causal$_{c}urve$

Release 1.0.6

Apr 20, 2022

Getting Started

1 Introduction to causal-curve 1
1.1 Interpreting the causal curve . . . 1
1.2 A caution about causal inference assumptions . . . 3
1.3 References . . . 4

2 Installation, testing and development 5
2.1 Dependencies . . . 5
2.2 User installation . . . 6
2.3 Testing . . . 6
2.4 Development . . . 6

3 Contributing guide 7
3.1 Submitting a bug report or a feature request . . . 7
3.2 Contributing code . . . 8
3.3 Contributing to the documentation . . . 9

4 Health data: generating causal curves and examining mediation 11
4.1 The causal effect of blood lead levels on cognitive performance in children . . . 11
4.2 Do blood lead levels mediate the relationship between poverty and cognitive performance? . . . 14
4.3 References . . . 15

5 GPS_Regressor Tool (continuous treatments, continuous outcomes) 17
5.1 References . . . 19

6 GPS_Classifier Tool (continuous treatments, binary outcomes) 21
6.1 References . . . 23

7 TMLE_Regressor Tool (continuous treatments, continuous outcomes) 25
7.1 References . . . 27

8 Mediation Tool (continuous treatment, mediator, and outcome) 29
8.1 References . . . 31

9 causal_curve 33
9.1 causal_curve package . . . 33

10 Change Log 45
10.1 Version 1.0.6 . . . 45

10.2 Version 1.0.5 45
10.3 Version 1.0.4 45
10.4 Version 1.0.3 45
10.5 Version 1.0.2 45
10.6 Version 1.0.1 46
10.7 Version 1.0.0: **Major Update** 46
10.8 Version 0.5.2 46
10.9 Version 0.5.1 46
10.10 Version 0.5.0 46
10.11 Version 0.4.1 46
10.12 Version 0.4.0 46
10.13 Version 0.3.8 47
10.14 Version 0.3.7 47
10.15 Version 0.3.6 47
10.16 Version 0.3.5 47
10.17 Version 0.3.4 47
10.18 Version 0.3.3 47
10.19 Version 0.3.2 47
10.20 Version 0.3.1 48
10.21 Version 0.3.0 48
10.22 Version 0.2.4 48
10.23 Version 0.2.3 48
10.24 Version 0.2.2 48
10.25 Version 0.2.1 48
10.26 Version 0.2.0 48
10.27 Version 0.1.3 48
10.28 Version 0.1.2 49
10.29 Version 0.1.1 49
10.30 Version 0.1.0 49
10.31 Version 0.0.10 49
10.32 Version 0.0.9 49

11 Citation **51**

12 Summary **53**

13 Quick example (of the `GPS_Regressor` tool) **55**

Python Module Index **57**

Index **59**

CHAPTER 1

Introduction to causal-curve

In academia and industry, randomized controlled experiments (or simply experiments or "A/B tests") are considered the gold standard approach for assessing the true, causal impact of a treatment or intervention. For example:

- We want to increase the number of times per day new customers log into our business's website. Will it help if we send daily emails out to our customers? We take a group of 2000 new business customers and half is randomly chosen to receive daily emails while the other half receives one email per week. We follow both groups forward in time for a month compare each group's average number of logins per day.

However, for ethical or financial reasons, experiments may not always be feasible to carry out.

- It's not ethical to randomly assign some people to receive a possible carcinogen in pill form while others receive a sugar pill, and then see which group is more likely to develop cancer.
- It's not feasible to increase the household incomes of some New York neighborhoods, while leaving others unchanged to see if changing a neighborhood's income inequality would improve the local crime rate.

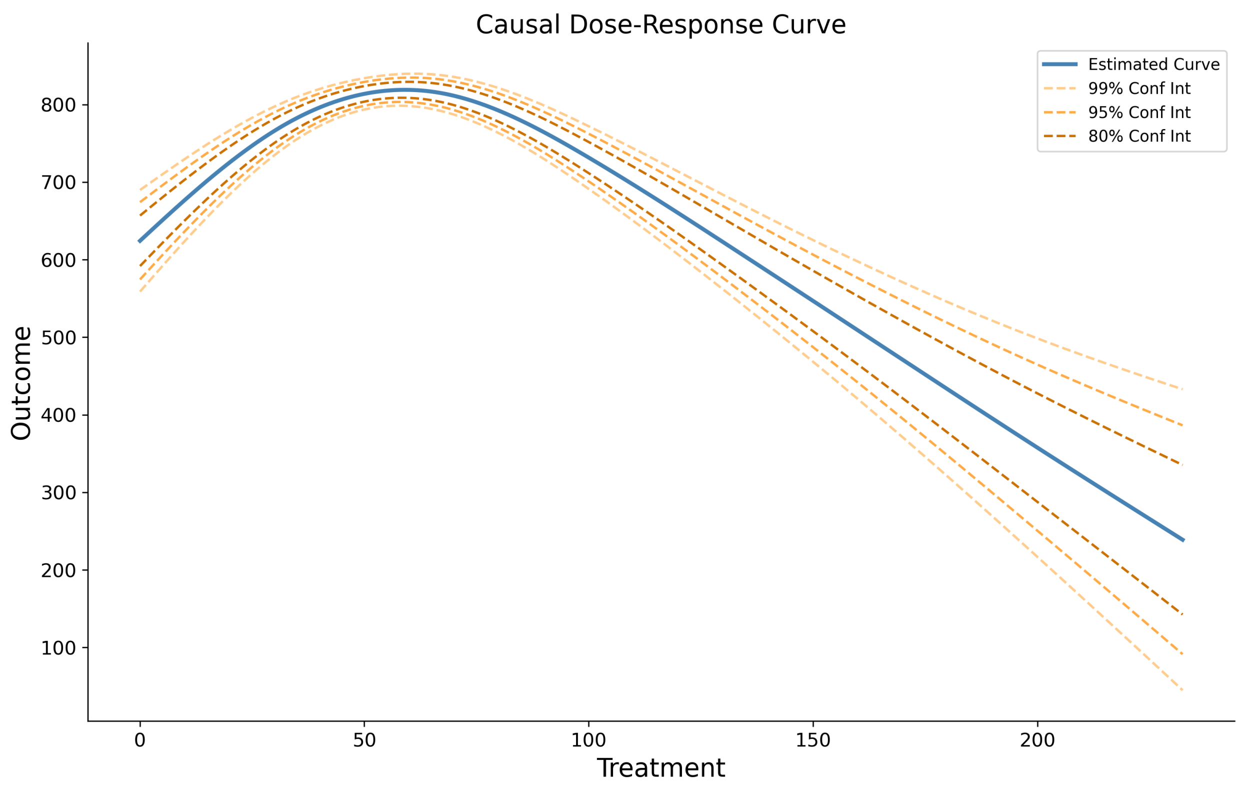
"Causal inference" methods are a set of approaches that attempt to estimate causal effects from observational rather than experimental data, correcting for the biases that are inherent to analyzing observational data (e.g. confounding and selection bias) [@Hernán:2020].

As long as you have varying observational data on some treatment, your outcome of interest, and potentially confounding variables across your units of analysis (in addition to meeting the assumptions described below), then you can essentially estimate the results of a proper experiment and make causal claims.

1.1 Interpreting the causal curve

Two of the methods contained within this package produce causal curves for continuous treatments (see the GPS and TMLE methods). Both continuous and binary treatments can be modeled (only the *GPS_Classifier* tool can handle binary outcomes).

Continuous outcome:

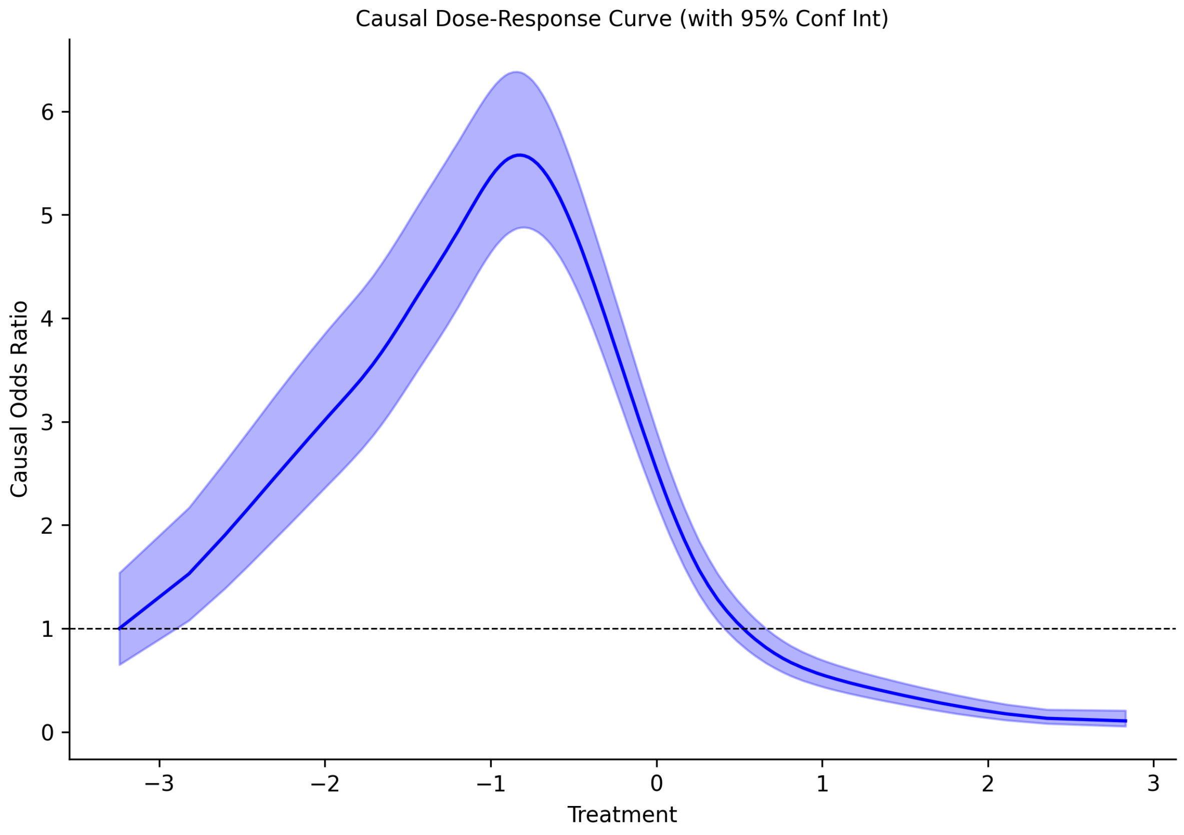


Using the above causal curve as an example, we see that employing a treatment value between 50 - 60 causally produces the highest outcome values. We also see that the treatment produces a smaller effect if lower or higher than that range. The confidence intervals become wider on the parts of the curve where we have fewer data points (near the minimum and maximum treatment values).

This curve differs from a simple bivariate plot of the treatment and outcome or even a similar-looking plot generated through standard multivariable regression modeling in a few important ways:

- This curve represents the estimated causal effect of a treatment on an outcome, not the association between treatment and outcome.
- This curve represents a population-level effect, and should not be used to infer effects at the individual-level (or whatever the unit of analysis is).
- To generate a similar-looking plot using multivariable regression, you would have to hold covariates constant, and any treatment effect that is inferred occurs within the levels of the covariates specified in the model. The causal curve averages out across all of these strata and gives us the population marginal effect.

Binary outcome:



In the case of binary outcome, the *GPS_Classifier* tool can be used to estimate a curve of odds ratios. Every point on the curve is relative to the lowest treatment value. The highest effect (relative to the lowest treatment value) is around a treatment value of -1.2. At this point in the treatment, the odds of a positive class occurring is 5.6 times higher compared with the lowest treatment value. This curve is always on the relative scale. This is why the odds ratio for the lowest point is always 1.0, because it is relative to itself. Odds ratios are bounded [0, inf] and cannot take on a negative value. Note that the confidence intervals at any given point in the curve isn't symmetric.

