Treatment effects

James Scott

Causal inference

We've used prediction as a basis for model building:

- choose a model to do the best job at forecasting y at new x drawn from the same distribution as data sample X.
- this is exactly the question tackled by ML with cross validation

Causal inference

We've used prediction as a basis for model building:

- choose a model to do the best job at forecasting y at new x drawn from the same distribution as data sample X.
- this is exactly the question tackled by ML with cross validation

But it's not enough for understanding cause and effect.

Today, we'll try to estimate the effect of a special "treatment variable" d. We want to know the **causal** or **treatment effect** (TE), or how y changes when d changes *independently of everything else.* For example:

- Medicine: $d = 1$ if new drug, $d = 0$ if placebo (control).
- Macro: d is a policy tool (interest rates, etc. . .)
- Commerce: d is the price you set for a product

Potential Outcomes

Potential outcomes are used to talk about causality in a formal way:

- Let Y_1 be the *potential outcome* if the treatment is received ($d = 1$), and Y_0 if the treatment is not received ($d = 0$).
- Then the causal effect for an individual is defined as defined as $\mathrm{TE} = Y_1 - Y_0$.

Potential Outcomes

Potential outcomes are used to talk about causality in a formal way:

- Let Y_1 be the *potential outcome* if the treatment is received ($d = 1$), and Y_0 if the treatment is not received ($d = 0$).
- Then the causal effect for an individual is defined as defined as $\mathrm{TE} = Y_1 - Y_0$.

Of course, in reality, we can only observe one of Y_1 and Y_0 for a given individual, not both. This is the **fundamental problem of causal inference:**

- One of these outcomes is *actual*, i.e. directly observed.
- The other is *counterfactual*, and never observed.
- For each unit, we can only observe the potential outcome corresponding to the treatment that was actually received.
- Formally, causal inference is like a missing-data problem!

Assumptions for Estimating Treatment Effects

So estimating treatment effects is impossible, right?

Assumptions for Estimating Treatment Effects

So estimating treatment effects is impossible, right?

Well, kind of! It *is* the hardest game in data-science town. It always requires two *strong assumptions*:

1. **Unconfoundedness:** Treatment assignment is conditionally independent of the potential outcomes, given what we know about the individuals involved.
2. **Overlap:** Each unit has a positive probability of receiving either treatment.

Average Treatment Effect (ATE)

The Average Treatment Effect (ATE) is defined as the expected difference in outcomes between treated and untreated units:

Mathematically, ATE is defined as $\gamma = E[Y_1 - Y_0]$.

This measures the average effect of the treatment on the "population" (however defined).

One setting where it's possible to estimate an average treatment effect is in a designed experiment. We'll start with the simplest case: a completely randomized design.

Completely Randomized Designs

In a **completely randomized** design, each unit is independently assigned to treatment or control with the same probability.

- For example, you randomize your website visitors into groups 'A' and 'B'.
- Those in A see the current website (control, d=0), while those in B see a new layout (treatment, d=1).
- y is the visitor's total spend on that visit.

Under complete randomization, the treatment indicator d is independent of both the potential outcome under treatment (y_1) and the potential outcome under control (y_0).

- Mathematically, this is expressed as $(y_0, y_1) \perp d$
- This independence implies that the treatment and control groups are, on average identical, in terms of *potential* outcomes.

Average Treatment Effects under Randomization

The average treatment effect (ATE) can be estimated in the "obvious" way:

- Let Y_1 and Y_0 be the potential outcomes under treatment and control respectively
- Y is the observed outcome.
- d is the treatment assignment.

Then the ATE is

$$\gamma = E[Y_1 - Y_0] = E[Y \mid d = 1] - E[Y \mid d = 0]$$

Just use the sample means in treatment and control groups to get unbiased estimator of the ATE under complete randomization. **It's just Stat 101 analysis of designed experiments.**

Can you do better?

Completely randomized designs (experiments, RCTs, A/B tests. . .) are immensely popular and immensely useful.

