Double/Debiased Machine Learning

There is a rapidly growing recent literature on **double machine learning**, also called **double/debiased machine learning**:

Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins, "Double/debiased machine learning for treatment and structural parameters," *Econometrics Journal*, 2018, 21, C1–C68. (3595 cites)

Alexandre Belloni, Victor Chernozhukov, and Christian Hansen, "High-dimensional methods and inference on structural and treatment effects," *Journal of Economic Perspectives*, 2014, 28, 29–50. (1119 cites)

Philipp Bach, Victor Chernozhukov, Malte S. Kurz, and Martin Spindler, "DoubleML–An object-oriented implementation of double machine learning in python," *Journal of Machine Learning Research*, 2022, 23, 1–6. (120 cites)

And many others! Early methods were called **double selection**.

Victor Chernozhukov, Christian Hansen, Nathan Kallus, Martin Spindler, and Vasilis Syrgkanis, *Applied Causal Inference Powered by ML and AI*, arXiv:2403.02467 provides a detailed introduction to these methods; go to `https://causalml-book.org`.

An R (and Python) package for double/debiased machine learning called `DoubleML` is available at

`http://docs.doubleml.org/`

It provides a great many capabilities and extensive documentation. It is not to be confused with the Python `causalml` package.

Several methods for double ML are implemented in Stata. The `ddml` package implements random forests, boosting, and lasso.

Stata itself (V16 and later) implements several types of lasso, including double-selection lasso, partialing-out lasso, and DML lasso for both linear regression and logit models.

See `https://www.stata.com/manuals/lasso.pdf`.

Key Ideas of DDML

Suppose we want to estimate the **partially linear regression**, or **PLR**, model,

$$y_i = \beta d_i + g(\boldsymbol{x}_i) + u_i \tag{1}$$

where d_i is a "treatment" variable, but not one that is assigned at random, and $\boldsymbol{x}_i$ is a vector that contains p control variables.

We want to obtain an estimate of β and make inferences about it.

The model (1) is just a special case of a generalized additive model.

There are n observations, p may be large relative to n, and the function $g(\boldsymbol{x}_i)$ is unknown.

It might seem that we could simply use some type of machine learning procedure to estimate $g(\boldsymbol{x}_i)$.

If treatment were assigned at random, this would probably work fine.

When p is large and possible nonlinearities have been taken into account by including powers and cross-products in $\boldsymbol{x}_i$, we could use lasso or ridge. Lasso would make sense if sparsity is expected.

When p is not large, and we are fairly sure what inputs really matter, we could use a GAM, with $g(\boldsymbol{x}_i)$ treated as a sum of smooth functions, each involving one or a few inputs.

When p is either large or small, we could use a random forest, or perhaps a set of boosted trees, or a neural network.

Suppose the treatment is not assigned at random. Instead,

$$d_i = m(\boldsymbol{x}_i) + v_i, \tag{2}$$

where $m(\boldsymbol{x}_i)$ is an unknown function for the **propensity score**.

The propensity score is just the probability of treatment, so (2) might well be replaced by some sort of binary response model.

In the classic case where both unknown functions are linear and $\boldsymbol{x}_i$ is known and of low dimension, there is usually no real problem.

We just regress $\boldsymbol{y}$ on $\boldsymbol{d}$ and $\boldsymbol{X}$, where $\boldsymbol{y}$ and $\boldsymbol{d}$ have typical elements y_i and d_i, and $\boldsymbol{X}$ has typical row $\boldsymbol{x}_i$. The presence of the latter ensures that we obtain consistent estimates of β.

Provided every explanatory variable that appears in either of the two equations is included in $\boldsymbol{x}_i$, we don't even have to estimate the propensity score function (2).

But when $\boldsymbol{x}_i$ is of high dimension, simply regressing $\boldsymbol{y}$ on $\boldsymbol{d}$ and $\boldsymbol{X}$ will not work well. We will probably get very imprecise estimates of β.

This problem was investigated by Chernozhkov and coauthors in more than one paper. When $\boldsymbol{x}_i$ contains a lot of variables, but most of them are expected to have zero coefficients, they proposed using variations of the lasso.

In the general case, the model is given by equations (1) and (2).

When we don't know $g(\cdot)$ in (1), or $m(\cdot)$ in (2), we run into the problem of **regularization bias**.

Any machine-learning estimator of $g(\cdot)$ has to employ some sort of regularization, which induces bias.

Let $\hat{g}_i$ denote an ML estimator for the $g(\boldsymbol{x}_i)$ part of (1).

Then the obvious way to estimate β is just to regress $y_i - \hat{g}_i$ on d_i.

If $\boldsymbol{y}$, $\boldsymbol{d}$, and $\boldsymbol{g}$ are vectors with typical elements y_i, d_i, and $g(\boldsymbol{x}_i)$, this regression yields the OLS estimator

$$\hat{\beta} = (\boldsymbol{d}^\top \boldsymbol{d})^{-1} \boldsymbol{d}^\top (\boldsymbol{y} - \hat{\boldsymbol{g}}). \tag{3}$$

The vector $\hat{\boldsymbol{g}}$ here is a **generated regressor**.

It is often tempting, or even essential, to use generated regressors. But doing so can cause serious problems for estimation and inference.

In general, the properties of generated regressors differ from those of ordinary ones.

Even when they converge to the right things asymptotically, they may do so too slowly for standard asymptotic analysis to yield good approximations.

If $\hat{g}$ were unbiased for g_0, the estimator $\hat{\beta}$ would itself be unbiased, and it would probably work well in many cases.

Unfortunately, for essentially all ML estimators, $\hat{g}$ is not unbiased. This is especially true when p is large.