1.2 A caution about causal inference assumptions

There is a well-documented set of assumptions one must make to infer causal effects from observational data. These are covered elsewhere in more detail, but briefly:

- Causes always occur before effects: The treatment variable needs to have occurred before the outcome.
- SUTVA: The treatment status of a given individual does not affect the potential outcomes of any other individuals.
- Positivity: Any individual has a positive probability of receiving all values of the treatment variable.
- Ignorability: All major confounding variables are included in the data you provide.

Violations of these assumptions will lead to biased results and incorrect conclusions!

In addition, any covariates that are included in *causal-curve* models are assumed to only be **confounding** variables.

None of the methods provided in causal-curve rely on inference via instrumental variables, they only rely on the data from the observed treatment, confounders, and the outcome of interest (like the above GPS example).

1.3 References

Hernán M. and Robins J. Causal Inference: What If. Chapman & Hall, 2020.

Ahern J, Hubbard A, and Galea S. Estimating the Effects of Potential Public Health Interventions on Population Disease Burden: A Step-by-Step Illustration of Causal Inference Methods. American Journal of Epidemiology. 169(9), 2009. pp.1140–1147.

CHAPTER 2

Installation, testing and development

2.1 Dependencies

causal-curve requires:

- black
- coverage
- future
- joblib
- numpy
- numpydoc
- pandas
- patsy
- progressbar2
- pygam
- pytest
- python-dateutil
- python-utils
- pytz
- scikit-learn
- scipy
- six
- sphinx_rtd_theme
- statsmodels

2.2 User installation

If you already have a working installation of numpy, pandas, pygam, scipy, and statsmodels, you can easily install causal-curve using `pip`:

```
pip install causal-curve
```

You can also get the latest version of causal-curve by cloning the repository:

```
git clone https://github.com/ronikobrosly/causal-curve.git
cd causal-curve
pip install .
```

2.3 Testing

After installation, you can launch the test suite from outside the source directory using `pytest`:

```
pytest
```

2.4 Development

Please reach out if you are interested in adding additional tools, or have ideas on how to improve the package!

CHAPTER 3

Contributing guide

Thank you for considering contributing to causal-curve. Contributions from anyone are welcomed. There are many ways to contribute to the package, such as reporting bugs, adding new features and improving the documentation. The following sections give more details on how to contribute.

Important links:

- The project is hosted on GitHub: https://github.com/ronikobrosly/causal-curve

3.1 Submitting a bug report or a feature request

If you experience a bug using causal-curve or if you would like to see a new feature being added to the package, feel free to open an issue on GitHub: https://github.com/ronikobrosly/causal-curve/issues

3.1.1 Bug report

A good bug report usually contains:

- a description of the bug,
- a self-contained example to reproduce the bug if applicable,
- a description of the difference between the actual and expected results,
- the versions of the dependencies of causal-curve.

The last point can easily be done with the following commands:

```
import numpy; print("NumPy", numpy.__version__)
```

These guidelines make reproducing the bug easier, which make fixing it easier.

3.1.2 Feature request

A good feature request usually contains:

- a description of the requested feature,
- a description of the relevance of this feature to causal inference,
- references if applicable, with links to the papers if they are in open access.

This makes reviewing the relevance of the requested feature easier.

3.2 Contributing code

In order to contribute code, you need to create a pull request on https://github.com/ronikobrosly/causal-curve/pulls

3.2.1 How to contribute

To contribute to causal-curve, you need to fork the repository then submit a pull request:

1. Fork the repository.
2. Clone your fork of the causal-curve repository from your GitHub account to your local disk:

   ```
   git clone https://github.com/yourusername/causal-curve.git
   cd causal-curve
   ```

   where `yourusername` is your GitHub username.
3. Install the development dependencies:

   ```
   pip install pytest pylint black
   ```

4. Install causal-curve in editable mode:

   ```
   pip install -e .
   ```

5. Add the `upstream` remote. It creates a reference to the main repository that can be used to keep your repository synchronized with the latest changes on the main repository:

   ```
   git remote add upstream https://github.com/ronikobrosly/causal-curve.git
   ```

6. Fetch the `upstream` remote then create a new branch where you will make your changes and switch to it:

   ```
   git fetch upstream
   git checkout -b my-feature upstream/main
   ```

   where `my-feature` is the name of your new branch (it's good practice to have an explicit name). You can now start making changes.
7. Make the changes that you want on your new branch on your new local machine. When you are done, add the changed files using `git add` and then `git commit`:

   ```
   git add modified_files
   git commit
   ```

   Then push your commits to your GitHub account using `git push`:

```
git push origin my-feature
```

8. Create a pull request from your work. The base fork is the fork you would like to merge changes into, that is `ronikobrosly/causal-curve` on the `main` branch. The head fork is the repository where you made your changes, that is `yourusername/causal-curve` on the `my-feature` branch. Add a title and a description of your pull request, then click on **Create Pull Request**.

3.2.2 Pull request checklist

Before pushing to your GitHub account, there are a few rules that are usually worth complying with.

- **Make sure that your code passes tests**. You can do this by running the whole test suite with the `pytest` command. If you are experienced with `pytest`, you can run specific tests that are relevant for your changes. It is still worth it running the whole test suite when you are done making changes since it does not take very long. For more information, please refer to the pytest documentation. If your code does not pass tests but you are looking for help, feel free to do so (but mention it in your pull request).
- **Make sure to add tests if you add new code**. It is important to test new code to make sure that it behaves as expected. Ideally code coverage should increase with any new pull request. You can check code coverage using `pytest-cov`:

```
pip install pytest-cov
pytest --cov causal-curve
```

- **Make sure that the documentation renders properly**. To build the documentation, please refer to the *Contributing to the documentation* guidelines.
- **Make sure that your PR does not add PEP8 violations**. You can run *black* and *pylint* to only test the modified code. Feel free to submit another pull request if you find other PEP8 violations.

3.3 Contributing to the documentation

Documentation is as important as code. If you see typos, find docstrings unclear or want to add examples illustrating functionalities provided in causal-curve, feel free to open an issue to report it or a pull request if you want to fix it.

3.3.1 Building the documentation

Building the documentation requires installing some additional packages:

```
pip install sphinx==3.0.2 sphinx-rtd-theme numpydoc
```

To build the documentation, you must be in the `doc` folder:

```
cd doc
```

To generate the website with the example gallery, run the following command:

```
make html
```

The documentation will be generated in the `_build/html`. You can double click on `index.html` to open the index page, which will look like the first page that you see on the online documentation. Then you can move to the pages that you modified and have a look at your changes.

Finally, repeat this process until you are satisfied with your changes and open a pull request describing the changes you made.

CHAPTER 4

Health data: generating causal curves and examining mediation

To provide an end-to-end example of the sorts of analyses *cause-curve* can be used for, we'll begin with a health topic. A notebook containing the pipeline to produce the following output is available here. Note: Specific examples of the individual *causal-curve* tools with code are available elsewhere in this documentation.

4.1 The causal effect of blood lead levels on cognitive performance in children

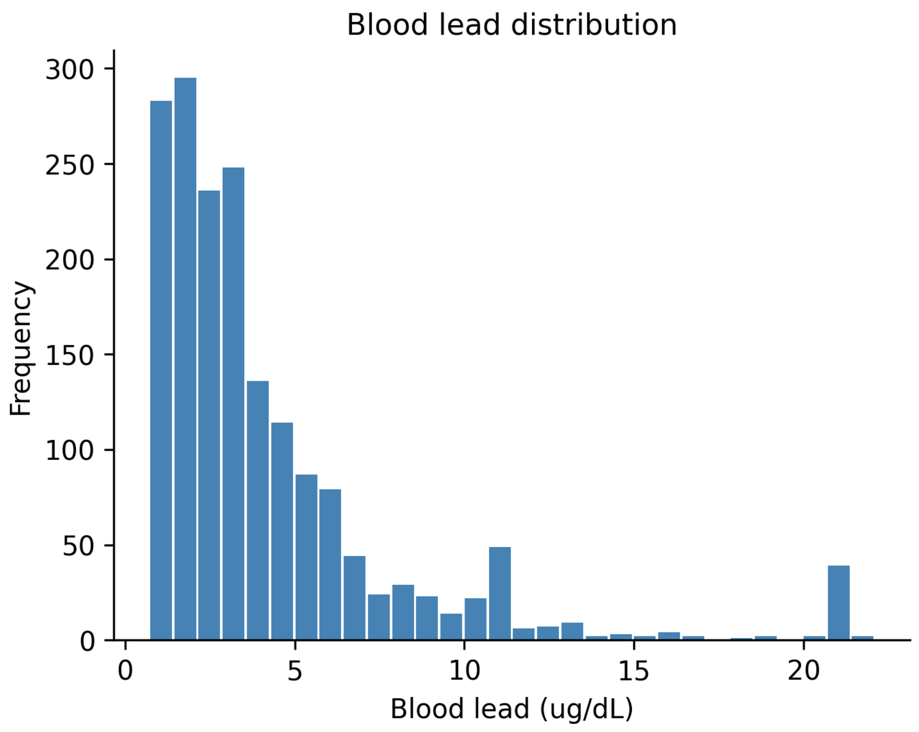
Despite the banning of the use of lead-based paint and the use of lead in gasoline in the United States, lead exposure remains an enormous public health problem for children and adolescents. This is particularly true for poorer children living in older homes in inner-city environments. For children, there is no known safe level of exposure to lead, and even small levels of lead measured in their blood have been shown to affect IQ and academic achievement. One of the scariest parts of lead exposure is that its effects are permanent. Blood lead levels (BLLs) of 5 ug/dL or higher are considered elevated.

There are much research around and many government programs for lead abatement. In terms of public policy, it would be helpful to understand how childhood cognitive outcomes would be affected by reducing BLLs in children. This is the causal question to answer, with blood lead levels being the continuous treatment, and the cognitive outcomes being the outcome of interest.

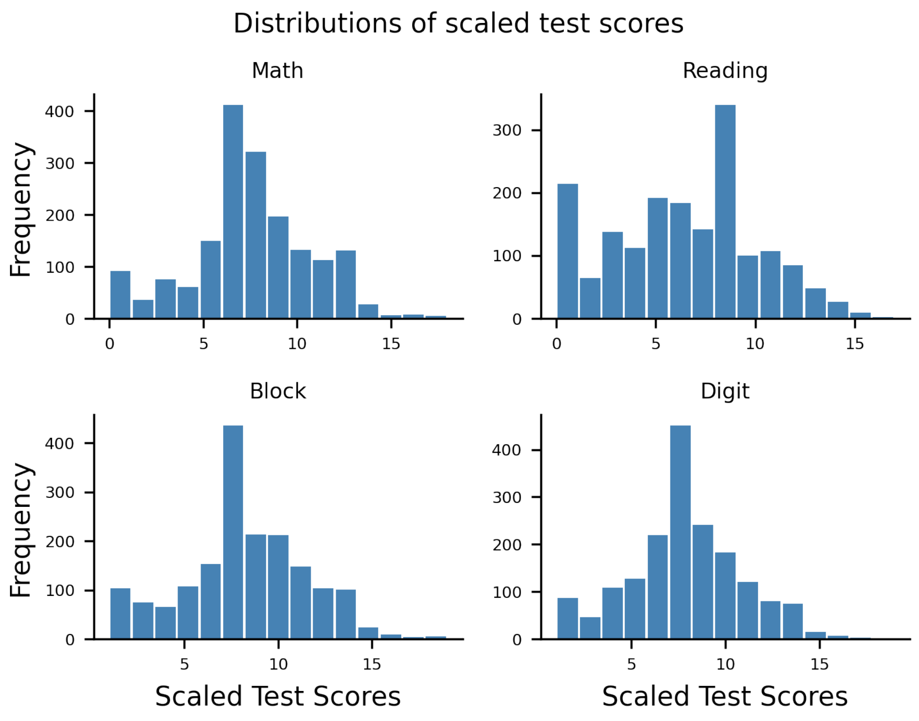
(Photo attribution: Thester11 / CC BY (https://creativecommons.org/licenses/by/3.0))

To explore that problem, we can analyze data collected from the National Health and Nutrition Examination Survey (NHANES) III. This was a large, national study of families throughout the United States, carried out between 1988 and 1994. Participants were involved in extensive interviews, medical examinations, and provided biological samples. As part of this project, BLLs were measured, and four scaled sub-tests of the Wechsler Intelligence Scale for Children-Revised and the Wide Range Achievement Test-Revised (WISC/WRAT) cognitive test were carried out. This data is de-identified and publicly available on the Centers for Disease Control and Prevention (CDC) government website.

