'*, *effect_types='diff'*)

Estimates an effect given two potential outcomes.

Parameters

- **outcome1** (*pd.Series*) – A potential outcome.
- **outcome2** (*pd.Series*) – A potential outcome.
- **agg** (*str*) – Either "population" or "individual" - whether to calculate individual effect or population effect.
- **effect_types** (*list[str] | str*) – Any iterable of strings from the set of EffectEstimator.CALCULATE_EFFECT keys

Returns

A Series if population effect (input is scalar) with index are the effect types and values are the corresponding computed effect. A DataFrame if individual effect (input is a vector) where columns are effects types and rows are effect in each individual. Always: Value type is the same as outcome_1 and outcome_2 type.

Return type pd.Series | pd.DataFrame

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)

Estimates individual outcome under different treatment values (interventions).

Notes

This method utilizes only the standardization model behind the doubly-robust model. Namely, this is an uncorrected outcome (that does not incorporates the weighted observed outcome). To get a true doubly-robust estimation use the estimate_population_outcome, rather than an individual outcome.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

estimate_population_outcome(*X*, *a*, *y=None*, *treatment_values=None*, *predict_proba=None*, *agg_func='mean'*)

Doubly-robust averaging, combining the individual counterfactual predictions from the standardization model and the weighted observed outcomes to estimate population outcome for each treatment subgroup.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to stratify upon before aggregating individual into population outcome. If not supplied, calculates for all available treatment values.
- **predict_proba** (*bool | None*) – To be used when provoking estimate_individual_outcome. In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications.
- **agg_func** (*str*) – Type of aggregation function (e.g. "mean" or "median").

Returns

Series which index are treatment values, and the values are numbers - the aggregated outcome for the strata of people whose assigned treatment is the key.

Return type pd.Series

fit(*X*, *a*, *y*, *refit_weight_model=True*, ***kwargs*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

class causallib.estimation.doubly_robust.**BaseDoublyRobust**(*outcome_model*, *weight_model*, *outcome_covariates=None*, *weight_covariates=None*)

Bases: *causallib.estimation.base_estimator.IndividualOutcomeEstimator*

Abstract class defining the interface and general initialization of specific doubly-robust methods.

Parameters

- **outcome_model** (IndividualOutcomeEstimator) – A causal model that estimate on individuals level (e.g. Standardization).
- **weight_model** (WeightEstimator) – A causal model for weighting individuals (e.g. IPW).

- **outcome_covariates** (*array*) – Covariates to use for outcome model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **weight_covariates** (*array*) – Covariates to use for weight model. If None - all covariates passed will be used. Either list of column names or boolean mask.

abstract fit(*X*, *a*, *y*, *refit_weight_model=True*, ***kwargs*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

class causallib.estimation.doubly_robust.**PropensityFeatureStandardization**(*outcome_model*, *weight_model*, *outcome_covariates=None*, *weight_covariates=None*, *feature_type='weight_vector'*)

Bases: *causallib.estimation.doubly_robust.BaseDoublyRobust*

A doubly-robust estimator of the effect of treatment. This model adds the weighting (inverse probability weighting) as additional feature to the outcome model.

References

- Bang and Robins, https://doi.org/10.1111/j.1541-0420.2005.00377.x
- Kang and Schafer, section 3.3, https://dx.doi.org/10.1214/07-STS227

Parameters

- **outcome_model** (IndividualOutcomeEstimator) – A causal model that estimate on individuals level
- **weight_model** (WeightEstimator | PropensityEstimator) – A causal model for weighting individuals (e.g. IPW).
- **outcome_covariates** (*array*) – Covariates to use for outcome model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **weight_covariates** (*array*) – Covariates to use for weight model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **feature_type** (*str*) – the type of covariate to add. One of the following options: * "weight_vector": uses a signed weight vector. Only defined for binary treatment.

  For example, if *weight_model* is IPW then: 1/Pr[A=a_i|X] for each sample *i*. As described in Bang and Robins (2005).

- "signed_weight_vector": as *'weight_vector'*, but negates the weights of the control group. For example, if *weight_model* is IPW then: 1/Pr[A|X] for treated and 1/Pr[A|X] for controls. As described in the correction for Bang and Robins (2008)
- **"weight_matrix": uses the entire weight matrix.**

  For example, if *weight_model* is IPW then: 1/Pr[A_i=a|X_i=x], for all treatment values *a* and for every sample *i*.
- "masked_weighted_matrix": uses the entire weight matrix, but masks it with a dummy-encoding of the treatment assignment. For example, if weight_model` is IPW then: 1/Pr[A=a_i|X=x_i] and 0 for all other *aa_i* columns. As described in Bang and Robins (2005).
- **"propensity_vector": uses the probabilities for being in treatment group: Pr[A=1|X].** Better defined for binary treatment. Equivalent to Scharfstein, Rotnitzky, and Robins (1999) that use its inverse.
- **"logit_propensity_vector": uses logit transformation of the propensity to treat Pr[A=1|X].** As described in Kang and Schafer (2007)
- **"propensity_matrix": uses the probabilities for all treatment options,** Pr[A_i=a|X_i=x] for all treatment values *a* and samples *i*.

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)

Estimates individual outcome under different treatment values (interventions)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None - parameter is ignored and behaviour is as specified when initializing the IndividualOutcomeEstimator.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

fit(*X*, *a*, *y*, *refit_weight_model=True*, ***kwargs*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

class causallib.estimation.doubly_robust.**WeightedStandardization**(*outcome_model*, *weight_model*, *outcome_covariates=None*, *weight_covariates=None*)

Bases: *causallib.estimation.doubly_robust.BaseDoublyRobust*

This model uses the weights from the weight-model (e.g. inverse probability weighting) as individual weights for fitting the outcome model.

References

- Kang and Schafer, section 3.2, https://dx.doi.org/10.1214/07-STS227

Parameters

- **outcome_model** (IndividualOutcomeEstimator) – A causal model that estimate on individuals level (e.g. Standardization).
- **weight_model** (WeightEstimator) – A causal model for weighting individuals (e.g. IPW).
- **outcome_covariates** (*array*) – Covariates to use for outcome model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **weight_covariates** (*array*) – Covariates to use for weight model. If None - all covariates passed will be used. Either list of column names or boolean mask.

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)

Estimates individual outcome under different treatment values (interventions)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None - parameter is ignored and behaviour is as specified when initializing the IndividualOutcomeEstimator.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

fit(*X*, *a*, *y*, *refit_weight_model=True*, ***kwargs*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

causallib.estimation.ipw module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Apr 25, 2018

class causallib.estimation.ipw.**IPW**(*learner*, *clip_min=None*, *clip_max=None*, *use_stabilized=False*, *verbose=False*)

Bases: *causallib.estimation.base_weight.PropensityEstimator*, *causallib.estimation.base_estimator.PopulationOutcomeEstimator*

Causal model implementing inverse probability (propensity score) weighting. w_i = 1 / Pr[A=a_i|Xi]

Parameters

- **learner** – Initialized sklearn model.
- **clip_min** (*None | float*) – Optional value between 0 to 0.5 to lower bound the propensity estimation by clipping it. Will clip probabilities under clip_min to this value.
- **clip_max** (*None | float*) – Optional value between 0.5 to 1 to upper bound the propensity estimation by clipping it. Will clip probabilities above clip_max to this value.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- **verbose** (*bool*) – Whether to print summary statistics regarding the number of samples clipped due to clip_min and clip_max.

compute_propensity(*X*, *a*, *treatment_values=None*, *clip_min=None*, *clip_max=None*)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any | None*) – A desired value/s to extract propensity to (i.e. probabilities to what treatment value should be calculated). If not specified, then the maximal treatment value is chosen. This is since the usual case is of treatment (A=1) control (A=0) setting.

- **clip_min** (*None | float*) – Optional value between 0 to 0.5 to lower bound the propensity estimation by clipping it. Will clip probabilities under clip_min to this value.
- **clip_max** (*None | float*) – Optional value between 0.5 to 1 to upper bound the propensity estimation by clipping it. Will clip probabilities above clip_max to this value.

Returns

A matrix/vector num_subjects rows and number of columns is the number of values provided to treatment_value. The content is probabilities for every individual to have the specified treatment_value. If treatment_value is a list/vector, than a pd.DataFrame is returned. If treatment_value is sort of scalar, than a pd.Series is returned.

(just like slicing a DataFrame's columns)

Return type pd.DataFrame | pd.Series

compute_propensity_matrix(*X*, *a=None*, *clip_min=None*, *clip_max=None*)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **clip_min** (*None | float*) – Optional value between 0 to 0.5 to lower bound the propensity estimation by clipping it. Will clip probabilities under clip_min to this value.
- **clip_max** (*None | float*) – Optional value between 0.5 to 1 to upper bound the propensity estimation by clipping it. Will clip probabilities above clip_max to this value.

Returns

A matrix of size (num_subjects, num_treatments) with probability for every individual and e very treatment.

Return type pd.DataFrame

compute_weight_matrix(*X*, *a*, *clip_min=None*, *clip_max=None*, *use_stabilized=None*)

Computes individual weight across all possible treatment values. w_ij = 1 / Pr[A=a_j | X_i] for all individual i and treatment j.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **clip_min** (*None | float*) – Optional value between 0 to 0.5 to lower bound the propensity estimation by clipping it. Will clip probabilities under clip_min to this value.
- **clip_max** (*None | float*) – Optional value between 0.5 to 1 to upper bound the propensity estimation by clipping it. Will clip probabilities above clip_max to this value.
- **use_stabilized** (*None | bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. If True provided, but the model was initialized with

use_stabilized=False, then prevalence is calculated from data at hand, rather than the prevalence from the training data. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

Returns

A matrix of size (num_subjects, num_treatments) with weight for every individual and every treatment.

Return type pd.DataFrame

compute_weights(*X*, *a*, *treatment_values=None*, *clip_min=None*, *clip_max=None*, *use_stabilized=None*)

Computes individual weight given the individual's treatment assignment. w_i = 1 / Pr[A=a_i|X_i] for each individual i.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any | None*) – A desired value/s to extract weights to (i.e. weights to what treatment value should be calculated). If not specified, then the weights are chosen by the individual's actual treatment assignment.
- **clip_min** (*None|float*) – Optional value between 0 to 0.5 to lower bound the propensity estimation by clipping it. Will clip probabilities under clip_min to this value.
- **clip_max** (*None|float*) – Optional value between 0.5 to 1 to upper bound the propensity estimation by clipping it. Will clip probabilities above clip_max to this value.
- **use_stabilized** (*None|bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. If True provided, but the model was initialized with use_stabilized=False, then prevalence is calculated from data at hand, rather than the prevalence from the training data. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

Returns

If treatment_values is not supplied (None) or is a scalar, then a vector of n_samples with a weight for each sample is returned. If treatment_values is a list/array, then a DataFrame is returned.

Return type pd.Series | pd.DataFrame

estimate_population_outcome(*X*, *a*, *y*, *w=None*, *treatment_values=None*)

Calculates weighted population outcome for each subgroup stratified by treatment assignment.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **w** (*pd.Series | None*) – Individual (sample) weights calculated. Used to achieved unbiased average outcome. If not provided, will be calculated on the data.
- **treatment_values** (*Any*) – Desired treatment value/s to stratify upon. Must be a subset of values found in *a*. If not supplied, calculates for all available treatment values.

Returns

Series which index are treatment values, and the values are numbers - the aggregated outcome for the strata of people whose assigned treatment is the key.

Return type pd.Series[Any, float]

fit(*X*, *a*, *y=None*)

Trains a model to predict treatment assignment given the covariates: Pr[A|X].

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** – IGNORED.

Returns A causal weight model with an inner learner fitted.

Return type *WeightEstimator*

causallib.estimation.marginal_outcome module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Apr 25, 2018

class causallib.estimation.marginal_outcome.**MarginalOutcomeEstimator**(*learner*, *use_stabilized=False*, **args*, ***kwargs*)

Bases: *causallib.estimation.base_weight.WeightEstimator*, *causallib.estimation.base_estimator.PopulationOutcomeEstimator*

A marginal outcome predictor. Assumes the sample is marginally exchangeable, and therefore does not correct (adjust, control) for covariates. Predicts the outcome/effect as if the sample came from a randomized control trial: $Pr[Y|A]$.

Parameters

- **learner** – Initialized sklearn model.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

compute_weight_matrix(*X*, *a*, *use_stabilized=None*, ***kwargs*)

Computes individual weight across all possible treatment values. f(Pr[A=a_j | X_i]) for all individual i and treatment j.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).

- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- ****kwargs** –

Returns

A matrix of size (num_subjects, num_treatments) with weight for every individual and every treatment.

Return type pd.DataFrame

compute_weights(*X*, *a*, *treatment_values=None*, *use_stabilized=None*, ***kwargs*)

Computes individual weight given the individual's treatment assignment. f(Pr[A=a_i | X_i]) for each individual i.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any | None*) – A desired value/s to extract weights to (i.e. weights to what treatment value should be calculated). If not specified, then the weights are chosen by the individual's actual treatment assignment.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- ****kwargs** –

Returns A vector of size (num_subjects,) with a weight for each individual

Return type pd.Series

estimate_population_outcome(*X*, *a*, *y*, *w=None*, *treatment_values=None*)

Calculates potential population outcome for each treatment value.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **w** (*pd.Series | None*) – Individual (sample) weights calculated. Used to achieved unbiased average outcome. If not provided, will be calculated on the data.
- **treatment_values** (*Any*) – Desired treatment value/s to stratify upon before aggregating individual into population outcome. If not supplied, calculates for all available treatment values.

Returns

Series which index are treatment values, and the values are numbers - the aggregated outcome for the strata of people whose assigned treatment is the key.

Return type pd.Series[Any, float]

fit(*X=None*, *a=None*, *y=None*)

Dummy implementation to match the API. MarginalOutcomeEstimator acts as a WeightEstimator that weights each sample as 1

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).

Returns a fitted model.

Return type *MarginalOutcomeEstimator*

causallib.estimation.matching module

class causallib.estimation.matching.**KNN**(*learner*, *index*)

Bases: tuple

Create new instance of KNN(learner, index)

index
Alias for field number 1

learner
Alias for field number 0

class causallib.estimation.matching.**Matching**(*propensity_transform=None*, *caliper=None*, *with_replacement=True*, *n_neighbors=1*, *matching_mode='both'*, *metric='mahalanobis'*, *knn_backend='sklearn'*, *estimate_observed_outcome=False*)

Bases: *causallib.estimation.base_estimator.IndividualOutcomeEstimator*, *causallib.estimation.base_weight.WeightEstimator*

Match treatment and control samples with similar covariates.

Parameters

- **propensity_transform** (*causallib.transformers.PropensityTransformer*) – an object for data preprocessing which adds the propensity score as a feature (default: None)
- **caliper** (*float*) – maximal distance for a match to be accepted. If not defined, all matches will be accepted. If defined, some samples may not be matched and their outcomes will not be estimated. (default: None)
- **with_replacement** (*bool*) – whether samples can be used multiple times for matching. If set to False, the matching process will optimize the linear sum of distances between pairs of treatment and control samples and only *min(N_treatment, N_control)* samples will be estimated. Matching with no replacement does not make use of the *fit* data and is therefore not implemented for out-of-sample data (default: True)
- **n_neighbors** (*int*) – number of nearest neighbors to include in match. Must be 1 if *with_replacement* is *False*. If larger than 1, the estimate is calculated using the *regress_agg_function* or *classify_agg_function* across the *n_neighbors*. Note that when the *caliper* variable is set, some samples will have fewer than *n_neighbors* matches. (default: 1).
- **matching_mode** (*str*) – Direction of matching: *treatment_to_control*, *control_to_treatment* or *both* to indicate which set should be matched to which. All sets are cross-matched in *match* and when *with_replacement* is *False* all matching modes coincide. With replacement there is a difference.

- **metric** (*str*) – Distance metric string for calculating distance between samples. Note: if an external built *knn_backend* object with a different metric is supplied, *metric* needs to be changed to reflect that, because *Matching* will set its inverse covariance matrix if "mahalanobis" is set. (default: "mahalanobis", also supported: "euclidean")
- **knn_backend** (*str or callable*) – Backend to use for nearest neighbor search. Options are "sklearn" or a callable which returns an object implementing *fit*, *kneighbors* and *set_params* like the sklearn *NearestNeighbors* object. (default: "sklearn").
- **estimate_observed_outcome** (*bool*) – Whether to allow a match of a sample to a sample other than itself when looking within its own treatment value. If True, the estimated potential outcome for the observed outcome may differ from the true observed outcome. (default: False)

classify_agg_function
Aggregating function for outcome estimation when classifying. (default: majority_rule) Usage is determined by type of *y* during *fit*

Type callable

regress_agg_function
Aggregating function for outcome estimation when regressing or predicting prob_a. (default: np.mean) Usage is determined by type of *y* during *fit*

Type callable

treatments_
DataFrame of treatments (created after *fit*)

Type pd.DataFrame

outcomes_
DataFrame of outcomes (created after *fit*)

Type pd.DataFrame

match_df_
Dataframe of most recently calculated matches. For details, see *match*. (created after *match*)

Type pd.DataFrame

samples_used_
Series with count of samples used during most recent match. Series includes a count for each treatment value. (created after *match*)

Type pd.Series

compute_weight_matrix(*X*, *a*, *use_stabilized=None*, ***kwargs*)
Computes individual weight across all possible treatment values. f(Pr[A=a_j | X_i]) for all individual i and treatment j.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title
- ****kwargs** –

Returns

A matrix of size (num_subjects, num_treatments) with weight for every individual and every treatment.

Return type pd.DataFrame

compute_weights(*X*, *a*, *treatment_values=None*, *use_stabilized=None*, ***kwargs*)

Calculate weights based on a given set of matches.

Only applicable for *matching_mode* "control_to_treatment" or "treatment_to_control".

Parameters

- **X** (*pd.DataFrame*) – DataFrame of shape (n,m) containing m covariates for n samples.
- **a** (*pd.Series*) – Series of shape (n,) containing discrete treatment values for the n samples.
- **treatment_values** – IGNORED.
- **use_stabilized** – IGNORED.
- ****kwargs** –

Returns a Series of shape (n,) with a weight per sample.

Return type pd.Series

Raises `ValueError if Matching().matching_mode == 'both'.` –

estimate_individual_outcome(*X*, *a*, *y=None*, *treatment_values=None*, *predict_proba=True*, *dropna=True*)

Calculate the potential outcome for each sample and treatment value.

Execute match and calculate, for each treatment value and each sample, the expected outcome.

Note: Out of sample estimation for matching without replacement requires passing a *y* vector here. If no 'y' is passed here, the values received by *fit* are used, and if the estimation indices are not a subset of the fitted indices, the estimation will fail.

If the attribute *estimate_observed_outcome* is *True*, estimates will be calculated for the observed outcomes as well. If not, then the observed outcome will be passed through from the corresponding element of *y* passed to *fit*.

Parameters

- **X** (*pd.DataFrame*) – DataFrame of shape (n,m) containing m covariates for n samples.
- **a** (*pd.Series*) – Series of shape (n,) containing discrete treatment values for the n samples.
- **y** (*pd.Series*) – Series of shape (n,) containing outcome values for n samples. This is only used when *with_replacemnt=False*. Otherwise, the outcome values passed to *fit* are used.
- **predict_proba** (*bool*) – whether to output classifications or probabilties for a classification task. If set to False and data is non-integer, a warning is issued. (default True)
- **dropna** (*bool*) – For samples that were unmatched due to caliper restrictions, drop from outcome_df leading to a potentially smaller sized output, or include them as NaN. (default: True)
- **treatment_values** – IGNORED

Note: The args are assumed to share the same index.

Returns outcome_df (pd.DataFrame)

fit(*X*, *a*, *y*, *sample_weight=None*)

Load the treatments and outcomes and fit search trees.

Applies transform to covariates X, initializes search trees for each treatment value for performing nearest neighbor searches. Note: Running *fit* a second time overwrites any information from previous *fit or `match* and re-fits the propensity_transform object.

Parameters

- **X** (*pd.DataFrame*) – DataFrame of shape (n,m) containing m covariates for n samples.
- **a** (*pd.Series*) – Series of shape (n,) containing discrete treatment values for the n samples.
- **y** (*pd.Series*) – Series of shape (n,) containing outcomes for the n samples.
- **sample_weight** – IGNORED In signature for compatibility with other estimators.

Note: *X*, *a* and *y* must share the same index.

Returns self (Matching) the fitted object

get_covariates_of_matches(*s*, *t*, *covariates*)

Look up covariates of closest matches for a given matching.