Sometimes simple is best. If you have the ability to randomize, it is very tough to find a TE estimate that is much better than $\hat{\gamma} = \bar{y}_1 - \bar{y}_0$ from an RCT. Beware of those making extravagant claims otherwise.

However, we *can* sometimes do better, especially if:

- there are many treatments to choose among
- the treatment effect is heterogeneous, i.e. we view $\gamma(x)$ as being a function of other features.

Multi-Arm Bandit Problem

We'll consider the case of multiple treatments as an example of the **multi-armed bandit** problem. This is a good working model for a lot of experiments in industry:

- you want to learn what's best...
- but you also want to quickly *make use of* what's best.

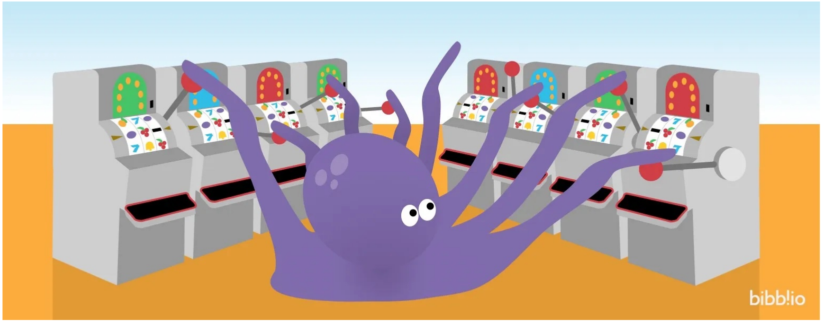


Figure 1: This guy can play all the arms at once... we're not so lucky.

Multi-Arm Bandit Problem

In the classic problem, a gambler has to decide which arm of K slot machines (or "bandits") to pull to maximize their total reward over a series of trials.

- Some bandits are more favorable than others...
- But we don't know which ones or by how much. We can only find out by actually pulling them.

This is a classic model in reinforcement learning, which is about balancing the trade-off between *exploration* and *exploitation.*

- Exploration: learning which treatments work best
- Exploitation: assigning users to the treatment that seems best in light of our current (partial) information

Example: online ad campaigns

In the context of online ad campaigns, each ‘arm’ can be considered as a different ad campaign.

- We have J different ads to show, denoted as $j = 1, ..., J$.
- Each time a user comes to our site, we can show them one ad, indicated by $d_i = j$.
- The objective is to maximize ad-clicks over all visitors, which can be seen as the rewards in the bandit problem.

Formulation as a Multi-Armed Bandit Problem

To formulate this scenario as a multi-armed bandit problem, we define:

- **States:** Each unique combination of user characteristics is a different state.
- **Actions:** Showing a particular ad campaign is an action.
- **Rewards:** Clicks on ads are rewards.

At each time step i (when a user arrives), we choose an action d_i (choose an ad to show), then we receive a reward based on whether the user clicks the ad.

Choosing Ads with Bandit Algorithms

To solve the multi-armed bandit problem, we can use time-tested algorithms like epsilon-greedy or Thompson sampling.

- These algorithms balance the trade-off between exploiting ads that have performed well in the past and exploring new ads that might perform better.
- The choice of algorithm can have a significant impact on the total reward (total ad-clicks) over time.

Notation:

- Let's say that $s_n = [s_{n1}, \ldots, s_{nJ}]$ are the number of times each ad has been shown up through user N.
- Let's also say that $c_n = [s_{c1}, \ldots, c_{nJ}]$ are the number of clicks on each ad through user N.

The Epsilon-Greedy Algorithm

The epsilon-greedy algorithm is a simple, effective method for balancing exploration and exploitation.

- At each time step n (when a new user arrives), with a small probability ϵ, we randomly select an ad to display (exploration).
- With probability $1 - \epsilon$, we display the ad with the highest observed click-through rate so far (exploitation).

We calculate the observed click-through rate for ad j as c_{nj}/s_{nj}, the currently observed success rate.

- Note: we can add a small pseudo-count and use e.g. $(c_{nj} + 1)/(s_{nj} + 1)$ to avoid division by zero.