Chernozhukov et al. (2018) studies the asymptotic properties of $\hat{\beta}$. They turn out to be pretty bad, in general.

Since $y = \beta_0 d + g_0 + u$,

$$\hat{\beta} = \beta_0 + (d^\top d)^{-1} d^\top (g_0 - \hat{g} + u). \tag{4}$$

After a little algebra and the insertion of appropriate factors of n, we find that

$$n^{1/2}(\hat{\beta} - \beta_0) = (n^{-1}\boldsymbol{d}^\top\boldsymbol{d})^{-1}n^{-1/2}\boldsymbol{d}^\top\boldsymbol{u} + (n^{-1}\boldsymbol{d}^\top\boldsymbol{d})^{-1}n^{-1/2}\boldsymbol{d}^\top(\boldsymbol{g}_0 - \hat{\boldsymbol{g}}). \tag{5}$$

The first term has standard properties. It has mean 0 and is $O_p(1)$.

But the second term does not converge. The quantity $n^{-1/2}\boldsymbol{d}^\top(\boldsymbol{g}_0 - \hat{\boldsymbol{g}})$ is a sum of n terms, divided by $n^{1/2}$. Each of these terms has a non-zero mean, because $\mathrm{E}(\boldsymbol{g}_0 - \hat{\boldsymbol{g}}) \neq \mathbf{0}$.

Although the bias diminishes as n increases, it always does so more slowly than $n^{-1/2}$. Therefore, the second term actually diverges.

This does not imply that $\hat{\beta} - \beta_0$ diverges. It does not. But because $n^{1/2}$ times it diverges, $\hat{\beta} - \beta_0$ converges more slowly than it should under standard asymptotics, and the bias can be large.

How fast $\hat{\beta}$ converges to β_0 depends on how fast $\hat{\boldsymbol{g}}$ converges to $\boldsymbol{g}_0$. For all machine-learning methods, this is slower than $n^{-1/2}$.

The way to overcome regularization bias is to estimate *two* equations via machine learning.

This is called **double machine learning**.

The first equation is not (1). It is just

$$y_i = g(\boldsymbol{x}_i) + u_i, \tag{6}$$

and the object is to obtain the residuals $\hat{\boldsymbol{u}}$.

The second equation is (2), the equation that explains the treatment variable. The idea is to "partial out" the effects of $\boldsymbol{x}_i$ on d_i. What we need to obtain are the residuals $\hat{\boldsymbol{v}} = \boldsymbol{d} - \hat{\boldsymbol{m}}(\boldsymbol{X})$.

Regressing $\hat{u} = y - \hat{g}$ on $\hat{v} = d - \hat{m}$ yields the double-ML, or **partialing-out**, estimator

$$\begin{aligned}\check{\beta} &= \left((d - \hat{m})^\top (d - \hat{m})\right)^{-1} (d - \hat{m})^\top (y - \hat{g}) \\ &= (\hat{v}^\top \hat{v})^{-1} \hat{v}^\top \hat{u}.\end{aligned} \tag{7}$$

This looks as if we are using the FWL theorem! We are regressing a vector of residuals on another vector of residuals.

If everything were linear, and we just used OLS, the estimate $\check{\beta}$ from (7) and the estimate $\hat{\beta}$ from (3) would be identical:

$$\hat{\beta} = (d^\top M_X d)^{-1} d^\top M_X y \tag{8}$$

$$\check{\beta} = (\hat{v}^\top \hat{v})^{-1} \hat{v}^\top M_X y = (d^\top M_X d)^{-1} d^\top M_X y. \tag{9}$$

The last equality uses the fact that M_X is idempotent; that is, $M_X M_X = M_X$.

But since nothing is linear here, $\check{\beta} \neq \hat{\beta}$, and it turns out that $\check{\beta}$ has much better properties than $\hat{\beta}$.

Chernozhukov et al. show that $n^{1/2}(\check{\beta} - \beta_0)$ is the sum of three terms. The first term is

$$\mathrm{E}(v^2)^{-1} n^{-1/2} \boldsymbol{v}^\top \boldsymbol{u} \xrightarrow{d} \mathrm{N}(0, \sigma_\beta^2), \tag{10}$$

where σ_β^2 is the asymptotic variance of $n^{1/2}(\check{\beta} - \beta_0)$, which can be estimated consistently in the usual way for the regression of $\hat{\boldsymbol{u}}$ on $\hat{\boldsymbol{v}}$.

The second term is

$$\mathrm{E}(v^2)^{-1} n^{-1/2} (\hat{\boldsymbol{m}} - \boldsymbol{m}_0)^\top (\hat{\boldsymbol{g}} - \boldsymbol{g}_0). \tag{11}$$

This depends on the product of the estimation errors in the two first-stage equations.

Even if both $\hat{\boldsymbol{m}}$ and $\hat{\boldsymbol{g}}$ suffer from considerable regularization bias, the product in (11) should vanish much faster than either of their squares would vanish.

There is a third term as well. It will go away asymptotically even without sample splitting, but sample splitting (to be discussed below) makes it go away faster.

Figure 20.1 illustrates how double-ML estimation can dramatically reduce bias. In this case, the $g(\cdot)$ and $m(\cdot)$ functions are estimated using random forests.

This figure is taken from the 2018 paper. The function $g_0(\cdot)$ is a smooth function of a small number of variables, so random forests should work well.

The red curves are normal approximations based on standard asymptotic theory. The blue histograms show the actual distributions.

- The naive estimator in the left panel performs dreadfully.
- The DML estimator in the right panel is approximately unbiased and is much closer to the asymptotic approximation.