Using *self.match_df_* and the supplied *covariates*, look up the covariates of the last match. The function can only be called after *match* has been run.

Args: s (int) : source treatment value t (int) : target treatment value covariates (pd.DataFrame) : The same covariates which were

passed to *fit*.

Returns: covariate_df (pd.DataFrame) : a DataFrame of size (n_matched_samples, n_covariates * 3 + 2) with the covariate values of the sample, covariates of its match, calculated distance and number of neighbors found within the given caliper (with no caliper this will equal self.n_neighbors)

match(*X*, *a*, *use_cached_result=True*, *successful_matches_only=False*)

Matching the samples in X according to the treatment values in a.

Returns a DataFrame including all the results, which is also set as the attribute *self.match_df_*. The arguments *X* and *a* define the "needle" where the "haystack" is the data that was previously passed to fit, for matching with replacement. As such, treatment and control samp les from within *X* will not be matched with each other, unless the same *X* and *a* were passed to *fit*. For matching without replacement, the *X* and *a* passed to *match* provide the "needle" and the "haystack". If the attribute *caliper* is set, the matches are limited to those with a distance less than *caliper*.

Parameters

- **X** (*pd.DataFrame*) – DataFrame of shape (n,m) containing m covariates for n samples.
- **a** (*pd.Series*) – Series of shape (n,) containing discrete treatment values for the n samples.
- **use_cached_result** (*bool*) – Whether or not to return the *match_df* from the most recent matching operation. The cached result will only be used if the sample indices of *X* and those of *match_df* are identical, otherwise it will rematch.
- **successful_matches_only** (*bool*) – Whether or not to filter the matches to those which matched successfully. If set to *False*, the resulting DataFrame will have shape

(n* len(a.unique()), 2), otherwise it may have a smaller shape due to unsuccessful matches.

Note: The args are assumed to share the same index.

Returns

The resulting matches DataFrame is indexed so that

` **match_df.loc[treatment_value, sample_id]` has columns *matches*** and *distances* containing lists of indices to samples and the respective distances for the matches discovered for *sample_id* from within the fitted samples with the given *treatment_value*. The indices in the *matches* column are from the fitted data, not the X argument in *match*. If *sample_id* had no match, *match_df.loc[treatment_value, sample_id].matches = []*. The DataFrame has shape (n* len(a.unique()), 2), if *successful_matches_only* is set to `False.

Return type match_df

Raises `NotImplementedError` – Raised when with_replacement is False and n_neighbors is not 1.

`matches_to_weights`(*match_df=None*)

Calculate weights based on a given set of matches.

For each matching from one treatment value to another, a weight vector is generated. The weights are calculated as the number of times a sample was selected in a matching, with each occurrence weighted according to the number of other samples in that matching. The weights can be used to estimate outcomes or to check covariate balancing. The function can only be called after *match* has been run.

Parameters `match_df` (*pd.DataFrame*) – a DataFrame of matches returned from *match*. If not supplied, will use the *match_df_* attribute if available, else raises ValueError. Will not execute *match* to generate a *match_df*.

Returns

DataFrame of shape (n,M) where M is the number of permutations of *a.unique()*.

Return type weights_df (pd.DataFrame)

`class` causallib.estimation.matching.`PropensityMatching`(*learner*, ***kwargs*)

Bases: *causallib.estimation.matching.Matching*

Matching on propensity score only.

This is a convenience class to execute the common task of propensity score matching. It shares all of the methods of the *Matching* class but offers a shortcut for initialization.

Parameters

- `learner` (*sklearn.estimator*) – a trainable propensity model that implements *fit* and *predict_proba*. Will be passed to the *PropensityTransformer* object.
- `**kwargs` – see Matching.__init__ for supported kwargs.

causallib.estimation.matching.`majority_rule`(*x*)

causallib.estimation.overlap_weights module

(C) Copyright 2021 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Jun 09, 2021

class causallib.estimation.overlap_weights.**OverlapWeights**(*learner*, *use_stabilized=False*)

Bases: *causallib.estimation.ipw.IPW*

Implementation of overlap (propensity score) weighting:

https://www.tandfonline.com/doi/full/10.1080/01621459.2016.1260466

A method to balance observed covariates between treatment groups in observational studies. Down-weigh observations with extreme propensity and weigh up Put less importance to observations with extreme propensity scores, and put more emphasis on observations with a central tendency towards (i.e. overlapping propensity scores).

Each unit's weight is proportional to the probability of that unit being assigned to the opposite group: w_i = 1 - Pr[A=a_i|Xi]

This method assumes only two treatment groups exist.

Parameters

- **learner** – Initialized sklearn model.
- **use_stabilized** (*bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

compute_weight_matrix(*X*, *a*, *clip_min=None*, *clip_max=None*, *use_stabilized=None*)

Computes individual weight across all possible treatment values. w_ij = 1 - Pr[A=a_j | X_i] for all individual i and treatment j.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **clip_min** (*None* | *float*) – Lower bound for propensity scores. Better be left *None*.
- **clip_max** (*None* | *float*) – Upper bound for propensity scores. Better be left *None*.
- **use_stabilized** (*None* | *bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. If True provided, but the model was initialized with use_stabilized=False, then prevalence is calculated from data at hand, rather than the prevalence from the training data. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

Returns

A matrix of size (num_subjects, num_treatments) with weight for every individual and every treatment.

Return type pd.DataFrame

stabilize_weights(*a*, *weight_matrix*, *use_stabilized=False*)

Adjust sample weights according to class prevalence: Pr[A=a_i] * w_i

Parameters

- **weight_matrix** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **use_stabilized** (*None|bool*) – Whether to re-weigh the learned weights with the prevalence of the treatment. This overrides the use_stabilized parameter provided at initialization. If True provided, but the model was initialized with use_stabilized=False, then prevalence is calculated from data at hand, rather than the prevalence from the training data. See Also: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4351790/#S6title

Returns

A matrix of size (num_subjects, num_treatments) with stabilized (if True) weight for every individual and every treatment.

Return type pd.DataFrame

causallib.estimation.rlearner module

(C) Copyright 2021 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on April 4, 2021

class causallib.estimation.rlearner.**RLearner**(*effect_model*, *outcome_model*, *treatment_model*, *outcome_covariates=None*, *treatment_covariates=None*, *effect_covariates=None*, *n_splits=5*, *refit=True*, *caliper=1e-06*, *non_parametric=False*)

Bases: *causallib.estimation.base_estimator.IndividualOutcomeEstimator*

Given the measured outcome Y, the assignment A, and the coefficients X calculate an R-learner estimator of the effect of the treatment Let e(X) be the estimated propensity score and m(X) is the estimated outcome (E[Y|X]) by an estimator, then the R-learner minimize the following:

||Y - m(X) - (A-e(X)) au(X)||^2_2 + lambda (au)

where au(X) is a conditional average treatment effect and lambda is a regularize coefficient.

If the effect_model is Linear, than minimizing squared loss with the target variable (Y-m(X)) and the features (A-e(X))X, otherwise it corresponds to a weighted regression problem, where the weights are (A-e(X))**2. This can be used with any scikit-learn regressor that accepts sample weights

References: Nie, X., & Wager, S.(2017). Quasi - oracle estimation of heterogeneous treatment effects https://arxiv.org/abs/1712.04912

Chernozhukov, V., et al. (2018). Double/debiased machine learning for treatment and structural parameters. https://academic.oup.com/ectj/article/21/1/C1/5056401

Parameters

- **effect_model** – An sklearn model that estimate that estimate the conditional average treatment effect au(X)
- **outcome_model** – An sklearn model that estimate the regressor Y|X (without the treatment). Note: it is recommended to use a regressor, even for binary outcome.
- **treatment_model** – An sklearn model that estimate the treatment model or the probability to be treated, i.e A|X or P(A=1|X)
- **outcome_covariates** (*array*) – Covariates to use for the outcome model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **treatment_covariates** (*array*) – Covariates to use for treatment model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **effect_covariates** (*array*) – Covariates to use for the effect model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **n_splits** (*int*) – number of sample-splitting in the cross-fitting procedure
- **refit** (*bool*) – if True - Nuisance models are fitted over the whole training set, otherwise Nuisance models are fitted per folds
- **non_parametric** (*bool*) – if True - the effect_model is estimated as weighted regression task, otherwise the effect_model is considered linear.

estimate_individual_effect(*X*)

Predict the individual treatment effect :param X: Covariate matrix of size (num_subjects, num_features). :type X: pd.DataFrame

Returns

The series is a vector in size (num_subjects) that contains the estimated treatment effect, each row is an individual

Return type pd.Series

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=False*)

Estimating corrected individual counterfactual outcomes.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome. If not supplied, calculates for all available treatment values.
- **predict_proba** – IGNORED. Not used, present for API consistency by convention.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

fit(*X*, *a*, *y*, *caliper=None*)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).

- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **caliper** (*None | float*) – minimal value of treatment-probability residual. used to avoid division by zero when fitting the effect-model. If None - no clipping is done. The caliper is irrelevant if the effect_model is Linear.

class causallib.estimation.rlearner.**VotingEstimator**(*estimators*)

Bases: object

predict(*X*)

Aggregate results of different estimators

causallib.estimation.standardization module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Apr 25, 2018

class causallib.estimation.standardization.**Standardization**(*learner*, *encode_treatment=False*, *predict_proba=False*)

Bases: *causallib.estimation.base_estimator.IndividualOutcomeEstimator*

Standard standardization model for causal inference. Learns a model that takes into account the treatment assignment, and later, this value can be intervened, changing the predicted outcome.

Parameters

- **learner** – Initialized sklearn model.
- **encode_treatment** (*bool*) – Whether to encode the treatment as one-hot matrix. Usually good if n_treatment > 2.
- **predict_proba** (*bool*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications.

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)

Estimates individual outcome under different treatment values (interventions)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.

- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None - parameter is ignored and behaviour is as specified when initializing the IndividualOutcomeEstimator.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

fit(*X*, *a*, *y*, *sample_weight=None*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

class causallib.estimation.standardization.**StratifiedStandardization**(*learner*, *treatment_values=None*, *predict_proba=False*)

Bases: *causallib.estimation.base_estimator.IndividualOutcomeEstimator*

Standardization model that learns a model for each treatment group (i.e. subgroup of subjects with the same treatment assignment).

Parameters

- **learner** – Initialized sklearn model or a mapping (dict) between treatment value and initialized model, For example: {0: Ridge(alpha=5), 1: Ridge(alpha=0.1)}, or even different models all over: {0: Ridge(), 1: RandomForestRegressor} Make sure these treatment_values keys represent all treatment values found in later use.
- **treatment_values** (*list*) – list of unique values of treatment (can be a single value as well). If known beforehand (on initialization time), can be passed now to init, otherwise would be inferred during fit (where treatment assignment must be supplied). Make sure these treatment_values represent all treatment values found in later use.
- **predict_proba** (*bool*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications.

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)

Estimates individual outcome under different treatment values (interventions)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).

- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None - parameter is ignored and behaviour is as specified when initializing the IndividualOutcomeEstimator.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

fit(*X*, *a*, *y*, *sample_weight=None*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

causallib.estimation.tmle module

class causallib.estimation.tmle.**BaseCleverCovariate**(*weight_model*)

Bases: object

abstract clever_covariate_fit(*X*, *a*)

abstract clever_covariate_inference(*X*, *a*, *treatment_value*)

abstract sample_weights(*X*, *a*)

class causallib.estimation.tmle.**CleverCovariateFeatureMatrix**(*weight_model*)

Bases: *causallib.estimation.tmle.CleverCovariateImportanceSamplingMatrix*

Clever covariate uses a matrix of inverse propensity weights of all treatment values as a predictor to the targeting regression.

References

- Gruber S, van der Laan M. tmle: An R package for targeted maximum likelihood estimation. 2012. https://doi.org/10.18637/jss.v051.i13

clever_covariate_fit(*X*, *a*)

clever_covariate_inference(*X*, *a*, *treatment_value*)

sample_weights(*X*, *a*)

class causallib.estimation.tmle.**CleverCovariateFeatureVector**(*weight_model*)
Bases: *causallib.estimation.tmle.BaseCleverCovariate*

Clever covariate uses a signed vector of inverse propensity weights, with control group have their weights negated. The vector is then used as a predictor to the targeting regression.

References

- Schuler MS, Rose S. Targeted maximum likelihood estimation for causal inference in observational studies. 2017 https://doi.org/10.1093/aje/kww165

clever_covariate_fit(*X*, *a*)

clever_covariate_inference(*X*, *a*, *treatment_value*)

sample_weights(*X*, *a*)

class causallib.estimation.tmle.**CleverCovariateImportanceSamplingMatrix**(*weight_model*)
Bases: *causallib.estimation.tmle.BaseCleverCovariate*

Clever covariate of inverse propensity weight vector is used as weight for the targeting regression. The predictors are a one-hot (full dummy) encoding of the treatment assignment.

References

- Gruber S, van der Laan M. tmle: An R package for targeted maximum likelihood estimation. 2012. https://doi.org/10.18637/jss.v051.i13

clever_covariate_fit(*X*, *a*)

clever_covariate_inference(*X*, *a*, *treatment_value*)

sample_weights(*X*, *a*)

class causallib.estimation.tmle.**CleverCovariateImportanceSamplingVector**(*weight_model*)
Bases: *causallib.estimation.tmle.BaseCleverCovariate*

Clever covariate of inverse propensity weight vector is used as weight for the targeting regression. The predictors are a signed vector with negative 1 for the control group.

clever_covariate_fit(*X*, *a*)

clever_covariate_inference(*X*, *a*, *treatment_value*)

sample_weights(*X*, *a*)

class causallib.estimation.tmle.**TMLE**(*outcome_model*, *weight_model*, *outcome_covariates=None*, *weight_covariates=None*, *reduced=False*, *importance_sampling=False*, *glm_fit_kwargs=None*)

Bases: *causallib.estimation.doubly_robust.BaseDoublyRobust*

Targeted Maximum Likelihood Estimation. A model that takes an outcome model that was optimized to predict E[Y|X,A], and "retargets" ("updates") it to estimate E[Y^A|X] using a "clever covariate" constructed from the inverse propensity weights.

Steps:

1. Fit an outcome model Y=Q(X,A).
2. Fit a weight model A=g(X,A).
3. Construct a clever covariate using g(X,A).
4. Fit a logistic regression model Q* to predict Y using g(X,A) as features and Q(X,A) as offset.
5. Predict counterfactual outcome for treatment value *a* Q*(X,a) by plugging in Q(X,a) as offset, g(X,a) as covariate.

Implements 4 flavours of TMLE controlled by the *reduced* and *importance_sampling* parameters. *importance_sampling=True* moves the clever covariate from being a feature to being a sample weight in the targeted regression. 'reduced=True' use a clever covariate vector of 1s and -1s, therefore only good for binary treatment. Otherwise, the clever covariate are the entire IPW matrix and can be used for multiple treatments.

References

- TMLE: Van Der Laan MJ, Rubin D. Targeted maximum likelihood learning. 2006. https://doi.org/10.2202/1557-4679.1043
- TMLE with a vector version of clever covariate: Schuler MS, Rose S. Targeted maximum likelihood estimation for causal inference in observational studies. 2017. https://doi.org/10.1093/aje/kww165
- TMLE with a matrix version of clever covariate: Gruber S, van der Laan M. tmle: An R package for targeted maximum likelihood estimation. 2012. https://doi.org/10.18637/jss.v051.i13
- TMLE with weighted regression and matrix of clever covariate: Gruber S, van der Laan M, Kennedy C. tmle: Targeted Maximum Likelihood Estimation. Cran documentation. https://cran.r-project.org/web/packages/tmle/index.html
- TMLE for continuous outcomes Gruber S, van der Laan MJ. A targeted maximum likelihood estimator of a causal effect on a bounded continuous outcome. 2010. https://doi.org/10.2202/1557-4679.1260

Parameters

- **outcome_model** (*IndividualOutcomeEstimator*) – An initial prediction of the outcome
- **weight_model** (*PropensityEstimator*) – An IPW model predicting the treatment.
- **outcome_covariates** (*array*) – Covariates to use for outcome model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **weight_covariates** (*array*) – Covariates to use for weight model. If None - all covariates passed will be used. Either list of column names or boolean mask.
- **reduced** (*bool*) – If *True* uses a vector version of the clever covariate (rather than a matrix of all treatment values). If *True* enforces a binary treatment assignment.

- **importance_sampling** (*bool*) – If *True* moves the clever covariate from being a feature to being a weight in the regression.
- **glm_fit_kwargs** (*dict*) – Additional kwargs for statsmodels' *GLM.fit()*. Can be used for example for refining the optimizers. see: https://www.statsmodels.org/stable/generated/statsmodels.genmod.generalized_linear_model.GLM.fit.html

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)
Estimates individual outcome under different treatment values (interventions)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None - parameter is ignored and behaviour is as specified when initializing the IndividualOutcomeEstimator.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

fit(*X*, *a*, *y*, *refit_weight_model=True*, ***kwargs*)
Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining scikit-learn's fit method.

Returns A causal weight model with an inner learner fitted.

Return type *IndividualOutcomeEstimator*

class causallib.estimation.tmle.**TargetMinMaxScaler**(*feature_range=(0, 1)*, ***, *copy=True*, *clip=False*)
Bases: sklearn.preprocessing._data.MinMaxScaler

A MinMaxScaler that operates on a vector (Series)

fit(*X*, *y=None*)
Compute the minimum and maximum to be used for later scaling.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – The data used to compute the per-feature minimum and maximum used for later scaling along the features axis.

- **y** (*None*) – Ignored.

Returns **self** – Fitted scaler.

Return type object

inverse_transform(*X*)

Undo the scaling of X according to feature_range.

Parameters **X** (*array-like of shape (n_samples, n_features)*) – Input data that will be transformed. It cannot be sparse.

Returns **Xt** – Transformed data.

Return type ndarray of shape (n_samples, n_features)

transform(*X*)

Scale features of X according to feature_range.

Parameters **X** (*array-like of shape (n_samples, n_features)*) – Input data that will be transformed.

Returns **Xt** – Transformed data.

Return type ndarray of shape (n_samples, n_features)

causallib.estimation.xlearner module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Sep 9, 2021

class causallib.estimation.xlearner.**XLearner**(*outcome_model*, *effect_model=None*, *treatment_model=None*, *predict_proba=True*, *effect_types='diff'*)

Bases: *causallib.estimation.base_estimator.IndividualOutcomeEstimator*

An X-learner model for causal inference (künzel et al. 2018. pnas, https://www.pnas.org/content/116/10/4156). Uses two outcome estimators. The first is used to calculate the response while the second is used invertly to calculate the treatment which is averaged according to the propensity of the treatment assignment.

Parameters

- **outcome_model** (IndividualOutcomeEstimator) – Initialized causallib estimator that will be used to predict the outcome of each treatment given a case and a certain. To adhere

  to the XLearner algorithm a StratifiedStandardization object should be used for both outcome and cate model initialized with comparable sklearn learners. Xlearner algorithm is suitable for a binary outcome, if a non binary outcome will be used the class will view the last outcome versus the rest as the binary outcome.

- **effect_model** (IndividualOutcomeEstimator | None) – Initialized causallib estimator that will be used to predict the treatment effect of each case. The treatment effect is estimated on the observed set using the outcome model if the treatment effect is continuous use a regression model. The default estimator is cloned from the outcome model. The cloning is done after the outcome model is fitted to enable warm start of the cate model by the outcome model if outcome_model has its warm_start attribute on.
- **treatment_model** – Initialized sklearn prediction model that will predict the probability of each treatment. Xlearner algorithm is suitable for binary treatment.
- **predict_proba** (*bool*) –

  In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. Xlearner effect estimation (in the case of binary effect)

  requires the outcome estimator to predict probabilities of classification (predict_proba=True)
- **effect_types** (*str*) – string from the set of EffectEstimator.CALCULATE_EFFECT keys if none the sklearn DummyClassifier with prior strategy will be used.

estimate_effect(*X*, *a*, *agg='population'*, *predict_proba=None*, *effect_types=None*)

Estimates the causal effect between treatment groups.

Parameters

- **X** (*pd.DataFrame*) – Covariates to predict on.
- **a** (*pd.Series*) – Corresponding treatment assignment to utilize for prediction. Assumes treated group is coded as 1, and control group as 0.
- **agg** (*str*) – Either "population" or "individual" - whether to calculate individual effect or population effect.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None, will use the object's initialized predict_proba value
- **effect_types** (*None*) – IGNORED

Returns the estimated causal effect

Return type pd.Series

estimate_individual_outcome(*X*, *a*, *treatment_values=None*, *predict_proba=None*)

Estimates individual outcome under different treatment values (interventions)

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **treatment_values** (*Any*) – Desired treatment value/s to use when estimating the counterfactual outcome/ If not supplied, calculates for all available treatment values.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples,

n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None - parameter is ignored and behaviour is as specified when initializing the IndividualOutcomeEstimator.

