Do-sampler

While most potential-outcomes based estimators focus on estimating the specific contrast, Pearlian inference focuses on more fundamental quantities like the joint distribution of a set of outcomes Y, which can be used to derive other statistics of interest.

Generally, it's hard to represent a probability distribution non-parametrically. Even if you could, you wouldn't want to gloss over finite-sample problems with you data you used to generate it. With these issues in mind, we decided to represent interventional distributions by sampling from them with an object called to "do-sampler". With these samples, we can hope to compute finite-sample statistics of our interventional data. If we bootstrap many such samples, we can even hope for good sampling distributions for these statistics.

The user should note that this is still an area of active research, so you should be careful about being too confident in bootstrapped error bars from do-samplers. Note that do samplers sample from the outcome distribution, and so will vary significantly from sample to sample. To use them to compute outcomes, it's recommended to generate several such samples to get an idea of the posterior variance of your statistic of interest.

Pearlian Interventions

Following the notion of an intervention in a Pearlian causal model, our do-samplers implement a sequence of steps:

- Disrupt causes
- Make Effective
- Propagate and sample

In the first stage, we imagine cutting the in-edges to all of the variables we're intervening on. In the second stage, we set the value of those variables to their interventional quantities. In the third stage, we propagate that value forward through our model to compute interventional outcomes

Skip to main content

In practice, there are many ways we can implement these steps. They're most explicit when we build the model as a linear bayesian network in PyMC3, which is what underlies the MCMC do sampler. In that case, we fit one bayesian network to the data, then construct a new network representing the interventional network. The structural equations are set with the parameters fit in the initial network, and we sample from that new network to get our do sample.

In the weighting do sampler, we abstractly think of "disrupting the causes" by accounting for selection into the causal state through propensity score estimation. These scores contain the information used to block back-door paths, and so have the same statistics effect as cutting edges into the causal state. We make the treatment effective by selecting the subset of our data set with the correct value of the causal state. Finally, we generated a weighted random sample using inverse propensity weighting to get our do sample.

There are other ways you could implement these three steps, but the formula is the same. We've abstracted them out as abstract class methods which you should override if you'd like to create your own do sampler!

Statefulness

The do sampler when accessed through the high-level pandas API is stateless by default.This makes it intuitive to work with, and you can generate different samples with repeated calls to the pandas.DataFrame.causal.do. It can be made stateful, which is sometimes useful.

The 3-stage process we mentioned before is implemented by passing an internal pandas.DataFrame through each of the three stages, but regarding it as temporary. The internal dataframe is reset by default before returning the result.

It can be much more efficient to maintain state in the do sampler between generating samples. This is especially true when step 1 requires fitting an expensive model, as is the case with the MCMC do sampler, the kernel density sampler, and the weighting sampler. Instead of re-fitting the model for each sample, you'd like to fit it once, and then generate many samples from the do sampler. You can do this by setting the kwarg stateful=True when you call the pandas.DataFrame.causal.do method. To reset the state of the dataframe (deleting the model as well as the internal dataframe), you can call the pandas.DataFrame.causal.reset method.

Through the lower-level API, the sampler is stateful by default. The assumption is that a "power

Skip to main content

case, state is carried by internal dataframe self._df, which is a copy of the dataframe passed on instantiation. The original dataframe is kept in self._data, and is used when the user resets state.

Using do-sampler

The do-sampler is built on top of the identification abstraction used throughout dowhy. It uses a dowhy.CausalModel to perform identification, and builds any models it needs automatically using this identification.

For details, check out the do-sampler notebook.

Previous
Propensity-based methods

Next
Estimating average causal effect with natural experiments

© Copyright 2022, PyWhy contributors.

Created using Sphinx 7.1.2.

Built with the PyData Sphinx Theme 0.14.4.