When processing the data and missing values were dropped, there were 1,764 children between 6 and 12 years of age with complete data. BLLs among these children were log-normally distributed, as one would expect:



The four scaled sub-tests of the WISC/WRAT included a math test, a reading test, a block design test (a test of spatial visualization ability and motor skill), and a digit spanning test (a test of memory). Their distributions are shown here:



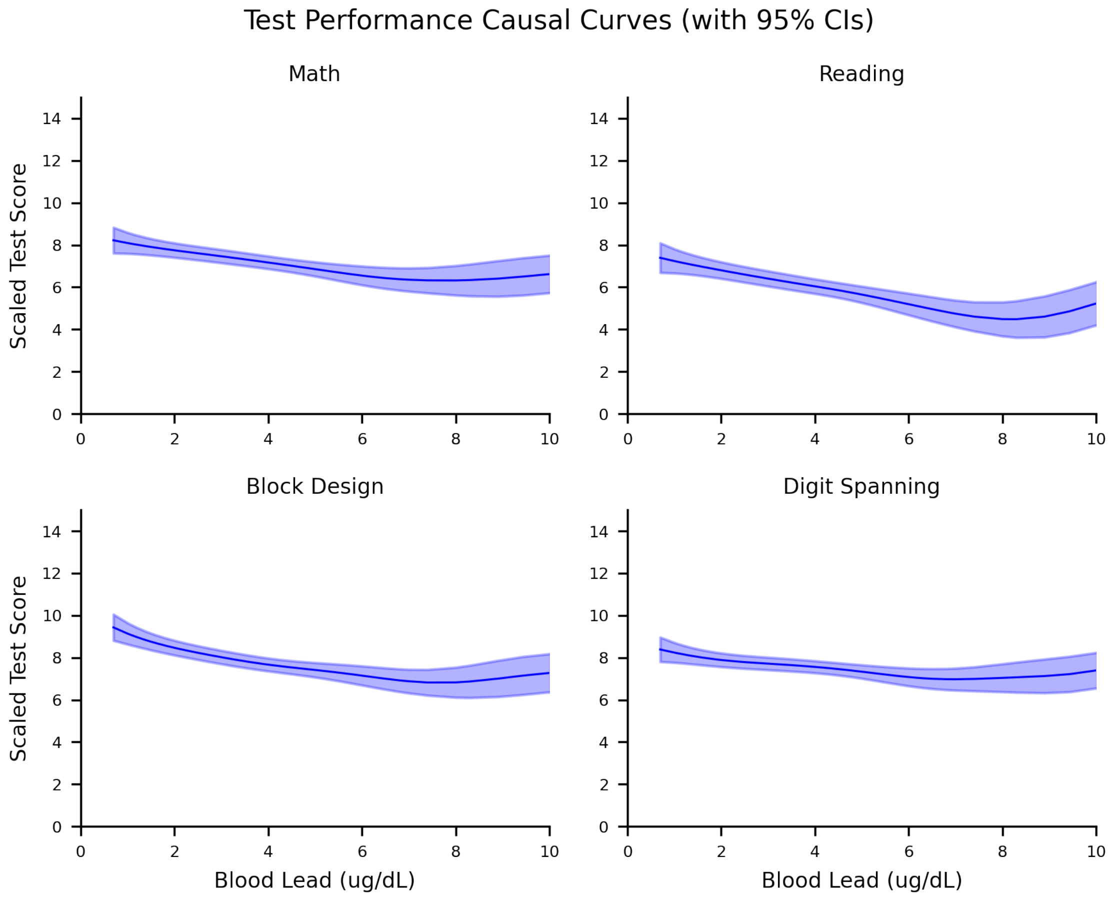
Using a well-known study by Bruce Lanphear conducted in 2000 as a guide, we used the following features as potentially confounding "nuisance" variables:

- Child age

- Child sex (in 1988 - 1994 the CDC assumed binary sex)
- Child race/ethnicity
- The education level of the guardian
- Whether someone smokes in the child's home
- Whether the child spent time in a neonatal intensive care unit as a baby
- Whether the child is experiencing food insecurity (is food sometimes not available due to lack of resources?).

In our "experiment", these above confounders will be controlled for.

By using either the GPS or TMLE tools included in *causal-curve* one can generate the causal dose-response curves for BLLs in relation to the four outcomes:



Note that the lower limit of detection for the blood lead test in this version of NHANES was 0.7 ug/dL. So lead levels below that value are not possible.

In the case of the math test, these results indicate that by reducing BLLs in this population to their lowest value would cause scaled math scores to increase by around 2 points, relative to the BLLs around 10 ug/dL. Similar results are found for the reading and block design test, although the digit spanning test causal curve appears possibly flat (although with the sparse observations at the higher end of the BLL range and the wide confidence intervals it is difficult to say).

The above curves differ from standard regression curves in a few big ways:

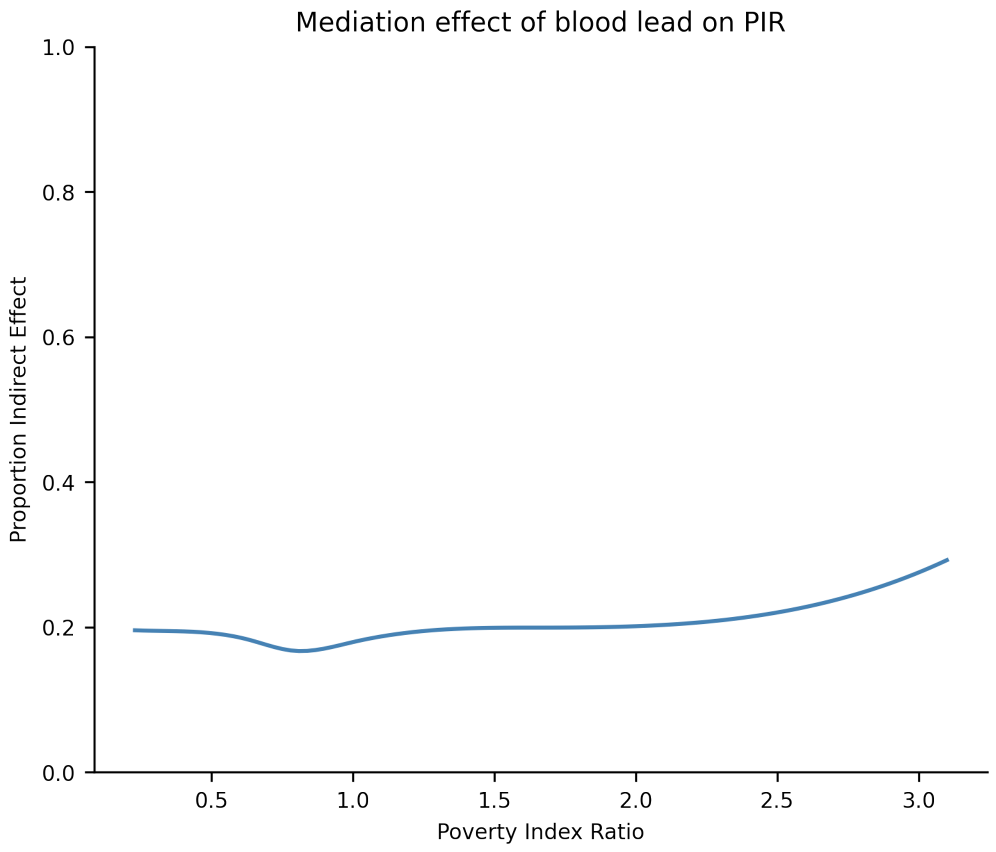
- Even though the data that we used to generate these curves are observational, if causal inference assumptions are met, these curves can be interpretted as causal.

- These models were created using the potential outcomes / counterfactual framework, while standard models are not. Also, the approach we used here essentially simulates experimental conditions by balancing out treatment assignment across the various confounders, and controlling for their effects.
- Even if complex interactions between the variables are modelled, these curves average over the various interaction effects and subgroups. In this sense, these are "marginal" curves.
- These curves should not be used to make predictions at the individual level. These are population level estimates and should remain that way.

4.2 Do blood lead levels mediate the relationship between poverty and cognitive performance?

There is a well-known link between household income and child academic performance. Now that we have some evidence of a potentially causal relationship between BLLs and test performance in children, one might wonder if lead exposure might mediate the relationship between household income academic performance. In other words, in this population does low income cause one to be exposed more to lead, which in turn causes lower performance? Or is household income directly linked with academic performance or through other variables?

NHANES III captured each household's Poverty Index Ratio (the ratio of total family income to the federal poverty level for the year of the interview). For this example, let's focus just on the math test as an outcome. Using *causal-curve*'s mediation tool, we found that the overall, mediating indirect effect of BLLs are 0.20 (0.17 - 0.23). This means that lead exposure accounts for 20% of the relationship between low income and low test performance in this population. The mediation tool also allows you to see how the indirect effect varies as a function of the treatment. As the plot shows, the mediating effect is relatively flat, although interesting there is a hint of an increase as income increases relative to the poverty line.



4.3 References

Centers for Disease Control and Prevention. NHANES III (1988-1994). https://wwwn.cdc.gov/nchs/nhanes/nhanes3/default.aspx. Accessed on July 2, 2020.

Centers for Disease Control and Prevention. Blood Lead Levels in Children. https://www.cdc.gov/nceh/lead/prevention/blood-lead-levels.htm. Accessed on July 2, 2020.

Environmental Protection Agency. Learn about Lead. https://www.epa.gov/lead/learn-about-lead. Accessed on July 2, 2020.

Pirkle JL, Kaufmann RB, Brody DJ, Hickman T, Gunter EW, Paschal DC. Exposure of the U.S. population to lead, 1991-1994. Environmental Health Perspectives, 106(11), 1998, pp. 745–750.

Lanphear BP, Dietrich K, Auinger P, Cox C. Cognitive Deficits Associated with Blood Lead Concentrations <10 pg/dL in US Children and Adolescents. In: Public Health Reports, 115, 2000, pp.521-529.

CHAPTER 5

GPS_Regressor Tool (continuous treatments, continuous outcomes)

In this example, we use this package's GPS_Regressor tool to estimate the marginal causal curve of some continuous treatment on a continuous outcome, accounting for some mild confounding effects. To put this differently, the result of this will be an estimate of the average of each individual's dose-response to the treatment. To do this we calculate generalized propensity scores (GPS) to correct the treatment prediction of the outcome.

Compared with the package's TMLE method, the GPS methods are more computationally efficient, better suited for large datasets, but produces wider confidence intervals.

In this example we use simulated data originally developed by Hirano and Imbens but adapted by others (see references). The advantage of this simulated data is it allows us to compare the estimate we produce against the true, analytically-derived causal curve.