Returns

DataFrame which columns are treatment values and rows are individuals: each column is a vector size (num_samples,) that contains the estimated outcome for each individual under the treatment value in the corresponding key.

Return type pd.DataFrame

fit(*X*, *a*, *y*, *sample_weight=None*, *predict_proba=None*)

Trains a causal model from observed data.

Parameters

- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **sample_weight** – To be passed to the underlining outcome model fit method.
- **predict_proba** (*bool | None*) – In case the outcome task is classification and in case *learner* supports the operation, if True - prediction will utilize learner's *predict_proba* or *decision_function* which returns a continuous matrix of size (n_samples, n_classes). If False - *predict* will be used and return value will be based on a vector of class classifications. If None, will use the object's initialized predict_proba value

Returns A causal model with an inner models fitted.

Return type *IndividualOutcomeEstimator*

3.1.3.4.3 Module contents

3.1.3.5 Module causallib.evaluation

This submodule allows evaluating the performance of the estimation models defined in `causallib.estmation`.

The intended usage is to use `evaluate` from `causalib.evaluation` to generate `EvaluationResults` objects. If the cross-validation parameter `cv` is not supplied, a simple evaluation without cross-validation will be performed. And an object will be returned that can generate various plots, accessible by name (see the docs) or all at once via `plot_all()`. The object also includes the model's predictions, evaluated metrics, the fitted models as `models` and a copy of the original data as (`X`, `a`, and `y`).

If the `cv` parameter is set to `"auto"`, `evaluate` generates a k-fold cross-validation with train and validation phases, refitting the model `k` times, with `k=5`. Other options are also supported for customizing cross-validation, see the docs. The `EvaluationResults` will also contain a list of train/test split indices used by cross-validation in `cv`.

3.1.3.5.1 Example: Inverse probability weighting

An IPW method with logistic regression can be evaluated in cross-validation using

```
from sklearn.linear_model import LogisticRegression
from causallib.estimation import IPW
from causallib.datasets.data_loader import fetch_smoking_weight
from causallib.evaluation import evaluate

data = fetch_smoking_weight()

model = LogisticRegression()
ipw = IPW(learner=model)
ipw.fit(data.X, data.a, data.y)
res = evaluate(ipw, data.X, data.a, data.y, cv="auto")

res.plot_all()
```

This will train the models and create evaluation plots showing the performance on both the training and validation data.

```
print(res.all_plot_names)
# {'weight_distribution', 'pr_curve', 'covariate_balance_love', 'roc_curve', 'calibration',
↪'covariate_balance_slope'}
res.plot_covariate_balance(kind="love", phase="valid")
res.plot_weight_distribution()
res.plot_roc_curve()
res.plot_calibration_curve()
```

3.1.3.5.2 Submodule structure

This section is intended for future contributors and those seeking to customize the evaluation logic.

The `evaluate` function is defined in `evaluator.py`. To generate predictions it instantiates a `Predictor` object as defined in `predictor.py`. This handles refitting and generating the necessary predictions for the different models. The predictions objects are defined in `predictions.py`. Metrics are defined in `metrics.py`. These are simple functions and do not depend on the structure of the objects. The metrics are applied to the individual predictions via the scoring functions defined in `scoring.py`. The results of the predictors and scorers across multiple phases and folds are combined in the `EvaluationResults` object which is defined in `results.py`.

evaluation.plots submodule structure

In order to generate the correct plots from the `EvaluationResults` objects, we need `PlotDataExtractor` objects. The responsibility of these objects is to extract the correct data for a given plot from `EvaluationResults`, and they are defined in `plots/data_extractors.py`. Enabling plotting as member functions for `EvaluationResults` objects is accomplished using the plotter mixins, which are defined in `plots/mixins.py`. When an `EvaluationResults` object is produced by evaluate, the `EvaluationResults.make` factory ensures that it has the correct extractors and plotting mixins.

Finally, `plots/curve_data_makers.py` contains a number of methods for aggregating and combining data to produce curves for ROC, PR and calibration plots. And `plots/plots.py` contains the individual plotting functions.

3.1.3.5.3 How to add a new plot

If there is a model evaluation plot that you would like to add to the codebase, you must first determine for what models it would be relevant. For example, a confusion matrix makes sense for a classification task but not for continuous outcome prediction, or sample weight calculation.

Currently, the types of models are

- Individual outcome predictions (continuous outcome)
- Individual outcome predictions (binary outcome)
- Sample weight predictions
- Propensity predictions

Propensity predictions combine binary individual outcome predictions (because "is treated" is a binary feature) with sample weight predictions. Something like a confusion matrix would make sense for binary outcome predictions and for propensity predictions, but not for the other categories. In that sense it would behave like the ROC curve, and PR curve which are already implemented.

Assuming you want to add a new plot, you would add the basic plotting function to `plots/plots.py`. Then you would add a case to the relevant extractors' `get_data_for_plot` members to extract the data for the plot, based on its name, in `plots/data_extractors.py` . You would also add the name as an available plot in the relevant `frozenset` and in the `lookup_name` function, both in `plots/plots.py`. At this point, the plot should be drawn automatically when you run `plot_all` on the relevant `EvaluationResults` object. To expose the plot as a member `plot_my_new_plot`, you must add it to the correct mixin in `plots/mixins.py`.

3.1.3.5.4 Subpackages

causallib.evaluation.plots package

Submodules

causallib.evaluation.plots.curve_data_makers module

Functions that calculate curve data for cross validation plots.

causallib.evaluation.plots.curve_data_makers.**calculate_curve_data_binary_outcome**(*folds_predictions*, *targets*, *curve_metric*, *area_metric*, *stratify_by=None*)

Calculate different performance (ROC or PR) curves

Parameters

- **folds_predictions** (*list[pd.Series]*) – Predictions for each fold.
- **targets** (*pd.Series*) – True labels
- **curve_metric** (*callable*) – Performance metric returning 3 output vectors - metric1, metric2 and thresholds. Where metric1 and metric2 depict the curve when plotted on x-axis and y-axis.
- **area_metric** (*callable*) – Performance metric of the area under the curve.
- **stratify_by** (*pd.Series*) – Group assignment to stratify by.

Returns

Evaluation of the metric for each fold and for each curve. One curve for each group level in *stratify_by*. On general: {curve_name: {metric1: [evaluation_fold_1, ...]}}. For example: {"Treatment=1": {"FPR": [FPR_fold_1, FPR_fold_2, FPR_fold_3]}}

Return type dict[str, dict[str, list[np.ndarray]]]

causallib.evaluation.plots.curve_data_makers.**calculate_curve_data_propensity**(*fold_predictions: List[causallib.evaluation.weight_p*, *targets*, *curve_metric*, *area_metric*)

Calculate different performance (ROC or PR) curves

Parameters

- **fold_predictions** (*list[PropensityEvaluatorPredictions]*) – Predictions for each fold.
- **targets** (*pd.Series*) – True labels
- **curve_metric** (*callable*) – Performance metric returning 3 output vectors - metric1, metric2 and thresholds. Where metric1 and metric2 depict the curve when plotted on x-axis and y-axis.
- **area_metric** (*callable*) – Performance metric of the area under the curve.
- ****kwargs** –

Returns

Evaluation of the metric for each fold and for each curve. 3 curves:

- "unweighted" (regular)
- "weighted" (weighted by inverse propensity)
- "expected" (duplicated population, weighted by propensity)

On general: {curve_name: {metric1: [evaluation_fold_1, ...]}}. For example: {"weighted": {"FPR": [FPR_fold_1, FPR_fold_2, FPR_fold3]}}

Return type dict[str, dict[str, list[np.ndarray]]]

causallib.evaluation.plots.curve_data_makers.**calculate_performance_curve_data_on_folds**(*folds_predictions*, *folds_targets*, *sample_weights=None*, *area_metric=<functi roc_auc_score>*, *curve_metric=<funct roc_curve>*, *pos_label=None*)

Calculates performance curves of the predictions across folds.

Parameters

- **folds_predictions** (*list[pd.Series]*) – Score prediction (as in continuous output of classifier, *predict_proba* or *decision_function*) for every fold.
- **folds_targets** (*list[pd.Series]*) – True labels for every fold.
- **sample_weights** (*list[pd.Series] | None*) – weight for each sample for every fold.

- **area_metric** (*callable*) – Performance metric of the area under the curve.
- **curve_metric** (*callable*) – Performance metric returning 3 output vectors - metric1, metric2 and thresholds. Where metric1 and metric2 depict the curve when plotted on x-axis and y-axis.
- **pos_label** – What label in *targets* is considered the positive label.

Returns For every fold, the calculated metric1 and metric2 (the curves), the thresholds and the area calculations.

Return type (list[np.ndarray], list[np.ndarray], list[np.ndarray], list[float])

causallib.evaluation.plots.curve_data_makers.**calculate_pr_curve**(*curve_data*, *targets*)
Calculates precision-recall curve on the folds.

Parameters

- **curve_data** (*dict*) – dict of curves produced by BaseEvaluationPlotDataExtractor.calculate_curve_data
- **targets** (*pd.Series*) – True labels.

Returns

Keys being "Precision", "Recall" and "AP" (PR metrics) and values are a list the size of number of folds with the evaluation of each fold. Additional "prevalence" key, with positive-label "prevalence" is added to be used by the chance curve.

Return type dict[str, list[np.ndarray]]

causallib.evaluation.plots.curve_data_makers.**calculate_roc_curve**(*curve_data*)
Calculates ROC curve on the folds

Parameters **curve_data** (*dict*) – dict of curves produced by BaseEvaluationPlotDataExtractor.calculate_curve_data

Returns

Keys being "FPR", "TPR" and "AUC" (ROC metrics) and values are a list the size of number of folds with the evaluation of each fold.

Return type dict[str, list[np.ndarray]]

causallib.evaluation.plots.data_extractors module

Plot data extractors.

The responsibility of these classes is to extract the data from the EvaluationResults objects to match the requested plot.

class causallib.evaluation.plots.data_extractors.**BaseEvaluationPlotDataExtractor**(*evaluation_results:* causallib.evaluation.results.Evaluati

Bases: abc.ABC

Extractor to get plot data from EvaluationResults.

Subclasses also have a *plot_names* property.

cv_by_phase(*phase='train'*)
Get the cross-validation indices of all folds for a given phase.

Parameters **phase** (*str, optional*) – Requested phase: "train" or "valid. Defaults to "train".

Returns _description_

Return type List

abstract get_data_for_plot(*plot_name*, *phase='train'*)
Get data for plot with name *plot_name*.

class causallib.evaluation.plots.data_extractors.**BinaryOutcomePlotDataExtractor**(*evaluation_results:* causallib.evaluation.results.Evaluatio

Bases: *causallib.evaluation.plots.data_extractors.BaseEvaluationPlotDataExtractor*

Extractor to get plot data from OutcomeEvaluatorPredictions.

Note that the available plots are different if the outcome predictions are binary/classification or continuous/regression.

get_data_for_plot(*plot_name*, *phase='train'*)
Retrieve the data needed for each provided plot. Plot interfaces are at the plots module.

Parameters **plot_name** (*str*) – Plot name.

Returns Plot data

Return type tuple

plot_names = frozenset({'calibration', 'pr_curve', 'roc_curve'})

class causallib.evaluation.plots.data_extractors.**ContinuousOutcomePlotDataExtractor**(*evaluation_results:* causallib.evaluation.results.Eva

Bases: *causallib.evaluation.plots.data_extractors.BaseEvaluationPlotDataExtractor*

Extractor to get plot data from OutcomeEvaluatorPredictions.

Note that the available plots are different if the outcome predictions are binary/classification or continuous/regression.

get_data_for_plot(*plot_name*, *phase='train'*)
Retrieve the data needed for each provided plot. Plot interfaces are at the plots module.

Parameters **plot_name** (*str*) – Plot name.

Returns Plot data

Return type tuple

plot_names = frozenset({'common_support', 'continuous_accuracy', 'residuals'})

class causallib.evaluation.plots.data_extractors.**PropensityPlotDataExtractor**(*evaluation_results:* causallib.evaluation.results.EvaluationRe

Bases: *causallib.evaluation.plots.data_extractors.WeightPlotDataExtractor*

Extractor to get plot data from PropensityEvaluatorPredictions.

get_data_for_plot(*plot_name*, *phase='train'*)
Retrieve the data needed for each provided plot. Plot interfaces are at the plots.py module.

Parameters

- **plot_name** (*str*) – Plot name.

- **fold_predictions** (*list[PropensityEvaluatorPredictions]*) – Predictions for each fold.
- **list[np.ndarray]** (*cv*) – Indices (in iloc positions) of each fold.

Returns Plot data

Return type tuple

```
plot_names = frozenset({'calibration', 'covariate_balance_love',
'covariate_balance_slope', 'pr_curve', 'roc_curve', 'weight_distribution'})
```

class causallib.evaluation.plots.data_extractors.**WeightPlotDataExtractor**(*evaluation_results:* causallib.evaluation.results.EvaluationResults)

Bases: *causallib.evaluation.plots.data_extractors.BaseEvaluationPlotDataExtractor*

Extractor to get plot data from WeightEvaluatorPredictions.

get_data_for_plot(*plot_name*, *phase='train'*)

Retrieve the data needed for each provided plot.

Plot functions are in plots module.

Parameters **plot_name** (*str*) – Plot name.

Returns Plot data

Return type tuple

```
plot_names = frozenset({'covariate_balance_love', 'covariate_balance_slope',
'weight_distribution'})
```

causallib.evaluation.plots.mixins module

Mixins for plotting.

To work the mixin requires the class to implement *get_data_for_plot* with the supported plot names. See .data_extractors for examples.

class causallib.evaluation.plots.mixins.**ClassificationPlotterMixin**

Bases: object

Mixin to add members to for classification/binary prediction estimation.

This occurs for propensity models (treatment assignment is inherently binary) and for outcome models where the outcome is binary.

Class must implement:

- *get_data_for_plot(plots.ROC_CURVE_PLOT)*
- *get_data_for_plot(plots.PR_CURVE_PLOT)*
- *get_data_for_plot(plots.CALIBRATION_PLOT)*

plot_calibration_curve(*phase='train'*, *n_bins=10*, *plot_se=True*, *plot_rug=False*, *plot_histogram=False*, *quantile=False*, *ax=None*)

Plot calibration curves for multiple models (presumably in folds)

Parameters

- **phase** (*str, optional*) – Phase to plot: “train” or “valid”. Defaults to “train”.
- **n_bins** (*int*) – number of bins to evaluate in the plot

- **plot_se** (*bool*) – Whether to plot standard errors around the mean bin-probability estimation.
- **plot_rug** (*bool*) –
- **plot_histogram** (*bool*) –
- **quantile** (*bool*) – If true, the binning of the calibration curve is by quantiles. Defaults to False.
- **ax** (*matplotlib.axes.Axes, optional*) – axis to plot on, if None creates new axis. Defaults to None.

Note: One of plot_propensity or plot_model must be True.

Returns matplotlib.axes.Axes

plot_pr_curve(*phase='train'*, *plot_folds=False*, *label_folds=False*, *label_std=False*, *ax=None*)
Plot precision-recall (PR) curve.

Parameters

- **phase** (*str, optional*) – Phase to plot: "train" or "valid". Defaults to "train".
- **plot_folds** (*bool, optional*) – Whether to plot individual folds. Defaults to False.
- **label_folds** (*bool, optional*) – Whether to label folds. Defaults to False.
- **label_std** (*bool, optional*) – Whether to label std. Defaults to False.
- **ax** (*matplotlib.axes.Axes, optional*) – axis to plot on, if None creates new axis. Defaults to None.

Returns matplotlib.axes.Axes

plot_roc_curve(*phase='train'*, *plot_folds=False*, *label_folds=False*, *label_std=False*, *ax=None*)
Plot ROC curve.

Parameters

- **phase** (*str, optional*) – Phase to plot: "train" or "valid". Defaults to "train".
- **plot_folds** (*bool, optional*) – Whether to plot individual folds. Defaults to False.
- **label_folds** (*bool, optional*) – Whether to label folds. Defaults to False.
- **label_std** (*bool, optional*) – Whether to label std. Defaults to False.
- **ax** (*matplotlib.axes.Axes, optional*) – axis to plot on, if None creates new axis. Defaults to None.

Returns matplotlib.axes.Axes

class causallib.evaluation.plots.mixins.**ContinuousOutcomePlotterMixin**
Bases: object

Mixin to add members to for continous outcome estimation.

Class must implement:

- *get_data_for_plot(plots.CONTINUOUS_ACCURACY_PLOT)*

- *get_data_for_plot(plots.RESIDUALS_PLOT)*
- *get_data_for_plot(plots.CONTINUOUS_ACCURACY_PLOT)*

plot_common_support(*phase='train'*, *alpha_by_density=True*, *ax=None*)
Plot the scatter plot of y0 vs. y1 for multiple scoring results, colored by the treatment

Parameters

- **alpha_by_density** (*bool*) – Whether to calculate points alpha value (transparent-opaque) with density estimation. This can take some time to compute for a large number of points. If False, alpha calculation will be a simple fast heuristic.
- **ax** (*plt.Axes*) – The axes on which the plot will be displayed. Optional.

plot_continuous_accuracy(*phase='train'*, *alpha_by_density=True*, *plot_residuals=False*, *ax=None*)
Plot continuous accuracy,

Parameters

- **phase** (*str, optional*) – Phase to plot: "train" or "valid". Defaults to "train".
- **alpha_by_density** (*bool, optional*) – Whether to calculate points alpha value (transparent-opaque) with density estimation. This can take some time to compute for a large number of points. If False, alpha calculation will be a simple fast heuristic.
- **plot_residuals** (*bool, optional*) – Whether to plot residuals. Defaults to False.
- **ax** (*matplotlib.axes.Axes, optional*) – axis to plot on, if None creates new axis. Defaults to None.

Returns matplotlib.axes.Axes

plot_residuals(*phase='train'*, *alpha_by_density=True*, *ax=None*)
Plot residuals of predicted outcome vs ground truth.

Parameters

- **phase** (*str, optional*) – Phase to plot: "train" or "valid". Defaults to "train".
- **alpha_by_density** (*bool, optional*) – Whether to calculate points alpha value (transparent-opaque) with density estimation. This can take some time to compute for a large number of points. If False, alpha calculation will be a simple fast heuristic.
- **ax** (*matplotlib.axes.Axes, optional*) – axis to plot on, if None creates new axis. Defaults to None.

Returns matplotlib.axes.Axes

class causallib.evaluation.plots.mixins.**PlotAllMixin**
Bases: object

Mixin to make all the train and validation plots.

Class must implement:

- *all_plot_names*
- *get_data_for_plot(name)* for every name in *all_plot_names*

plot_all(*phase=None*)
Create plot of all available EvaluationResults.

Will create a figure with a subplot for each plot name in *all_plot_names*. If *results* have train and validation data, will create separate "train" and "valid" figures. If a single plot is requested, only that plot is created.

Parameters **phase** (*Union[str, None], optional*) – phase to plot "train" or "valid". If not supplied, defaults to both if available.

Returns

the Axis objects of the plots in a nested dictionary:

- First key is the phase ("train" or "valid")
- Second key is the plot name.

Return type Dict[str, matplotlib.axis.Axis]]

class causallib.evaluation.plots.mixins.**WeightPlotterMixin**

Bases: object

Mixin to add members to for weight estimation plotting.

Class must implement:

- *get_data_for_plot(plots.COVARIATE_BALANCE_GENERIC_PLOT)*
- *get_data_for_plot(plots.WEIGHT_DISTRIBUTION_PLOT)*

plot_covariate_balance(*kind='love'*, *phase='train'*, *ax=None*, *aggregate_folds=True*, *thresh=None*, *plot_semi_grid=True*, *label_imbalanced=True*, ***kwargs*)

Plot covariate balance before and after weighting.

Parameters

- **kind** (*str, optional*) – Plot kind, "love" ,"slope" or "scatter". Defaults to "love".
- **phase** (*str, optional*) – Phase to plot: "train" or "valid". Defaults to "train".
- **ax** (*matplotlib.axes.Axes, optional*) – axis to plot on, if None creates new axis. Defaults to None.
- **aggregate_folds** (*bool, optional*) – Whether to aggregate folds. Defaults to True. Ignored when kind="slope".
- **thresh** (*float, optional*) – Draw threshold line at value. Defaults to None.
- **plot_semi_grid** (*bool, optional*) – Defaults to True. only for kind="love".
- **label_imbalanced** (*bool*) – Label covariates that weren't properly balanced. Ignored when kind="love".

