NOTES FROM INDUSTRY

Causal Inference in the Wild: Elasticity Pricing

How can machine learning improve causal inference in industry? I describe a real example with code for retail pricing.

Lars Roemheld · Follow

Published in Towards Data Science · 13 min read · Jul 8, 2021

Causal inference is a hot topic in machine learning, and there are many excellent primers on the theory of causal inference available [1–4]. But much fewer examples of real-world applications of machine-learning-powered causal inference exist. This article introduces one such example from an industry context, using a (public) real-world dataset. It is aimed at a technical audience with an understanding of the basics of causality.

Specifically, I will look at the "ideal" scenario of price elasticity estimation [2, 5]. This scenario is highly relevant for retailers wishing to optimize prices, for example to better manage inventory or just to be more competitive.

In section 1, I will motivate the problem and provide a brief background. This will also illustrate the more general point that estimating causal effects requires domain expertise on top of "just" data analytics. In section 2, I will

explain that learning from historical data is difficult because of confounding. Section 3 will sketch a machine-learning-powered solution to this problem, which is detailed in code samples in section 4. A complete, runnable notebook is available here. To conclude, I will discuss the business relevance of such causally valid estimation.

1 Why care about elasticity?

A core concept of most Econ101 courses, "price elasticity of demand" describes how price-sensitive demand is for a given product. If a product is "more elastic," price increases will lead to greater demand reductions. As a causal concept, elasticity allows us to quantify how many units more we *could* sell (counterfactually) if we *were* to decrease prices by, say, 5%.

Elasticity as an effect between price and demanded quantity describes a complex causal system: Price of course doesn't directly impact demand, but is rather mediated through many individual consumer decisions to buy or not buy, along with secondary effects such as being featured on bargain-hunting websites. In trying to estimate elasticity, a retailer skips over all this complexity to directly measure the effect of something they control (price) on something they care about (demand).

It is somewhat surprising, then, that elasticity can actually be estimated. But since its formal definition in 1890 [6], economists have found price elasticity of demand to be a decent approximation of reality in many scenarios [5]. This approximation is particularly convenient in its canonical definition of *relative changes*: elasticity measures the percent-change of demanded quantity (q), given a percent-change of price (p):

$$\theta = \frac{\partial q / q}{\partial p / p}$$

The intuition here is simple: a $1 change in price for a product costing $5 will lead to more dramatic changes in demand than for a product costing $100. In aggregate, consumers care about "relative" changes. Empirically, this definition is convenient because it means the parameter to be estimated, θ, remains approximately constant as p changes.

With a valid estimate of θ, the retailer can now reason counterfactually about their prices: "if I were to increase my product's price by 5%, I could expect to sell 5θ% more units" (θ is typically negatively valued).

To build some intuition, assume they currently sell 100 units for a price of $5 and at a cost of $4. Then they currently make 100*(5–4)=$100 profit. Assume they estimated elasticity to be θ=-3, i.e. they expect a 5% price increase (from $5 to $5.25) to yield a 15% demand reduction (from 100 to 85). Then the new profit would be slightly increased, 85*(5.25–4)=$106. If, however, they decreased their price by 5%, they would lose profits: 115*(4.75–4)=$86. Note that higher prices do not always lead to higher profits: for example, if the

unit cost were just $3: a 5% price increase would then reduce profits, from 100*(5–3)=$200 to 85*(5.25–3)=$191.

You may convince yourself that different values of elasticity change the optimal price: generally speaking, the more elastic demand is (if θ=-5 instead of -3, for example), the lower the markup between cost and price should be.

Being able to make such counterfactual statements saves the retailer a lot of "trying out new prices" by giving an indication whether prices should be increased or decreased. Of course, good estimates of elasticity provide additional value when a retailer wants to manage their inventory to sell all stock until the end of the season, for example. In short, highly accurate estimates of elasticity give the retailer a concise summary of a complex causal system of individual buying decisions, allowing business decisions that are confident in their expected consequences.

This short excursion into the basic economic theory of demand serves as an example of expert knowledge in causal inference: Knowledge about data generating processes informs the type of question to answer, the mechanisms to look out for, and often the assumed functional form. It would be difficult to learn all this structure from data — and certainly data-inefficient.

2 Appreciating the challenge

After this primer on why a retailer would want to estimate elasticity, this section describes why that makes for an interesting problem.