Figure 20.1 — Naive versus double-ML estimators

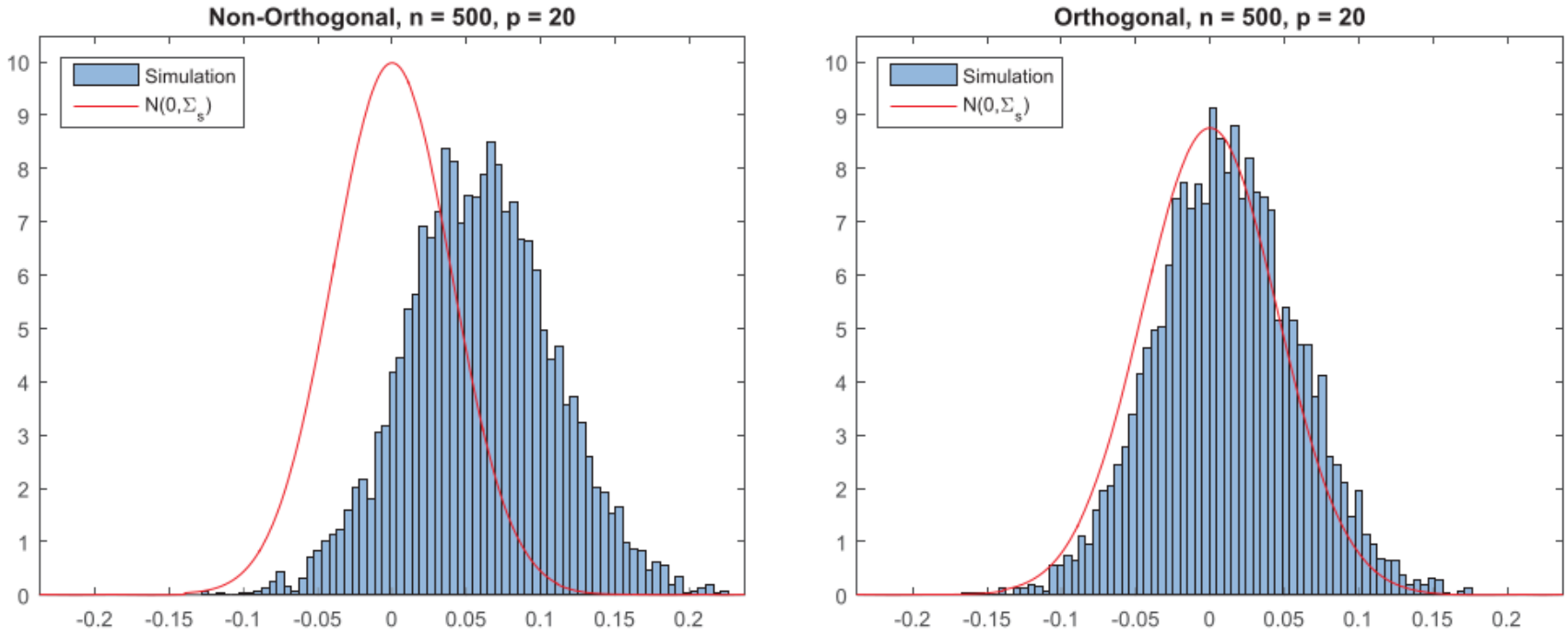


Figure 1. Comparison of the conventional and double ML estimators. [Colour figure can be viewed at wileyonlinelibrary.com]

Double Selection

If we are willing to assume linearity, but with p large and potentially zero coefficients on many columns of $\boldsymbol{X}$, we can use the lasso.

The obvious approach is a **naive lasso** procedure, which applies lasso to a regression of $\boldsymbol{y}$ on $\boldsymbol{d}$ and $\boldsymbol{X}$.

If $\boldsymbol{X}^*$ is a matrix that contains the columns of $\boldsymbol{X}$ which survive the lasso, we then run the regression

$$\boldsymbol{y} = \beta \boldsymbol{d} + \boldsymbol{X}^* \boldsymbol{\gamma} + \boldsymbol{u} \tag{12}$$

Chapter 4 of the book explains theoretically why this fails.

In this case, the bias in $\hat{\gamma}$ causes $\hat{\beta}$ to be biased, and the bias goes away so slowly that t-statistics on $\hat{\beta}$ do not have mean 0 in finite samples.

Instead of the naive lasso, Chernozhukov et al. propose two different procedures.

The first is a **double-ML** procedure that the book calls **double lasso**. It starts with two separate lasso regressions.

- Run a lasso of y_i on $\boldsymbol{x}_i$ and another lasso of d_i on $\boldsymbol{x}_i$. These yield residuals $\check{\boldsymbol{y}}$ and $\check{\boldsymbol{d}}$.
- Optionally, replace these residuals by ones from the relaxed lasso, or from post-lasso regressions using all the selected columns of $\boldsymbol{X}$ in both of them.
- Regress $\check{\boldsymbol{y}}$ on $\check{\boldsymbol{d}}$ to estimate β.

Under certain (fairly strong) assumptions, the resulting $\tilde{\beta}$ is almost unbiased, and suitable t-statistics based on it yield valid inferences.

This only works if the (true) parameters of the lasso regressions satisfy certain sparsity conditions, and the tuning parameters are chosen carefully.

The authors recommend a plug-in procedure for choosing the tuning parameter. Cross-validation may not work well.