Returns axis with plot

Return type matplotlib.axes.Axes

plot_weight_distribution(*phase='train'*, *reflect=True*, *kde=False*, *cumulative=False*, *norm_hist=True*, *ax=None*)

Plot the distribution of propensity score.

Parameters

- **phase** (*str, optional*) – Phase to plot: "train" or "valid". Defaults to "train".
- **reflect** (*bool*) – Whether to plot treatment groups on opposite sides of the x-axis. This can only work if there are exactly two groups.
- **kde** (*bool*) – Whether to plot kernel density estimation
- **cumulative** (*bool*) – Whether to plot cumulative distribution.
- **norm_hist** (*bool*) – If False - use raw counts on the y-axis. If kde=True, then norm_hist should be True as well.

- **ax** (*matplotlib.axes.Axes, optional*) – axis to plot on, if None creates new axis. Defaults to None.

Returns matplotlib.axes.Axes

causallib.evaluation.plots.plots module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Aug 22, 2018

causallib.evaluation.plots.plots.**calibration_curve**(*y_true*, *y_prob*, *bins=5*)

Compute calibration curve of a classifier given its scores output and true label assignment.

Parameters

- **y_true** (*pd.Series*) – True binary label assignment.
- **y_prob** (*pd.Series*) – Predicted probability of each sample being the positive label.
- **bins** (*int | list | np.ndarray | pd.Series*) – If int, it defines the number of equal-width bins in the given range (5, by default). If bins a sequence, it defines the bin edges, including the rightmost edge, allowing for non-uniform bin widths.

Returns

empirical_prob, predicted_prob, bin_counts empirical_prob: The fraction of positive labels in each bins predicted_prob: The average of predicted probability in each bin bin_counts: The number of samples fallen in each bin

Return type (pd.Series, pd.Series, pd.Series)

References

[1] Zadrozny, B., & Elkan, C. (2002, July). Transforming classifier scores into accurate multiclass probability estimates

causallib.evaluation.plots.plots.**get_subplots**(*n_features*, *max_cols=5*, *fig_size=(16, 16)*, *sharex=False*, *sharey=False*)

Initializes the grid of subplots and returns the axes

Parameters

- **n_features** (*int*) – The total number of features to plot
- **max_cols** (*int*) – The maximal number of figures in each row of figures
- **fig_size** (*tuple[int, int]*) – Passed on to matplotlib
- **sharex** (*str|bool*) – will be passed to subplots
- **sharey** (*str|bool*) – will be passed to subplots

Returns the figure and the array of axes

Return type tuple[Figure, np.ndarray]

causallib.evaluation.plots.plots.**lookup_name**(*name: str*) → Callable

Lookup function for plot name.

Canonical plot names are defined in this file as globals. Incorrect names will raise KeyError.

Parameters **name** (*str*) – plot name to lookup

Returns plot function

Return type Callable

causallib.evaluation.plots.plots.**plot_calibration**(*predictions*, *targets*, *n_bins=10*, *plot_se=True*, *plot_rug=False*, *plot_histogram=True*, *quantile=False*, *ax=None*)

causallib.evaluation.plots.plots.**plot_calibration_folds**(*predictions*, *targets*, *cv*, *n_bins=10*, *plot_se=True*, *plot_rug=False*, *plot_histogram=False*, *quantile=False*, *ax=None*)

Plot calibration curves for multiple models (presumably in folds)

Parameters

- **predictions** (*list[pd.Series]*) – list (each entry of a fold) of arrays - probability ("scores") predictions.
- **targets** (*pd.Series*) – true labels to calibrate against on the overall data (not divided to folds).
- **cv** (*list[np.array]*) –
- **n_bins** (*int*) – number of bins to evaluate in the plot
- **plot_se** (*bool*) – Whether to plot standard errors around the mean bin-probability estimation.
- **plot_rug** –
- **plot_histogram** –
- **quantile** (*bool*) – If true, the binning of the calibration curve is by quantiles. Default is false
- **ax** (*plt.Axes*) – Optional

Note: One of plot_propensity or plot_model must be True.

Returns:

causallib.evaluation.plots.plots.**plot_continuous_prediction_accuracy**(*predictions*, *y*, *a*, *alpha_by_density=True*, *ax=None*)

causallib.evaluation.plots.plots.**plot_continuous_prediction_accuracy_folds**(*predictions*, *y*, *a*, *cv*, *alpha_by_density=True*, *plot_residuals=False*, *ax=None*)

causallib.evaluation.plots.plots.**plot_counterfactual_common_support**(*prediction*, *a*, *ax=None*)

causallib.evaluation.plots.plots.**plot_counterfactual_common_support_folds**(*predictions*, *hue_by*, *cv*, *alpha_by_density=True*, *ax=None*)

Plot the scatter plot of y0 vs. y1 for multiple scoring results, colored by the treatment

Parameters

- **predictions** (*list[pd.Series]*) – List, the size of number of folds, of outcome prediction values.
- **hue_by** (*pd.Series*) – Group assignment (as in treatment assignment) of the entire dataset. (indices from *cv* will be used to slice this vector)
- **cv** (*list[np.array]*) – List, the size of number of folds, of row indices (as in iloc locations) - the indices of samples participating the fold.
- **alpha_by_density** (*bool*) – Whether to calculate points alpha value (transparent-opaque) with density estimation. This can take some time to compute for large number of points. If False, alpha calculation will be a simple fast heuristic.
- **ax** (*plt.Axes*) – The axes on which the plot will be displayed. Optional.

causallib.evaluation.plots.plots.**plot_mean_features_imbalance_love_folds**(*table1_folds*, *cv=None*, *aggregate_folds=True*, *thresh=None*, *plot_semi_grid=True*, *ax=None*)

causallib.evaluation.plots.plots.**plot_mean_features_imbalance_scatter_plot**(*table1_folds*, *aggregate_folds=True*, *thresh=None*, *label_imbalanced=True*, *ax=None*)

causallib.evaluation.plots.plots.**plot_mean_features_imbalance_slope_folds**(*table1_folds*, *cv=None*, *thresh=None*, *label_imbalanced=True*, *ax=None*)

causallib.evaluation.plots.plots.**plot_precision_recall_curve_folds**(*curve_data*, *ax=None*, *plot_folds=False*, *label_folds=False*, *label_std=False*, ***kwargs*)

causallib.evaluation.plots.plots.**plot_propensity_score_distribution**(*propensity*, *treatment*, *reflect=True*, *kde=False*, *cumulative=False*, *norm_hist=True*, *ax=None*)

Plot the distribution of propensity score

Parameters

- **propensity** (*pd.Series*) –
- **treatment** (*pd.Series*) –

- **reflect** (*bool*) – Whether to plot second treatment group on the opposite sides of the x-axis. This can only work if there are exactly two groups.
- **kde** (*bool*) – Whether to plot kernel density estimation
- **cumulative** (*bool*) – Whether to plot cumulative distribution.
- **norm_hist** (*bool*) – If False - use raw counts on the y-axis. If kde=True, then norm_hist should be True as well.
- **ax** (*plt.Axes | None*) –

Returns:

causallib.evaluation.plots.plots.**plot_propensity_score_distribution_folds**(*predictions*, *hue_by*, *cv*, *reflect=True*, *kde=False*, *cumulative=False*, *norm_hist=True*, *ax=None*)

Parameters

- **predictions** (*list[pd.Series]*) –
- **X** (*pd.DataFrame*) –
- **hue_by** (*pd.Series*) –
- **y** (*pd.Series*) –
- **cv** (*list[np.array]*) –
- **reflect** (*bool*) – Whether to plot second treatment group on the opposite sides of the x-axis. This can only work if there are exactly two groups.
- **kde** (*bool*) – Whether to plot kernel density estimation
- **cumulative** (*bool*) – Whether to plot cumulative distribution.
- **norm_hist** (*bool*) – If False - use raw counts on the y-axis. If kde=True, then norm_hist should be True as well.
- **ax** (*plt.Axis*) –

Returns:

causallib.evaluation.plots.plots.**plot_residual**(*predictions*, *y*, *a*, *alpha_by_density=True*, *ax=None*)

causallib.evaluation.plots.plots.**plot_residual_folds**(*predictions*, *y*, *a*, *cv*, *alpha_by_density=True*, *ax=None*)

causallib.evaluation.plots.plots.**plot_roc_curve_folds**(*curve_data*, *ax=None*, *plot_folds=False*, *label_folds=False*, *label_std=False*, ***kwargs*)

causallib.evaluation.plots.plots.**slope_graph**(*left*, *right*, *thresh=None*, *label_imbalanced=True*, *color_below='C0'*, *color_above='C1'*, *marker='o'*, *ax=None*)

Module contents

Causal model evaluation plotting functions.

3.1.3.5.5 Submodules

causallib.evaluation.evaluator module

Methods for evaluating causal inference models.

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Dec 25, 2018

causallib.evaluation.evaluator.**evaluate**(*estimator*, *X*, *a*, *y*, *cv=None*, *metrics_to_evaluate='defaults'*, *plots=False*)

Evaluate model in cross-validation of the provided data

Parameters

- **|** (*estimator (causallib.estimation.base_estimator.IndividualOutcomeEstimator*) – causallib.estimation.base_weight.WeightEstimator | causallib.estimation.base_weight.PropensityEstimator) : an estimator. If using cv, it will be refit, otherwise it should already be fit.
- **X** (*pd.DataFrame*) – Covariates.
- **a** (*pd.Series*) – Treatment assignment.
- **y** (*pd.Series*) – Outcome.
- **cv** (*list[tuples] | generator[tuples] | None*) – list the number of folds containing tuples of indices (train_idx, validation_idx) in an iloc manner (row number). If None, there will be no cross-validation. If *cv="auto"*, a stratified Kfold with 5 folds will be created and used for cross-validation.
- **metrics_to_evaluate** (*dict | "defaults" | None*) – key: metric's name, value: callable that receives true labels, prediction, and sample_weights (the latter may be ignored). If *"defaults"*, default metrics are selected. If *None*, no metrics are evaluated.
- **plots** (*bool*) – whether to generate plots

Returns EvaluationResults

causallib.evaluation.evaluator.**evaluate_bootstrap**(*estimator*, *X*, *a*, *y*, *n_bootstrap*, *n_samples=None*, *replace=True*, *refit=False*, *metrics_to_evaluate=None*)

Evaluate model on a bootstrap sample of the provided data

Parameters

- **X** (*pd.DataFrame*) – Covariates.

- **a** (*pd.Series*) – Treatment assignment.
- **y** (*pd.Series*) – Outcome.
- **n_bootstrap** (*int*) – Number of bootstrap sample to create.
- **n_samples** (*int | None*) – Number of samples to sample in each bootstrap sampling. If None - will use the number samples (first dimension) of the data.
- **replace** (*bool*) – Whether to use sampling with replacements. If False - n_samples (if provided) should be smaller than X.shape[0])
- **refit** (*bool*) – Whether to refit the estimator on each bootstrap sample. Can be computational intensive if n_bootstrap is large.
- **metrics_to_evaluate** (*dict | None*) – key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter is allowed to be ignored). If not provided, default from causallib.evaluation.metrics are used.

Returns EvaluationResults

causallib.evaluation.metrics module

Apply machine learning metrics to causal models for evaluation.

causallib.evaluation.metrics.**evaluate_metrics**(*metrics_to_evaluate*, *y_true*, *y_pred=None*, *y_pred_proba=None*, *sample_weight=None*)

Evaluates the metrics against the supplied predictions and labels.

Note that some metrics operate on proba predictions (*y_pred_proba*) and others on direct predictions. The function will select the correct input based on the name of the metric, if it knows about the metric. Otherwise it defaults to using the direct prediction (*y_pred*).

Parameters

- **metrics_to_evaluate** (*dict*) – key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter is allowed to be ignored).
- **y_true** (*pd.Series*) – True labels
- **y_pred_proba** (*pd.Series*) – continuous output of predictor, as in *predict_proba* or *decision_function*.
- **y_pred** (*pd.Series*) – label (i.e., categories, decisions) predictions.
- **sample_weight** (*pd.Series | None*) – weight of each sample.

Returns name of metric as index and the evaluated score as value.

Return type pd.Series

causallib.evaluation.metrics.**get_default_binary_metrics**(*only_numeric_metric=False*)

Get default metrics for evaluating binary models.

Parameters **only_numeric_metric** (*bool*) – If metrics_to_evaluate not provided and default is used, whether to use only numerical metrics. Ignored if metrics_to_evaluate is provided. Non-numerical metrics are for example roc_curve, that returns vectors and not scalars).

Returns

metrics dict with key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter is allowed to be ignored).

Return type dict [str, callable]

causallib.evaluation.metrics.**get_default_regression_metrics**()
Get default metrics for evaluating continuous prediction models.

Returns

metrics dict with key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter is allowed to be ignored).

Return type dict [str, callable]

causallib.evaluation.predictions module

Predictions from single folds.

Predictions are generated by predictors for causal models. They contain the estimates for single folds and are combined in the EvaluationResults objects for further analysis.

class causallib.evaluation.predictions.**OutcomePredictions**(*prediction*, *prediction_event_prob=None*)

Bases: object

Data structure to hold outcome-model predictions

evaluate_metrics(*a*, *y*, *metrics_to_evaluate*)
Evaluate metrics for this model prediction.

Parameters

- **a** (*pd.Series*) – treatment assignment
- **y** (*pd.Series*) – ground truth outcomes
- **metrics_to_evaluate** (*Dict[str,Callable]*) – key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter may be ignored). If not provided, defaults from causallib.evaluation.metrics are used.

Returns evaluated metrics

Return type pd.DataFrame

get_prediction_by_treatment(*a: pandas.core.series.Series*)
Get proba if available else prediction

get_proba_by_treatment(*a: pandas.core.series.Series*)
Get proba of prediction

class causallib.evaluation.predictions.**PropensityEvaluatorScores**(*prediction_scores*, *covariate_balance*)

Bases: tuple

Create new instance of PropensityEvaluatorScores(prediction_scores, covariate_balance)

covariate_balance
Alias for field number 1

prediction_scores
Alias for field number 0

class causallib.evaluation.predictions.**PropensityPredictions**(*weight_by_treatment_assignment*, *weight_for_being_treated*, *treatment_assignment_pred*, *propensity*, *propensity_by_treatment_assignment*)

Bases: *causallib.evaluation.predictions.WeightPredictions*

Data structure to hold propensity-model predictions

evaluate_metrics(*X*, *a_true*, *metrics_to_evaluate*)

Evaluate metrics on prediction.

Parameters

- **X** (*pd.DataFrame*) – Covariates.
- **a_true** (*pd.Series*) – ground truth treatment assignment
- **metrics_to_evaluate** (*dict | None*) – key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter may be ignored).

Returns

Object with two data attributes: "predictions" and "covariate_balance"

Return type WeightEvaluatorScores

class causallib.evaluation.predictions.**WeightPredictions**(*weight_by_treatment_assignment*, *weight_for_being_treated*)

Bases: object

Data structure to hold weight-model predictions

evaluate_metrics(*X*, *a_true*, *metrics_to_evaluate*)

Evaluate covariate balancing of the weight model

Parameters

- **X** (*pd.DataFrame*) – Covariates.
- **a_true** (*pd.Series*) – ground truth treatment assignment
- **metrics_to_evaluate** (*dict | None*) – IGNORED.

Returns a *covariate_balance* dataframe

Return type pd.DataFrame

causallib.evaluation.predictor module

Predictor classes.

Predictors generate sets of predictions for a single fold with no cross-validation or train-test logic.

class causallib.evaluation.predictor.**BasePredictor**(*estimator*)

Bases: object

Generate predictions from estimator for evaluation (base class).

abstract fit(*X*, *a*, *y*)

Fit an estimator.

static from_estimator(*estimator: Union[causallib.estimation.base_estimator.IndividualOutcomeEstimator, causallib.estimation.base_weight.PropensityEstimator, causallib.estimation.base_weight.WeightEstimator]*)

Select subclass based on estimator.

Parameters **estimator** (*Union[IndividualOutcomeEstimator, PropensityEstimator, WeightEstimator]*) – Estimator to generate evaluation predictions from.

Returns

the correct predictor for the supplied estimator

Return type Union[*PropensityPredictor*, *WeightPredictor*, *OutcomePredictor*]

abstract predict(*X*, *a*)

Predict (weights, outcomes, etc. depending on the model). The output can be as flexible as desired, but score_estimation should know to handle it.

class causallib.evaluation.predictor.**OutcomePredictor**(*estimator*)

Bases: *causallib.evaluation.predictor.BasePredictor*

Generate evaluation predictions for IndividualOutcomeEstimator models.

Parameters **estimator** (IndividualOutcomeEstimator) –

fit(*X*, *a*, *y*)

Fit estimator.

predict(*X*, *a*)

Predict on data.

class causallib.evaluation.predictor.**PropensityPredictor**(*estimator*)

Bases: *causallib.evaluation.predictor.WeightPredictor*

Generate evaluation predictions for PropensityEstimator models.

Parameters **estimator** (PropensityEstimator) –

predict(*X*, *a*)

Predict on data.

Parameters

- **X** (*pd.DataFrame*) – Covariates.
- **a** (*pd.Series*) – Target variable - treatment assignment

Returns PropensityEvaluatorPredictions

class causallib.evaluation.predictor.**WeightPredictor**(*estimator*)

Bases: *causallib.evaluation.predictor.BasePredictor*

Generate evaluation predictions for WeightEstimator models.

Parameters **estimator** (WeightEstimator) –

fit(*X*, *a*, *y=None*)

Fit estimator. *y* is ignored.

predict(*X*, *a*)

Predict on data.

Parameters

- **X** (*pd.DataFrame*) – Covariates.
- **a** (*pd.Series*) – Target variable - treatment assignment

Returns WeightEvaluatorPredictions

causallib.evaluation.predictor.**predict_cv**(*estimator*, *X*, *a*, *y*, *cv*, *refit=True*, *phases=('train', 'valid')*)
Obtain predictions on the provided data in cross-validation

Parameters

- **X** (*pd.DataFrame*) – Covariates.
- **a** (*pd.Series*) – Treatment assignment.
- **y** (*pd.Series*) – Outcome.
- **cv** (*list[tuples]*) – list the number of folds containing tuples of indices (train_idx, validation_idx)
- **refit** (*bool*) – Whether to refit the model on each fold.
- **phases** (*list[str]*) – {["train", "valid"], ["train"], ["valid"]}. Phases names to evaluate on - train ("train"), validation ("valid") or both. 'train' corresponds to cv[i][0] and 'valid' to cv[i][1]

Returns

A two-tuple containing:

- **predictions: dictionary with keys being the phases provided and values are** list the size of the number of folds in cv and containing the output of the estimator on that corresponding fold. For example, predictions["valid"][3] contains the prediction of the estimator on untrained data of the third fold (i.e. validation set of the third fold)
- **models: list the size of the number of folds in cv containing the fitted estimator** on the training data of that fold.

Return type (dict[str, list], list)

causallib.evaluation.results module

Evaluation results objects for plotting and further analysis.

These objects are generated by the *evaluate* method.

class causallib.evaluation.results.**BinaryOutcomeEvaluationResults**(*evaluated_metrics: Union[pandas.core.frame.DataFrame, causallib.evaluation.predictions.PropensityEvaluatorSc models: Union[List[causallib.estimation.base_weight.We List[causallib.estimation.base_estimator.Individu List[causallib.estimation.base_weight.Propensity predictions: Dict[str, List[Union[causallib.evaluation.predictions.Prop causallib.evaluation.predictions.WeightPredictions, causallib.evaluation.predictions.OutcomePredictions]]] cv: List[Tuple[List[int], List[int]]], X: pandas.core.frame.DataFrame, a: pandas.core.series.Series, y: pandas.core.series.Series*)

Bases: *causallib.evaluation.results.EvaluationResults*, *causallib.evaluation.plots.mixins.ClassificationPlotterMixin*, *causallib.evaluation.plots.mixins.PlotAllMixin*

Data structure to hold evaluation results including cross-validation.

