Causal Inference in Python: A Vignette

Laurence Wong

August 30, 2015

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

1 Initialization

The main object of interest in CausalInference is the class `CausalModel`, which we can import with

```
>>> from causalinference import CausalModel
```

`CausalModel` takes as inputs three NumPy arrays: `Y`, an N-vector of observed outcomes; `D`, an N-vector of treatment status indicators; and `X`, an N-by-K matrix of covariates. To initialize a `CausalModel` instance, simply run:

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

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.

`CausalModel` is *stateful*. As we employ some of the methods to be discussed subsequently, the instance `causal` will mutate, with new data being added or existing data being modified or dropped. Running

```
>> causal.reset()
```

will return `causal` to its initial state.

2 Summary Statistics

Once an instance of the class `CausalModel` has been created, basic summary statistics will be computed and stored in the attribute `summary_stats`. We can display it by running:

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

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

                       Controls (N_c=392)         Treated (N_t=608)
       Variable         Mean         S.d.         Mean         S.d.     Nor-diff
--------------------------------------------------------------------------------
             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

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

where $\bar{X}_{k,t}$ and $s_{k,t}$ are the sample mean and sample standard deviation of the kth 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.

3 Propensity Score Estimation

The propensity score, defined as the probability of getting treatment conditional on the covariates, plays a central role in much of what follows. Two methods, `est_propensity` and `est_propensity_s`, are provided for propensity score estimation. Both involve running a logistic regression of the treatment indicator D on functions of the covariates. `est_propensity` allows the user to specify the covariates to include linearly and/or quadratically, while `est_propensity_s` will make this choice automatically based on a sequence of likelihood ratio tests.

In the following, we run `est_propensity_s` and display the estimation results. In this example, the specification selection algorithm decided to include both covariates and all the interaction and quadratic terms.

```
>>> 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` attribule is in reality 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 calls to it, most of the propensity-based techniques discussed subsequently are based on this vector.

4 Improving Covariate Balance

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. One way of doing so is by dropping units with extreme values of propensity score. For these subjects, their covariate values are such that the probability of being in the treatment (or control) group is so overwhelmingly high that we cannot reliably find comparable units in the opposite group. We may wish to forego estimating treatment effects for such units since nothing much can be credibly said about them.

A good rule-of-thumb is to drop units whose estimated propensity score is less than $\alpha = 0.1$ or greater than $1 - \alpha = 0.9$. By default, once the propensity score has been estimated by running either `est_propensity` or `est_propensity_s`, a value of 0.1 will be set for the attribute `cutoff`:

```
>>> causal.cutoff
0.1
```

Calling `causal.trim()` at this point will drop every unit that has propensity score outside of the $[\alpha, 1 - \alpha]$ interval. Alternatively, a procedure exists that will estimate the optimal cutoff. The method `trim_s` will perform this calculation, set the `cutoff` to the optimal α, and then invoke `trim` to construct the subsample. For our example, the optimal α was estimated to be slightly less than 0.1:

```
>>> causal.trim_s()
>>> causal.cutoff
0.0954928016329
```

If we now print `summary_stats` again to view the summary statistics of the trimmed sample, we see that the normalized differences in average covariates has fallen noticeably.

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

                       Controls (N_c=371)         Treated (N_t=363)
       Variable         Mean         S.d.         Mean         S.d.     Raw-diff
--------------------------------------------------------------------------------
              Y       41.331       29.608       66.067       28.108       24.736

                       Controls (N_c=371)         Treated (N_t=363)
       Variable         Mean         S.d.         Mean         S.d.     Nor-diff
--------------------------------------------------------------------------------
             X0        3.709        2.872        4.658        2.522        0.351
             X1        3.407        2.784        4.661        2.517        0.472
```

5 Stratifying the Sample

With the propensity score estimated, one may wish to stratify the sample into blocks that have units that are more similar in terms of their covariates. This makes the treatment and control groups within each propensity bin more comparable, and therefore treatment effect estimates more credible.

CausalInference provides two methods for subclassification based on propensity score. The first, `stratify`, splits the sample based on what is specified in the attribute `blocks`. The default value of `blocks` is set to 5, which means that `stratify` will split the sample into 5 equal-sized bins. In contrast, the second method, `stratify_s`, will use a data-driven procedure for selecting both the number of blocks and their boundaries, with the expectation that the number of blocks should increase with the sample size.

To inspect the results of the stratification, we can invoke `print` on the attribute `strata` to display some summary statistics, as follows:

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

              Propensity Score         Sample Size     Ave. Propensity   Outcome
   Stratum      Min.      Max.  Controls   Treated  Controls   Treated  Raw-diff
--------------------------------------------------------------------------------
         1     0.095     0.265       157        28     0.188     0.187    11.885
         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
```

Under the hood, the attribute `strata` is actually a list-like object that contains, as each of its elements, a full instance of the class `CausalModel`, with the input data being those that correspond to the units that are in the propensity bin. We can thus, for example, access each stratum and inspect its `summary_stats` attribute, or as the following illustrates, loop through `strata` and estimate within-bin treatment effects using least squares.