There is an alternative, and asymptotically equivalent, procedure called **double selection** that starts in the same way, with the two lasso regressions. After that, it proceeds as follows:

- Find all regressors that have non-zero coefficients in either lasso. Let $\tilde{\boldsymbol{X}}$ denote the union of those two sets of regressors.
- Run the OLS regression

$$\boldsymbol{y} = \beta \boldsymbol{d} + \tilde{\boldsymbol{X}}\boldsymbol{\gamma} + \boldsymbol{u}. \tag{13}$$

- This yields the double selection estimator

$$\hat{\beta} = (\boldsymbol{d}^\top \boldsymbol{M}_{\tilde{\boldsymbol{X}}} \boldsymbol{d})^{-1} \boldsymbol{d}^\top \boldsymbol{M}_{\tilde{\boldsymbol{X}}} \boldsymbol{y}. \tag{14}$$

Compare (14) with (9). The only difference is that $\boldsymbol{M}_{\tilde{\boldsymbol{X}}}$ has replaced $\boldsymbol{M}_{\boldsymbol{X}}$.

Amazingly, conventional standard errors for $\hat{\beta}$ are asymptotically valid. These should usually be hetero-robust or cluster-robust.

Figure 20.2, which is taken from Chapter 4 of the book, illustrates how the naive lasso procedure can work badly when the double lasso procedure works well.

It is hard to see the true value, so I have added "x" in two places

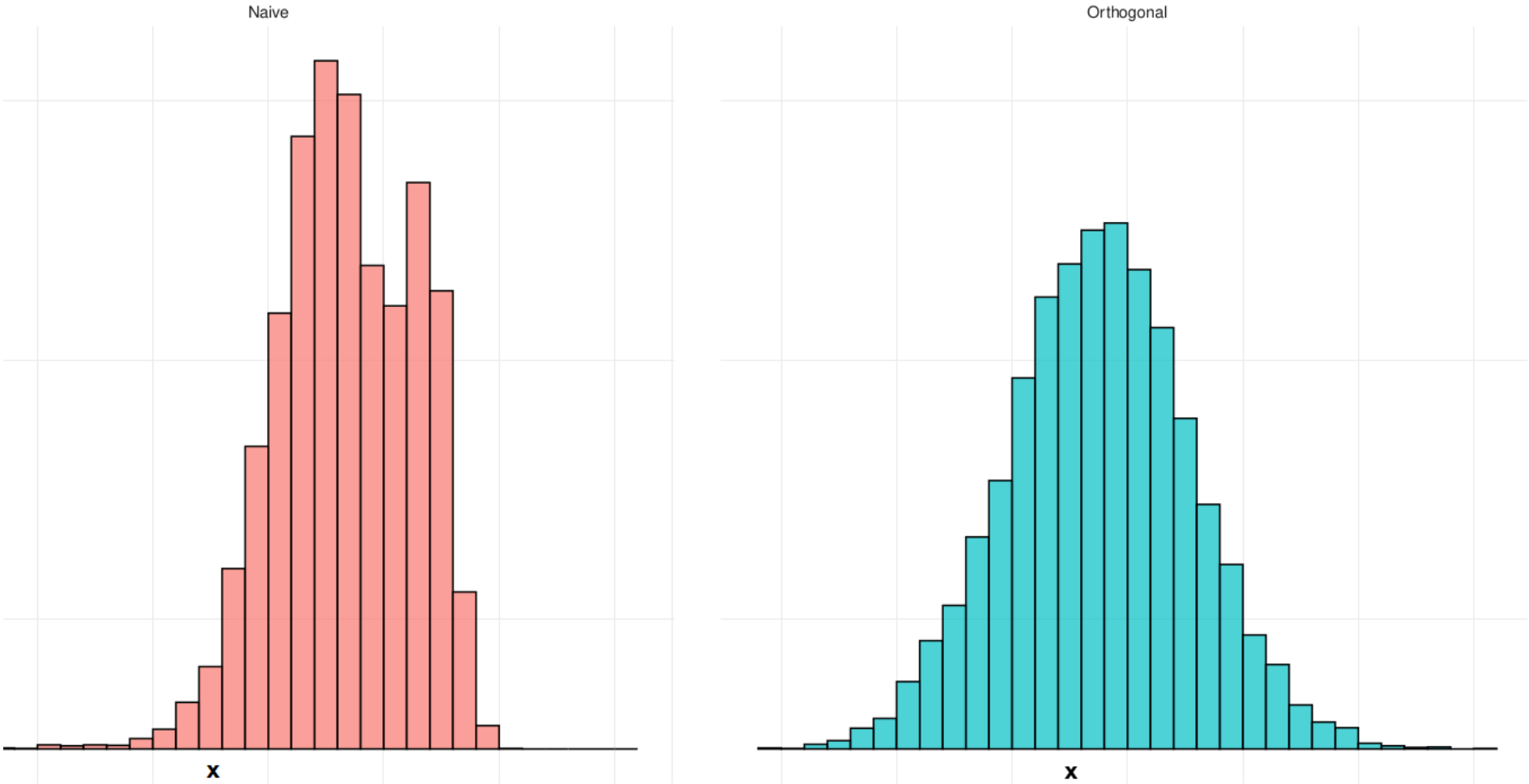
It is evident that $\hat{\beta}$ is severely biased for naive lasso, but much less biased for double lasso.

Figure 20.2 looks a lot like Figure 20.1, even though the models and machine-learning procedures are quite different.

Chapter 4 of the book also discusses yet another lasso estimator that is analogous to IV. It is called **desparsified lasso**. We will discuss this sort of estimator later.

Although `Stata` is generally not as good as `R` for machine learning, it does provide an extensive set of commands for double-selection lasso (`dsregress`, `dslogit`), partialing-out lasso (`poresgress`, `pologit`, and partialing out that uses cross-fitting (`xporegress`, `xpologit`).