Attrs: evaluated_metrics (Union[pd.DataFrame, PropensityEvaluatorScores, None]): models (dict[str, Union[list[WeightEstimator], list[IndividualOutcomeEstimator]):

Models trained during evaluation. May be dict or list or a model directly.

predictions (dict[str, List[SingleFoldPredictions]]): dict with keys "train" and "valid" (if produced through cross-validation) and values of the predictions for the respective fold

cv (list[tuple[list[int], list[int]]]): the cross validation indices, used to generate the results, used for constructing plots correctly

X (pd.DataFrame): features data a (pd.Series): treatment assignment data y (pd.Series): outcome data

X: pandas.core.frame.DataFrame

a: pandas.core.series.Series

cv: List[Tuple[List[int], List[int]]]

evaluated_metrics: Union[pandas.core.frame.DataFrame, *causallib.evaluation.predictions.PropensityEvaluatorScores*]

models: Union[List[*causallib.estimation.base_weight.WeightEstimator*], List[*causallib.estimation.base_estimator.IndividualOutcomeEstimator*], List[*causallib.estimation.base_weight.PropensityEstimator*]]

predictions: Dict[str, List[Union[*causallib.evaluation.predictions.PropensityPredictions*, *causallib.evaluation.predictions.WeightPredictions*, *causallib.evaluation.predictions.OutcomePredictions*]]]

y: pandas.core.series.Series

class causallib.evaluation.results.**ContinuousOutcomeEvaluationResults**(*evaluated_metrics: Union[pandas.core.frame.DataFrame, causallib.evaluation.predictions.PropensityEvalua models: Union[List[causallib.estimation.base_weigh List[causallib.estimation.base_estimator.Ind List[causallib.estimation.base_weight.Prope predictions: Dict[str, List[Union[causallib.evaluation.predictions. causallib.evaluation.predictions.WeightPredictions causallib.evaluation.predictions.OutcomePredictio cv: List[Tuple[List[int], List[int]]], X: pandas.core.frame.DataFrame, a: pandas.core.series.Series, y: pandas.core.series.Series*)

Bases: *causallib.evaluation.results.EvaluationResults*, *causallib.evaluation.plots.mixins.ContinuousOutcomePlotterMixin*, *causallib.evaluation.plots.mixins.PlotAllMixin*

Data structure to hold evaluation results including cross-validation.

Attrs: evaluated_metrics (Union[pd.DataFrame, PropensityEvaluatorScores, None]): models (dict[str, Union[list[WeightEstimator], list[IndividualOutcomeEstimator]):

Models trained during evaluation. May be dict or list or a model directly.

predictions (dict[str, List[SingleFoldPredictions]]): dict with keys "train" and "valid" (if produced through cross-validation) and values of the predictions for the respective fold

cv (list[tuple[list[int], list[int]]]): the cross validation indices, used to generate the results, used for constructing plots correctly

X (pd.DataFrame): features data a (pd.Series): treatment assignment data y (pd.Series): outcome data

X: pandas.core.frame.DataFrame

a: pandas.core.series.Series

cv: List[Tuple[List[int], List[int]]]

evaluated_metrics: Union[pandas.core.frame.DataFrame, *causallib.evaluation.predictions.PropensityEvaluatorScores*]

models: Union[List[*causallib.estimation.base_weight.WeightEstimator*], List[*causallib.estimation.base_estimator.IndividualOutcomeEstimator*], List[*causallib.estimation.base_weight.PropensityEstimator*]]

predictions: Dict[str, List[Union[*causallib.evaluation.predictions.PropensityPredictions*, *causallib.evaluation.predictions.WeightPredictions*, *causallib.evaluation.predictions.OutcomePredictions*]]]

y: pandas.core.series.Series

class causallib.evaluation.results.**EvaluationResults**(*evaluated_metrics: Union[pandas.core.frame.DataFrame, causallib.evaluation.predictions.PropensityEvaluatorScores], models: Union[List[causallib.estimation.base_weight.WeightEstimator], List[causallib.estimation.base_estimator.IndividualOutcomeEstim List[causallib.estimation.base_weight.PropensityEstimator]], predictions: Dict[str, List[Union[causallib.evaluation.predictions.PropensityPredictions causallib.evaluation.predictions.WeightPredictions, causallib.evaluation.predictions.OutcomePredictions]]], cv: List[Tuple[List[int], List[int]]], X: pandas.core.frame.DataFrame, a: pandas.core.series.Series, y: pandas.core.series.Series*)

Bases: abc.ABC

Data structure to hold evaluation results including cross-validation.

Attrs: evaluated_metrics (Union[pd.DataFrame, PropensityEvaluatorScores, None]): models (dict[str, Union[list[WeightEstimator], list[IndividualOutcomeEstimator]):

Models trained during evaluation. May be dict or list or a model directly.

predictions (dict[str, List[SingleFoldPredictions]]): dict with keys "train" and "valid" (if produced through cross-validation) and values of the predictions for the respective fold

cv (list[tuple[list[int], list[int]]]): the cross validation indices, used to generate the results, used for constructing plots correctly

X (pd.DataFrame): features data a (pd.Series): treatment assignment data y (pd.Series): outcome data

X: pandas.core.frame.DataFrame

a: pandas.core.series.Series

property **all_plot_names**

Available plot names.

Returns string names of supported plot names for these results

Return type set[str]

cv: List[Tuple[List[int], List[int]]]

evaluated_metrics: Union[pandas.core.frame.DataFrame, *causallib.evaluation.predictions.PropensityEvaluatorScores*]

get_data_for_plot(*plot_name*, *phase='train'*)

Get data for a given plot

Parameters

- **plot_name** (*str*) – plot name from *self.all_plot_names*
- **phase** (*str, optional*) – phase of interest. Defaults to "train".

Returns the data required for the plot in question

Return type Any

static make(*evaluated_metrics: Union[pandas.core.frame.DataFrame, causallib.evaluation.predictions.PropensityEvaluatorScores], models: Union[List[causallib.estimation.base_weight.WeightEstimator], List[causallib.estimation.base_estimator.IndividualOutcomeEstimator], List[causallib.estimation.base_weight.PropensityEstimator]], predictions: Dict[str, List[Union[causallib.evaluation.predictions.PropensityPredictions, causallib.evaluation.predictions.WeightPredictions, causallib.evaluation.predictions.OutcomePredictions]]], cv: List[Tuple[List[int], List[int]]], X: pandas.core.frame.DataFrame, a: pandas.core.series.Series, y: pandas.core.series.Series*)

Make EvaluationResults object of correct type.

This is a factory method to dispatch the initializing data to the correct subclass of EvaluationResults. This is the only supported way to instantiate EvaluationResults objects.

Parameters

- **evaluated_metrics** (*Union[pd.DataFrame, WeightEvaluatorScores]*) – evaluated metrics
- **(Union[** (*models*) – List[WeightEstimator], List[IndividualOutcomeEstimator], List[PropensityEstimator],]): fitted models
- **predictions** (*Dict[str, List[SingleFoldPrediction]]*) – predictions by phase and fold
- **cv** (*List[Tuple[List[int], List[int]]]*) – cross validation indices
- **X** (*pd.DataFrame*) – features data
- **a** (*pd.Series*) – treatment assignment data
- **y** (*pd.Series*) – outcome data

Raises **ValueError** – raised if invalid estimator is passed

Returns object with results of correct type

Return type *EvaluationResults*

models: Union[List[*causallib.estimation.base_weight.WeightEstimator*], List[*causallib.estimation.base_estimator.IndividualOutcomeEstimator*], List[*causallib.estimation.base_weight.PropensityEstimator*]]

predictions: Dict[str, List[Union[*causallib.evaluation.predictions.PropensityPredictions*, *causallib.evaluation.predictions.WeightPredictions*, *causallib.evaluation.predictions.OutcomePredictions*]]]

remove_spurious_cv()

Remove redundant information accumulated due to the use of cross-validation process.

y: pandas.core.series.Series

class causallib.evaluation.results.**PropensityEvaluationResults**(*evaluated_metrics: Union[pandas.core.frame.DataFrame, causallib.evaluation.predictions.PropensityEvaluatorScores*, *models: Union[List[causallib.estimation.base_weight.WeightE*, *List[causallib.estimation.base_estimator.IndividualO*, *List[causallib.estimation.base_weight.PropensityEsti*, *predictions: Dict[str, List[Union[causallib.evaluation.predictions.Propensi causallib.evaluation.predictions.WeightPredictions, causallib.evaluation.predictions.OutcomePredictions]]]*, *cv: List[Tuple[List[int], List[int]]]*, *X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *y: pandas.core.series.Series*)

Bases: *causallib.evaluation.results.EvaluationResults*, *causallib.evaluation.plots.mixins.ClassificationPlotterMixin*, *causallib.evaluation.plots.mixins.WeightPlotterMixin*, *causallib.evaluation.plots.mixins.PlotAllMixin*

Data structure to hold evaluation results including cross-validation.

Attrs: evaluated_metrics (Union[pd.DataFrame, PropensityEvaluatorScores, None]): models (dict[str, Union[list[WeightEstimator], list[IndividualOutcomeEstimator]):

Models trained during evaluation. May be dict or list or a model directly.

predictions (dict[str, List[SingleFoldPredictions]]): dict with keys "train" and "valid" (if produced through cross-validation) and values of the predictions for the respective fold

cv (list[tuple[list[int], list[int]]]): the cross validation indices, used to generate the results, used for constructing plots correctly

X (pd.DataFrame): features data a (pd.Series): treatment assignment data y (pd.Series): outcome data

X: pandas.core.frame.DataFrame

a: pandas.core.series.Series

cv: List[Tuple[List[int], List[int]]]

evaluated_metrics: Union[pandas.core.frame.DataFrame, *causallib.evaluation.predictions.PropensityEvaluatorScores*]

models: Union[List[*causallib.estimation.base_weight.WeightEstimator*], List[*causallib.estimation.base_estimator.IndividualOutcomeEstimator*], List[*causallib.estimation.base_weight.PropensityEstimator*]]

predictions: Dict[str, List[Union[*causallib.evaluation.predictions.PropensityPredictions*, *causallib.evaluation.predictions.WeightPredictions*, *causallib.evaluation.predictions.OutcomePredictions*]]]

y: pandas.core.series.Series

class causallib.evaluation.results.**WeightEvaluationResults**(*evaluated_metrics: Union[pandas.core.frame.DataFrame, causallib.evaluation.predictions.PropensityEvaluatorScores], models: Union[List[causallib.estimation.base_weight.WeightEstim List[causallib.estimation.base_estimator.IndividualOutcon List[causallib.estimation.base_weight.PropensityEstimato predictions: Dict[str, List[Union[causallib.evaluation.predictions.PropensityPre causallib.evaluation.predictions.WeightPredictions, causallib.evaluation.predictions.OutcomePredictions]]], cv: List[Tuple[List[int], List[int]]], X: pandas.core.frame.DataFrame, a: pandas.core.series.Series, y: pandas.core.series.Series*)

Bases: *causallib.evaluation.results.EvaluationResults*, *causallib.evaluation.plots.mixins.WeightPlotterMixin*, *causallib.evaluation.plots.mixins.PlotAllMixin*

Data structure to hold evaluation results including cross-validation.

Attrs: evaluated_metrics (Union[pd.DataFrame, PropensityEvaluatorScores, None]): models (dict[str, Union[list[WeightEstimator], list[IndividualOutcomeEstimator]):

Models trained during evaluation. May be dict or list or a model directly.

predictions (dict[str, List[SingleFoldPredictions]]): dict with keys "train" and "valid" (if produced through cross-validation) and values of the predictions for the respective fold

cv (list[tuple[list[int], list[int]]]): the cross validation indices, used to generate the results, used for constructing plots correctly

X (pd.DataFrame): features data a (pd.Series): treatment assignment data y (pd.Series): outcome data

X: pandas.core.frame.DataFrame

a: pandas.core.series.Series

cv: List[Tuple[List[int], List[int]]]

evaluated_metrics: Union[pandas.core.frame.DataFrame, *causallib.evaluation.predictions.PropensityEvaluatorScores*]

models: Union[List[*causallib.estimation.base_weight.WeightEstimator*], List[*causallib.estimation.base_estimator.IndividualOutcomeEstimator*], List[*causallib.estimation.base_weight.PropensityEstimator*]]

predictions: Dict[str, List[Union[*causallib.evaluation.predictions.PropensityPredictions*, *causallib.evaluation.predictions.WeightPredictions*, *causallib.evaluation.predictions.OutcomePredictions*]]]

y: pandas.core.series.Series

causallib.evaluation.scoring module

Scoring functions that operate on the evaluation results objects.

These functions depend on the causallib.evalutation results objects and are less reusable than the functions in metrics.py.

causallib.evaluation.scoring.**score_cv**(*predictions*, *X*, *a*, *y*, *cv*, *metrics_to_evaluate='defaults'*)
Evaluate the prediction against the true data using evaluation score metrics.

Parameters

- **predictions** (*dict[str, list]*) – the output of predict_cv.
- **X** (*pd.DataFrame*) – Covariates.
- **a** (*pd.Series*) – Treatment assignment.
- **y** (*pd.Series*) – Outcome.
- **cv** (*list[tuples]*) – list the number of folds containing tuples of indices: (train_idx, validation_idx)
- **metrics_to_evaluate** (*dict | "defaults"*) – key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter is allowed to be ignored). If *"defaults"*, default metrics are selected.

Returns DataFrame whose columns are different metrics and each row is a product of phase x fold x strata. PropensityEvaluatorScores also has a covariate-balance result in a DataFrame.

Return type pd.DataFrame | WeightEvaluatorScores

causallib.evaluation.scoring.**score_estimation**(*prediction*, *X*, *a_true*, *y_true*, *metrics_to_evaluate=None*)
Should know how to handle the _estimator_predict output provided in *prediction*. Can utilize any of the true values provided:

covariates *X*, treatment assignment *a* or outcome *y*.

3.1.3.5.6 Module contents

Objects and methods to evaluate accuracy of causal models.

causallib.evaluation.**evaluate**(*estimator*, *X*, *a*, *y*, *cv=None*, *metrics_to_evaluate='defaults'*, *plots=False*)
Evaluate model in cross-validation of the provided data

Parameters

- **|** (*estimator (causallib.estimation.base_estimator.IndividualOutcomeEstimator*) – causallib.estimation.base_weight.WeightEstimator | causallib.estimation.base_weight.PropensityEstimator) : an estimator. If using cv, it will be refit, otherwise it should already be fit.
- **X** (*pd.DataFrame*) – Covariates.
- **a** (*pd.Series*) – Treatment assignment.
- **y** (*pd.Series*) – Outcome.
- **cv** (*list[tuples] | generator[tuples] | None*) – list the number of folds containing tuples of indices (train_idx, validation_idx) in an iloc manner (row number). If None, there will be no cross-validation. If *cv="auto"*, a stratified Kfold with 5 folds will be created and used for cross-validation.

- **metrics_to_evaluate** (*dict | "defaults" | None*) – key: metric's name, value: callable that receives true labels, prediction, and sample_weights (the latter may be ignored). If *"defaults"*, default metrics are selected. If *None*, no metrics are evaluated.
- **plots** (*bool*) – whether to generate plots

Returns EvaluationResults

causallib.evaluation.**evaluate_bootstrap**(*estimator*, *X*, *a*, *y*, *n_bootstrap*, *n_samples=None*, *replace=True*, *refit=False*, *metrics_to_evaluate=None*)

Evaluate model on a bootstrap sample of the provided data

Parameters

- **X** (*pd.DataFrame*) – Covariates.
- **a** (*pd.Series*) – Treatment assignment.
- **y** (*pd.Series*) – Outcome.
- **n_bootstrap** (*int*) – Number of bootstrap sample to create.
- **n_samples** (*int | None*) – Number of samples to sample in each bootstrap sampling. If None - will use the number samples (first dimension) of the data.
- **replace** (*bool*) – Whether to use sampling with replacements. If False - n_samples (if provided) should be smaller than X.shape[0])
- **refit** (*bool*) – Whether to refit the estimator on each bootstrap sample. Can be computational intensive if n_bootstrap is large.
- **metrics_to_evaluate** (*dict | None*) – key: metric's name, value: callable that receives true labels, prediction and sample_weights (the latter is allowed to be ignored). If not provided, default from causallib.evaluation.metrics are used.

Returns EvaluationResults

3.1.3.6 Module preprocessing

This module provides several useful filters and transformers to augment the ones provided by scikit-learn.

Specifically, the various filters remove features for the following criteria:

- Features that are almost constant (not by variance but by actual value).
- Features that are highly correlated with other features.
- Features that have a low variance (can deal with NaN values).
- Features that are mostly NaN.
- Features that are highly associated with the outcome (not just correlation)

Various transformers are provided:

- A standard scaler that deals with Nan values.
- A min/max scaler.

A transformer that accepts numpy arrays and turns them into pandas will be added soon.

These filters and transformers can be used as part of a scikit-learn pipeline.

3.1.3.6.1 Example:

This example combines a scikit-learn filter with a causallib scaler. The pipeline scales the data, then removes covariates with low variance, and then applies IPW with logistic regression.

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import VarianceThreshold
from sklearn.pipeline import make_pipeline
from causallib.estimation import IPW
from causallib.datasets import load_nhefs
from causallib.preprocessing.transformers import MinMaxScaler