Let t_i be the treatment for the i-th unit, let x_1 and x_2 be the confounding covariates, and let y_i be the outcome measure. We assume that the covariates and treatment are exponentially-distributed, and the treatment variable is associated with the covariates in the following way:

```
>>> import numpy as np
>>> import pandas as pd
>>> from scipy.stats import expon
```

```
>>> np.random.seed(333)
>>> n = 5000
>>> x_1 = expon.rvs(size=n, scale = 1)
>>> x_2 = expon.rvs(size=n, scale = 1)
>>> treatment = expon.rvs(size=n, scale = (1/(x_1 + x_2)))
```

The GPS is given by

$$f(t, x_1, x_2) = (x_1 + x_2) * e^{-(x_1+x_2)*t}$$

If we generate the outcome variable by summing the treatment and GPS, the true causal curve is derived analytically to be:

$$f(t) = t + \frac{2}{(1+t)^3}$$

The following code completes the data generation:

```
>>> gps = ((x_1 + x_2) * np.exp(-(x_1 + x_2) * treatment))
>>> outcome = treatment + gps + np.random.normal(size = n, scale = 1)
```

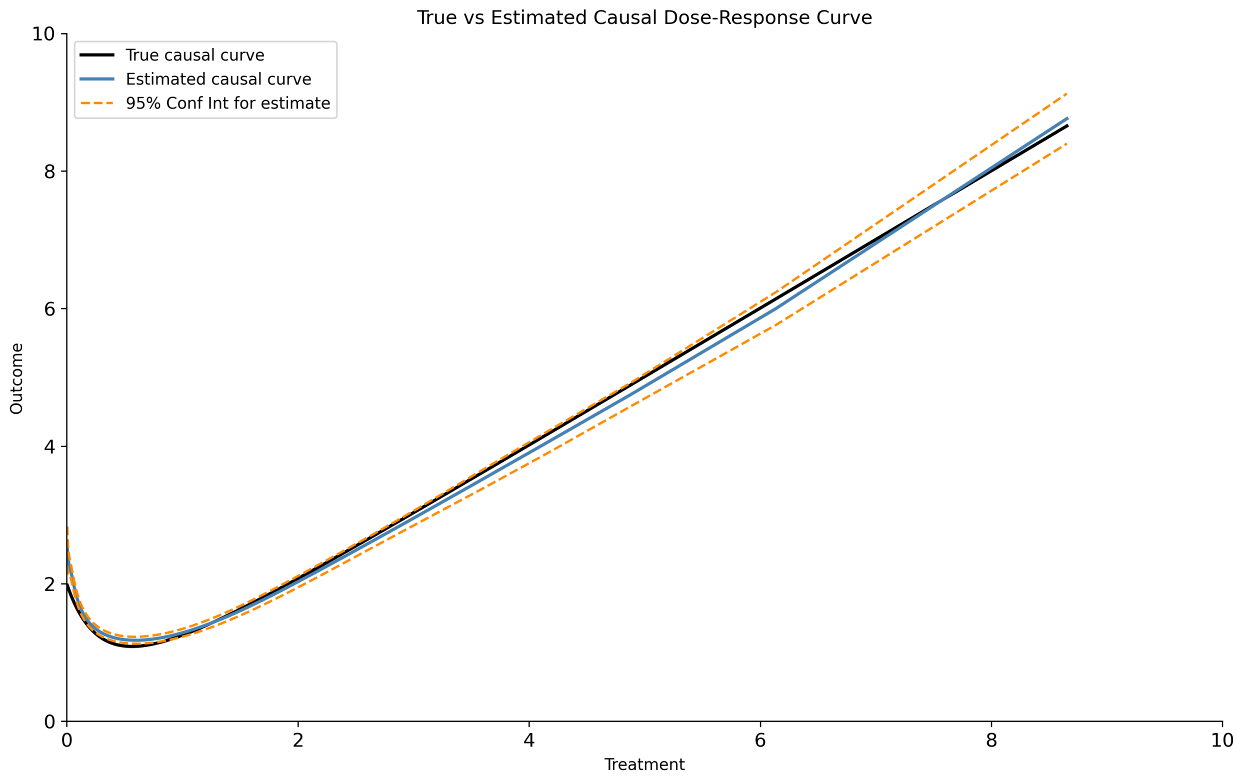
```
>>> truth_func = lambda treatment: (treatment + (2/(1 + treatment)**3))
>>> vfunc = np.vectorize(truth_func)
>>> true_outcome = vfunc(treatment)
```

```
>>> df = pd.DataFrame(
>>>     {
>>>         'X_1': x_1,
>>>         'X_2': x_2,
>>>         'Treatment': treatment,
>>>         'GPS': gps,
>>>         'Outcome': outcome,
>>>         'True_outcome': true_outcome
>>>     }
>>> ).sort_values('Treatment', ascending = True)
```

With this dataframe, we can now calculate the GPS to estimate the causal relationship between treatment and outcome. Let's use the default settings of the GPS_Regressor tool:

```
>>> from causal_curve import GPS_Regressor
>>> gps = GPS_Regressor()
>>> gps.fit(T = df['Treatment'], X = df[['X_1', 'X_2']], y = df['Outcome'])
>>> gps_results = gps.calculate_CDRC(0.95)
```

You now have everything to produce the following plot with matplotlib. In this example with only mild confounding, the GPS-calculated estimate of the true causal curve produces has approximately half the error of a simple LOESS estimate using only the treatment and the outcome.



The GPS_Regressor tool also allows you to estimate a specific set of points along the causal curve. Use the *predict* and *predict_interval* methods to produce a point estimate and prediction interval, respectively.

5.1 References

Galagate, D. Causal Inference with a Continuous Treatment and Outcome: Alternative Estimators for Parametric Dose-Response function with Applications. PhD thesis, 2016.

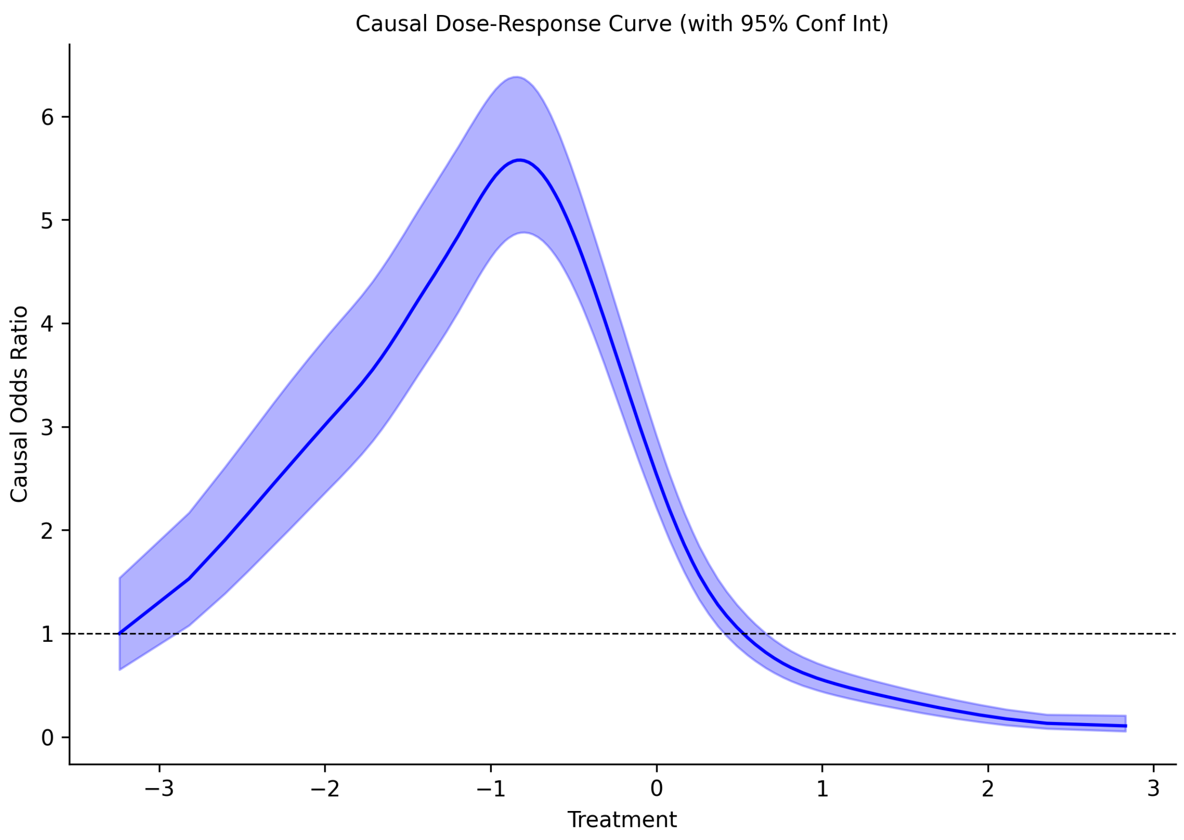
Moodie E and Stephens DA. Estimation of dose–response functions for longitudinal data using the generalised propensity score. In: Statistical Methods in Medical Research 21(2), 2010, pp.149–166.

Hirano K and Imbens GW. The propensity score with continuous treatments. In: Gelman A and Meng XL (eds) Applied bayesian modeling and causal inference from incomplete-data perspectives. Oxford, UK: Wiley, 2004, pp.73–84.

CHAPTER 6

GPS_Classifier Tool (continuous treatments, binary outcomes)

As with the other GPS tool, we calculate generalized propensity scores (GPS) but with the classifier we can estimate the point-by-point causal contribution of a continuous treatment to a binary outcome. The GPS_Classifier does this by estimating the log odds of a positive outcome and odds ratio (odds of positive outcome / odds of negative outcome) along the entire range of treatment values:



Currently, the causal-curve package does not contain a TMLE implementation that is appropriate for a binary outcome, so the GPS_Classifier tool will have to suffice for this sort of outcome.

This tool works much like the _GPS_Regressor tool; as long as the outcome series in your dataframe contains binary integer values (e.g. 0's and 1's) the `fit()` method will work as it's supposed to:

```
>>> df.head(5) # a pandas dataframe with your data
          X_1       X_2   Treatment    Outcome
0    0.596685  0.162688    0.000039     1
1    1.014187  0.916101    0.000197     0
2    0.932859  1.328576    0.000223     0
3    1.140052  0.555203    0.000339     0
4    1.613471  0.340886    0.000438     1
```

With this dataframe, we can now calculate the GPS to estimate the causal relationship between treatment and outcome. Let's use the default settings of the GPS tool:

```
>>> from causal_curve import GPS_Classifier
>>> gps = GPS()
>>> gps.fit(T = df['Treatment'], X = df[['X_1', 'X_2']], y = df['Outcome'])
>>> gps_results = gps.calculate_CDRC(0.95)
```

The `gps_results` object (a dataframe) now contains all of the data to produce the above plot.

If you'd like to estimate the log odds at a specific point on the curve, use the `predict_log_odds` to do so.

6.1 References

Galagate, D. Causal Inference with a Continuous Treatment and Outcome: Alternative Estimators for Parametric Dose-Response function with Applications. PhD thesis, 2016.

Moodie E and Stephens DA. Estimation of dose–response functions for longitudinal data using the generalised propensity score. In: Statistical Methods in Medical Research 21(2), 2010, pp.149–166.

Hirano K and Imbens GW. The propensity score with continuous treatments. In: Gelman A and Meng XL (eds) Applied bayesian modeling and causal inference from incomplete-data perspectives. Oxford, UK: Wiley, 2004, pp.73–84.

CHAPTER 7

TMLE_Regressor Tool (continuous treatments, continuous outcomes)

In this example, we use this package's Targeted Maximum Likelihood Estimation (TMLE) tool to estimate the marginal causal curve of some continuous treatment on a continuous outcome, accounting for some mild confounding effects.

The TMLE algorithm is doubly robust, meaning that as long as one of the two models contained with the tool (the `g` or `q` models) performs well, then the overall tool will correctly estimate the causal curve.

Compared with the package's GPS methods incorporates more powerful machine learning techniques internally (gradient boosting) and produces significantly smaller confidence intervals. However it is less computationally efficient and will take longer to run. In addition, **the treatment values provided should be roughly normally-distributed**, otherwise you may encounter internal math errors.