Epsilon-greedy: pseudo-code

```
Initialize s_n and c_n as zero vectors of length J
for each user n do
  Generate a random number r from U(0,1)
  if r < epsilon then
    Select a random ad j to display
  else
    ctr_j = c_{nj}/s_{nj}
    Display ad with the highest ctr_j
  end if
  Update s_n and c_n based on whether the user clicks
end for
```

Thompson Sampling Algorithm

Thompson Sampling is a probabilistic algorithm that balances exploration and exploitation by maintaining a Bayesian posterior distribution for each ad's click-through rate.

- At each time step n, we sample a click-through rate θ_j from the posterior distribution for each ad j's true, unknown click-through rate.
- We show the ad with the highest sampled click-through rate.
- The uncertainty in our posterior distribution ensures that we will do some exploration rather than always choose the current best option.

Under a common choice of a Beta prior for each ad's click-through rate, the posterior distribution after seeing s_{nj} displays and c_{nj} clicks for ad j is a Beta distribution with parameters $c_{nj} + 1$ and $s_{nj} - c_{nj} + 1$.

Thompson sampling: pseudo-code

```
Initialize s_n and c_n as zero vectors of length J
for each user n do
  for each ad j do
    Draw theta_j ~ Be(c_{nj} + 1, s_{nj} - c_{nj} + 1)
  end for
  Display the ad with the highest theta_j
  Update s_n and c_n based on whether the user clicks
end for
```

Example

Let's look at `MAB.R` to compare these two strategies versus a completely randomized design.

High-Dimensional Confounding

We now turn to causal inference with non-experimental data, i.e. in the presence of confounding.

Recall the Stat 101 setup and recipe here:

- y is the response, d is the treatment, and d seems strongly predictive of y...
- But d is also correlated with some other variables x (the *confounders*).
- So run a regression of y on d and x to get an estimate for the *partial effect* of d on y, holding x constant.

High-Dimensional Confounding

We now turn to causal inference with non-experimental data, i.e. in the presence of confounding.

Recall the Stat 101 setup and recipe here:

- y is the response, d is the treatment, and d seems strongly predictive of y...
- But d is also correlated with some other variables x (the *confounders*).
- So run a regression of y on d and x to get an estimate for the *partial effect* of d on y, holding x constant.

Let's see a quick example as a refresher:

- y = price of an Airbnb rental in Sante Fe, NM
- d = distance to the center of town
- x = other stuff about the rental

Airbnb example

The relationship between `Price` and `PlazaDist` seems strong:

```
lm0 = lm(Price ~ PlazaDist, data=airbnb)
get_regression_table(lm0) %>%
  select(term, estimate, std_error)
```

```
## # A tibble: 2 x 3
##   term      estimate std_error
##   <chr>        <dbl>     <dbl>
## 1 intercept    218.       18.6
## 2 PlazaDist    -45.6      14.6
```

Airbnb example

But this is a naive answer because bigger places tend to be a bit closer to the center of town:

```
cor(PlazaDist ~ Bedrooms, data=airbnb)
```

```
## [1] -0.1096495
```

```
cor(PlazaDist ~ Baths, data=airbnb)
```

```
## [1] -0.2732923
```

So in estimating a `PlazaDist` effect on `Price`, we are also implicitly including a size effect! Causal confusion, a.k.a. **confounding.**

Airbnb example

What if we adjust for size by adding bedrooms and bathrooms? Now the distance effect looks weaker:

```
lm1 = lm(Price ~ PlazaDist + Bedrooms + Baths, data=airbnb)
get_regression_table(lm1) %>%
  select(term, estimate, std_error)
```

```
## # A tibble: 4 x 3
##   term      estimate std_error
##   <chr>        <dbl>     <dbl>
## 1 intercept     10.5     19.7
## 2 PlazaDist    -16.7      9.10
## 3 Bedrooms      30.0     11.9
## 4 Baths        100.      15.3
```

That's because beds/baths both have large effects on y *and* were correlated with distance. This led to causal confusion in our first model!