pipeline = make_pipeline(MinMaxScaler(), VarianceThreshold(0.1), LogisticRegression())
data = load_nhefs()
ipw = IPW(pipeline)
ipw.fit(data.X, data.a)
ipw.estimate_population_outcome(data.X, data.a, data.y)
```

Submodules

causallib.preprocessing.confounder_selection module

class causallib.preprocessing.confounder_selection.**DoubleLASSO**(*treatment_lasso=None*, *outcome_lasso=None*, *mask_fn=None*, *threshold=1e-06*, *importance_getter='auto'*, *covariates=None*)

Bases: causallib.preprocessing.confounder_selection._BaseConfounderSelection

A method for selecting confounders using sparse regression on both the treatment and the outcomes, and select for

Implementing "Inference on Treatment Effects after Selection among High-Dimensional Controls" https://academic.oup.com/restud/article/81/2/608/1523757

Parameters

- **treatment_lasso** – Lasso learner to fit confounders and treatment. For example using scikit-learn, continuous treatment may use: *Lasso()*, discrete treatment may use: *LogisticRegression(penalty='l1')*. If *None* will try to automatically assign a lasso model with cross validation.
- **outcome_lasso** – Lasso learner to fit confounders and outcome. For example using scikit-learn, continuous outcome may use: *Lasso()*, discrete outcome may use: *LogisticRegression(penalty='l1')*. If *None* will try to automatically assign lasso model cross-validation.
- **mask_fn** – Function that takes input as two fitted lasso learners and returns a mask of the length of number of columns where True corresponds to columns that need to be selected. When set to None, the default implementation returns a mask based on non-zero coefficients in either learner. User can supply their own function, which must return a boolean array (of the length of columns of X) to indicate which columns are to be included.
- **threshold** – For default mask_fn, absolute value below which a lasso coefficient is treated as zero.

- **importance_getter** (*str | callable*) – how to obtain feature importance. either a callable that inputs an estimator, a string of *'coef_'* or *'feature_importance_'*, or *'auto'* will detect *'coef_'* or *'feature_importance_'* automatically.
- **covariates** (*list | np.ndarray*) – Specifying a subset of columns to perform selection on. Columns in *X* but not in *covariates* will be included after *transform* no matter the selection. Can be either a list of column names, or an array of boolean indicators length of *X*, or anything compatible with pandas *loc* function for columns. if *None* then all columns are participating in the selection process. This is similar to using sklearn's *ColumnTransformer* or *make_column_selector*.

fit(*X*, **args*, ***kwargs*)

class causallib.preprocessing.confounder_selection.**RecursiveConfounderElimination**(*estimator*, *n_features_to_select: int = 1*, *step: int = 1*, *importance_getter='auto'*, *covariates=None*)

Bases: causallib.preprocessing.confounder_selection._BaseConfounderSelection

Recursively eliminate confounders to prune confounders.

Parameters

- **estimator** – Estimator to fit for every step of recursive elimination.
- **n_features_to_select** (*int*) – The number of confounders to keep.
- **step** (*int*) – The number of confounders to eliminate in one iteration.
- **importance_getter** (*str | callable*) – how to obtain feature importance. either a callable that inputs an estimator, a string of *'coef_'* or *'feature_importance_'*, or *'auto'* will detect *'coef_'* or *'feature_importance_'* automatically.
- **covariates** (*list | np.ndarray*) – Specifying a subset of columns to perform selection on. Columns in *X* but not in *covariates* will be included after *transform* no matter the selection. Can be either a list of column names, or an array of boolean indicators length of *X*, or anything compatible with pandas *loc* function for columns. if *None* then all columns are participating in the selection process. This is similar to using sklearn's *ColumnTransformer* or *make_column_selector*.

fit(*X*, **args*, ***kwargs*)

causallib.preprocessing.filters module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class causallib.preprocessing.filters.**BaseFeatureSelector**
Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

abstract fit(*X*, *y=None*)

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used for filtering.
- **y** – Passthrough for Pipeline compatibility.

Returns BaseFeatureSelector

property selected_features

transform(*X*)

Parameters **X** (*pd.DataFrame*) –

Return type pd.DataFrame

class causallib.preprocessing.filters.**ConstantFilter**(*threshold=0.95*)
Bases: *causallib.preprocessing.filters.BaseFeatureSelector*

Removes features that are almost constant

Parameters **threshold** (*float*) –

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

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used for filtering.
- **y** – Passthrough for Pipeline compatibility.

Returns BaseFeatureSelector

class causallib.preprocessing.filters.**CorrelationFilter**(*threshold=0.9*)
Bases: *causallib.preprocessing.filters.BaseFeatureSelector*

Removes features that are strongly correlated to other features

Parameters **threshold** (*float*) –

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

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used for filtering.
- **y** – Passthrough for Pipeline compatibility.

Returns BaseFeatureSelector

class causallib.preprocessing.filters.**HrlVarFilter**(*threshold=0.0*)
Bases: *causallib.preprocessing.filters.BaseFeatureSelector*

Removes features with a small variance, while allowing for missing values

Parameters **threshold** (*float*) –

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

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used for filtering.
- **y** – Passthrough for `Pipeline` compatibility.

Returns BaseFeatureSelector

class causallib.preprocessing.filters.**SparseFilter**(*threshold=0.2*)

Bases: *causallib.preprocessing.filters.BaseFeatureSelector*

Removes features with many missing values

Parameters **threshold** (*float*) –

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

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used for filtering.
- **y** – Passthrough for `Pipeline` compatibility.

Returns BaseFeatureSelector

class causallib.preprocessing.filters.**StatisticalFilter**(*threshold=0.2*, *isLinear=True*)

Bases: *causallib.preprocessing.filters.BaseFeatureSelector*

Removes features according to univariate association

Parameters

- **isLinear** (*bool*) –
- **threshold** (*float*) –

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

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used for filtering.
- **y** – Passthrough for `Pipeline` compatibility.

Returns BaseFeatureSelector

class causallib.preprocessing.filters.**UnivariateAssociationFilter**(*is_linear=True*, *threshold=0.2*)

Bases: *causallib.preprocessing.filters.BaseFeatureSelector*

Removes features according to univariate association

Parameters

- **is_linear** (*bool*) –
- **threshold** (*float*) –

compute_pvals(*X*, *y*)

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

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used for filtering.
- **y** – Passthrough for `Pipeline` compatibility.

Returns BaseFeatureSelector

causallib.preprocessing.filters.**track_selected_features**(*pipeline_stages*, *num_features*)

Parameters

- **pipeline_stages** (*list [tuple[str, TransformerMixin]]*) – list of steps. each step is a tuple of Name and Transformer Object.
- **num_features** (*int*) –

Return type np.ndarray

causallib.preprocessing.transformers module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class causallib.preprocessing.transformers.**Imputer**(*, *missing_values=nan*, *strategy='mean'*, *fill_value=None*, *verbose='deprecated'*, *copy=True*, *add_indicator=False*)

Bases: `sklearn.impute._base.SimpleImputer`

transform(*X*)

Impute all missing values in *X*.

Parameters **X** (*{array-like, sparse matrix}, shape (n_samples, n_features)*) – The input data to complete.

Returns **X_imputed** – *X* with imputed values.

Return type {ndarray, sparse matrix} of shape (n_samples, n_features_out)

class causallib.preprocessing.transformers.**MatchingTransformer**(*propensity_transform=None*, *caliper=None*, *with_replacement=True*, *n_neighbors=1*, *matching_mode='both'*, *metric='mahalanobis'*, *knn_backend='sklearn'*)

Bases: `object`

Transform data by removing poorly matched samples.

Parameters

- **propensity_transform** (*causallib.transformers.PropensityTransformer*) – an object for data preprocessing which adds the propensity score as a feature (default: None)
- **caliper** (*float*) – maximal distance for a match to be accepted. If not defined, all matches will be accepted. If defined, some samples may not be matched and their outcomes will not be estimated. (default: None)
- **with_replacement** (*bool*) – whether samples can be used multiple times for matching. If set to False, the matching process will optimize the linear sum of distances between pairs of treatment and control samples and only *min(N_treatment, N_control)* samples will be estimated. Matching with no replacement does not make use of the *fit* data and is therefore not implemented for out-of-sample data (default: True)
- **n_neighbors** (*int*) – number of nearest neighbors to include in match. Must be 1 if *with_replacement* is *False*. If larger than 1, the estimate is calculated using the *regress_agg_function* or *classify_agg_function* across the *n_neighbors*. Note that when the *caliper* variable is set, some samples will have fewer than *n_neighbors* matches. (default: 1).
- **matching_mode** (*str*) – Direction of matching: *treatment_to_control*, *control_to_treatment* or *both* to indicate which set should be matched to which. All sets are cross-matched in *match* and when *with_replacement* is *False* all matching modes coincide. With replacement there is a difference.
- **metric** (*str*) – Distance metric string for calculating distance between samples. Note: if an external built *knn_backend* object with a different metric is supplied, *metric* needs to be changed to reflect that, because *Matching* will set its inverse covariance matrix if "mahalanobis" is set. (default: "mahalanobis", also supported: "euclidean")
- **knn_backend** (*str or callable*) – Backend to use for nearest neighbor search. Options are "sklearn" or a callable which returns an object implementing *fit*, *kneighbors* and *set_params* like the sklearn *NearestNeighbors* object. (default: "sklearn").

find_indices_of_matched_samples(*X*, *a*)

Find indices of samples which matched successfully.

Given a DataFrame of samples *X* and treatment assignments *a*, return a list of indices of samples which matched successfully.

Parameters

- **X** (*pd.DataFrame*) – Covariates of samples
- **a** (*pd.Series*) – Treatment assignments

Returns indices of matched samples to be passed to *X.loc*

Return type pd.Series

fit(*X*, *a*, *y*)

Fit data to transform

This function loads the data for matching and must be called before *transform*. For convenience, consider using *fit_transform*.

Parameters

- **X** (*pd.DataFrame*) – DataFrame of shape (n,m) containing m covariates for n samples.

- **a** (*pd.Series*) – Series of shape (n,) containing discrete treatment values for the n samples.
- **y** (*pd.Series*) – Series of shape (n,) containing outcomes for the n samples.

Returns Fitted object

Return type self (*MatchingTransformer*)

fit_transform(*X*, *a*, *y*)

Match data and return matched subset.

This is a convenience method, calling *fit* and *transform* at once. For details, see documentation of each function.

Parameters

- **X** (*pd.DataFrame*) – DataFrame of shape (n,m) containing m covariates for n samples.
- **a** (*pd.Series*) – Series of shape (n,) containing discrete treatment values for the n samples.
- **y** (*pd.Series*) – Series of shape (n,) containing outcomes for the n samples.

Returns Covariates of samples that were matched am (pd.Series): Treatment values of samples that were matched ym (pd.Series): Outcome values of samples that were matched

Return type Xm (pd.DataFrame)

set_params(***kwargs*)

Set parameters of matching engine. Supported parameters are:

Keyword Arguments

- **propensity_transform** (*causallib.transformers.PropensityTransformer*) – an object for data preprocessing which adds the propensity score as a feature (default: None)
- **caliper** (*float*) – maximal distance for a match to be accepted (default: None)
- **with_replacement** (*bool*) – whether samples can be used multiple times for matching (default: True)
- **n_neighbors** (*int*) – number of nearest neighbors to include in match. Must be 1 if *with_replacement* is False (default: 1).
- **matching_mode** (*str*) – Direction of matching: *treatment_to_control*, *control_to_treatment* or *both* to indicate which set should be matched to which. All sets are cross-matched in *match* and without replacement there is no difference in outcome, but with replacement there is a difference and it impacts the results of *transform*.
- **metric** (*str*) – Distance metric string for calculating distance between samples (default: "mahalanobis",

  also supported: "euclidean")
- **knn_backend** (*str or callable*) – Backend to use for nearest neighbor search. Options are "sklearn" or a callable which returns an object implementing *fit*, *kneighbors* and *set_params* like the sklearn *NearestNeighbors* object. (default: "sklearn").

Returns (MatchingTransformer) object with new parameters set

Return type self

transform(*X*, *a*, *y*)

Transform data by restricting it to samples which are matched

Following a matching process, not all of the samples will find matches. Transforming the data by only allowing samples in treatment that have close matches in control, or in control that have close matches in treatment can make other causal methods more effective. This function will call *match* on the underlying Matching object.

The attribute *matching_mode* changes the behavior of this function. If set to *control_to_treatment* each control will attempt to find a match among the treated, hence the transformed data will have a maximum size of N_c + min(N_c,N_t). If set to *treatment_to_control*, each treatment will attempt to find a match among the control and the transformed data will have a maximum size of N_t + min(N_c,N_t). If set to *both*, both matching operations will be executed and if a sample succeeds in either direction it will be included, hence the maximum size of the transformed data will be *len(X)*.

If *with_replacement* is *False*, *matching_mode* does not change the behavior. There will be up to *min(N_c,N_t)* samples in the returned DataFrame, regardless.

Parameters

- **X** (*pd.DataFrame*) – DataFrame of shape (n,m) containing m covariates for n samples.
- **a** (*pd.Series*) – Series of shape (n,) containing discrete treatment values for the n samples.
- **y** (*pd.Series*) – Series of shape (n,) containing outcomes for the n samples.

Raises

- **NotImplementedError** – Raised if a value of attribute *matching_mode*
- **other than the supported values is set.** –

Returns Covariates of samples that were matched am (pd.Series): Treatment values of samples that were matched ym (pd.Series): Outcome values of samples that were matched

Return type Xm (pd.DataFrame)

class causallib.preprocessing.transformers.**MinMaxScaler**(*only_binary_features=True*, *ignore_nans=True*)

Bases: sklearn.base.BaseEstimator, sklearn.base.TransformerMixin

Scales features to 0-1, allowing for NaNs.

X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

Parameters

- **only_binary_features** (*bool*) – Whether to apply only on binary features or across all.
- **ignore_nans** (*bool*) – Whether to ignore NaNs during calculation.

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

Compute the minimum and maximum to be used for later scaling.

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used to compute the mean and standard deviation used for later scaling along the features axis (axis=0).
- **y** – Passthrough for Pipeline compatibility.

Returns a fitted MinMaxScaler

Return type *MinMaxScaler*

`inverse_transform`(*X*)

Scaling chosen features of X to the range of 0 - 1.

Parameters **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] Input data that will be transformed.

Returns array-like, shape [n_samples, n_features]. Transformed data.

Return type pd.DataFrame

`transform`(*X*)

Undo the scaling of X according to feature_range.

Parameters **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] Input data that will be transformed.

Returns array-like, shape [n_samples, n_features]. Transformed data.

Return type pd.DataFrame

`class causallib.preprocessing.transformers.PropensityTransformer`(*learner*, *include_covariates=False*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin`

Transform covariates by adding/replacing with the propensity score.

Parameters

- **learner** (*sklearn.estimator*) – A learner implementing *fit* and *predict_proba* to use for predicting the propensity score.
- **include_covariates** (*bool*) – Whether to return the original covariates alongside the "propensity" column.

`fit`(*X*, *a*)

`transform`(*X*, *treatment_values=None*)

Append propensity or replace covariates with propensity.

Parameters

- **X** (*pd.DataFrame*) – A DataFrame of samples to transform. This will be input to the learner trained by fit. If the columns are different, the results will not be valid.
- **treatment_values** (*Any | None*) – A desired value/s to extract propensity to (i.e. probabilities to what treatment value should be calculated). If not specified, then the maximal treatment value is chosen. This is since the usual case is of treatment (A=1) control (A=0) setting.

Returns DataFrame with a "propensity" column. If "include_covariates" is *True*, it will include all of the original features plus "propensity", else it will only have the "propensity" column.

Return type pd.DataFrame

`class causallib.preprocessing.transformers.StandardScaler`(*with_mean=True*, *with_std=True*, *ignore_nans=True*)

Bases: `sklearn.base.BaseEstimator`, `sklearn.base.TransformerMixin`

Standardize continuous features by removing the mean and scaling to unit variance while allowing nans.

X = (X - X.mean()) / X.std()

Parameters

- **with_mean** (*bool*) – Whether to center the data before scaling.
- **with_std** (*bool*) – Whether to scale the data to unit variance.
- **ignore_nans** (*bool*) – Whether to ignore NaNs during calculation.

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

Compute the mean and std to be used for later scaling.

Parameters

- **X** (*pd.DataFrame*) – The data used to compute the mean and standard deviation used for later scaling along the features axis (axis=0).
- **y** – Passthrough for `Pipeline` compatibility.

Returns A fitted standard-scaler

Return type *StandardScaler*

inverse_transform(*X*)

Scale back the data to the original representation

Parameters **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used to compute the mean and standard deviation used for later scaling along the features axis (axis=0).

Returns Un-scaled dataset.

Return type pd.DataFrame

transform(*X*, *y='deprecated'*)

Perform standardization by centering and scaling

Parameters

- **X** (*pd.DataFrame*) – array-like, shape [n_samples, n_features] The data used to compute the mean and standard deviation used for later scaling along the features axis (axis=0).