Let's first generate some simple toy data:

```
>>> import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from causal_curve import TMLE_Regressor
np.random.seed(200)
```

```
>>> def generate_data(t, A, sigma, omega, noise=0, n_outliers=0, random_state=0):
        y = A * np.exp(-sigma * t) * np.sin(omega * t)
        rnd = np.random.RandomState(random_state)
        error = noise * rnd.randn(t.size)
        outliers = rnd.randint(0, t.size, n_outliers)
        error[outliers] *= 35
        return y + error
```

```
>>> treatment = np.linspace(0, 10, 1000)
outcome = generate_data(
        t = treatment,
        A = 2,
        sigma = 0.1,
        omega = (0.1 * 2 * np.pi),
```

(continues on next page)

(continued from previous page)

```
        noise = 0.1,
        n_outliers = 5
)
x1 = np.random.uniform(0,10,1000)
x2 = (np.random.uniform(0,10,1000) * 3)
```

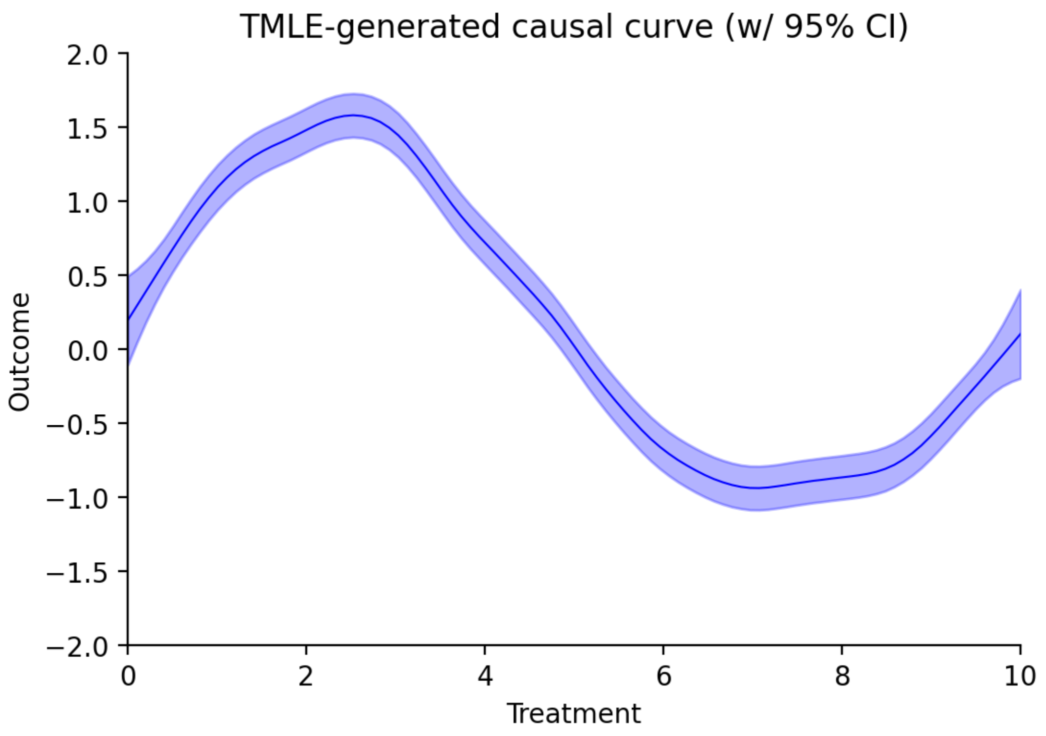
```
>>> df = pd.DataFrame(
        {
                'x1': x1,
                'x2': x2,
                'treatment': treatment,
                'outcome': outcome
        }
)
```

All we do now is employ the TMLE_Regressor class, with mostly default settings:

```
>>> from causal_curve import TMLE_Regressor
tmle = TMLE_Regressor(
    random_seed=111,
                bandwidth=10
)
```

```
>>> tmle.fit(T = df['treatment'], X = df[['x1', 'x2']], y = df['outcome'])
gps_results = tmle.calculate_CDRC(0.95)
```

The resulting dataframe contains all of the data you need to generate the following plot:



To generate user-specified points along the curve, use the `point_estimate` and `point_estimate_interval` methods:

```
>>> tmle.point_estimate(np.array([5.5]))
tmle.point_estimate_interval(np.array([5.5]))
```

7.1 References

Kennedy EH, Ma Z, McHugh MD, Small DS. Nonparametric methods for doubly robust estimation of continuous treatment effects. Journal of the Royal Statistical Society, Series B. 79(4), 2017, pp.1229-1245.

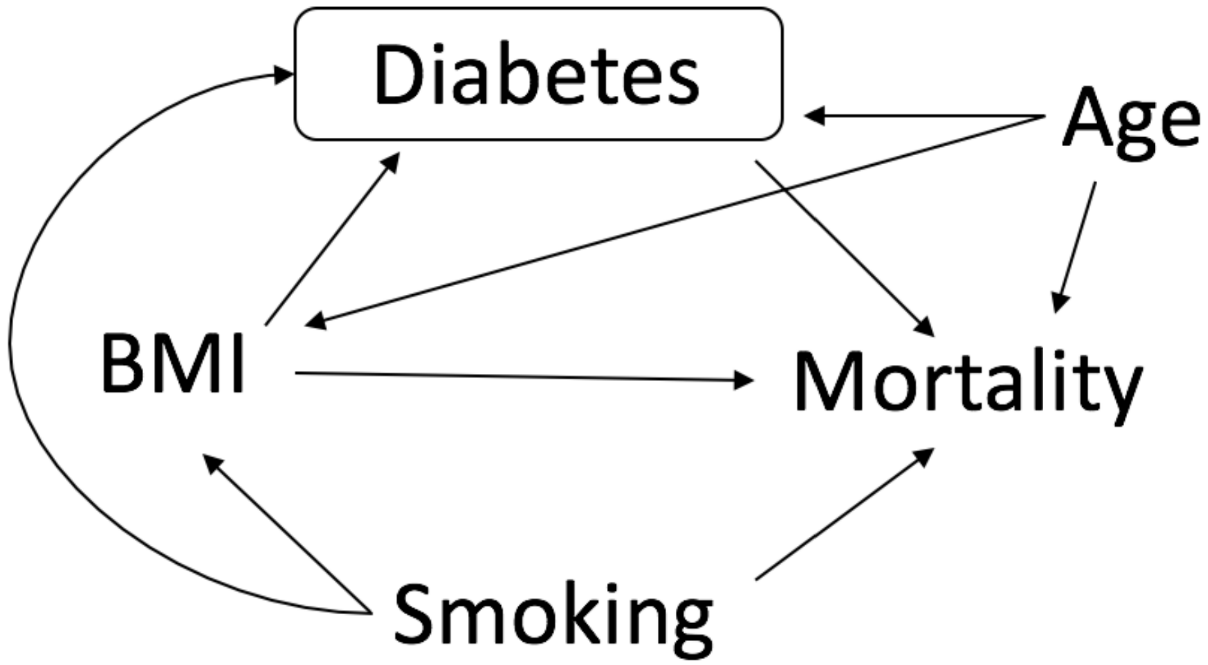
van der Laan MJ and Rubin D. Targeted maximum likelihood learning. In: U.C. Berkeley Division of Biostatistics Working Paper Series, 2006.

van der Laan MJ and Gruber S. Collaborative double robust penalized targeted maximum likelihood estimation. In: The International Journal of Biostatistics 6(1), 2010.

CHAPTER 8

Mediation Tool (continuous treatment, mediator, and outcome)

In trying to explore the causal relationships between various elements, oftentimes you'll use your domain knowledge to sketch out your initial ideas about the causal connections. See the following causal DAG of the expected relationships between smoking, diabetes, obesity, age, and mortality (Havumaki et al.):



At some point though, it's helpful to validate these ideas with empirical tests. This tool provides a test that can estimate the amount of mediation that occurs between a treatment, a purported mediator, and an outcome. In keeping with the causal curve theme, this tool uses a test developed by Imai et al. when handling a continuous treatment and mediator.

In this example we use the following simulated data, and assume that the *mediator* variable is decided to be a mediator by expert judgement.

```
>>> import numpy as np
>>> import pandas as pd
```

```
>>> np.random.seed(132)
>>> n_obs = 500
```

```
>>> treatment = np.random.normal(loc=50.0, scale=10.0, size=n_obs)
>>> mediator = np.random.normal(loc=70.0 + treatment, scale=8.0, size=n_obs)
>>> outcome = np.random.normal(loc=(treatment + mediator - 50), scale=10.0, size=n_
↪obs)
```

```
>>> df = pd.DataFrame(
>>>     {
>>>         "treatment": treatment,
>>>         "mediator": mediator,
>>>         "outcome": outcome
>>>     }
>>> )
```

Now we can instantiate the Mediation class:

```
>>> from causal_curve import Mediation
>>> med = Mediation(
>>>         bootstrap_draws=100,
>>>         bootstrap_replicates=100,
>>>         spline_order=3,
>>>         n_splines=5,
>>>         verbose=True,
>>> )
```

We then fit the data to the *med* object:

```
>>> med.fit(
>>>     T=df["treatment"],
>>>     M=df["mediator"],
>>>     y=df["outcome"],
>>> )
```

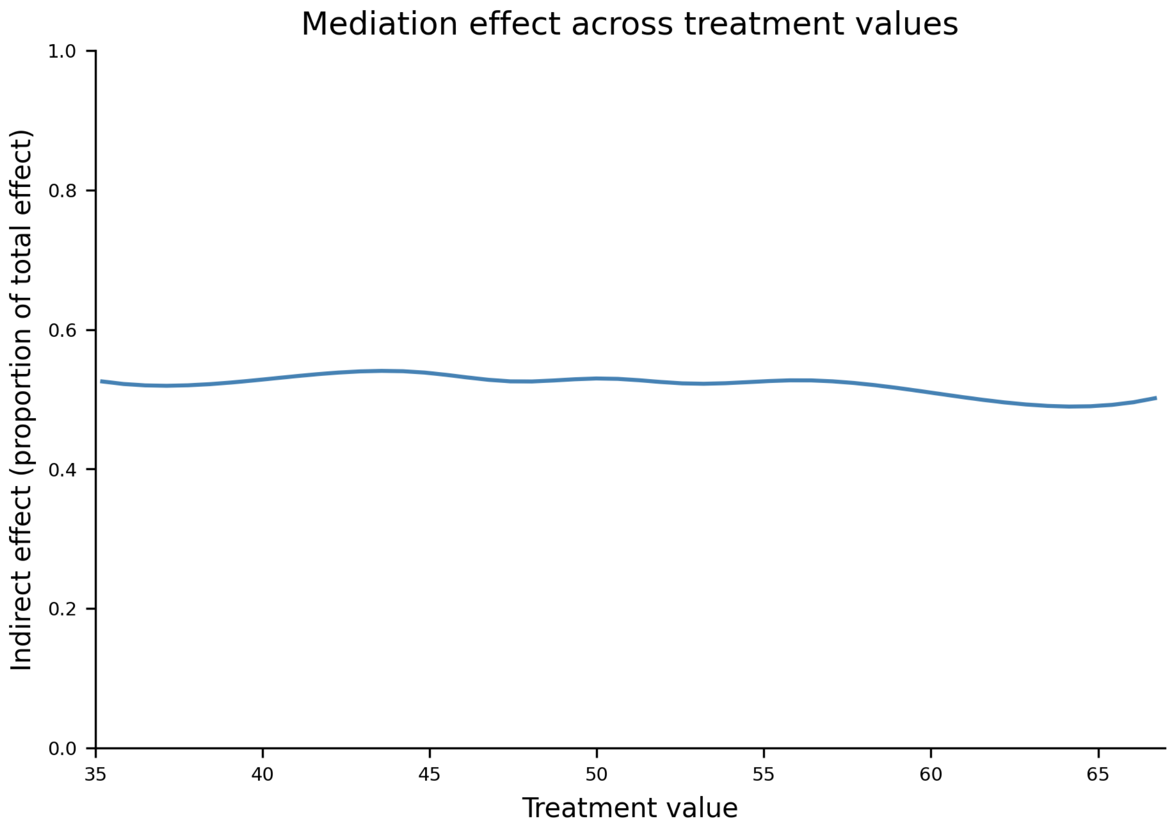
With the internal models of the mediation test fit with data, we can now run the *calculate_mediation* method to produce the final report:

```
>>> med.calculate_mediation(ci = 0.95)
>>>
>>> ----------------------------------
>>> Mean indirect effect proportion: 0.5238 (0.5141 - 0.5344)
>>>
>>> Treatment_Value  Proportion_Direct_Effect  Proportion_Indirect_Effect
>>> 35.1874                    0.4743                       0.5257
>>> 41.6870                    0.4638                       0.5362
>>> 44.6997                    0.4611                       0.5389
>>> 47.5672                    0.4745                       0.5255
>>> 50.1900                    0.4701                       0.5299
>>> 52.7526                    0.4775                       0.5225
>>> 56.0204                    0.4727                       0.5273
>>> 60.5174                    0.4940                       0.5060
>>> 66.7243                    0.4982                       0.5018
```

The final analysis tells us that overall, the mediator is estimated to account for around 52% (+/- 1%) of the effect of the treatment on the outcome. This indicates that moderate mediation is occurring here. The remaining 48% occurs through a direct effect of the treatment on the outcome.