The preferred way to causal inference is, of course, (randomized) experimentation: the retailer might spend some time randomly adjusting product prices up and down (or better yet, randomize prices across users). But such experimentation is expensive, since products are sold at suboptimal

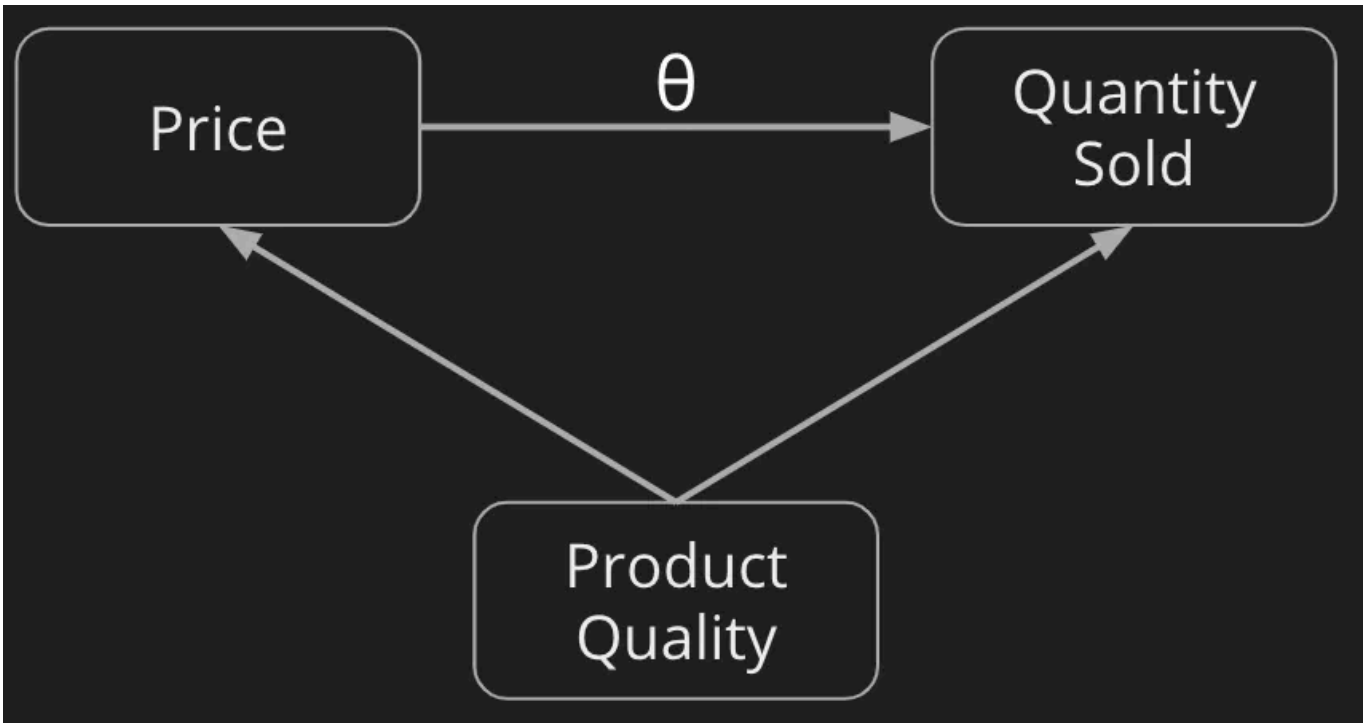
prices, and customer experience suffers. To make matters worse, signal-to-noise in retail data is typically low, since most shops offer a vast number of products of which a large share never sells in any given timeframe. Thus experiments would need to run for a long time before enough products have been sold to allow for inference. Finally, the retailer might be concerned that short experiments might not generalize across seasons, for example.

The ability to learn causally valid elasticity from observational data is therefore key; observational data in this example is simply the retailer's history of prices and units sold over time. But estimating causal effects from observational data is difficult because of confounding. To see what this means, consider (1) product quality and (2) seasons as two important examples of many potential confounders:

1. MacBooks are more expensive than, say, Chromebooks. Assuming a retailer sells more MacBooks than Chromebooks (and nothing else), the observational data indicates that high prices correlate with high sales. But it would be foolish to (counterfactually) expect that raising a Chromebook's price to Apple- levels would allow selling more Chromebooks.
2. Demand for many products is seasonal, for example due to holidays (Christmas) and weather changes (summer). Typically, prices are high during high season (and yet a lot of products are sold), and lower during off-season (when fewer products are sold); yet despite this correlation it would be foolish to expect higher sales from raising off-season prices to high season levels.