```
>>> for stratum in causal.strata:
...     stratum.est_via_ols(adj=1)
...
>>> [stratum.estimates['ols']['ate'] for stratum in causal.strata]
[10.379170390195197, 9.2918973715823707, 9.67876709257445, 9.6722830043583023,
9.2239596078238222]
```

Note these estimates are much more stable and closer to the true value of 10 than the within-bin raw differences in average outcomes that were reported in the stratification summary table, highlighting the virtue of further controlling for covariates even within blocks.

Instead of manually looping through the `strata` attribute, estimating within-bin treatment effects, and then averaging appropriately to arrive at an overall estimate, we can also simply call `est_via_blocking`. We will report the resulting estimates in the next section along with estimates obtained from other, alternative estimators.

6 Treatment Effect Estimation

In addition to least squares and the blocking estimator described in the last section, CausalInference provides two alternative treatment effect estimators. The first is the nearest neighborhood matching estimator of Abadie and Imbens (2006). Instead of relying on the propensity score, this estimator constructs matching treatment and control units by matching directly on the covariate vectors themselves. The method `est_via_matching` implements this estimator, and provides several optional arguments that can be used to specify certain key parameters.

The last estimator is a version of the Horvitz-Thompson estimator, modified to further adjust for covariates. Mechanically, this involves running the following weight least squares regression:

$$Y_i = \alpha + \tau D_i + X_i \beta + \varepsilon_i,$$

where the weight for unit i is $1/\hat{p}(X)$ if i is in the treatment group, and $1/\big(1 - \hat{p}(X)\big)$ if i is in the control group. This estimator is also sometimes called the doubly-robust estimator, referring to the fact that this estimator is consistent if either the specification of the propensity score is correct, or the specification of the regression function is correct. We can invoke it by calling `est_via_weighting`.

In the following we invoke each of the four estimators (including least squares, since the input data has changed now that the sample has been trimmed), and print out the resulting estimates.

```
>>> causal.est_via_ols()
>>> causal.est_via_weighting()
>>> causal.est_via_blocking()
>>> causal.est_via_matching(bias_adj=True)
Treatment Effect Estimates: Weighting

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE     17.821      1.684     10.585      0.000     14.521     21.121
```

```
Treatment Effect Estimates: OLS

                   Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
         ATE      2.913      0.803      3.627      0.000      1.339      4.487
         ATC      2.435      0.824      2.956      0.003      0.820      4.049
         ATT      3.401      0.885      3.843      0.000      1.667      5.136

Treatment Effect Estimates: Blocking

                   Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
         ATE      9.702      0.381     25.444      0.000      8.954     10.449
         ATC      9.847      0.527     18.701      0.000      8.815     10.879
         ATT      9.553      0.332     28.771      0.000      8.903     10.204

Treatment Effect Estimates: Matching

                   Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
         ATE      9.624      0.245     39.354      0.000      9.145     10.103
         ATC      9.642      0.270     35.776      0.000      9.114     10.170
         ATT      9.606      0.318     30.159      0.000      8.981     10.230
```

7 Simulated Data

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

i.) Unconfoundedness, $\big(Y(0), Y(1)\big) \perp\!\!\!\perp D \,|\, X$, 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.

Attribute	Description
`summary_stats`	Dictionary-like object containing summary statistics for the covariate variables.
`propensity`	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.
`cutoff`	Floating point number specifying the cutoff point for trimming on propensity score.
`blocks`	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 each stratum.
`strata`	List-like object containing the list of stratified propensity bins.
`estimates`	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.

References

Abadie, A., & Imbens, G. (2006). Large sample properties of matching estimators for average treatment effects. *Econometrica*, *74*, 235-267.

Crump, R., Hotz, V. J., Imbens, G., & Mitnik, O. (2009). Dealing with limited overlap in estimation of average treatment effects. *Biometrika*, *96*, 187-199.

Imbens, G. W., & Rubin, D. B. (2015). *Causal inference in statistics, social, and biomedical sciences: An introduction.* Cambridge University Press.

Method	Description
`reset`	Reinitializes data to original inputs, and drop any estimated results.
`est_propensity`	Estimates via logistic regression the propensity score using specified linear and quadratic terms.
`est_propensity_s`	Estimates via logistic regression the propensity score using the covariate selection algorithm of Imbens and Rubin (2015).
`trim`	Trims data based on propensity score using the threshold specified by the attribute `cutoff`.
`trim_s`	Trims data based on propensity score using the cutoff selected by the procedure of Crump, Hotz, Imbens, and Mitnik (2009).
`stratify`	Stratifies the sample based on propensity score as specified by the attribute `blocks`.
`stratify_s`	Stratifies the sample based on propensity score using the bin selection procedure suggested by Imbens and Rubin (2015).
`est_via_blocking`	Estimates average treatment effects using regression within blocks.
`est_via_matching`	Estimates average treatment effects using matching with replacement.
`est_via_weighting`	Estimates average treatment effects using the Horvitz-Thompson weighting estimator modified to incorporate covariates.
`est_via_ols`	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.