High-Dimensional Confounding

Unfortunately, this "Stat 101" approach breaks down in the presence of lots and lots of confounders.

Why? **Overfitting.**

- With a large number of confounders relative to the number of observations, the model is likely to overfit the data.
- When you force the model to control for every crazy possibility in a high-D x, it will!
- Result: massively inflated variance of the estimated treatment effect.

High-Dimensional Confounding

Unfortunately, this “Stat 101” approach breaks down in the presence of lots and lots of confounders.

Why? **Overfitting.**

- With a large number of confounders relative to the number of observations, the model is likely to overfit the data.
- When you force the model to control for every crazy possibility in a high-D x, it will!
- Result: massively inflated variance of the estimated treatment effect.

OK, so why not just run the LASSO?!

- It performs L1 regularization, which has the effect of shrinking some regression coefficients exactly to zero.
- This performs both variable selection and regularization, helping to mitigate overfitting.

LASSO for treatment effect estimation?

Seems like a no-brainer to use LASSO regression to estimate the treatment effect in a regression framework:

```
y ~ d + x
```

where:

- y is the outcome,
- d is the treatment indicator, and
- x is the vector of potential confounders.
- the whole model is fit with a single run of LASSO-CV.

LASSO for treatment effect estimation?

Seems like a no-brainer to use LASSO regression to estimate the treatment effect in a regression framework:

```
  y ~ d + x
```

where:

- y is the outcome,
- d is the treatment indicator, and
- x is the vector of potential confounders.
- the whole model is fit with a single run of LASSO-CV.

Please don't do it! It's a causal-inference disaster.

Let's take a simple counter-example to show why it won't work.

Why naive LASSO is a (potential) disaster

Suppose the true data-generating process is this:

- $y = x_1 + x_2 + e_y$. No treatment effect, but the confounders have true effects on the outcome.
- $d = x_1 + x_2 + e_d$. The confounders strongly predict the treatment.
- E.g.: y is getting a fancy job, d is Harvard attendance, x_1 is whether parents are wealthy, x_2 is Harvard legacy status.

This model is *structural*, in the sense that it is assumed to generate the *correct potential outcomes*, conditional on covariates. To wit:

- $E(Y_0 \mid x_1, x_2) = x_1 + x_2$
- $E(Y_1 \mid x_1, x_2) = x_1 + x_2$
- Therefore $TE = E(Y_1) - E(Y_0) = 0$. No treatment effect!

Why naive LASSO is a (potential) disaster

But notice that, since $E(d \mid x_1, x_2) = x_1 + x_2$, I could consider two perfectly good "sparse" regression models for $E(y)$:

- $E(y) = \beta_1 x_1 + \beta_2 x_2$
- $E(y) = \beta_d d$

The second model isn't structural:

- It doesn't specify the correct potential outcomes, which we know don't depend on d! (It's what econometricians would call a "reduced-form" model.)
- But the LASSO would strongly prefer it: it predicts just as well as the correct structural model, but it only costs $1 \cdot \lambda$ rather than $2 \cdot \lambda$ to "unzero" its coefficients.
- The lasso cares about *prediction* and *parsimony*, not correct causal identification.

(Note: x is low-D and so "naive" OLS would do great!)

Why naive LASSO is a (potential) disaster

This is a quite general problem with the LASSO (or anything similar that "regularizes" the model fit – i.e. *everything* in ML!).

Using LASSO regression directly for causal inference can lead to biased treatment-effect estimates:

- LASSO can zero out important confounders due to the L1 penalty.
- This is particularly problematic when the confounders are highly correlated with the treatment assignment.