As the saying goes, the retailer must be careful not to confuse correlation and causation. The following causal graph represents a simple confounder relationship: failing to control for product quality (and season, and others, not displayed) will significantly bias estimates of θ. Such biases will lead the

retailer to wrong conclusions about optimal prices, directly hurting their business.



A minimum (i.e. incomplete) causal graph between price and quantity, with product quality as only confounder.

As an aside: price elasticity is an instance of a problem known as "simultaneous equation model" in econometrics [7]— the core insight being that price is actually determined by simultaneous decisions on the supply- *and* demand-sides. In this example, the retailer believes that they have sufficient data on their own pricing decisions to separate out a consistent estimate of market reaction to price changes.

Knowing the importance of correctly controlling for all available confounders, the retailer now turns to machine-learning-powered causal inference.

3 Double Machine Learning, or Controlling for Observables 2.0

The solution I am describing here primarily solves two issues: First, regularization picks appropriate controls out of a large number of potential confounders. Second, since I'm using a flexible machine learning algorithm, the functional form of and interaction between control variables (i.e. their preprocessing) matters less than it would in a traditional regression approach, where confounders are controlled for using vanilla linear regression. Naturally, a lot of thought still needs to go into finding variables that might be confounders — and what variables should be avoided since they might introduce "collider bias" [8], which this method is *not* immune to.

My solution implements Double Machine Learning (DML) [9]. The main idea is relatively intuitive: given some observed potential confounders, I use nonparametric, flexible estimators (machine learning models) to efficiently control for various functional forms and interaction effects. In other words, I approximate the actual confounding mechanism in the data generating process with ML models.

To do this, I train *two* separate, "auxiliary" models to predict the treatment (price, P) and the outcome (quantity demanded, Q), respectively. The models are both trained independently with a set X of potential confounding variables (e.g. product quality and season), so that their predictions approximate expected values, $E[P|X]$ and $E[Q|X]$.

Using these predictions, I then *residualize out* the parts of treatment and effect that are predictable through the set of control variables:

$$\tilde{P} = P - E[P|X]$$

$$\tilde{Q} = Q - E[Q|X]$$

In my implementation below, I choose RandomForests to estimate the conditional expectations, chosen for their robust performance. You can think of these two auxiliary models as "control models": If X contains enough information about the relevant confounders, and if at least one of the models "fits well", by construction $\tilde{P}$ and $\tilde{Q}$ are now "unconfounded" and only the true causal relationship remains (there is a rich literature on this, beginning with [9]).

So after residualizing, I am interested in

$$\theta = \frac{\partial \tilde{Q}/\tilde{Q}}{\partial \tilde{P}/\tilde{P}}$$

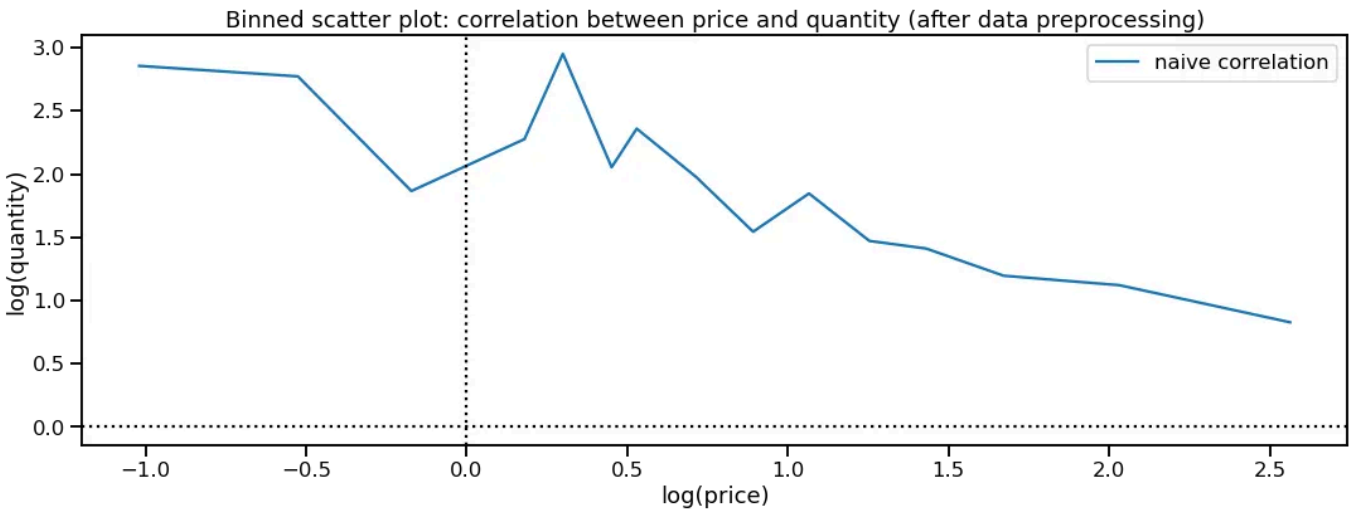
This quantity is still difficult to estimate, but fortunately there is a standard way of estimating elasticities in econometrics: log-log regression. θ is exactly equivalent to the parameter in the following regression model:

$$\log \tilde{Q} \sim \theta \log \tilde{P} + \text{intercept}$$

Exactly how this works is beyond the scope of this article (if you do want to follow, see [10]). The important takeaway is that in the actual implementation I will (linearly) regress residualized log-quantity on residualized log-price to obtain θ.

In the binned scatterplot below, I show the "naive" result of just taking the correlation between log-quantity and log-price in my sample dataset: The relationship appears to be somewhat linear even before controlling for confounders such as season or product quality. Elasticity, θ, is the slope of

this line ([10]). As you will see, controlling for confounders significantly changes (improves) the estimate.



In a naive correlation analysis after data cleaning, visualized here as a binned scatterplot, a (somewhat) linear relationship between log-quantity and log-price can be observed.

Now, to put the approach together:

1. I'll take *logs* of p and q.
2. I'll construct a set of potential confounders, X.
3. I'll train an auxiliary RandomForest model to predict $log(p)$ from X.
4. I'll train a separate auxiliary RandomForest model to predict $log(q)$ from X.

If you follow this process closely, you'll notice that the process of residualization as described is "in-sample prediction", i.e. the auxiliary models are trained on the same data that they predict on. This is likely to introduce overfit and result in bias — to counteract this, I'll split my data into 2 halves. I'll use one half of the data for fitting the auxiliary models, and then use the models to residualize $log(p)$ and $log(q)$ in the other half. Then I can measure causal elasticity in the latter data set. Rinse and repeat, switching out the two sets and averaging over both estimates. So:

1. Split the dataset into two halves, and take *logs* of p and q.
2. Construct a set of potential confounders X (same variables in both halves).
3. Use the **first** half to train an auxiliary RandomForest model to predict $log(p)$ from X.
4. Use the **first** half to train a separate auxiliary RandomForest model to predict $log(q)$ from X.
5. Use the two auxiliary models to predict $log(p)$ and $log(q)$ from X in the **second** half of the data, to obtain after residualizing:

$$\tilde{\log} p = \log p - \hat{\log} p$$

$$\tilde{\log} q = \log q - \hat{\log} q$$

6. Infer elasticity as the coefficient of a linear regression,

$$\tilde{\log} Q \sim \theta \tilde{\log} P + \text{intercept}$$

7. Repeat steps 3–6 with the two halves swapped out. Use the mean of both θ-estimates as final result.

To better illustrate how this mechanism works in practice, let's look at some step-by-step example code. For your own experiments, DML and many other modern methods are conveniently available in EconML, a new python package maintained by Microsoft Research [11]. As of early 2021, I have found the API to be rather unstable, but the ongoing development is very promising. Similarly, DoubleML offers a DML implementation for python and R [12].

4 Sample code

The code uses this idealized real-world dataset: there is a row for every day and every product, along with some additional features of this (day x product) combination that can be used as potential confounders, *X*. Most importantly, in every row, the data records *Q* and *P*.

					Q	P
item_id	date	item_description	day_of_week	stock_age_in_days		
A123	2021-01-01	Hardwood bench with chair	3	23	2	230.99
A563	2021-01-01	Brand name perfume 100ml	3	399	0	120.99
B879	2021-01-01	Cashmere wool turtleneck sweater	3	4	3	99.99
A123	2021-01-02	Hardwood bench with chair	4	24	1	230.99
A563	2021-01-02	Brand name perfume 100ml	4	400	1	119.99
B879	2021-01-02	Cashmere wool turtleneck sweater	4	5	5	89.99

First, I'll define two standard ML models to predict *log(p)* and *log(q)* from all available potential confounders *X*. Notice that since there are individual control variables for every day and every product, the models can capture both product quality and seasonality.

