

Causal Inference in Python: A Vignette

Laurence Wong

August 23, 2015

In this document we illustrate the use of `CausalInference` with a simple simulated data set.

Simulated Data

The data generating process that generated the example data set was chosen so that:

- i.) Unconfoundedness, $(Y(0), Y(1)) \perp\!\!\!\perp D \mid X$, is satisfied.
- ii.) Unconditionally, selection is not random. In particular, the probability of being assigned treatment is a function of the covariates X .
- iii.) Potential outcomes $Y(0)$ and $Y(1)$ are nonlinear functions of X plus random error.

Property (i) ensures that the tools available in `CausalInference` are actually appropriate. Properties (ii) and (iii) were adopted to illustrate how standard linear methods can fail even if selection is only on observables. A detailed description of the data simulation can be found in the Appendix.

Initialization

The main object of interest in `CausalInference` is the class `CausalModel`. It takes as inputs three NumPy arrays: \mathbf{Y} , an N -vector of observed outcomes; \mathbf{D} , an N -vector of treatment status indicators; and \mathbf{X} , an N -by- K matrix of covariates. To initialize a `CausalModel` instance, simply run:

```
>>> causal = CausalModel(Y, D, X)
```

Summary Statistics

Once an instance of the class `CausalModel` has been created, we can compute some basic summary statistics. This can be done by running:

```
>>> print causal.summary_stats
Summary Statistics
```

Variable	Controls (N_c=392)		Treated (N_t=608)		Raw-diff
	Mean	S.d.	Mean	S.d.	
Y	43.097	31.353	90.911	41.815	47.814

Variable	Controls (N_c=392)		Treated (N_t=608)		Nor-diff
	Mean	S.d.	Mean	S.d.	
X0	3.810	2.950	5.762	2.566	0.706
X1	3.436	2.848	5.849	2.634	0.880

The attribute `summary_stats` is in reality just a dictionary-like object with special method defined to enable the display of the above table. In many situations it is more convenient to simply access the relevant statistic directly. To retrieve the vector of covariate mean for the treatment group, for example, we simply run:

```
>>> causal.summary_stats['X_t_mean']
array([ 5.76232357,  5.8489734 ])
```

Since `summary_stats` behaves like a dictionary, it is equipped with the usual Python dictionary methods. To list the dictionary keys, for instance, we go:

```
>>> causal.summary_stats.keys()
['Y_c_mean', 'X_t_sd', 'N_t', 'K', 'ndiff', 'N', 'Y_t_sd', 'rdiff', 'Y_t_mean',
'X_c_mean', 'X_t_mean', 'Y_c_sd', 'X_c_sd', 'N_c']
```

Most of the statistics appearing in the summary table should be self-explanatory, with the possible exception of the normalized differences in average covariates. This statistic is defined as

$$\frac{\bar{X}_{k,t} - \bar{X}_{k,c}}{\sqrt{(s_{k,t}^2 + s_{k,c}^2) / 2}},$$

where $\bar{X}_{k,t}$ and $s_{k,t}$ are the sample mean and sample standard deviation of the k th covariate of the treatment group, and $\bar{X}_{k,c}$ and $s_{k,c}$ are the analogous statistics for the control group.

The normalized differences in average covariates provide a way to measure the covariate balance between the treatment and the control groups. Unlike the t-statistic, its absolute magnitude does not increase (in expectation) as the sample size increases.

Propensity Score Estimation

When there is indication of covariate imbalance, we may wish to construct a sample where the treatment and control groups are more similar than the original full sample.

```
>>> causal.est_propensity_s()
>>> print causal.propensity
Estimated Parameters of Propensity Score
```

	Coef.	S.e.	z	P> z	[95% Conf. int.]	
Intercept	-2.839	0.526	-5.401	0.000	-3.870	-1.809
X1	0.486	0.153	3.178	0.001	0.186	0.786
X0	0.466	0.155	3.011	0.003	0.163	0.770
X1*X0	0.080	0.015	5.391	0.000	0.051	0.109
X0*X0	-0.045	0.012	-3.579	0.000	-0.069	-0.020
X1*X1	-0.045	0.013	-3.542	0.000	-0.070	-0.020

Like `summary_stats`, the `propensity` attribute is another dictionary-like container of results. The dictionary keys of `propensity` can be found by running:

```
>>> causal.propensity.keys()
['coef', 'lin', 'qua', 'loglike', 'fitted', 'se']
```

The estimated propensity scores can be recovered by accessing `causal.propensity['fitted']`. Though we won't make direct use of it, most of the propensity-based techniques discussed subsequently are based on this vector.