Example

Let's see an example on simulated data to build our intuition.

```r
N = 100
P = 200 # lots of confounders

# matrix of confounders
X = matrix(runif(N*P), N, P)

# same 10 confounders affect treatment/outcome
D = rowSums(X[,1:10]) + rnorm(N, 0, 0.1)
Y = rowSums(X[,1:10]) + rnorm(N, 0, 1)
```

Example

Now we run the naive lasso:

```r
lm_naive = gamlr::cv.gamlr(cbind(D, X), Y)
coef(lm_naive) %>% head(12)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                seg19
## intercept 2.0002861
## D         0.6204517
##           .
##           .
##           .
##           .
##           .
##           .
##           .
##           .
##           .
##           .
```

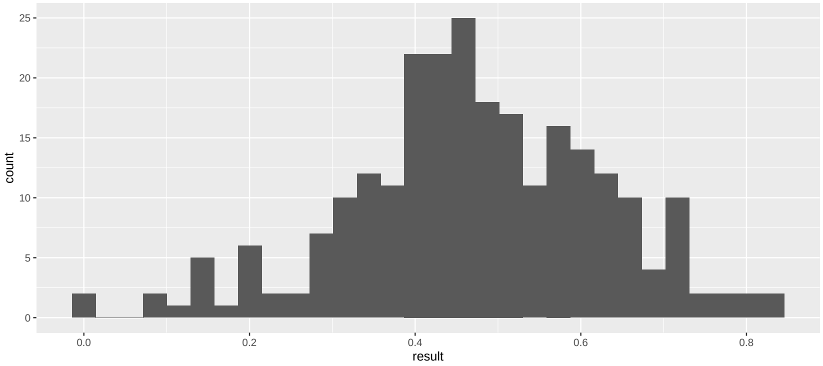
Example

This isn't just bad luck. Here's we are simulating the same situation many times:

```
sim = do(250)*{
  X = matrix(runif(N*P), N, P)
  D = rowSums(X[,1:10]) + rnorm(N, 0, 0.1)
  Y = rowSums(X[,1:10]) + rnorm(N, 0, 1)
  lm_naive = gamlr::cv.gamlr(cbind(D, X), Y)
  coef(lm_naive)[2]
}
```

Example

The histogram of our treatment-effect estimates looks like this:



The bias is terrible.

Double-Selection or "Double LASSO"

A particularly simple solution is the so-called "double-selection" procedure of Belloni, Chernozhukov, and Hansen (2014):

1. **Selection Step 1**: Use LASSO to select variables that are predictive of the treatment d.
2. **Selection Step 2**: Use LASSO to select variables that are predictive of the outcome y.
3. **Inference Step**: Use the variables selected from both steps in an ordinary regression of the outcome on the treatment, plus the selected controls: `y ~ d + x_selected`

The coefficient on the treatment indicator d in this regression represents the estimated treatment effect.

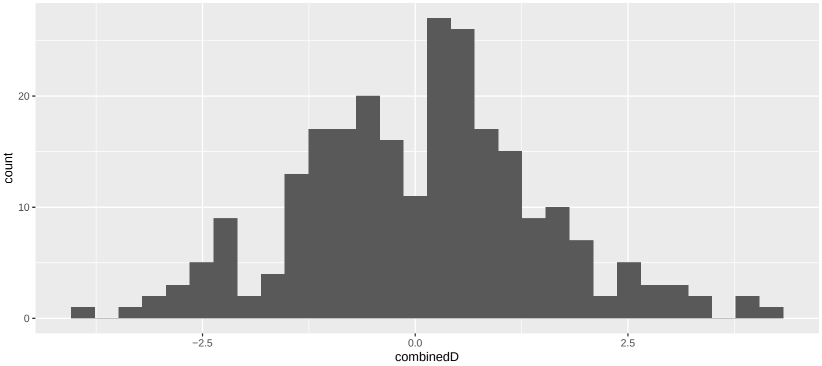
Not a panacea, but not a bad approach! Theory says that the standard error of the d coefficient should be about right.