To improve the predictive quality of the models, to further reduce confounding, and to show off the advantages of DML, I add some further feature engineering: *n-grams* of product descriptions to recover similarities between products, seasonality-descriptors, and a measure of product novelty. I then pipe the result of this feature engineering, a set of potential confounders, into a standard RandomForest model.

Much more effort could be spent here, of course; but this set of potential confounders showcases that the RandomForests might find nontrivial functional forms and interaction effects beyond what a standard regression could do.

```python
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestRegressor

feature_generator = ColumnTransformer(
    [
        ('item_id', OneHotEncoder(), ['item_id']),
        ('date', OneHotEncoder(), ['date']),
        ('item_description',
             CountVectorizer(min_df=0.05, ngram_range=(1, 3)),
             'item_description'),
        ('numeric_feats', StandardScaler(),
             ['day_of_week', 'stock_age_in_days'])
    ], remainder='drop'
)

model_q = Pipeline([
    ('feat_proc', feature_generator),
    ('model_q', RandomForestRegressor())
])
model_p = Pipeline([
    ('feat_proc', feature_generator),
    ('model_p', RandomForestRegressor())
])
```

causal_inf_article_model.py hosted with ❤️ by GitHub view raw

Next up, for the actual inference I'll use a linear regression model. This will estimate the slope θ of the log-log-regression, as discussed above.

The original DML paper proposes a slightly different estimator for this regression than OLS, to achieve better robustness: Instead of taking the standard OLS-solution,

$$\hat{\theta} = (\tilde{P}' \tilde{P})^{-1} \tilde{P}' \tilde{Q}$$

Standard OLS solution to Q~θP+intercept

they use the following, de-biased estimator. Note that the first P is residualized, while the second is *not*, and the third is:

$$\hat{\theta} = (\tilde{P}' P)^{-1} \tilde{P}' \tilde{Q}$$

Debiased regression estimator [9]

In the following snippet, I implement steps (1) through (6) as described above: divide the data into two halves, train the auxiliary models on the first half, residualize the second half, and infer elasticity. After repeating with the two halves swapped, I take the mean of the two estimates, to obtain a robust causal estimate (assuming I was able to capture the relevant confounders).

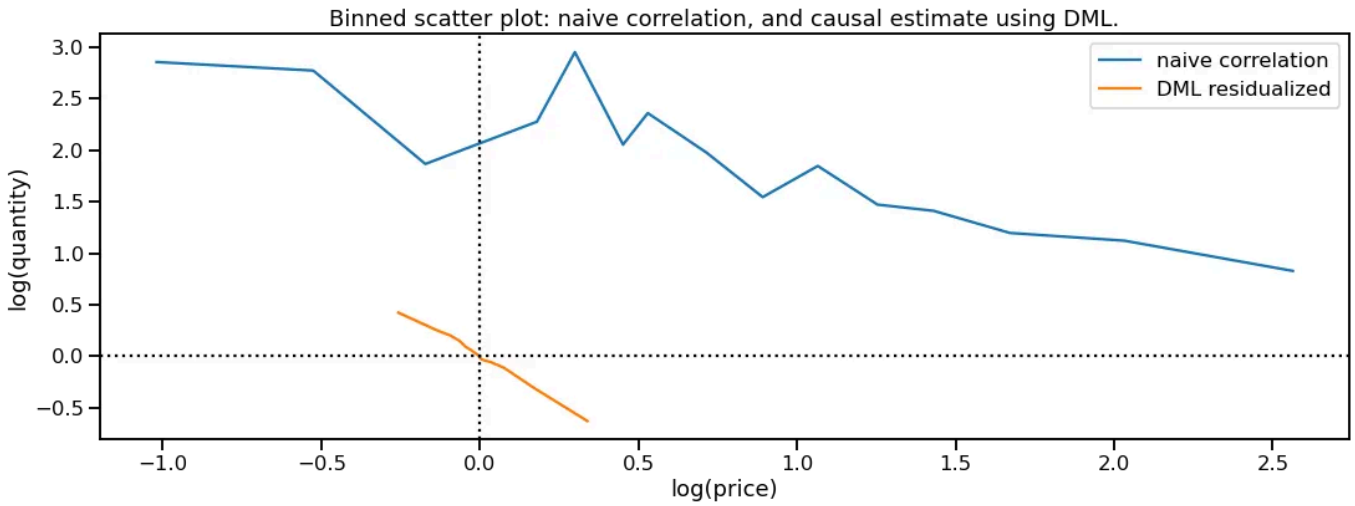
```python
from sklearn.model_selection import KFold

# Since Q might be 0, can't just take logs. This is a quick
# workaround for demonstration. Better options exist.
df_mdl['LnP'] = np.log1p(df_mdl['P'])
df_mdl['LnQ'] = np.log1p(df_mdl['Q'])
elast_estimates = list()

# Step 1: split into two halves
for idx_aux, idx_inf in KFold(
    n_splits=2, shuffle=True).split(df_mdl):

    df_aux = df_mdl.iloc[idx_aux]
    df_inf = df_mdl.iloc[idx_inf].copy()

    # Step 2+3: fit auxiliary models on first half
    model_q.fit(df_aux, df_aux['LnQ'])
    model_p.fit(df_aux, df_aux['LnP'])

    # Step 4: residualize in second half
    df_inf = df_inf.assign(
        LnP_res = df_inf['LnP'] - model_p.predict(df_inf),
        LnQ_res = df_inf['LnQ'] - model_q.predict(df_inf),
    )

    # Step 5: DML inference
    elast = (
        df_inf['LnP_res'].dot(df_inf['LnQ_res'])
        /
        df_inf['LnP_res'].dot(df_inf['LnP'])
        # the last part here deviates from standard OLS solution
    )

    print('DML elasticity:', elast)
    elast_estimates.append(elast)

    print('OLS elasticity for comparison:',
        df_inf['LnP_res'].dot(df_inf['LnQ_res'])
        /
        df_inf['LnP_res'].dot(df_inf['LnP_res'])
    )

# Step 6: Take the mean of both estimates
print("DML efficient estimate of elasticity:", np.mean(elast_estimates))
```