So long as we are confident that the mediator doesn't play another role in the causal graph (it isn't a confounder of the treatment and outcome association), this supports the idea that the mediator is in fact a mediator.

The report also shows how this mediation effect various as a function of the continuous treatment. In this case, it looks the effect is relatively flat (as expected). With a little processing and some basic interpolation, we can plot this mediation effect:



8.1 References

Imai K., Keele L., Tingley D. A General Approach to Causal Mediation Analysis. Psychological Methods. 15(4), 2010, pp.309–334.

Havumaki J., Eisenberg M.C. Mathematical modeling of directed acyclic graphs to explore competing causal mechanisms underlying epidemiological study data. medRxiv preprint. doi: https://doi.org/10.1101/19007922. Accessed June 23, 2020.

CHAPTER 9

causal_curve

9.1 causal_curve package

9.1.1 causal_curve.core module

Core classes (with basic methods) that will be invoked when other, model classes are defined

class `causal_curve.core.`**`Core`**

Bases: `object`

Base class for causal_curve module

static **`calculate_z_score`**(*ci*)

Calculates the critical z-score for a desired two-sided, confidence interval width.

Parameters

ci: float, the confidence interval width (e.g. 0.95)

Returns

Float, critical z-score value

static **`clip_negatives`**(*number*)

Helper function to clip negative numbers to zero

Parameters

number: int or float, any number that needs a floor at zero

Returns

Int or float of modified value

`get_params`()

Returns a dict of all of the object's user-facing parameters

Parameters

None

Returns

self: object

if_verbose_print(*string*)
Prints the input statement if verbose is set to True

Parameters

string: str, some string to be printed

Returns

None

static rand_seed_wrapper(*random_seed=None*)
Sets the random seed using numpy

Parameters

random_seed: int, random seed number

Returns

None

9.1.2 causal_curve.gps_core module

Defines the Generalized Prospensity Score (GPS) Core model class

class causal_curve.gps_core.**GPS_Core**(*gps_family=None*, *treatment_grid_num=100*, *lower_grid_constraint=0.01*, *upper_grid_constraint=0.99*, *spline_order=3*, *n_splines=30*, *lambda_=0.5*, *max_iter=100*, *random_seed=None*, *verbose=False*)

Bases: *causal_curve.core.Core*

In a multi-stage approach, this computes the generalized propensity score (GPS) function, and uses this in a generalized additive model (GAM) to correct treatment prediction of the outcome variable. Assumes continuous treatment, but the outcome variable may be continuous or binary.

WARNING:

-This algorithm assumes you've already performed the necessary transformations to categorical covariates (i.e. these variables are already one-hot encoded and one of the categories is excluded for each set of dummy variables).

-Please take care to ensure that the "ignorability" assumption is met (i.e. all strong confounders are captured in your covariates and there is no informative censoring), otherwise your results will be biased, sometimes strongly so.

Parameters

gps_family: str, optional (default = None) Is used to determine the family of the glm used to model the GPS function. Look at the distribution of your treatment variable to determine which family is more appropriate. Possible values:

- 'normal'
- 'lognormal'
- 'gamma'

- None : (best-fitting family automatically chosen)

treatment_grid_num: int, optional (default = 100) Takes the treatment, and creates a quantile-based grid across its values. For instance, if the number 6 is selected, this means the algorithm will only take the 6 treatment variable values at approximately the 0, 20, 40, 60, 80, and 100th percentiles to estimate the causal dose response curve. Higher value here means the final curve will be more finely estimated, but also increases computation time. Default is usually a reasonable number.

lower_grid_constraint: float, optional(default = 0.01) This adds an optional constraint of the lower side of the treatment grid. Sometimes data near the minimum values of the treatment are few in number and thus generate unstable estimates. By default, this clips the bottom 1 percentile or lower of treatment values. This can be as low as 0, indicating there is no lower limit to how much treatment data is considered.

upper_grid_constraint: float, optional (default = 0.99) See above parameter. Just like above, but this is an upper constraint. By default, this clips the top 99th percentile or higher of treatment values. This can be as high as 1.0, indicating there is no upper limit to how much treatment data is considered.

spline_order: int, optional (default = 3) Order of the splines to use fitting the final GAM. Must be integer >= 1. Default value creates cubic splines.

n_splines: int, optional (default = 30) Number of splines to use for the treatment and GPS in the final GAM. Must be integer >= 2. Must be non-negative.

lambda_: int or float, optional (default = 0.5) Strength of smoothing penalty. Must be a positive float. Larger values enforce stronger smoothing.

max_iter: int, optional (default = 100) Maximum number of iterations allowed for the maximum likelihood algo to converge.

random_seed: int, optional (default = None) Sets the random seed.

verbose: bool, optional (default = False) Determines whether the user will get verbose status updates.

References

Galagate, D. Causal Inference with a Continuous Treatment and Outcome: Alternative Estimators for Parametric Dose-Response function with Applications. PhD thesis, 2016.

Moodie E and Stephens DA. Estimation of dose–response functions for longitudinal data using the generalised propensity score. In: Statistical Methods in Medical Research 21(2), 2010, pp.149–166.

Hirano K and Imbens GW. The propensity score with continuous treatments. In: Gelman A and Meng XL (eds) Applied bayesian modeling and causal inference from incomplete-data perspectives. Oxford, UK: Wiley, 2004, pp.73–84.

Examples

```
>>> # With continuous outcome
>>> from causal_curve import GPS_Regressor
>>> gps = GPS_Regressor(treatment_grid_num = 200, random_seed = 512)
>>> gps.fit(T = df['Treatment'], X = df[['X_1', 'X_2']], y = df['Outcome'])
>>> gps_results = gps.calculate_CDRC(0.95)
>>> point_estimate = gps.point_estimate(np.array([5.0]))
>>> point_estimate_interval = gps.point_estimate_interval(np.array([5.0]), 0.95)
>>> # With binary outcome
>>> from causal_curve import GPS_Classifier
>>> gps = GPS_Classifier()
>>> gps.fit(T = df['Treatment'], X = df[['X_1', 'X_2']], y = df['Binary_Outcome'])
>>> gps_results = gps.calculate_CDRC(0.95)
>>> log_odds = gps.estimate_log_odds(np.array([5.0]))
```

Attributes

grid_values: array of shape (treatment_grid_num,) The gridded values of the treatment variable. Equally spaced.

best_gps_family: str If no gps_family is specified and the algorithm chooses the best glm family, this is the name of the family that was chosen.

gps_deviance: float The GPS model deviance

gps: array of shape (number of observations,) The calculated GPS for each observation

gam_results: 'pygam.LinearGAM' class trained model of *LinearGAM* class, from pyGAM library

Methods

fit: (self, T, X, y)	Fits the causal dose-response model.
calculate_CDRC: (self, ci)	Calculates the CDRC (and confidence interval) from trained model.
print_gam_summary: (self)	Prints pyGAM text summary of GAM predicting outcome from the treatment and the GPS.

calculate_CDRC(*ci=0.95*)

Using the results of the fitted model, this generates a dataframe of point estimates for the CDRC at each of the values of the treatment grid. Connecting these estimates will produce the overall estimated CDRC. Confidence interval is returned as well.

Parameters

ci: float (default = 0.95) The desired confidence interval to produce. Default value is 0.95, corresponding to 95% confidence intervals. bounded (0, 1.0).

Returns

dataframe: Pandas dataframe Contains treatment grid values, the CDRC point estimate at that value, and the associated lower and upper confidence interval bounds at that point.

self: object

fit(*T*, *X*, *y*)

Fits the GPS causal dose-response model. For now, this only accepts pandas columns. While the treatment variable must be continuous (or ordinal with many levels), the outcome variable may be continuous or binary. You *must* provide at least one covariate column.

Parameters

T: array-like, shape (n_samples,) A continuous treatment variable.

X: array-like, shape (n_samples, m_features) Covariates, where n_samples is the number of samples and m_features is the number of features. Features can be a mix of continuous and nominal/categorical variables.

y: array-like, shape (n_samples,) Outcome variable. May be continuous or binary. If continuous, this must be a series of type *float*, if binary must be a series of type *integer*.

Returns

self [object]

print_gam_summary()

Prints the GAM model summary (uses pyGAM's output)

Parameters

None

Returns

self: object

9.1.3 causal_curve.gps_regressor module

Defines the Generalized Prospensity Score (GPS) regressor model class

class causal_curve.gps_regressor.**GPS_Regressor**(*gps_family=None*, *treatment_grid_num=100*, *lower_grid_constraint=0.01*, *upper_grid_constraint=0.99*, *spline_order=3*, *n_splines=30*, *lambda_=0.5*, *max_iter=100*, *random_seed=None*, *verbose=False*)

Bases: *causal_curve.gps_core.GPS_Core*

A GPS tool that handles continuous outcomes. Inherits the GPS_core base class. See that base class code its docstring for more details.

Methods

point_estimate: (self, T)	Calculates point estimate within the CDRC given treatment values. Can only be used when outcome is continuous.
point_estimate_interval: (self, T, ci)	Calculates the prediction confidence interval associated with a point estimate within the CDRC given treatment values. Can only be used when outcome is continuous.

point_estimate(*T*)

Calculates point estimate within the CDRC given treatment values. Can only be used when outcome is continuous. Can be estimate for a single data point or can be run in batch for many observations. Extrapolation will produce untrustworthy results; the provided treatment should be within the range of the training data.

Parameters

T: Numpy array, shape (n_samples,) A continuous treatment variable.

Returns

array: Numpy array Contains a set of CDRC point estimates

point_estimate_interval(*T*, *ci=0.95*)

Calculates the prediction confidence interval associated with a point estimate within the CDRC given treatment values. Can only be used when outcome is continuous. Can be estimate for a single data point or can be run in batch for many observations. Extrapolation will produce untrustworthy results; the provided treatment should be within the range of the training data.

Parameters

T: Numpy array, shape (n_samples,) A continuous treatment variable.

ci: float (default = 0.95) The desired confidence interval to produce. Default value is 0.95, corresponding to 95% confidence intervals. bounded (0, 1.0).

Returns

array: Numpy array Contains a set of CDRC prediction intervals ([lower bound, higher bound])

9.1.4 causal_curve.gps_classifier module

Defines the Generalized Prospensity Score (GPS) classifier model class

class causal_curve.gps_classifier.**GPS_Classifier**(*gps_family=None*, *treatment_grid_num=100*, *lower_grid_constraint=0.01*, *upper_grid_constraint=0.99*, *spline_order=3*, *n_splines=30*, *lambda_=0.5*, *max_iter=100*, *random_seed=None*, *verbose=False*)

Bases: *causal_curve.gps_core.GPS_Core*

A GPS tool that handles binary outcomes. Inherits the GPS_core base class. See that base class code its docstring for more details.

Methods

estimate_log_odds: (self, T)	Calculates the predicted log odds of the highest integer class. Can only be used when the outcome is binary.

estimate_log_odds(*T*)

Calculates the estimated log odds of the highest integer class. Can only be used when the outcome is binary. Can be estimate for a single data point or can be run in batch for many observations. Extrapolation will produce untrustworthy results; the provided treatment should be within the range of the training data.