Double lasso for our toy example

```
sim2 = do(250)*{
  X = matrix(runif(N*P), N, P)
  D = rowSums(X[,1:10]) + rnorm(N, 0, 0.1)
  Y = rowSums(X[,1:10]) + rnorm(N, 0, 1)
  lm_d = gamlr::cv.gamlr(X, D)
  include_from_d = which(coef(lm_d) != 0) - 1
  lm_y = gamlr::cv.gamlr(X, Y)
  include_from_y = which(coef(lm_y) != 0) - 1
  include = union(include_from_d, include_from_y)
  combined = cbind(D, X[,include])
  lm_double = lm(Y ~ combined)
  coef(lm_double)[2]
}
```

Double lasso for our toy example

The histogram of our treatment-effect estimates now looks like this:



Huge variability, centered at 0. **No bias.**

A useful diagnostic plot

A really useful plot in these regressions is to check how much independent or "quasi-experimental" variation remains in the treatment variable d, once we've regressed it on the control variables.

The thinking goes like this:

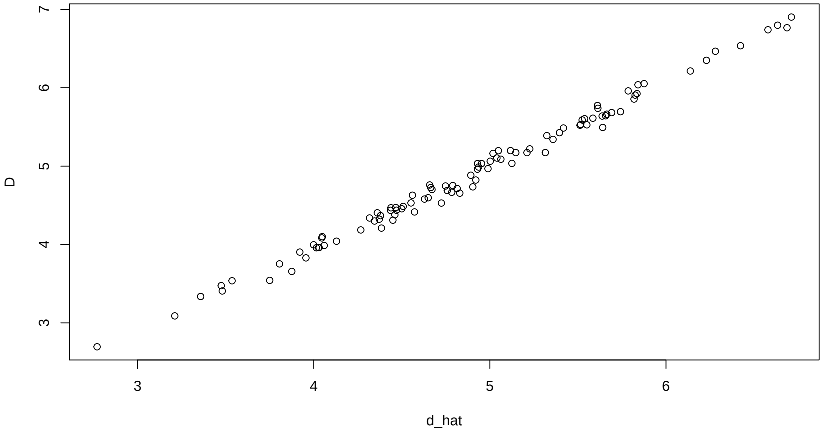
- we judge cause and effect by trying to correlate variation in y with variation in d, *after* we've controlled for variation that can be explained by the confounders.
- high R^2 for y vs. d: bad. The confounders *strongly* predict the treatment. No variation leftover for causal inference.
- low R^2 for y vs. d: we might be OK. The treatment seems to vary at least somewhat, independently of the confounders.

A useful diagnostic plot

```
X = matrix(runif(N*P), N, P)
D = rowSums(X[,1:10]) + rnorm(N, 0, 0.1)
Y = rowSums(X[,1:10]) + rnorm(N, 0, 1)
lm_d = gamlr::cv.gamlr(X, D)
d_hat = predict(lm_d, X)
```

A useful diagnostic plot

Our example? No hope.



A real example

Donahoe and Levitt argue a controversial thesis: easier access to abortion causes decreased crime.

Made famous in Freakonomics. Maybe you read it.

There's obviously no experiment here. How have they controlled for confounders?

A real example

The treatment variable d is by-state, by-year lagged abortion rate, and for response we look at $y =$ murder rate.

DL control for bunch of state-specific confounders: income, poverty, child tax credits, weapons laws, beer consumption...

They also include state effects (factor 's') and a linear time trend (numeric 't') to control for missed confounders.

```
> orig = glm(y ~ d + t + s + ., data=controls)
> summary(orig)$coef['d',]
     Estimate    Std. Error       t value      Pr(>|t|)
-2.098119e-01  4.109177e-02 -5.105936e+00  4.505925e-07
```

A real example

The treatment variable d is by-state, by-year lagged abortion rate, and for response we look at $y =$ murder rate.

DL control for bunch of state-specific confounders: income, poverty, child tax credits, weapons laws, beer consumption...

They also include state effects (factor 's') and a linear time trend (numeric 't') to control for missed confounders.

```
> orig = glm(y ~ d + t + s + ., data=controls)
> summary(orig)$coef['d',]
     Estimate    Std. Error       t value      Pr(>|t|)
-2.098119e-01  4.109177e-02 -5.105936e+00  4.505925e-07
```

Skeptical? You should be! Let's visit `abortion.R`.