The final output looks like this:

```
1st Fold:
DML elasticity:  -1.90
OLS elasticity for comparison: -1.83
2nd Fold:
DML elasticity:  -1.92
OLS elasticity for comparison: -1.85
DML efficient estimate of elasticity: -1.91
```

So there we have it: our doubly robust, ML-controlled estimate of price elasticity of demand. That is, of the average treatment effect of price on quantity sold. In an industry setting, the retailer may now try to validate this estimate using small experiments (i.e. random price variation).

This final result is visualized in the following binned scatterplot:



This visualization of the DML result shows that controlling for confounders reduces “noisy” variation in our treatment variable and changes the effect size.

In the plot, you can see the “naive” approach in blue. This is simply plotting log-quantity against log-price, without controlling for confounders (i.e., confusing correlation and causation). Even though the relationship is

“messy”, you can already see a negative slope: higher prices lead to lower demand. The regression parameter here is around $\theta \approx -0.6$.

The line in orange is a visualization of the DML result, having controlled for potential confounders. Notice that the slope has gotten steeper, indicating more elastic demand ($\theta \approx -1.9$). You can also observe that the relationship has gotten significantly less noisy, and you can see drastically reduced variance in prices (the range of the line on the x-axis) as a result of DML residualization “explaining away” much of this variation.

5 Discussion

Similar to the example from the beginning, consider a retailer currently selling 100 units of a product for $5 each with a cost of $2. With an estimated elasticity of -1.9, the retailer should decrease prices for more profit. But with a (biased, naive) estimate of -0.6, they would have increased prices.

In this example, appropriate causal inference reverses the recommended decision, with direct impact on the retailer’s profit. Confusing correlation and causation would have cost the retailer real money. In contrast, using a carefully designed DML mechanism, the retailer is able to confidently predict the effect of price changes before they happen.

This article aimed to explain one industry application end-to-end. You can follow each step in much more detail in the runnable notebook. Many business problems benefit from similar approaches; I hope this has been a useful example.

To further optimize prices, the retailer might now move towards heterogeneous treatment effects, i.e. exploring differences in elasticity. This also opens a question of what “average treatment effect” I actually estimated:

whether the mix of products in my training data is representative of the future product mix. I'll explore heterogeneous effects in a follow-up article.

Acknowledgements

Thanks to Rima Rahal, Jonathan Haas, and Paul Hünermund for insightful discussions and comments on early drafts.

References / Endnotes

[1] Taddy, Matt: Business Data Science. McGraw-Hill Education, 2019.