Figure 20.2 — Naive versus double lasso



The concept of double selection preceded the concept of double machine learning. The key paper is

Alexandre Belloni, Victor Chernozhukov, and Christian Hansen, "Inference on treatment effects after selection among high-dimensional controls," *Review of Economic Studies*, 2014, 81, 608–650. (2039 cites)

- With double selection, all inputs that help to explain either d_i or y_i (according to lasso) are included in (13).
- If instead we use the double-ML (or partialing out) estimator (7), the two machine-learning procedures do not have to be the same.
- When asymptotic theory provides a good guide, and both $g(\cdot)$ and $m(\cdot)$ are approximately linear, the double-selection and partialing-out (double-ML) estimators should be similar.
- When they are not similar, we should abandon the linearity assumption and lasso. The first-stage regressions should use some more flexible machine-learning procedure.

Almost any machine-learning method can be used for the two first-stage regressions, that is, regressing y_i on $g(\boldsymbol{x}_i)$ and d_i on $m(\boldsymbol{x}_i)$.

- When p is large and sparsity seems plausible, we can use lasso, relaxed lasso, or post-lasso estimates of $g(\cdot)$ and $m(\cdot)$.
- In the treatment case, we should minimize deviance instead of SSR when estimating the propensity score function $m(\cdot)$.
- When p is small, effects are additive, and only a few variables are thought to enter nonlinearly, we can use GAMs.
- Whether or not p is large, we can use random forests or boosting.
- Since d_i is often a binary variable, it seems natural to minimize deviance instead of the SSR when estimating $m(\boldsymbol{x}_i)$.
- Neural networks can also be used, if they seem to be appropriate.
- We can use several different methods and then average their outputs, perhaps giving higher weights to ones that work better.

The Partially Linear IV Regression Model

The **partially linear IV regression model**, or **PLIV model**, can be written as

$$\boldsymbol{y} = \beta \boldsymbol{d} + \boldsymbol{g}(\boldsymbol{X}) + \boldsymbol{u}, \quad \mathrm{E}(\boldsymbol{u} \,|\, \boldsymbol{X}, \boldsymbol{Z}) = \boldsymbol{0} \tag{15}$$

$$\boldsymbol{Z} = \boldsymbol{m}(\boldsymbol{X}) + \boldsymbol{v}, \quad \mathrm{E}(\boldsymbol{v} \,|\, \boldsymbol{X}) = \boldsymbol{0}. \tag{16}$$

Here (15) is the structural equation of interest, and $\boldsymbol{Z}$ is a matrix of additional instruments (or excluded exogenous variables). We have stacked the y_i into $\boldsymbol{y}$, the $\boldsymbol{x}_i$ into $\boldsymbol{X}$, and so on.

The endogenous variable is $\boldsymbol{d}$, and the parameter we care about is β.

In the linear case, we could ignore the way the instruments are generated. Asymptotically, all we care about is that they be correlated with $\boldsymbol{d}$ and uncorrelated with $\boldsymbol{u}$.

Thus, in the linear case, we can simply include $\boldsymbol{X}$ in both the reduced form regression and the structural equation.

In the linear case, there are two regressions:

$$\boldsymbol{d} = \boldsymbol{X}\boldsymbol{\pi}_1 + \boldsymbol{Z}\boldsymbol{\pi}_2 + \boldsymbol{w} \tag{17}$$

$$\boldsymbol{y} = \beta\boldsymbol{d} + \boldsymbol{X}\boldsymbol{\gamma} + \boldsymbol{u}. \tag{18}$$

We first run the reduced-form regression (17) to generate a vector of fitted values $\hat{\boldsymbol{d}}$.

Then we either regress $\boldsymbol{y}$ on $\hat{\boldsymbol{d}}$ and $\boldsymbol{X}$ (yielding 2SLS) or explicitly compute the generalized IV estimator as

$$\hat{\beta}_{\text{IV}} = (\boldsymbol{d}^\top \boldsymbol{P}_{[\text{X Z}]}\boldsymbol{d})^{-1}\boldsymbol{d}^\top \boldsymbol{P}_{[\text{X Z}]}\boldsymbol{y}, \tag{19}$$

where $\boldsymbol{P}_{[X\,Z]}$ is the matrix that yields fitted values.

Of course, the IV estimator in (19) is identical to the 2SLS estimator. But an IV package gives asymptotically valid standard errors, and a second-stage OLS regression does not.

Although $\hat{\beta}_{\text{IV}}$ is consistent, its finite-sample properties can be very poor. The estimate of β can be severely biased, and standard errors can be too small.

DDML may yield much better estimates, especially when the dimensions of $\boldsymbol{X}$ and/or $\boldsymbol{Z}$ are large. That was one of the motivations of the double-selection literature.

In the partially linear case, with $g(\boldsymbol{X})$ and $\boldsymbol{m}(\boldsymbol{X})$ unknown, we need to estimate those functions and worry about bias.

For the PLIV model, we have to estimate at least two nonlinear functions:

$$\boldsymbol{y} = g(\boldsymbol{X}) \tag{20}$$

$$\boldsymbol{Z} = \boldsymbol{m}(\boldsymbol{X}). \tag{21}$$

There will be more than two functions to estimate if there is more than one instrument, because we need to estimate $\boldsymbol{m}_j(\boldsymbol{X})$ for each column of $\boldsymbol{Z}$, indexed by j.

Then we "regress" $y - \hat{g}$ on d using instruments $Z - \hat{m}$ in order to obtain what we really care about, which is the IV estimate of β.

However, the two (or more) sets of fitted values in this "regression" are not the usual ones. They come from **cross-fitting**, discussed below.