Parameters

T: Numpy array, shape (n_samples,) A continuous treatment variable.

Returns

array: Numpy array Contains a set of log odds

9.1.5 causal_curve.tmle_core module

Defines the Targetted Maximum likelihood Estimation (TMLE) model class

class causal_curve.tmle_core.**TMLE_Core**(*treatment_grid_num=100*, *lower_grid_constraint=0.01*, *upper_grid_constraint=0.99*, *n_estimators=200*, *learning_rate=0.01*, *max_depth=3*, *bandwidth=0.5*, *random_seed=None*, *verbose=False*)

Bases: `causal_curve.core.Core`

Constructs a causal dose response curve via a modified version of Targetted Maximum Likelihood Estimation (TMLE) across a grid of the treatment values. Gradient boosting is used for prediction of the Q model and G models, simple kernel regression is used processing those model results, and a generalized additive model is used in the final step to contruct the final curve. Assumes continuous treatment and outcome variable.

WARNING:

-The treatment values should be roughly normally-distributed for this tool to work. Otherwise you may encounter internal math errors.

-This algorithm assumes you've already performed the necessary transformations to categorical covariates (i.e. these variables are already one-hot encoded and one of the categories is excluded for each set of dummy variables).

-Please take care to ensure that the "ignorability" assumption is met (i.e. all strong confounders are captured in your covariates and there is no informative censoring), otherwise your results will be biased, sometimes strongly so.

Parameters

treatment_grid_num: int, optional (default = 100) Takes the treatment, and creates a quantile-based grid across its values. For instance, if the number 6 is selected, this means the algorithm will only take the 6 treatment variable values at approximately the 0, 20, 40, 60, 80, and 100th percentiles to estimate the causal dose response curve. Higher value here means the final curve will be more finely estimated, but also increases computation time. Default is usually a reasonable number.

lower_grid_constraint: float, optional(default = 0.01) This adds an optional constraint of the lower side of the treatment grid. Sometimes data near the minimum values of the treatment are few in number and thus generate unstable estimates. By default, this clips the bottom 1 percentile or lower of treatment values. This can be as low as 0, indicating there is no lower limit to how much treatment data is considered.

upper_grid_constraint: float, optional (default = 0.99) See above parameter. Just like above, but this is an upper constraint. By default, this clips the top 99th percentile or higher of treatment values. This can be as high as 1.0, indicating there is no upper limit to how much treatment data is considered.

n_estimators: int, optional (default = 200) Optional argument to set the number of learners to use when sklearn creates TMLE's Q and G models.

learning_rate: float, optional (default = 0.01) Optional argument to set the sklearn's learning rate for TMLE's Q and G models.

max_depth: int, optional (default = 3) Optional argument to set sklearn's maximum depth when creating TMLE's Q and G models.

bandwidth: float, optional (default = 0.5) Optional argument to set the bandwidth parameter of the internal kernel density estimation and kernel regression methods.

random_seed: int, optional (default = None) Sets the random seed.

verbose: bool, optional (default = False) Determines whether the user will get verbose status updates.

References

Kennedy EH, Ma Z, McHugh MD, Small DS. Nonparametric methods for doubly robust estimation of continuous treatment effects. Journal of the Royal Statistical Society, Series B. 79(4), 2017, pp.1229-1245.

van der Laan MJ and Rubin D. Targeted maximum likelihood learning. In: The International Journal of Biostatistics, 2(1), 2006.

van der Laan MJ and Gruber S. Collaborative double robust penalized targeted maximum likelihood estimation. In: The International Journal of Biostatistics 6(1), 2010.

Examples

```
>>> # With continuous outcome
>>> from causal_curve import TMLE_Regressor
>>> tmle = TMLE_Regressor()
>>> tmle.fit(T = df['Treatment'], X = df[['X_1', 'X_2']], y = df['Outcome'])
>>> tmle_results = tmle.calculate_CDRC(0.95)
>>> point_estimate = tmle.point_estimate(np.array([5.0]))
>>> point_estimate_interval = tmle.point_estimate_interval(np.array([5.0]), 0.95)
```

Attributes

grid_values: array of shape (treatment_grid_num,) The gridded values of the treatment variable. Equally spaced.

final_gam: 'pygam.LinearGAM' class trained final model of *LinearGAM* class, from pyGAM library

pseudo_out: array of shape (observations,) Adjusted, pseudo-outcome observations

Methods

fit: (self, T, X, y)	Fits the causal dose-response model
calculate_CDRC: (self, ci, CDRC_grid_num)	Calculates the CDRC (and confidence interval) from TMLE estimation

calculate_CDRC(*ci=0.95*)

Using the results of the fitted model, this generates a dataframe of CDRC point estimates at each of the values of the treatment grid. Connecting these estimates will produce the overall estimated CDRC. Confidence interval is returned as well.

Parameters

ci: float (default = 0.95) The desired confidence interval to produce. Default value is 0.95, corresponding to 95% confidence intervals. bounded (0, 1.0).

Returns

dataframe: Pandas dataframe Contains treatment grid values, the CDRC point estimate at that value, and the associated lower and upper confidence interval bounds at that point.

self: object

fit(*T*, *X*, *y*)

Fits the TMLE causal dose-response model. For now, this only accepts pandas columns. You *must* provide at least one covariate column.

Parameters

T: array-like, shape (n_samples,) A continuous treatment variable

X: array-like, shape (n_samples, m_features) Covariates, where n_samples is the number of samples and m_features is the number of features

y: array-like, shape (n_samples,) Outcome variable

Returns

self [object]

`one_dim_estimate_density`(*series*)
Takes in a numpy array, returns grid values for KDE and predicted probabilities

`pred_from_loess`(*train_x*, *train_y*, *x_to_pred*)
Trains simple loess regression and returns predictions

9.1.6 causal_curve.tmle_regressor module

Defines the Targetted Maximum likelihood Estimation (TMLE) regressor model class

class `causal_curve.tmle_regressor.`**`TMLE_Regressor`**(*treatment_grid_num=100*, *lower_grid_constraint=0.01*, *upper_grid_constraint=0.99*, *n_estimators=200*, *learning_rate=0.01*, *max_depth=3*, *bandwidth=0.5*, *random_seed=None*, *verbose=False*)

Bases: *`causal_curve.tmle_core.TMLE_Core`*

A TMLE tool that handles continuous outcomes. Inherits the TMLE_core base class. See that base class code its docstring for more details.

Methods

point_estimate: (self, T)	Calculates point estimate within the CDRC given treatment values. Can only be used when outcome is continuous.

`point_estimate`(*T*)
Calculates point estimate within the CDRC given treatment values. Can only be used when outcome is continuous. Can be estimate for a single data point or can be run in batch for many observations. Extrapolation will produce untrustworthy results; the provided treatment should be within the range of the training data.

Parameters

T: Numpy array, shape (n_samples,) A continuous treatment variable.

Returns

array: Numpy array Contains a set of CDRC point estimates

`point_estimate_interval`(*T*, *ci=0.95*)
Calculates the prediction confidence interval associated with a point estimate within the CDRC given treatment values. Can only be used when outcome is continuous. Can be estimate for a single data point or can be run in batch for many observations. Extrapolation will produce untrustworthy results; the provided treatment should be within the range of the training data.

Parameters

T: Numpy array, shape (n_samples,) A continuous treatment variable.

ci: float (default = 0.95) The desired confidence interval to produce. Default value is 0.95, corresponding to 95% confidence intervals. bounded (0, 1.0).

Returns

array: Numpy array Contains a set of CDRC prediction intervals ([lower bound, higher bound])

9.1.7 causal_curve.mediation module

Defines the Mediation test class

class causal_curve.mediation.**Mediation**(*treatment_grid_num=10, lower_grid_constraint=0.01, upper_grid_constraint=0.99, bootstrap_draws=500, bootstrap_replicates=100, spline_order=3, n_splines=5, lambda_=0.5, max_iter=100, random_seed=None, verbose=False*)

Bases: *causal_curve.core.Core*

Given three continuous variables (a treatment or independent variable of interest, a potential mediator, and an outcome variable of interest), Mediation provides a method to determine the average direct and indirect effect.

Parameters

treatment_grid_num: int, optional (default = 10) Takes the treatment, and creates a quantile-based grid across its values. For instance, if the number 6 is selected, this means the algorithm will only take the 6 treatment variable values at approximately the 0, 20, 40, 60, 80, and 100th percentiles to estimate the causal dose response curve. Higher value here means the final curve will be more finely estimated, but also increases computation time. Default is usually a reasonable number.

lower_grid_constraint: float, optional(default = 0.01) This adds an optional constraint of the lower side of the treatment grid. Sometimes data near the minimum values of the treatment are few in number and thus generate unstable estimates. By default, this clips the bottom 1 percentile or lower of treatment values. This can be as low as 0, indicating there is no lower limit to how much treatment data is considered.

upper_grid_constraint: float, optional (default = 0.99) See above parameter. Just like above, but this is an upper constraint. By default, this clips the top 99th percentile or higher of treatment values. This can be as high as 1.0, indicating there is no upper limit to how much treatment data is considered.

bootstrap_draws: int, optional (default = 500) Bootstrapping is used as part of the mediation test. The parameter determines the number of draws from the original data to create a single bootstrap replicate.

bootstrap_replicates: int, optional (default = 100) Bootstrapping is used as part of the mediation test. The parameter determines the number of bootstrapping runs to perform / number of new datasets to create.

spline_order: int, optional (default = 3) Order of the splines to use fitting the final GAM. Must be integer >= 1. Default value creates cubic splines.

n_splines: int, optional (default = 5) Number of splines to use for the mediation and outcome GAMs. Must be integer >= 2. Must be non-negative.

lambda_: int or float, optional (default = 0.5) Strength of smoothing penalty. Must be a positive float. Larger values enforce stronger smoothing.

max_iter: int, optional (default = 100) Maximum number of iterations allowed for the maximum likelihood algo to converge.

random_seed: int, optional (default = None) Sets the random seed.

verbose: bool, optional (default = False) Determines whether the user will get verbose status updates.

References

Imai K., Keele L., Tingley D. A General Approach to Causal Mediation Analysis. Psychological Methods. 15(4), 2010, pp.309–334.

Examples

```
>>> from causal_curve import Mediation
>>> med = Mediation(treatment_grid_num = 200, random_seed = 512)
>>> med.fit(T = df['Treatment'], M = df['Mediator'], y = df['Outcome'])
>>> med_results = med.calculate_effects(0.95)
```

Attributes

grid_values: array of shape (treatment_grid_num,) The gridded values of the treatment variable. Equally spaced.

Methods

fit: (self, T, M, y)	Fits the trio of relevant variables using generalized additive models.
calculate_effects: (self, ci)	Calculates the average direct and indirect effects.

`calculate_mediation`(*ci=0.95*)

Conducts mediation analysis on the fit data

Parameters

ci: float (default = 0.95) The desired bootstrap confidence interval to produce. Default value is 0.95, corresponding to 95% confidence intervals. bounded (0, 1.0).

Returns

dataframe: Pandas dataframe Contains the estimate of the direct and indirect effects and the proportion of indirect effects across the treatment grid values. The bootstrap confidence interval that is returned might not be symmetric.

self [object]

`fit`(*T, M, y*)

Fits models so that mediation analysis can be run. For now, this only accepts pandas columns.

Parameters

T: array-like, shape (n_samples,) A continuous treatment variable

M: array-like, shape (n_samples,) A continuous mediation variable

y: array-like, shape (n_samples,) A continuous outcome variable

Returns

self [object]

9.1.8 Module contents

causal_curve module

CHAPTER 10

Change Log

10.1 Version 1.0.6

- Latest version of python black can now run. Linted tmle_core.py.