Improving Covariate Balance

```
>>> causal.trim_s()
>>> causal.cutoff
0.095492801025642338
```

We can now print `summary_stats` again to view the summary statistics of the trimmed sample.

```
>>> print causal.summary_stats
Summary Statistics
```

Variable	Controls (N_c=371)		Treated (N_t=362)		Raw-diff
	Mean	S.d.	Mean	S.d.	
Y	41.331	29.608	65.984	28.102	24.653

Variable	Controls (N_c=371)		Treated (N_t=362)		Nor-diff
	Mean	S.d.	Mean	S.d.	
X0	3.709	2.872	4.645	2.514	0.347
X1	3.407	2.784	4.674	2.509	0.478

Stratifying the Sample

```
>>> causal.stratify_s()
>>> print causal.strata
Stratification Summary
```

Stratum	Propensity Score		Sample Size		Ave. Propensity		Outcome Raw-diff
	Min.	Max.	Controls	Treated	Controls	Treated	
1	0.096	0.265	157	27	0.188	0.190	10.052
2	0.266	0.474	111	72	0.360	0.367	12.025
3	0.477	0.728	70	113	0.598	0.601	11.696
4	0.728	0.836	23	69	0.781	0.787	10.510
5	0.838	0.904	10	81	0.865	0.873	3.405

```
>>> for stratum in causal.strata:
...     stratum.est_via_ols(adj=1)
...
>>> [stratum.estimates['ols']['ate'] for stratum in causal.strata]
[9.9532273440618457, 9.2918973715823778, 9.6787670925744465, 9.6722830043582491,
9.2239596078237955]
```

Treatment Effect Estimation

```
>>> causal.est_via_blocking()
>>> causal.est_via_matching(bias_adj=True)
```

```
>>> print causal.estimates
Treatment Effect Estimates: Blocking
```

	Est.	S.e.	z	P> z	[95% Conf. int.]	
ATE	9.594	0.373	25.706	0.000	8.862	10.325
ATC	9.667	0.511	18.911	0.000	8.665	10.668
ATT	9.519	0.331	28.723	0.000	8.870	10.169

```
Treatment Effect Estimates: Matching
```

	Est.	S.e.	z	P> z	[95% Conf. int.]	
ATE	9.573	0.243	39.469	0.000	9.098	10.049
ATC	9.544	0.272	35.035	0.000	9.010	10.078
ATT	9.603	0.313	30.723	0.000	8.990	10.216

Once an instance of the class `CausalModel` has been created, it will contain a number of attributes and methods that are relevant for conducting a causal analysis. Tables 1 and 2 contain a brief description of these attributes and methods.

Attribute	Description
<code>summary_stats</code>	Dictionary-like object containing summary statistics for the covariate variables.
<code>propensity</code>	Dictionary-like object containing propensity score data, including estimated logistic regression coefficients, predicted propensity score, maximized log-likelihood, and the lists of the linear and quadratic terms that are included in the regression.
<code>cutoff</code>	Floating point number specifying the cutoff point for trimming on propensity score.
<code>blocks</code>	Either an integer indicating the number of equal-sized blocks to stratify the sample into, or a list of ascending numbers specifying the boundaries of the strata.
<code>strata</code>	List-like object containing the list of stratified propensity bins.
<code>estimates</code>	Dictionary-like object containing treatment effect estimates for each estimator used.

Table 1: Attributes of the class `CausalModel`. Invoking `print` on any of the dictionary- or list-like attribute above yields customized summary tables. Note that some attributes are only created after the relevant methods have been called.

Method	Description
<code>reset</code>	Reinitializes data to original inputs, and drop any estimated results.
<code>est_propensity</code>	Estimates via logit the propensity score using specified linear and quadratic terms.
<code>est_propensity_s</code>	Estimates via logit the propensity score using the covariate selection algorithm of Imbens and Rubin (2015).
<code>trim</code>	Trims data based on propensity score using the threshold specified by the attribute <code>cutoff</code> .
<code>trim_s</code>	Trims data based on propensity score using the cutoff selected by the procedure of Crump, Hotz, Imbens, and Mitnik (2008).
<code>stratify</code>	Stratifies the sample based on propensity score as specified by the attribute <code>blocks</code> .
<code>stratify_s</code>	Stratifies the sample based on propensity score using the bin selection procedure suggested by Imbens and Rubin (2015).
<code>est_via_blocking</code>	Estimates average treatment effects using regression within blocks.
<code>est_via_matching</code>	Estimates average treatment effects using matching with replacement.
<code>est_via_weighting</code>	Estimates average treatment effects using the Horvitz-Thompson weighting estimator modified to incorporate covariates.
<code>est_via_ols</code>	Estimates average treatment effects using least squares.

Table 2: Methods of the class `CausalModel`. Invoke `help` on any of the above methods for more detailed documentation.

Installation

`CausalInference` can be installed using `pip`, and will run provided the necessary dependencies are in place. On Ubuntu systems, the following commands should take care of all the essential steps if you are starting from scratch:

```
$ sudo apt-get update
$ sudo apt-get install python-pip python-numpy python-scipy
$ sudo pip install causalinference
```