This should also be the case for the two sets of fitted values $\hat{g}$ and $\hat{m}$ that we need for the PLR model.

There are two different ways to run the final "regression."

The "partialling out" method involves an IV regression of $y - \hat{g}$ on d using instruments $Z - \hat{m}$. This seems to be the approach recommended by the `DoubleML` package.

The other method involves an IV regression of y on d and $g(X)$. But this means using a partially linear model to estimate the coefficient on d and the function $g(X)$ jointly.

Fully Nonlinear Models

The `DoubleML` package can also handle models in which d enters nonlinearly. Thus, instead of the PLR model (1), we can estimate the **interactive regression model**, or **IRM**,

$$y_i = g(d_i, \boldsymbol{x}_i) + u_i \tag{22}$$
$$d_i = m(\boldsymbol{x}_i) + v_i. \tag{23}$$

Instead of the PLIV model (15) and (16), we can estimate the **interactive instrumental variable model**, or **IIVM**,

$$y_i = g(d_i, \boldsymbol{x}_i) + u_i \tag{24}$$
$$z_i = m(\boldsymbol{x}_i) + v_i. \tag{25}$$

The documentation suggests that the IIVM is only available for d_i binary and a single binary instrument.

Neyman Orthogonal Scores

Chernozhukov et al. (2018) devotes 11 pages to discussing **Neyman orthogonal scores**! Section 4.3 of the book has a shorter treatment.

This is not an elementary concept, and the treatment in the paper is very abstract.

We are trying to estimate a low-dimensional parameter vector $\boldsymbol{\theta}$ based on the moment conditions

$$\mathrm{E}\big(\boldsymbol{\psi}(w; \boldsymbol{\theta}_0, \boldsymbol{\eta}_0)\big) = \mathbf{0}, \tag{26}$$

where w is a random data vector and $\boldsymbol{\eta}_0$ is the true value of a high-dimensional vector of nuisance parameters.

The dimension of $\boldsymbol{\psi}(\cdot)$ is the same as the dimension of $\boldsymbol{\theta}_0$.

Here $\boldsymbol{\theta}_0$ is the true value of $\boldsymbol{\theta}$, and $\boldsymbol{\eta}_0$ is the true value of $\boldsymbol{\eta}$. Equation (26) would not hold at any other values, at least not if all parameters are identified.

The idea of the (generalized) method of moments is to replace the expectation in (26) by its sample analog.

If we knew $\boldsymbol{\eta}_0$, we could simply solve the equations

$$\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\psi}(\boldsymbol{w}_i;\boldsymbol{\theta},\boldsymbol{\eta}_0) = \mathbf{0}. \tag{27}$$

When these are linear, we can obtain an analytic expression for $\hat{\boldsymbol{\theta}}$.

When they are nonlinear, we have to solve them somehow, which might well be equivalent to minimizing a certain objective function.

In either case, we can probably obtain theoretical results for the asymptotic distribution of $n^{1/2}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}_0)$.

These will depend on the matrix of derivatives of $\boldsymbol{\psi}$ and on the randomness in the $\boldsymbol{w}_i$.

But all of this assumes that η_0 is known. Because η is high-dimensional, we do not want to estimate θ and η jointly.

Instead, we would like to estimate η by some ML method and then estimate θ conditional on it.

Suppose we are maximizing the loglikelihood function

$$\ell(W;\theta,\eta) = \sum_{i=1}^{n} \ell(w_i;\theta,\eta). \tag{28}$$

The score function for θ is

$$\phi(W;\theta,\eta) = \sum_{i=1}^{n} \frac{\partial \ell(w_i;\theta,\eta)}{\partial \theta}. \tag{29}$$

Except in rare special cases, this score function is not orthogonal to the score function for η.

Thus, if try to estimate θ conditional on the ML estimator $\hat{\eta}$, the estimate will be biased whenever $\hat{\eta}$ is biased.

The solution is to replace $\boldsymbol{\phi}(\boldsymbol{W};\boldsymbol{\theta},\boldsymbol{\eta})$ by the **Neyman-orthogonal** score function

$$\boldsymbol{\psi}(\boldsymbol{W};\boldsymbol{\theta},\boldsymbol{\eta}) = \boldsymbol{\phi}(\boldsymbol{W};\boldsymbol{\theta},\boldsymbol{\eta}) - \boldsymbol{\mu}\sum_{i=1}^{n}\frac{\partial\ell(\boldsymbol{w}_i;\boldsymbol{\theta},\boldsymbol{\eta})}{\partial\boldsymbol{\eta}}, \tag{30}$$

where $\boldsymbol{\mu}$ is an **orthogonalization matrix** which depends on the second derivatives of $\ell(\boldsymbol{W};\boldsymbol{\theta},\boldsymbol{\eta})$ with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$.

This is all pretty abstract. Let us go back to the partially linear regression model, but now we make it fully linear:

$$\begin{aligned}\boldsymbol{y} &= \theta\boldsymbol{d} + \boldsymbol{X}\boldsymbol{\eta} + \boldsymbol{u}\\ \boldsymbol{d} &= \boldsymbol{X}\boldsymbol{\mu} + \boldsymbol{v}.\end{aligned} \tag{31}$$

In this case, the Neyman-orthogonal scores are

$$\boldsymbol{\psi}(\boldsymbol{W};\theta,\boldsymbol{\eta}) = (\boldsymbol{y} - \theta\boldsymbol{d} - \boldsymbol{X}\boldsymbol{\eta})^{\top}(\boldsymbol{d} - \boldsymbol{X}\boldsymbol{\mu}), \tag{32}$$

where $\boldsymbol{\mu} = \mathrm{E}(\boldsymbol{X}^{\top}\boldsymbol{X})^{-1}\mathrm{E}(\boldsymbol{X}^{\top}\boldsymbol{d})$.