10.2 Version 1.0.5

- Removed *master* branch, replaced with *main*
- Removed all mention of *master* branch from documentation

10.3 Version 1.0.4

- Fixed TMLE plot and code errors in documentation

10.4 Version 1.0.3

- Fixed bug with *random_seed* functionality in all tools

10.5 Version 1.0.2

- Updated end-to-end example notebook in */examples* folder
- Fixed various class docstrings if they still reference old v0.5.2 API
- Fixed bug where custom class input parameters weren't being used

10.6 Version 1.0.1

- Added to TMLE overview in the docs (including plot)

10.7 Version 1.0.0: Major Update

- Overhaul of the TMLE tool to make it dramatically more accurate and user-friendly.
- Improved TMLE example documentation
- Much like with *scikit-learn*, there are now separate model classes used for predicting binary or continuous outcomes
- Updating documentation to reflect API changes
- Added more tests
- Linted with *pylint* (added *.pylintrc* file)

10.8 Version 0.5.2

- Fixed bug that prevented *causal-curve* modules from being shown in Sphinx documentation
- Augmented tests to capture more error states and improve code coverage

10.9 Version 0.5.1

- Removed working test file

10.10 Version 0.5.0

- Added new *predict*, *predict_interval*, and *predict_log_odds* methods to GPS tool
- Slight updates to doc to reflect new features

10.11 Version 0.4.1

- When using GPS tool with a treatment with negative values, only the normal GLM family can be picked
- Added 'sphinx_rtd_theme' to dependency list in *.travis.yml* and *install.rst*
- core.py base class now has __version__ attribute

10.12 Version 0.4.0

- Added support for binary outcomes in GPS tool
- Small changes to repo README

10.13 Version 0.3.8

- Added citation (yay!)

10.14 Version 0.3.7

- Bumped version for PyPi

10.15 Version 0.3.6

- Fixed bug in Mediation.calculate_mediation that would clip treatments < 0 or > 1
- Fixed incorrect horizontal axis labels in lead example
- Fixed typos in documentation
- Added links to resources so users could learn more about causal inference theory

10.16 Version 0.3.5

- Re-organized documentation
- Added *Introduction* section to explain purpose and need for the package

10.17 Version 0.3.4

- Removed XGBoost as dependency.
- Now using sklearn's gradient boosting implementation.

10.18 Version 0.3.3

- Misc edits to paper and bibliography

10.19 Version 0.3.2

- Fixed random seed issue with Mediation tool
- Fixed Mediation bootstrap issue. Confidence interval bounded [0,1]
- Fixed issue with all classes not accepting non-sequential indicies in pandas Dataframes/Series
- Class init checks for all classes now print cleaner errors if bad input

10.20 Version 0.3.1

- Small fixes to end-to-end example documentation
- Enlarged image in paper

10.21 Version 0.3.0

- Added full, end-to-end example of package usage to documentation
- Cleaned up documentation
- Added example folder with end-to-end notebook
- Added manuscript to paper folder

10.22 Version 0.2.4

- Strengthened unit tests

10.23 Version 0.2.3

- codecov integration

10.24 Version 0.2.2

- Travis CI integration

10.25 Version 0.2.1

- Fixed Mediation tool error / removed *tqdm* from requirements
- Misc documentation cleanup / revisions

10.26 Version 0.2.0

- Added new Mediation class
- Updated documentation to reflect this
- Added unit and integration tests for Mediation methods

10.27 Version 0.1.3

- Simplifying unit and integration tests.

10.28 Version 0.1.2

- Added unit and integration tests

10.29 Version 0.1.1

- setup.py fix

10.30 Version 0.1.0

- Added new TMLE class
- Updated documentation to reflect new TMLE method
- Renamed CDRC method to more appropriate *GPS* method
- Small docstring corrections to GPS method

10.31 Version 0.0.10

- Bug fix in GPS estimation method

10.32 Version 0.0.9

- Project created

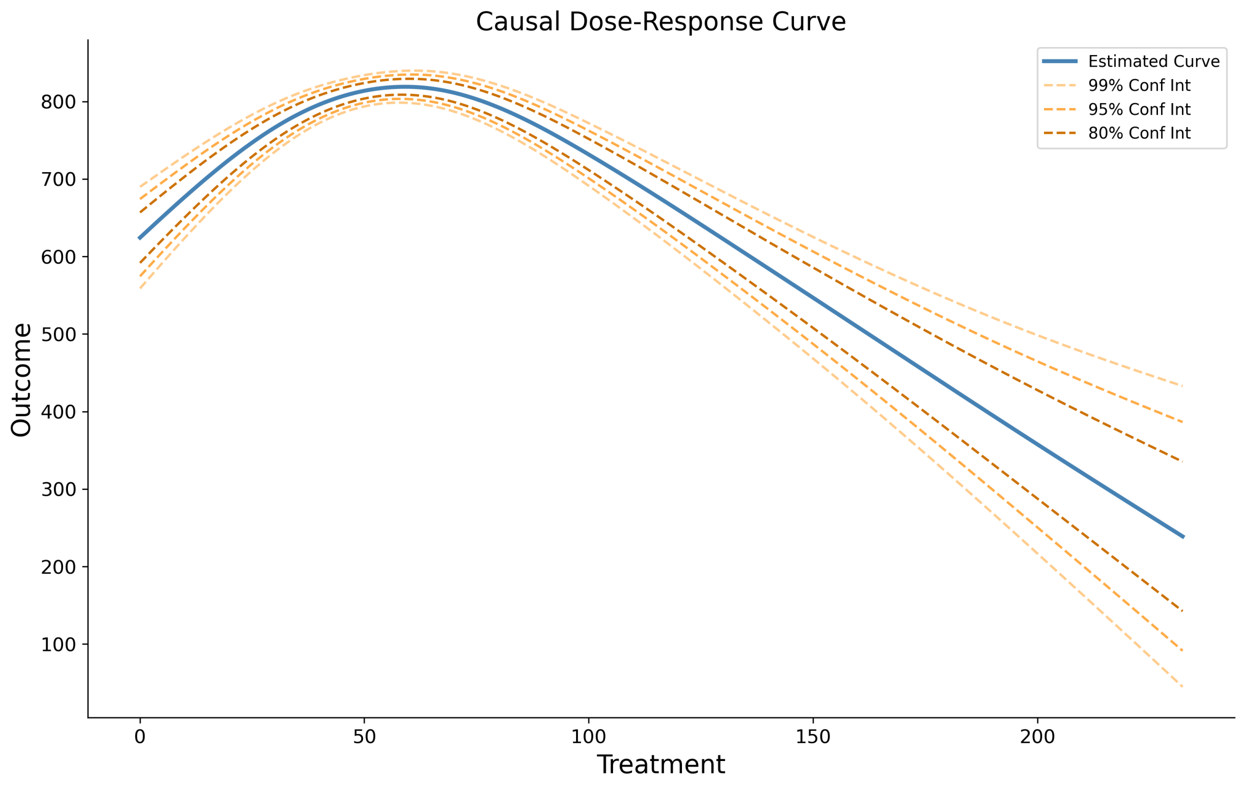
CHAPTER 11

Citation

Please consider citing us in your academic or industry project.

Kobrosly, R. W., (2020). causal-curve: A Python Causal Inference Package to Estimate Causal Dose-Response Curves. Journal of Open Source Software, 5(52), 2523, https://doi.org/10.21105/joss.02523

causal-curve is a Python package with tools to perform causal inference when the treatment of interest is continuous.



CHAPTER 12

Summary

(**Version 1.0.0 released in Jan 2021!**)

There are many available methods to perform causal inference when your intervention of interest is binary, but few methods exist to handle continuous treatments. This is unfortunate because there are many scenarios (in industry and research) where these methods would be useful. This library attempts to address this gap, providing tools to estimate causal curves (AKA causal dose-response curves). Both continuous and binary outcomes can be modeled with this package.

CHAPTER 13

Quick example (of the `GPS_Regressor` tool)

causal-curve uses a sklearn-like API that should feel familiar to python machine learning users. This includes `_Regressor` and `_Classifier` models, and `fit()` methods.

The following example estimates the causal dose-response curve (CDRC) by calculating generalized propensity scores.

```
>>> from causal_curve import GPS_Regressor
>>> import numpy as np
```

```
>>> gps = GPS_Regressor(treatment_grid_num = 200, random_seed = 512)
```

```
>>> df # a pandas dataframe with your data
           X_1        X_2  Treatment    Outcome
0     0.596685  0.162688   0.000039  -0.270533
1     1.014187  0.916101   0.000197  -0.266979
2     0.932859  1.328576   0.000223   1.921979
3     1.140052  0.555203   0.000339   1.461526
4     1.613471  0.340886   0.000438   2.064511
```

```
>>> gps.fit(T = df['Treatment'], X = df[['X_1', 'X_2']], y = df['Outcome'])
>>> gps_results = gps.calculate_CDRC(ci = 0.95)
>>> gps_point = gps.point_estimate(np.array([0.0003]))
>>> gps_point_interval = gps.point_estimate_interval(np.array([0.0003]), ci = 0.95)
```

1. First we import the *GPS_Regressor* class.
2. Then we instantiate the class, providing any of the optional parameters.
3. Prepare and organized your treatment, covariate, and outcome data into a pandas dataframe.
4. Fit the load the training and test sets by calling the `.fit()` method.
5. Estimate the points of the causal curve (along with 95% confidence interval bounds) with the `.calculate_CDRC()` method.
6. Generate point estimates along the causal curve with the `.point_estimate()`, `.point_estimate_interval()`, and `.estimate_log_odds()` methods.

7. Explore or plot your results!

Python Module Index

c

causal_curve, 44
causal_curve.core, 33
causal_curve.gps_classifier, 38
causal_curve.gps_core, 34
causal_curve.gps_regressor, 37
causal_curve.mediation, 42
causal_curve.tmle_core, 38
causal_curve.tmle_regressor, 41

Index

C

calculate_CDRC() (*causal_curve.gps_core.GPS_Core method*), 36

calculate_CDRC() (*causal_curve.tmle_core.TMLE_Core method*), 40

calculate_mediation() (*causal_curve.mediation.Mediation method*), 43

calculate_z_score() (*causal_curve.core.Core static method*), 33

causal_curve (*module*), 44

causal_curve.core (*module*), 33

causal_curve.gps_classifier (*module*), 38

causal_curve.gps_core (*module*), 34

causal_curve.gps_regressor (*module*), 37

causal_curve.mediation (*module*), 42

causal_curve.tmle_core (*module*), 38

causal_curve.tmle_regressor (*module*), 41

clip_negatives() (*causal_curve.core.Core static method*), 33

Core (*class in causal_curve.core*), 33

E

estimate_log_odds() (*causal_curve.gps_classifier.GPS_Classifier method*), 38

F

fit() (*causal_curve.gps_core.GPS_Core method*), 36

fit() (*causal_curve.mediation.Mediation method*), 43

fit() (*causal_curve.tmle_core.TMLE_Core method*), 40

G

get_params() (*causal_curve.core.Core method*), 33

GPS_Classifier (*class in causal_curve.gps_classifier*), 38

GPS_Core (*class in causal_curve.gps_core*), 34

GPS_Regressor (*class in causal_curve.gps_regressor*), 37

I

if_verbose_print() (*causal_curve.core.Core method*), 34

M

Mediation (*class in causal_curve.mediation*), 42

O

one_dim_estimate_density() (*causal_curve.tmle_core.TMLE_Core method*), 41

P

point_estimate() (*causal_curve.gps_regressor.GPS_Regressor method*), 37

point_estimate() (*causal_curve.tmle_regressor.TMLE_Regressor method*), 41

point_estimate_interval() (*causal_curve.gps_regressor.GPS_Regressor method*), 37

point_estimate_interval() (*causal_curve.tmle_regressor.TMLE_Regressor method*), 41

pred_from_loess() (*causal_curve.tmle_core.TMLE_Core method*), 41

print_gam_summary() (*causal_curve.gps_core.GPS_Core method*), 37

R

rand_seed_wrapper() (*causal_curve.core.Core static method*), 34

T

TMLE_Core (*class in causal_curve.tmle_core*), 38

`TMLE_Regressor` (*class in causal_curve.tmle_regressor*), 41