[2] Cunningham, Scott: Causal Inference: The Mixtape. Yale University Press, 2021. https://mixtape.scunning.com/introduction.html

[3] Wager, Stefan: STATS 361: Causal Inference. Stanford University, 2020. https://web.stanford.edu/~swager/stats361.pdf

[4] Varian, Hal R.: Causal inference in economics and marketing. Proceedings of the National Academy of Sciences, Jul 2016.

[5] https://en.wikipedia.org/wiki/Price_elasticity_of_demand

[6] Marshall, Alfred: The Principles of Economics. McMaster University Archive for the History of Economic Thought, 1890. https://socialsciences.mcmaster.ca/econ/ugcm/3ll3/marshall/prin/prinbk5.htm

[7] Chapter 16: Simultaneous Equation Models. In: Wooldridge, Jeffrey: Introductory Econometrics: A Modern Approach, 7th Edition. 2019.

[8] Lee H, Aronson JK, Nunan D: Collider bias. Catalogue Of Bias, 2019. https://catalogofbias.org/biases/collider-bias/

[9] Chernozhukov, Victor, et al.: Double/Debiased Machine Learning for Treatment and Structural Parameters. The Econometrics Journal, Volume 21, 2018. https://academic.oup.com/ectj/article/21/1/C1/5056401

[10] To see that linear log-log regression indeed estimates elasticity, consider this: The log-log regression assumes that

$$\log Q(P) = \theta \log P + \text{intercept} + \epsilon \ (\text{with } \epsilon \sim N(0, \sigma^2))$$

Taking the partial derivative of this regression equation with respect to *P*, you obtain

$$\frac{\partial Q}{\partial P}\frac{1}{Q} = \theta\frac{1}{P} \Rightarrow \theta = \frac{\partial Q/Q}{\partial P/P}$$

[11] Microsoft Research: EconML. A Python Package for ML-Based Heterogeneous Treatment Effects Estimation. https://github.com/microsoft/EconML

[12] Bach, P., Chernozhukov, V., Kurz, M. S., and Spindler, M.: DoubleML — An Object-Oriented Implementation of Double Machine Learning in Python, 2021. arXiv:2104.03220

Machine Learning | Causal Inference | Data Science | Deep Dives | Notes From Industry

Written by Lars Roemheld

Follow

152 Followers · Writer for Towards Data Science

Making machines learn, preferably about causality. http://mdl.fit/

More from Lars Roemheld and Towards Data Science

Lars Roemheld in Towards Data Science

Great Applied (Data) Science Work

What helps solve real-life problems end-to-end, from business requirements to...

13 min read · Aug 23, 2023

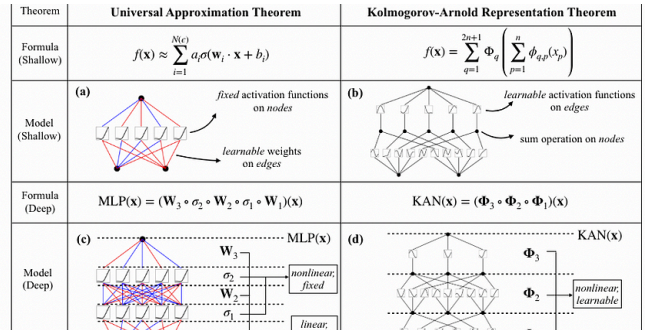
Torsten Walbaum in Towards Data Science

What 10 Years at Uber, Meta and Startups Taught Me About Data...

Advice for Data Scientists and Managers

9 min read · May 30, 2024

228 3



Theo Wolf in Towards Data Science

Kolmogorov-Arnold Networks: the latest advance in Neural Network...

The new type of network that is making waves in the ML world.