If we replace $\boldsymbol{\mu}$ in (32) by the above expression for $\boldsymbol{\mu}$ and then drop the expectation operators, we obtain

$$\begin{aligned}\boldsymbol{\psi}(\boldsymbol{W};\theta,\boldsymbol{\eta}) &= (\boldsymbol{y}-\theta\boldsymbol{d}-\boldsymbol{X}\boldsymbol{\eta})^\top(\boldsymbol{d}-\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{d}) \\ &= (\boldsymbol{y}-\theta\boldsymbol{d}-\boldsymbol{X}\boldsymbol{\eta})^\top(\boldsymbol{d}-\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{d}) \\ &= (\boldsymbol{y}-\theta\boldsymbol{d}-\boldsymbol{X}\boldsymbol{\eta})^\top\boldsymbol{M}_{\boldsymbol{X}}\boldsymbol{d} \\ &= -\theta\boldsymbol{d}^\top\boldsymbol{M}_{\boldsymbol{X}}\boldsymbol{d}+\boldsymbol{y}^\top\boldsymbol{M}_{\boldsymbol{X}}\boldsymbol{d}.\end{aligned} \tag{33}$$

Finding $\hat{\theta}$ means equating this expression to zero. The result is

$$\hat{\theta} = (\boldsymbol{d}^\top\boldsymbol{M}_{\boldsymbol{X}}\boldsymbol{d})^{-1}\boldsymbol{d}^\top\boldsymbol{M}_{\boldsymbol{X}}\boldsymbol{y}. \tag{34}$$

This is a special case of the DML, or partialing-out, estimator (7).

Here we just use OLS for partialing out, which would be OK if the number of regressors were small, and everything were linear.

Intuitively, the reason for using Neyman orthogonal scores is to make the estimate of the parameter we care about, which is θ here, locally invariant to the value of the nuisance parameter η.

If we just regress $\boldsymbol{y}$ on $\boldsymbol{d}$, there are no nuisance parameters to worry about, but we get inconsistent estimates unless treatment was assigned at random.

If both the treatment regression and the causal regression are linear, and there are only a few regressors, we can obtain consistent estimates by including $\boldsymbol{X}$ in the regression.

But this does not work if p is not small relative to n.

Using lasso, or any other machine-learning method, to estimate $g(\boldsymbol{x}_i)$ along with β leads to biased estimates of $g(\boldsymbol{x}_i)$, and this bias is transmitted to $\hat{\beta}$,

The various "double" methods are then designed to make the bias go away rapidly as n increases by ensuring that the scores are approximtely Neyman-orthogonal.

Cross-Fitting

A key part of DDML and the `DoubleML` package is the use of **cross-fitting**, which involves sample splitting. This works somewhat like cross-validation, but its purpose is different.

For simplicity, suppose we divide the sample into 2 equal-sized folds.

- First, estimate the model using the data for fold 1, with the data for fold 2 used to obtain the fitted values.
- Next, estimate the model using the data for fold 2, with the data for fold 1 used to obtain the fitted values.
- Finally, put the two sets of estimates together. There is more than one way to do this.

It is often recommended to use 4 or 5 folds instead of 2. In general, let K denote the number of folds.

When there are K folds, we use $K-1$ of them for estimation and the omitted one for $1/K^{\text{th}}$ of the fitted values. Repeat K times.

If tuning parameters need to be estimated, this may be done using cross-validation. The `DoubleML` package provides two different ways to perform cross-validation.

- The default (cheaper) procedure is to perform cross-validation using the entire sample (that is, the entire training sample).
- Alternatively, cross-validation can be done using the data for the estimation folds.
- This involves further dividing each set of estimation fold(s) so as to use cross-validation for each of them. Each set of estimates uses a different value of the tuning parameter.

In the simplest approach (DML1), each fold yields an estimator $\check{\beta}_k$, which is based on the other $K-1$ folds.

Then the $\check{\beta}_k$ are averaged across the K folds to obtain $\tilde{\beta}$.

In a second variant (DML2), an optimization problem is solved to obtain $\tilde{\beta}$ rather than simply averaging the $\check{\beta}_k$. In this case, the individual $\check{\beta}_k$ need not be calculated.

For the double-ML estimator (7) of the PLR model, which depends on two sets of ML residuals, DML2 just means replacing $\hat{v}$ and $\hat{u}$ by their cross-fit equivalents, say $\check{v}$ and $\check{u}$.

The elements of $\check{v}$ and $\check{u}$ that correspond to fold 1 are based on estimates from folds 2 to K, the ones that correspond to fold 2 are based on estimates from fold 1 and folds 3 to K, and so on.

The number of folds for the `DoubleML` routines is given by the parameter `n_folds`. The default value is 5.

Suppose there are 5 folds. Then DML2 is based on

$$\check{v} = \begin{bmatrix} \check{v}_1 \\ \check{v}_2 \\ \vdots \\ \check{v}_5 \end{bmatrix} \quad \text{and} \quad \check{u} = \begin{bmatrix} \check{u}_1 \\ \check{u}_2 \\ \vdots \\ \check{u}_5 \end{bmatrix}. \tag{35}$$

We simply form the vectors $\check{u}$ and $\check{v}$ and calculate $\tilde{\beta} = (\check{v}^\top \check{v})^{-1} \check{v}^\top \check{u}$.

In this case, DML2 is even easier than DML1, and it is what Stata does in its various xpo procedures.