- **y** – Passthrough for `Pipeline` compatibility.X:

Returns Scaled dataset.

Return type pd.DataFrame

Module contents

3.1.3.7 causallib.simulation package

3.1.3.7.1 Submodules

causallib.simulation.CausalSimulator3 module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Jun 21, 2017

class causallib.simulation.CausalSimulator3.**CausalSimulator3**(*topology*, *var_types*, *prob_categories*, *link_types*, *snr*, *treatment_importances*, *treatment_methods='gaussian'*, *outcome_types='categorical'*, *effect_sizes=None*, *survival_distribution='expon'*, *survival_baseline=1*, *params=None*)

Bases: object

Constructor

Parameters

- **topology** (*np.ndarray*) – A boolean adjacency matrix for variables (including covariates, treatment and outcome variables of the model). Every row is a binary vector for a variable, where v[i, j] = 1 iff j is a parent of i
- **var_types** (*Sequence[str]*) – Vector the size of variables stating every variable to be "covariate", "hidden", "outcome", "treatment", "censor". **Notes**: if type(pd.Series) variable names will be var_types.index, otherwise, if no-key-vector - var names will be just range(num-of-variables).
- **prob_categories** (*Sequence[float|None]*) – vector the size of the number of variables. if prob_categories[i] = None -> than variable i is considered continuous. otherwise -> prob_categories[i] should be a list (or any iterable) which size specifies number of categories variable i has, and it contains multinomial probabilities for those categories (i.e. list non negative and sums to 1).
- **link_types** (*str|Sequence[str]*) – set of string the size or string or specifying the relation between covariate parents to the covariate itself
- **snr** (*float|Sequence[float]*) – Signal to noise ratio (use 1.0 to eliminate noise in the system). May be a vector the size of number of variables for stating different snr values for different variables.
- **treatment_importances** (*float|Sequence[float]*) – The effect of treatment on the outcome. A float between 0 and 1.0 stating how much weight the treatment variable have vs. the other parents of an outcome variable. *To support multi-treatment* - place a list the size of the number of treatment variables (as stated in var_types). The matching between treatment variable and its importance will be according to the order of the treatment variables and the order of the list. If all treatments variables has the same importance - pass the float value.
- **treatment_methods** (*str|Sequence[str]*) – method for creating treatment assignment and propensities, can be one of {"random", "gaussian", "logistic"}. *To support multi-treatment* - place a list the size of the number of treatment variables. The matching between treatment variable and its creation method will be according to the order of the treatment variables and the order of the list. If all treatment variables has the same type - pass the str value.
- **outcome_types** (*str|Sequence[str]*) – outcome can either be 'survival' or 'binary'. *To support multi-outcome* - place a list the size of the number of outcome variables (as

stated in var_types). The matching between outcome variable and its type will be according to the order of the outcome variables and the order of the list. If all outcome variables has the same type - pass the str value.

- **effect_sizes** (*float|Sequence[float|None]|None*) – The wanted mean effect size between two counterfactuals. If None - The mean effect size will not be adjusted, but will be whatever generated. If float - The mean effect size will be adjusted to be approximately the given number (considering the noise) *To support multi-outcome* - a list the size the number of the outcome variables (as stated in var_types). The matching between outcome variable and its effect size will be according to the order of the outcome variables and the order of the list.
- **survival_distribution** (*Sequence[str] or str*) – The distribution family from which to generate the outcome values of outcome variables that their corresponding outcome_types is "survival". Default value is exponent distribution. The same survival distribution will be used for the corresponding censoring variable as well. *To support multi-outcome* - place a list the size of the number of outcome variables of type "survival" (as stated in outcome_types). The matching between survival outcome variable and its survival distribution will be according to the order of the outcome variables and the order of the list. If all outcome variables has the same survival distribution - pass the str value (if present). *Ignore if no outcome variable is of type survival*
- **survival_baseline** (*Sequence[float] or float*) – The survival baseline from the CoxPH model that will be the basics for the parameters of the corresponding survival_distribution. The same survival baseline will be used for the corresponding censoring variable as well (if present). Default value is 1 (no multiplicative meaning for baseline value). *To support multi-outcome* - place a list the size of the number of outcome variables of type "survival" (as stated in outcome_types). The matching between survival outcome variable and its survival distribution will be according to the order of the outcome variables and the order of the list. If all outcome variables has the same survival distribution - pass the str value. *Ignore if no outcome variable is of type survival*
- **params** (*dict | None*) – Various parameters related to the generation process (e.g. the slope for sigmoid-based functions etc.). The form of: {var_name: {param_name: param_value, ...}, ...}

G_LINKING_METHODS = {'affine': <function CausalSimulator3.<lambda>>, 'exp': <function CausalSimulator3.<lambda>>, 'linear': <function CausalSimulator3.<lambda>>, 'log': <function CausalSimulator3.<lambda>>, 'poly': <function CausalSimulator3.<lambda>>}

O_LINKING_METHODS = {'marginal_structural_model': <function CausalSimulator3.<lambda>>, None: <function CausalSimulator3.<lambda>>}

TREATMENT_METHODS = {'gaussian': <function CausalSimulator3.<lambda>>, 'logistic': <function CausalSimulator3.<lambda>>, 'odds_ratio': <function CausalSimulator3.<lambda>>, 'quantile_gauss_fit': <function CausalSimulator3.<lambda>>, 'random': <function CausalSimulator3.<lambda>>}

format_for_training(*X*, *propensities*, *cf*, *headers_chars=None*, *exclude_hidden_vars=True*)

prepare to output. merge the data into two DataFrames - an observed one and one gathering the counterfactuals.

Parameters

- **X** (*pd.DataFrame*) – Containing the data (covariates) , treatment and outcomes
- **propensities** (*pd.DataFrame*) – Containing the propensity values for the treatmetns

- **cf** (*pd.DataFrame*) – Containing the counterfactuals results for all possible treatments.
- **headers_chars** (*dict*) – Optional. Containing the column header prefix for different types of variables. Examples: {"covariate": "x", "treatment": "t", "outcome": "y"}
- **exclude_hidden_vars** – If to exclude hidden variables from the resulting dataset.

Returns

2-element tuple containing:

- **df_X** (*pd.DataFrame*): The observed dataset (if hidden variables are excluded).
- **df_cf** (*pd.DataFrame*): Containing the two counterfactuals, treatments and propensities.

Return type (pd.DataFrame, pd.DataFrame)

generate_censor_col(*X_parents*, *link_type*, *snr*, *prob_category*, *outcome_type*, *treatment_importance=None*, *survival_distribution=None*, *survival_baseline=None*, *var_name=None*)

Generates a single censor variable column.

Parameters

- **X_parents** (*pd.DataFrame*) – Sub-dataset containing only the relevant columns (features which are topological parents to the current covariate being created)
- **link_type** (*str*) – How the parents variables (parents covariate columns) influence the current generated column. What relation is there between them.
- **snr** (*float*) – Signal to noise ratio that controls the amount of noise to add (value of 1.0 will not generate noise)
- **prob_category** (*Sequence | None*) – A k-length distribution vector over k-1 treatments with the probability of being untreated in prob_category[0] (prob_category.iloc[0]) and all other k-1 probabilities corresponds to k-1 treatments.

  Notes: vector must sum to 1. If None - the covariate column is left untouched (i.e. continuous)
- **outcome_type** (*str*) – The type of the outcome variable that is dependent on the current censor variable. The censoring mechanism varies given different types of outcome variables.
- **treatment_importance** (*float*) – The effect power of the treatment on the current generated outcome variable, as opposed to other variables that may influence on it.
- **survival_distribution** (*str*) – The type of the distribution of which to sample the survival time from. relevant only if outcome_type is "survival"
- **survival_baseline** – The baseline value of the the cox ph model. relevant only if outcome_type is "survival"
- **var_name** (*int | str*) – The name of the variable currently being generated. Optional.

Returns

2-element tuple containing:

- **x_censor** (*pd.Series*): a column describing the censor variable
- **beta** (*pd.Series*): The coefficients used to generate current variable from it predecessors.

Return type (pd.Series, pd.Series)

generate_covariate_col(*X_parents*, *link_type*, *snr*, *prob_category*, *num_samples*, *var_name=None*)
Generates a single signal (covariate) column

Parameters

- **X_parents** (*pd.DataFrame*) – Sub-dataset containing only the relevant columns (features which are topological parents to the current covariate being created)
- **link_type** (*str*) – How the parents variables (parents covariate columns) influence the current generated column. What relation is there between them.
- **snr** (*float*) – Signal to noise ratio that controls the amount of noise to add (value of 1.0 will not generate noise)
- **prob_category** (*pd.Series|None*) – A vector which length states the number of classes (number of discrete values) and every value is fractional - the probability of the corresponding class.

  Notes: vector must sum to 1 If None - the covariate column is left untouched (i.e. continuous)
- **num_samples** (*int*) – number of samples to generate
- **var_name** (*int|str*) – The name of the variable currently being generated. Optional.

Returns

2-element tuple containing:

- **X_final** (*pd.Series*): The final (i.e. noised and discretize [if needed]) covariate column.
- **beta** (*pd.Series*): The coefficients used to generate current variable from it predecessors.

Return type (pd.Series, pd.Series)

Raises **ValueError** – if the given link_type is not a valid link_type. (Supported link types are placed in self.G_LINKING_METHODS)

generate_data(*X_given=None*, *num_samples=None*, *random_seed=None*)
Generates tables of dataset given the object's initial parameters.

Parameters

- **num_samples** (*int*) – Number of samples that will be in the dataset.
- **X_given** (*pd.DataFrame*) – A baseline dataset to generate from. This dataset may contain only some of variables stated in the initialized topology. The rest of the dataset (variables which are stated in the topology and not in this dataset) will be generated. **Notes**: The data given will not be overwritten and will be taken as is. It is

  user responsibility to see that the given table has no dependant variables since they will not be re-generated according to the graph.
- **random_seed** (*int*) – A seed for the pseudo-random-number-generator in order to reproduce results.

Returns

3-element tuple containing:

- **X (*pd.DataFrame*): A (num_samples x num_covariates) matrix of all covariates** (including treatments and outcomes) over samples.

- **propensities (*pd.DataFrame*): A (num_samples x num_treatments) matrix (or vector) of propensity** values of every treatment.
- **counterfactuals** (*pd.DataFrame*): A (num_samples x num_outcomes) matrix -

Return type (pd.DataFrame, pd.DataFrame, pd.DataFrame)

`generate_outcome_col`(*X_parents*, *link_type*, *snr*, *prob_category*, *outcome_type*, *treatment_importance=None*, *effect_size=None*, *survival_distribution=None*, *survival_baseline=None*, *var_name=None*)

Generates a single outcome variable column.

Parameters

- **X_parents** (*pd.DataFrame*) – Sub-dataset containing only the relevant columns (features which are topological parents to the current covariate being created)
- **link_type** (*str*) – How the parents variables (parents covariate columns) influence the current generated column. What relation is there between them.
- **treatment_importance** (*float*) – The effect power of the treatment on the current generated outcome variable, as opposed to other variables that may influence on it.
- **snr** (*float*) – Signal to noise ratio that controls the amount of noise to add (value of 1.0 will not generate noise)
- **prob_category** (*pd.Series|None*) – A k-length distribution vector over k-1 treatments with the probability of being untreated in prob_category[0] (prob_category.iloc[0]) and all other k-1 probabilities corresponds to k-1 treatments.

  Notes: vector must sum to 1. If None - the covariate column is left untouched (i.e. continuous)
- **effect_size** (*float*) – wanted mean effect size.
- **outcome_type** (*str*) – Type of outcome variable. Either categorical (and continuous) or survival
- **survival_distribution** (*str*) – The type of the distribution of which to sample the survival time from. relevant only if outcome_type is "survival"
- **survival_baseline** – The baseline value of the the cox ph model. relevant only if outcome_type is "survival"
- **var_name** (*int|str*) – The name of the variable currently being generated. Optional.

Returns

3-element tuple containing:

- **x_outcome** (*pd.Series*): Outcome assignment for each sample.
- **cf (*pd.DataFrame*): Holding the counterfactuals for every possible treatment category of the** outcome's treatment predecessor variable.
- **beta** (*pd.DataFrame*): The coefficients used to generate current variable from it predecessors.

Return type (pd.Series, pd.DataFrame, pd.DataFrame)

Raises

- **ValueError** – if the given link_type is not a valid link_type. (Supported link types are placed in self.G_LINKING_METHODS)

- **ValueError** – if prob_category is neither None nor a legitimate distribution vector.

generate_treatment_col(*X_parents*, *link_type*, *snr*, *prob_category*, *method='logistic'*, *var_name=None*)

Generates a single treatment variable column.

Parameters

- **X_parents** (*pd.DataFrame*) – Sub-dataset containing only the relevant columns (features which are topological parents to the current covariate being created)
- **link_type** (*str*) – How the parents variables (parents covariate columns) influence the current generated column. What relation is there between them.
- **snr** (*float*) – Signal to noise ratio that controls the amount of noise to add (value of 1.0 will not generate noise)
- **prob_category** (*pd.Series|None*) – A k-length distribution vector over k-1 treatments with the probability of being untreated in prob_category[0] (prob_category.iloc[0]) and all other k-1 probabilities corresponds to k-1 treatments.

  Notes: vector must sum to 1. If None - the covariate column is left untouched (i.e. continuous)
- **method** (*str*) – A type of method to generate the treatment signal and the corresponding propensities.
- **var_name** (*int|str*) – The name of the variable currently being generated. Optional.

Returns

3-element tuple containing:

- **treatment** (*pd.Series*): Treatment assignment to each sample.
- **propensity (*pd.DataFrame*): The marginal conditional probability of treatment given covariates.** A DataFrame shaped (num_samples x num_of_possible_treatment_categories).
- **beta** (*pd.Series*): The coefficients used to generate current variable from it predecessors.

Return type (pd.Series, pd.DataFrame, pd.Series)

Raises

- **ValueError** – if prob_category is None (treatment must be categorical)
- **ValueError** – If prob_category is not a legitimate probability vector (non negative, sums to 1)

reset_coefficients(*variables=None*)

Delete the linking coefficients that accumulated in the generating model so far.

Parameters **variables** (*list|None*) – list of variables to reset the coefficients linking *into* them (Not from them). if None - all the available coefficients will be deleted.

static to_csv(*data*, *out_file=None*)

causallib.simulation.CausalSimulator3.**generate_random_topology**(*n_covariates*, *p*, *n_treatments=1*, *n_outcomes=1*, *n_censoring=0*, *given_vars=()*, *p_hidden=0.0*)

Creates a random graph topology, suitable for describing a causal graph model. Generation is based on a G(n,p) random graph model (each edge independently generated or not by a coin toss).

Parameters

- **n_covariates** (*int*) – Number of simple covariates to generate
- **p** (*float*) – Probability to generate an edge.
- **n_treatments** (*int*) – Number of treatment variables.
- **n_outcomes** (*int*) – Number of outcome variables.
- **n_censoring** (*int*) – Number of censoring variables.
- **given_vars** (*Sequence[Any]*) – Vector of names of given variables. These variables are considered independent. These suppose to mimic a situation where a partial dataset can be supplied to the generation process. Those names will correspond to the variable names in this existing baseline dataset.
- **p_hidden** (*float*) – The probability to convert a simple covariate variable into a latent (i.e. hidden) variable.

Returns

2-element tuple containing:

- **topology (*pd.DataFrame*): A boolean matrix describing graph dependencies.**
  Where T[i,j] = True iff j is a predecessor of i.
- **var_types (*pd.Series*): A Series which index holds variable names and values are variable types.**
  (e.g. "treatment", "covariate", "hidden', "outcome"…) The given_vars will be the first variable, followed by the generated vars (covariates, then treatment, then outcome, then censors)

Return type (pd.DataFrame, pd.Series)

`causallib.simulation.CausalSimulator3.`**idx2var_vector**(*num_vars*, *args*)

3.1.3.7.2 Module contents

3.1.3.8 Module causallib.survival

This module allows estimating counterfactual outcomes in a setting of right-censored data (also known as survival analysis, or time-to-event modeling). In addition to the standard inputs of `X` - baseline covariates, `a` - treatment assignment and `y` - outcome indicator, a new variable `t` is introduced, measuring time from the beginning of observation period to an occurrence of event. An event may be right-censoring (where `y=0`) or an outcome of interest, or "death" (where `y=1`, which is also considered as censoring). Each of these methods uses an underlying machine learning model of choice, and can also integrate with the *`lifelines`* <https://github.com/CamDavidsonPilon/lifelines>`_ survival analysis Python package.

Additional methods will be added incrementally.

3.1.3.8.1 Available Methods

The methods that are currently available are:

1. Weighting: `causallib.survival.WeightedSurvival` - uses `causallib`'s `WeightEstimator` (e.g., `IPW`) to generate weighted pseudo-population for survival analysis.
2. Standardization (parametric g-formula): `causallib.survival.StandardizedSurvival` - fits a parametric hazards model that includes baseline covariates.
3. Weighted Standardization: `causallib.survival.WeightedStandardizedSurvival` - combines the two above-mentioned methods.

Example: Weighted survival analysis with Inverse Probability Weighting