· 9 min read · May 12, 2024

2.1K 18

Egor Howell in Towards Data Science

How I Use ChatGPT As A Data Scientist

How ChatGPT improved my productivity as a data scientist

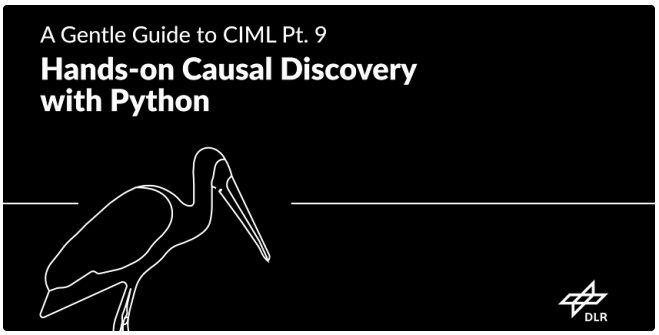
· 8 min read · Jun 2, 2024

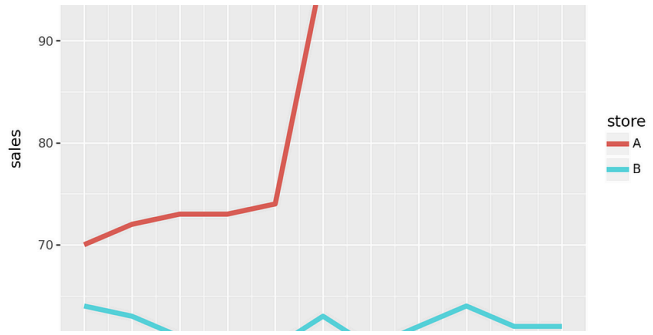
594 19

See all from Lars Roemheld

See all from Towards Data Science

Recommended from Medium





Jakob Runge in Causality in Data Science

Hands-on Causal Discovery with Python

A Gentle Guide to Causal Inference with Machine Learning Pt. 9

14 min read · Mar 4, 2024

234 3

Oliver Chai

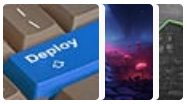
Unraveling Causal Effects with Difference-in-Differences: A...

Introduction: In the realm of data-driven decision-making, understanding the causal...

3 min read · May 10, 2024

4

Lists



Predictive Modeling w/ Python

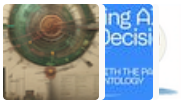
20 stories · 1307 saves

Practical Guides to Machine Learning

10 stories · 1570 saves

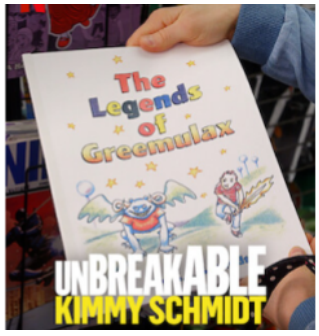
Natural Language Processing

1528 stories · 1061 saves

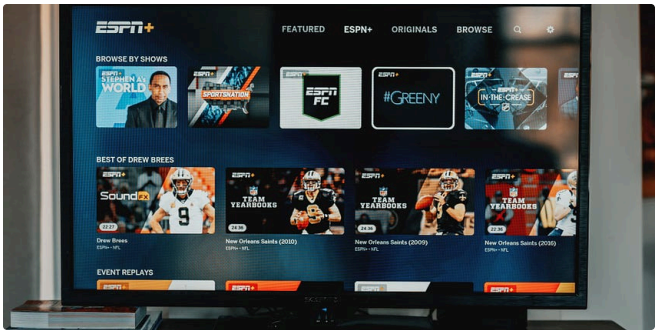


data science and AI

40 stories · 187 saves



Netflix Technology Blog in Netflix TechBlog



Harry Lu in Towards Data Science

Causal Machine Learning for Creative Insights

A framework to identify the causal impact of successful visual components.

13 min read · Jan 11, 2023

963 6

How to use Causal Inference when A/B testing is not available

Evaluating ad targeting product using causal inference: propensity score matching!

7 min read · Jan 8, 2024

548 2

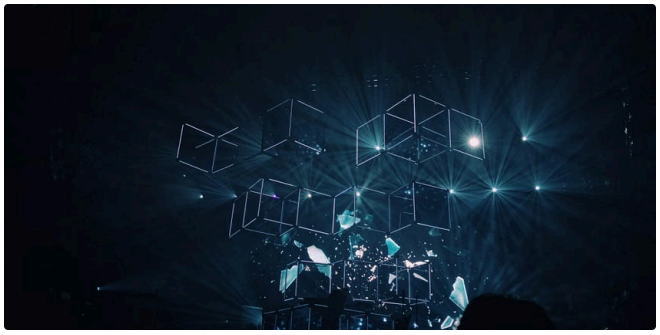
Heinrich Kögel

Causal Machine Learning in Marketing

This article provides a case study that demonstrates how we can leverage causal...

10 min read · Jul 31, 2023

619 13



Prasun Biswas in Walmart Global Tech Blog

Fundamentals of Causal Discovery and Causal Impact Analysis

Why Causal Analysis?

12 min read · Mar 1, 2024

29

See more recommendations