```
from sklearn.linear_model import LogisticRegression
from causallib.survival import WeightedSurvival
from causallib.estimation import IPW
from causallib.datasets import load_nhefs_survival

ipw = IPW(learner=LogisticRegression())
weighted_survival_estimator = WeightedSurvival(weight_model=ipw)
X, a, t, y = load_nhefs_survival()

weighted_survival_estimator.fit(X, a)
population_averaged_survival_curves = weighted_survival_estimator.estimate_population_
↪outcome(X, a, t, y)
```

Example: Standardized survival (parametric g-formula)

```
from causallib.survival import StandardizedSurvival

standardized_survival = StandardizedSurvival(survival_model=LogisticRegression())
standardized_survival.fit(X, a, t, y)
population_averaged_survival_curves = standardized_survival.estimate_poplatuon_outcome(X,
↪ a, t)
individual_survival_curves = standardized_survival.estimate_individual_outcome(X, a, t)
```

3.1.3.8.2 Submodules

causallib.survival.base_survival module

class causallib.survival.base_survival.**SurvivalBase**

Bases: abc.ABC

Interface class for causal survival analysis with fixed baseline covariates.

abstract estimate_population_outcome(***kwargs*) → pandas.core.frame.DataFrame

Returns population averaged survival curves.

Returns with time-step index, treatment values as columns and survival as entries

Return type pd.DataFrame

abstract fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*)

Fits internal learner(s).

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).

Returns self

class causallib.survival.base_survival.**SurvivalTimeVaryingBase**

Bases: *causallib.survival.base_survival.SurvivalBase*

Interface class for causal survival analysis estimators that support time-varying followup covariates. Followup covariates matrix (XF) needs to have a 'time' column, and indexed by subject IDs that correspond to the other inputs (X, a, y, t). All columns other than 'time' will be used for time-varying adjustments.

Example XF format: +—-+——+——+——+ | id | t | var1 | var2 | +—-+——+——+——+ | 1 | 0 | 1.4 | 22 | | 1 | 4 | 1.2 | 22 | | 1 | 8 | 1.5 | NaN | | 2 | 0 | 1.6 | 10 | | 2 | 11 | 1.6 | 11 | +—-+——+——+——+

abstract fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *XF: Optional[pandas.core.frame.DataFrame] = None*) → None

Fits internal survival functions.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **XF** (*pd.DataFrame*) – Time-varying followup covariate matrix

Returns A fitted estimator with precalculated survival functions.

causallib.survival.marginal_survival module

class causallib.survival.marginal_survival.**MarginalSurvival**(*survival_model: Optional[Any] = None*)

Bases: *causallib.survival.weighted_survival.WeightedSurvival*

Marginal (un-adjusted) survival estimator. Essentially it is a degenerated WeightedSurvival instance without a weight model.

Marginal (un-adjusted) survival estimator. :param survival_model: Three alternatives:

1. None - compute non-parametric KaplanMeier survival curve
2. Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a time-varying hazards model
3. lifelines UnivariateFitter - use lifelines fitter to compute survival curves from events and durations

causallib.survival.regression_curve_fitter module

class causallib.survival.regression_curve_fitter.**RegressionCurveFitter**(*learner: sklearn.base.BaseEstimator*)

Bases: object

Default implementation of a parametric survival curve fitter with covariates (pooled regression). API follows 'lifelines' convention for regression models, see here for example: https://lifelines.readthedocs.io/en/latest/fitters/regression/CoxPHFitter.html#lifelines.fitters.coxph_fitter.CoxPHFitter.fit

Parameters **learner** – scikit-learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a time-varying hazards model that includes baseline covariates. Note that the model is fitted on a person-time table with all covariates, and might be computationally and memory expansive.

fit(*df: pandas.core.frame.DataFrame*, *duration_col: str*, *event_col: Optional[str] = None*, *weights_col: Optional[str] = None*)
Fits a parametric curve with covariates.

Parameters

- **df** (*pd.DataFrame*) – DataFrame, must contain a 'duration_col', and optional 'event_col' / 'weights_col'. All other columns are treated as baseline covariates.
- **duration_col** (*str*) – Name of column with subjects' lifetimes (time-to-event)
- **event_col** (*Optional[str]*) – Name of column with event type (outcome=1, censor=0). If unspecified, assumes that all events are 'outcome' (no censoring).
- **weights_col** (*Optional[str]*) – Name of column with optional subject weights.

Returns Self

predict_survival_function(*X: Optional[Union[pandas.core.series.Series, pandas.core.frame.DataFrame]] = None*, *times: Optional[Union[List[float], numpy.ndarray, pandas.core.series.Series]] = None*) → pandas.core.frame.DataFrame
Predicts survival function (table) for individuals, given their covariates. :param X: Subjects covariates :type X: pd.DataFrame / pd.Series :param times: An iterable of increasing time points to predict cumulative hazard at.

If unspecified, predict all observed time points in data.

Returns Each column contains a survival curve for an individual, indexed by time-steps

Return type pd.DataFrame

causallib.survival.standardized_survival module

class causallib.survival.standardized_survival.**StandardizedSurvival**(*survival_model: Any*, *stratify: bool = True*, ***kwargs*)

Bases: *causallib.survival.base_survival.SurvivalBase*

Standardization survival estimator. Computes parametric curve by fitting a time-varying hazards model that includes baseline covariates. :param survival_model: Two alternatives:

1. **Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a** time-varying hazards model that includes baseline covariates. Note that the model is fitted on a person-time table with all covariates, and might be computationally and memory expansive.
2. **lifelines RegressionFitter - use lifelines fitter to compute survival curves from baseline covariates,** events and durations

Parameters **stratify** (*bool*) – if True, fit a separate model per treatment group

estimate_individual_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: Optional[Any] = None*, *timeline_start: Optional[int] = None*, *timeline_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Returns individual survival curves for each subject row in X/a/t

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup durations, size (num_subjects,).
- **y** – NOT USED (for API compatibility only).
- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event (t.min()).
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event (t.max()).

Returns with time-step index, subject IDs (X.index) as columns and point survival as entries

Return type pd.DataFrame

estimate_population_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: Optional[Any] = None*, *timeline_start: Optional[int] = None*, *timeline_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Returns population averaged survival curves.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup durations, size (num_subjects,).
- **y** – NOT USED (for API compatibility only).
- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event (t.min()).
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event (t.max()).

Returns with time-step index, treatment values as columns and survival as entries

Return type pd.DataFrame

fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *w: Optional[pandas.core.series.Series] = None*, *fit_kwargs: Optional[dict] = None*)

Fits parametric models and calculates internal survival functions.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).

- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **w** (*pd.Series*) – Optional subject weights.
- **fit_kwargs** (*dict*) – Optional kwargs for fit call of survival model

Returns self

causallib.survival.survival_utils module

causallib.survival.survival_utils.**add_random_suffix**(*name*, *suffix_length=4*)

Adds a random suffix to string, by computing uuid64.hex.

Parameters

- **name** – input string
- **suffix_length** – length of desired added suffix.

Returns string with suffix

causallib.survival.survival_utils.**canonize_dtypes_and_names**(*a=None*, *t=None*, *y=None*, *w=None*, *X=None*)

Housekeeping method that assign names for unnamed series and canonizes their data types.

Parameters

- **a** (*pd.Series|None*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series|None*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series|None*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **w** (*pd.Series|None*) – Optional subject weights
- **X** (*pd.DataFrame|None*) – Baseline covariate matrix of size (num_subjects, num_features).

Returns a, y, t, w, X

causallib.survival.survival_utils.**compute_survival_from_single_hazard_curve**(*hazard: List*, *logspace: bool = False*) → List

Computes survival curve from an array of point hazards. Note that trailing NaN are supported :param hazard: list/array of point hazards :type hazard: list :param logspace: whether to compute in logspace, for numerical stability :type logspace: bool

Returns survival at each time-step

Return type list

causallib.survival.survival_utils.**get_person_time_df**(*t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *a: Optional[pandas.core.series.Series] = None*, *w: Optional[pandas.core.series.Series] = None*, *X: Optional[pandas.core.frame.DataFrame] = None*, *return_individual_series: bool = False*) → pandas.core.frame.DataFrame

Converts standard input format into an expanded person-time format. Input series need to be indexed by subject IDs and have non-null names (including index name).

Parameters

- **t** (*pd.Series*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **w** (*pd.Series*) – Optional subject weights
- **X** (*pd.DataFrame*) – Optional baseline covariate matrix of size (num_subjects, num_features).
- **return_individual_series** (*bool*) – If True, returns a tuple of Series/DataFrames instead of a single DataFrame

Returns Expanded person-time format with columns from X and expanded 'a', 'y', 't' columns

Return type pd.DataFrame

Examples

This example standard input:

age height a y t

id 1 22 170 0 1 2 2 40 180 1 0 1 3 30 165 1 0 2

Will be expanded to:

age height a y t

id 1 22 170 0 0 0 1 22 170 0 0 1 1 22 170 0 1 2 2 40 180 1 0 0 2 40 180 1 0 1 3 30 165 1 0 0 3 30 165 1 0 1 3 30 165 1 0 2

causallib.survival.survival_utils.**get_regression_predict_data**(*X: pandas.core.frame.DataFrame*, *times: pandas.core.series.Series*)

Generates prediction data for a regression fitter: repeats patient covariates per time point in 'times'. .. rubric:: Example

age height

id 1 22 170 2 40 180

0 1 2

age height t

id 1 22 170 0 1 22 170 1 1 22 170 2 2 40 180 0 2 40 180 1 2 40 180 2

Parameters

- **X** (*pd.DataFrame*) – Covariates DataFrame

- **times** (*pd.Series*) – A Series of time points to predict

Returns

DataFrame with repeated covariates per time point. Index is subject ID with repeats, columns are X + a time column, which is a repeat of 'times' per subject.

t_name (str): Name of time column in pred_data_X. Default is 't', but since we concatenate a column to a covariates frame, we might need to add a random suffix to it.

Return type pred_data_X (pd.DataFrame)

causallib.survival.survival_utils.**safe_join**(*df: Optional[pandas.core.frame.DataFrame] = None*, *list_of_series: Optional[List[pandas.core.series.Series]] = None*, *return_series_names=False*)

Safely joins (concatenates on axis 1) a collection of Series (or one DataFrame and multiple Series), while renaming Series that have a duplicate name (a name that already exists in DataFrame or another Series). * Note that DataFrame columns are never changed (only Series names are).

Parameters

- **df** (*pd.DataFrame*) – optional DataFrame. If provided, will join Series to DataFrame
- **list_of_series** (*List[pd.Series]*) – list of Series for safe-join
- **return_series_names** (*bool*) – if True, returns a list of (potentially renamed) Series names

Returns

1. single concatenated DataFrame
2. list of (potentially renamed) Series names

causallib.survival.univariate_curve_fitter module

class causallib.survival.univariate_curve_fitter.**UnivariateCurveFitter**(*learner: Optional[sklearn.base.BaseEstimator] = None*)

Bases: object

Default implementation of a univariate survival curve fitter. Construct a curve fitter, either non-parametric (Kaplan-Meier) or parametric. API follows 'lifelines' convention for univariate models, see here for example: https://lifelines.readthedocs.io/en/latest/fitters/univariate/KaplanMeierFitter.html#lifelines.fitters.kaplan_meier_fitter.KaplanMeierFitter.fit :param learner: optional scikit-learn estimator (needs to implement *predict_proba*). If provided, will

compute parametric curve by fitting a time-varying hazards model. if None, will compute non-parametric Kaplan-Meier estimator.

fit(*durations*, *event_observed=None*, *weights=None*)

Fits a univariate survival curve (Kaplan-Meier or parametric, if a learner was provided in constructor)

Parameters

- **durations** (*Iterable*) – Duration subject was observed
- **event_observed** (*Optional[Iterable]*) – Boolean or 0/1 iterable, where True means 'outcome event' and False means 'right censoring'. If unspecified, assumes that all events are 'outcome' (no censoring).
- **weights** (*Optional[Iterable]*) – Optional subject weights

Returns Self

predict(*times=None*, *interpolate=False*)

Compute survival curve for time points given in 'times' param. :param times: sequence of time points for prediction :param interpolate: if True, linearly interpolate non-observed times. Otherwise, repeat last observed time point.

Returns with times index and survival values

Return type pd.Series

causallib.survival.weighted_standardized_survival module

class causallib.survival.weighted_standardized_survival.**WeightedStandardizedSurvival**(*weight_model: causallib.estimation.base_weig sur- vival_model: Any, strat- ify: bool = True, out- come_covariates=None weight_covariates=Non*

Bases: *causallib.survival.standardized_survival.StandardizedSurvival*

Combines WeightedSurvival and StandardizedSurvival:

1. Adjusts for treatment assignment by creating weighted pseudo-population (e.g., inverse propensity weighting).
2. Computes parametric curve by fitting a time-varying hazards model that includes baseline covariates.

Parameters

- **weight_model** – causallib compatible weight model (e.g., IPW)
- **survival_model** –

  Two alternatives:

  1. **Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a** time-varying hazards model that includes baseline covariates. Note that the model is fitted on a person-time table with all covariates, and might be computationally and memory expansive.
  2. **lifelines RegressionFitter - use lifelines fitter to compute survival curves from baseline covariates,** events and durations

  stratify (bool): if True, fit a separate model per treatment group outcome_covariates (array): Covariates to use for outcome model.

  If None - all covariates passed will be used. Either list of column names or boolean mask.

weight_covariates (array): Covariates to use for weight model. If None - all covariates passed will be used. Either list of column names or boolean mask.

estimate_individual_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: Optional[Any] = None*, *timeline_start: Optional[int] = None*, *timeline_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Returns individual survival curves for each subject row in X/a/t

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup durations, size (num_subjects,).
- **y** – NOT USED (for API compatibility only).
- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event (t.min()).
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event (t.max()).

Returns with time-step index, subject IDs (X.index) as columns and point survival as entries

Return type pd.DataFrame

fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *w: Optional[pandas.core.series.Series] = None*, *fit_kwargs: Optional[dict] = None*)

Fits parametric models and calculates internal survival functions.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **w** (*pd.Series*) – NOT USED (for compatibility only) optional subject weights.
- **fit_kwargs** (*dict*) – Optional kwargs for fit call of survival model

Returns self

causallib.survival.weighted_survival module

class causallib.survival.weighted_survival.**WeightedSurvival**(*weight_model: Optional[causallib.estimation.base_weight.WeightEstimator* = *None*, *survival_model: Optional[Any]* = *None*)

Bases: *causallib.survival.base_survival.SurvivalBase*

Weighted survival estimator

Weighted survival estimator. :param weight_model: causallib compatible weight model (e.g., IPW) :param survival_model: Three alternatives:

1. None - compute non-parametric KaplanMeier survival curve
2. **Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a** time-varying hazards model
3. lifelines UnivariateFitter - use lifelines fitter to compute survival curves from events and durations

estimate_population_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *timeline_start: Optional[int]* = *None*, *timeline_end: Optional[int]* = *None*) → pandas.core.frame.DataFrame

Returns population averaged survival curves.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series* | *int*) – Followup durations, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event.
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event.

Returns with timestep index, treatment values as columns and survival as entries

Return type pd.DataFrame

fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: Optional[pandas.core.series.Series]* = *None*, *y: Optional[pandas.core.series.Series]* = *None*, *fit_kwargs: Optional[dict]* = *None*)

Fits internal weight module (e.g. IPW module, adversarial weighting, etc).

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – NOT USED (for compatibility only)
- **y** (*pd.Series*) – NOT USED (for compatibility only)

- **fit_kwargs** (*dict*) – Optional kwargs for fit call of survival model (NOT USED, since fit call of survival model occurs in 'estimate_population_outcome' rather than here)

Returns self

3.1.3.8.3 Module contents

Causal Survival Analysis Models

class causallib.survival.**MarginalSurvival**(*survival_model: Optional[Any] = None*)
Bases: *causallib.survival.weighted_survival.WeightedSurvival*

Marginal (un-adjusted) survival estimator. Essentially it is a degenerated WeightedSurvival instance without a weight model.

Marginal (un-adjusted) survival estimator. :param survival_model: Three alternatives:

1. None - compute non-parametric KaplanMeier survival curve
2. Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a time-varying hazards model
3. lifelines UnivariateFitter - use lifelines fitter to compute survival curves from events and durations

class causallib.survival.**RegressionCurveFitter**(*learner: sklearn.base.BaseEstimator*)
Bases: object

Default implementation of a parametric survival curve fitter with covariates (pooled regression). API follows 'lifelines' convention for regression models, see here for example: https://lifelines.readthedocs.io/en/latest/fitters/regression/CoxPHFitter.html#lifelines.fitters.coxph_fitter.CoxPHFitter.fit

Parameters **learner** – scikit-learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a time-varying hazards model that includes baseline covariates. Note that the model is fitted on a person-time table with all covariates, and might be computationally and memory expansive.

fit(*df: pandas.core.frame.DataFrame*, *duration_col: str*, *event_col: Optional[str] = None*, *weights_col: Optional[str] = None*)
Fits a parametric curve with covariates.

Parameters

- **df** (*pd.DataFrame*) – DataFrame, must contain a 'duration_col', and optional 'event_col' / 'weights_col'. All other columns are treated as baseline covariates.
- **duration_col** (*str*) – Name of column with subjects' lifetimes (time-to-event)
- **event_col** (*Optional[str]*) – Name of column with event type (outcome=1, censor=0). If unspecified, assumes that all events are 'outcome' (no censoring).
- **weights_col** (*Optional[str]*) – Name of column with optional subject weights.

Returns Self

predict_survival_function(*X: Optional[Union[pandas.core.series.Series, pandas.core.frame.DataFrame]] = None*, *times: Optional[Union[List[float], numpy.ndarray, pandas.core.series.Series]] = None*) → pandas.core.frame.DataFrame
Predicts survival function (table) for individuals, given their covariates. :param X: Subjects covariates :type X: pd.DataFrame / pd.Series :param times: An iterable of increasing time points to predict cumulative hazard at.

If unspecified, predict all observed time points in data.

Returns Each column contains a survival curve for an individual, indexed by time-steps

Return type pd.DataFrame

class causallib.survival.**StandardizedSurvival**(*survival_model: Any*, *stratify: bool = True*, ***kwargs*)
Bases: causallib.survival.base_survival.SurvivalBase

Standardization survival estimator. Computes parametric curve by fitting a time-varying hazards model that includes baseline covariates. :param survival_model: Two alternatives:

1. **Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a** time-varying hazards model that includes baseline covariates. Note that the model is fitted on a person-time table with all covariates, and might be computationally and memory expansive.
2. **lifelines RegressionFitter - use lifelines fitter to compute survival curves from baseline covariates,** events and durations

Parameters **stratify** (*bool*) – if True, fit a separate model per treatment group

estimate_individual_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: Optional[Any] = None*, *timeline_start: Optional[int] = None*, *timeline_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Returns individual survival curves for each subject row in X/a/t

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup durations, size (num_subjects,).
- **y** – NOT USED (for API compatibility only).
- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event (t.min()).
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event (t.max()).

Returns with time-step index, subject IDs (X.index) as columns and point survival as entries

Return type pd.DataFrame

estimate_population_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: Optional[Any] = None*, *timeline_start: Optional[int] = None*, *timeline_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Returns population averaged survival curves.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup durations, size (num_subjects,).

- **y** – NOT USED (for API compatibility only).
- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event (t.min()).
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event (t.max()).

Returns with time-step index, treatment values as columns and survival as entries

Return type pd.DataFrame

fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *w: Optional[pandas.core.series.Series] = None*, *fit_kwargs: Optional[dict] = None*)

Fits parametric models and calculates internal survival functions.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **w** (*pd.Series*) – Optional subject weights.
- **fit_kwargs** (*dict*) – Optional kwargs for fit call of survival model

Returns self

class causallib.survival.**UnivariateCurveFitter**(*learner: Optional[sklearn.base.BaseEstimator] = None*)

Bases: object

Default implementation of a univariate survival curve fitter. Construct a curve fitter, either non-parametric (Kaplan-Meier) or parametric. API follows 'lifelines' convention for univariate models, see here for example: https://lifelines.readthedocs.io/en/latest/fitters/univariate/KaplanMeierFitter.html#lifelines.fitters.kaplan_meier_fitter.KaplanMeierFitter.fit :param learner: optional scikit-learn estimator (needs to implement *predict_proba*). If provided, will

compute parametric curve by fitting a time-varying hazards model. if None, will compute non-parametric Kaplan-Meier estimator.

fit(*durations*, *event_observed=None*, *weights=None*)

Fits a univariate survival curve (Kaplan-Meier or parametric, if a learner was provided in constructor)

Parameters

- **durations** (*Iterable*) – Duration subject was observed
- **event_observed** (*Optional[Iterable]*) – Boolean or 0/1 iterable, where True means 'outcome event' and False means 'right censoring'. If unspecified, assumes that all events are 'outcome' (no censoring).
- **weights** (*Optional[Iterable]*) – Optional subject weights

Returns Self

predict(*times=None*, *interpolate=False*)
Compute survival curve for time points given in 'times' param. :param times: sequence of time points for prediction :param interpolate: if True, linearly interpolate non-observed times. Otherwise, repeat last observed time point.

Returns with times index and survival values

Return type pd.Series

class causallib.survival.**WeightedStandardizedSurvival**(*weight_model: causallib.estimation.base_weight.WeightEstimator*, *survival_model: Any*, *stratify: bool = True*, *outcome_covariates=None*, *weight_covariates=None*)
Bases: *causallib.survival.standardized_survival.StandardizedSurvival*

Combines WeightedSurvival and StandardizedSurvival:

1. Adjusts for treatment assignment by creating weighted pseudo-population (e.g., inverse propensity weighting).
2. Computes parametric curve by fitting a time-varying hazards model that includes baseline covariates.

Parameters

- **weight_model** – causallib compatible weight model (e.g., IPW)
- **survival_model** –

  Two alternatives:

  1. **Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a** time-varying hazards model that includes baseline covariates. Note that the model is fitted on a person-time table with all covariates, and might be computationally and memory expansive.
  2. **lifelines RegressionFitter - use lifelines fitter to compute survival curves from baseline covariates,** events and durations

  stratify (bool): if True, fit a separate model per treatment group outcome_covariates (array): Covariates to use for outcome model.

  If None - all covariates passed will be used. Either list of column names or boolean mask.

  weight_covariates (array): Covariates to use for weight model. If None - all covariates passed will be used. Either list of column names or boolean mask.

estimate_individual_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: Optional[Any] = None*, *timeline_start: Optional[int] = None*, *timeline_end: Optional[int] = None*) → pandas.core.frame.DataFrame
Returns individual survival curves for each subject row in X/a/t

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup durations, size (num_subjects,).
- **y** – NOT USED (for API compatibility only).

- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event (t.min()).
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event (t.max()).

Returns with time-step index, subject IDs (X.index) as columns and point survival as entries

Return type pd.DataFrame

fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *w: Optional[pandas.core.series.Series] = None*, *fit_kwargs: Optional[dict] = None*)

Fits parametric models and calculates internal survival functions.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – Followup duration, size (num_subjects,).
- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **w** (*pd.Series*) – NOT USED (for compatibility only) optional subject weights.
- **fit_kwargs** (*dict*) – Optional kwargs for fit call of survival model

Returns self

class causallib.survival.**WeightedSurvival**(*weight_model: Optional[causallib.estimation.base_weight.WeightEstimator] = None*, *survival_model: Optional[Any] = None*)

Bases: *causallib.survival.base_survival.SurvivalBase*

Weighted survival estimator

Weighted survival estimator. :param weight_model: causallib compatible weight model (e.g., IPW) :param survival_model: Three alternatives:

1. None - compute non-parametric KaplanMeier survival curve
2. **Scikit-Learn estimator (needs to implement *predict_proba*) - compute parametric curve by fitting a** time-varying hazards model
3. lifelines UnivariateFitter - use lifelines fitter to compute survival curves from events and durations

estimate_population_outcome(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: pandas.core.series.Series*, *y: pandas.core.series.Series*, *timeline_start: Optional[int] = None*, *timeline_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Returns population averaged survival curves.

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series | int*) – Followup durations, size (num_subjects,).

- **y** (*pd.Series*) – Observed outcome (1) or right censoring event (0), size (num_subjects,).
- **timeline_start** (*int*) – Common start time-step. If provided, will generate survival curves starting from 'timeline_start' for all patients. If None, will predict from first observed event.
- **timeline_end** (*int*) – Common end time-step. If provided, will generate survival curves up to 'timeline_end' for all patients. If None, will predict up to last observed event.

Returns with timestep index, treatment values as columns and survival as entries

Return type pd.DataFrame

fit(*X: pandas.core.frame.DataFrame*, *a: pandas.core.series.Series*, *t: Optional[pandas.core.series.Series] = None*, *y: Optional[pandas.core.series.Series] = None*, *fit_kwargs: Optional[dict] = None*)
Fits internal weight module (e.g. IPW module, adversarial weighting, etc).

Parameters

- **X** (*pd.DataFrame*) – Baseline covariate matrix of size (num_subjects, num_features).
- **a** (*pd.Series*) – Treatment assignment of size (num_subjects,).
- **t** (*pd.Series*) – NOT USED (for compatibility only)
- **y** (*pd.Series*) – NOT USED (for compatibility only)
- **fit_kwargs** (*dict*) – Optional kwargs for fit call of survival model (NOT USED, since fit call of survival model occurs in 'estimate_population_outcome' rather than here)

Returns self

3.1.3.9 causallib.utils package

3.1.3.9.1 Submodules

causallib.utils.crossfit module

causallib.utils.crossfit.**cross_fitting**(*estimator*, *X*, *y*, *n_splits=5*, *predict_proba=False*, *return_estimator=True*)

Parameters

- **estimator** (*object*) – sklearn object
- **X** (*pd.DataFrame*) – Covariate matrix of size (num_subjects, num_features).
- **y** (*pd.Series*) – Observed outcome of size (num_subjects,).
- **n_splits** (*int*) – number of folds
- **predict_proba** (*bool*) –

  If True, the treatment model is a classifier and use 'predict_proba',

  If False, use 'predict'.
- **return_estimator** (*bool*) – If true return fitted estimators of each fold

Returns

array of held-out prediction, if return estimator:

a tuple of estimators on held-out-data

causallib.utils.general_tools module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Created on Jun 27, 2018

General (i.e. non-scientific) utils used throughout the package.

causallib.utils.general_tools.**check_learner_is_fitted**(*learner*)
Return True if fitted and False otherwise

causallib.utils.general_tools.**create_repr_string**(*o*)

Parameters **o** (*object*) – any core object

Returns repr string based on internal attributes

Return type str

causallib.utils.general_tools.**get_iterable_treatment_values**(*treatment_values*, *treatment_assignment*, *sort=True*)
Convert an optionally provided specification of unique treatment values to an iterable of the unique treatment options. Since user can provide treatment values as either an iterable or a single value, this conversion to an iterable allows a generic approach of going over all provided treatment values.

Parameters

- **treatment_values** (*None | Any | list[Any]*) – Unique values of possible treatment values. Can be either one value (scalar) or list of values (any iterable). Can be None, if None - treatment values are inferred from treatment assignment.
- **treatment_assignment** (*Series*) – The observed treatment assignment, used to infer a list of unique treatment values in case no treatment values are provided (None is passed to treatment_values).
- **sort** (*bool*) – Whether to sort the treatment values

Returns list of unique treatment values.

Return type list[Any]

causallib.utils.stat_utils module

(C) Copyright 2019 IBM Corp.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

causallib.utils.stat_utils.**areColumnsBinary**(*X*)

Assess whether all matrix columns are binary. :param X: Covariate matrix. :type X: np.ndarray | pdDataFrame

Returns

A boolean vector the length of number of features (columns). An entry is True iff the corresponding column is binary.

Return type np.ndarray

causallib.utils.stat_utils.**calc_weighted_ks2samp**(*x*, *y*, *wx*, *wy*)

Weighted Kolmogorov-Smirnov

References

[1] https://stackoverflow.com/a/40059727

causallib.utils.stat_utils.**calc_weighted_standardized_mean_differences**(*x*, *y*, *wx*, *wy*, *weighted_var=False*)

Standardized mean difference: frac{mu_1 - mu_2 }{sqrt{sigma_1^2 + sigma_2^2}}

References

[1]https://cran.r-project.org/web/packages/cobalt/vignettes/cobalt_A0_basic_use.html#details-on-calculations
[2]https://en.wikipedia.org/wiki/Strictly_standardized_mean_difference#Concept

Note on variance: - The variance is calculated on unadjusted to avoid paradoxical situation when adjustment decreases both the

mean difference and the spread of the sample, yielding a larger smd than that prior to adjustment, even though the adjusted groups are now more similar [1].

- The denominator is as depicted in the "statistical estimation" section: https://en.wikipedia.org/wiki/Strictly_standardized_mean_difference#Statistical_estimation, namely, disregarding the covariance term [2], and is unweighted as suggested above in [1].

causallib.utils.stat_utils.**chi2_test**(*X*, *y*)

Parameters

- **X** (*np.ndarray*) – Binary feature matrix
- **y** (*np.ndarray*) – Binary response vector

Returns A vector of p-values, one for every feature.

Return type np.ndarray

causallib.utils.stat_utils.**computeCorrPvals**(*X*, *y*, *is_X_binary*, *is_y_binary*, *isLinear=True*)

Parameters

- **X** (*pd.DataFrame*) – The covariate matrix
- **y** (*pdSeries*) – The response
- **is_X_binary** (*np.ndarray*) – Indication which columns are binary
- **is_y_binary** (*bool*) – Indication whether the response vector is binary or not.
- **isLinear** (*bool*) – Whether to perform a linear (slope) test (t-test) on the non-binary features or to perform a two-sample Kolmogorov-Smirnov test

Returns A vector of p-values, one for every feature.

Return type np.array

causallib.utils.stat_utils.**isBinary**(*x*)

Asses whether a vector is binary. :param x: :type x: pdSeries | np.ndarray

Returns True iff x is binary.

Return type bool

causallib.utils.stat_utils.**is_vector_binary**(*vec*)

causallib.utils.stat_utils.**robust_lookup**(*df*, *indexer*)

Robust way to apply pandas lookup when indices are not unique

Parameters

- **df** (*pdDataFrame*) –
- **indexer** (*pdSeries*) – A Series whose index is either same or a subset of *df.index* and whose values are values from *df.columns*. If *a.index* contains values not in *df.index* they will have NaN values.

Returns

a vector where (logically) ***extracted[i] = df.loc[indexer.index[i], indexer[i]].*** In most cases, when *indexer.index == df.index* this translates to *extracted[i] = df.loc[i, indexer[i]]*

Return type pdSeries

causallib.utils.stat_utils.**which_columns_are_binary**(*X*)

Parameters **X** (*pdDataFrame*) –

Returns:

3.1.3.9.2 Module contents

3.1.4 Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

F

G

H

I

K

L

M

O

P

R

S

T

U

V

W

X

Y