- $\tilde{\beta}$ it is not the only DML2 estimator based on 5 folds.
- There is an extremely large number of possible estimators, one for each random split of the sample into 5 folds.

The papers recommend repeating the DDML procedure S (or `n_rep`) times. Each repetition involves randomly partitioning the sample into K folds, then using the DDML procedure to obtain $\tilde{\beta}$ for that split.

Let $\tilde{\beta}_s$ denote the DML1 or DML2 estimate for split number s.

When $S = 1$, we just report $\tilde{\beta}_1$. When $S > 1$, there are many things we could report.

Let $\tilde{\beta}_{\text{med}}$ denote the median of the $\tilde{\beta}_s$. For calculating the median, it is convenient for S to be a number like 99, 199, and so on. If $S = 199$, then the median is number 100 in the sorted list.

When $S > 1$, the papers recommend reporting $\tilde{\beta}_{\text{med}}$ and an associated measure of variance, which is the median of

$$\hat{\sigma}_s^2 + (\tilde{\beta}_s - \tilde{\beta}_{\text{med}})^2, \quad s = 1, \ldots, S. \tag{36}$$

Here $\hat{\sigma}_s^2$ is the usual estimator of the variance of the $\tilde{\beta}_s$.

The `DoubleML` package can be used without cross-fitting, simply dividing the sample in half and using half for the ML estimation and the other half to estimate the parameter(s).

Although this should have little bias from overfitting, it is not recommended!

Cross-fitting, preferably with $K = 4$ or $K = 5$, increases efficiency.

Repeating this a number of times further increases efficiency.

The increased efficiency in going from sample splitting without cross-fitting to 2-fold cross-fitting is illustrated in Figure 20.3.

Figure 20.3 — Efficiency gain from 2-fold cross-fitting

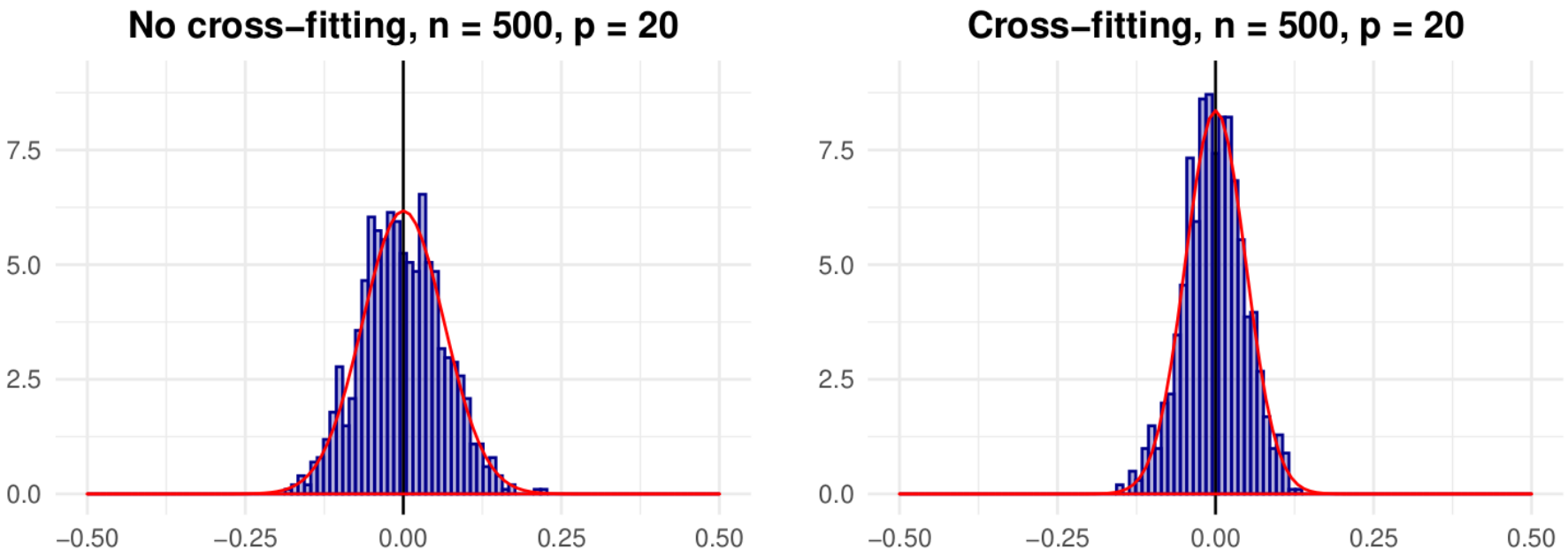


Figure 6: Illustration of efficiency gains due to the use of cross-fitting.

Left panel: Histogram of the centered dml estimator without cross-fitting, $\tilde{\theta}_0^{nocf} - \theta_0$. $\hat{\theta}_0^{nocf}$ is the double machine learning estimator obtained from a sample split into two folds. One fold is used for estimation of the nuisance parameters and the second fold is used for evaluation of the score function and estimation. The empirical distribution can be well-approximated by a normal distribution as indicated by the red curve. **Right panel:** Histogram of the centered dml estimator with cross-fitting, $\tilde{\theta}_0 - \theta_0$. The estimator is obtained from a split into two folds and application of Algorithm 2 (DML2). In both cases, the estimators are based on estimation of g_0 and m_0 with random forests and an orthogonal score function provided in Equation (17). Moreover, exactly the same data sets and exactly the same partitions are used for sample splitting. The empirical distribution of the estimator that is based on cross-fitting exhibits a more pronounced concentration around zero, which reflects the smaller standard errors.

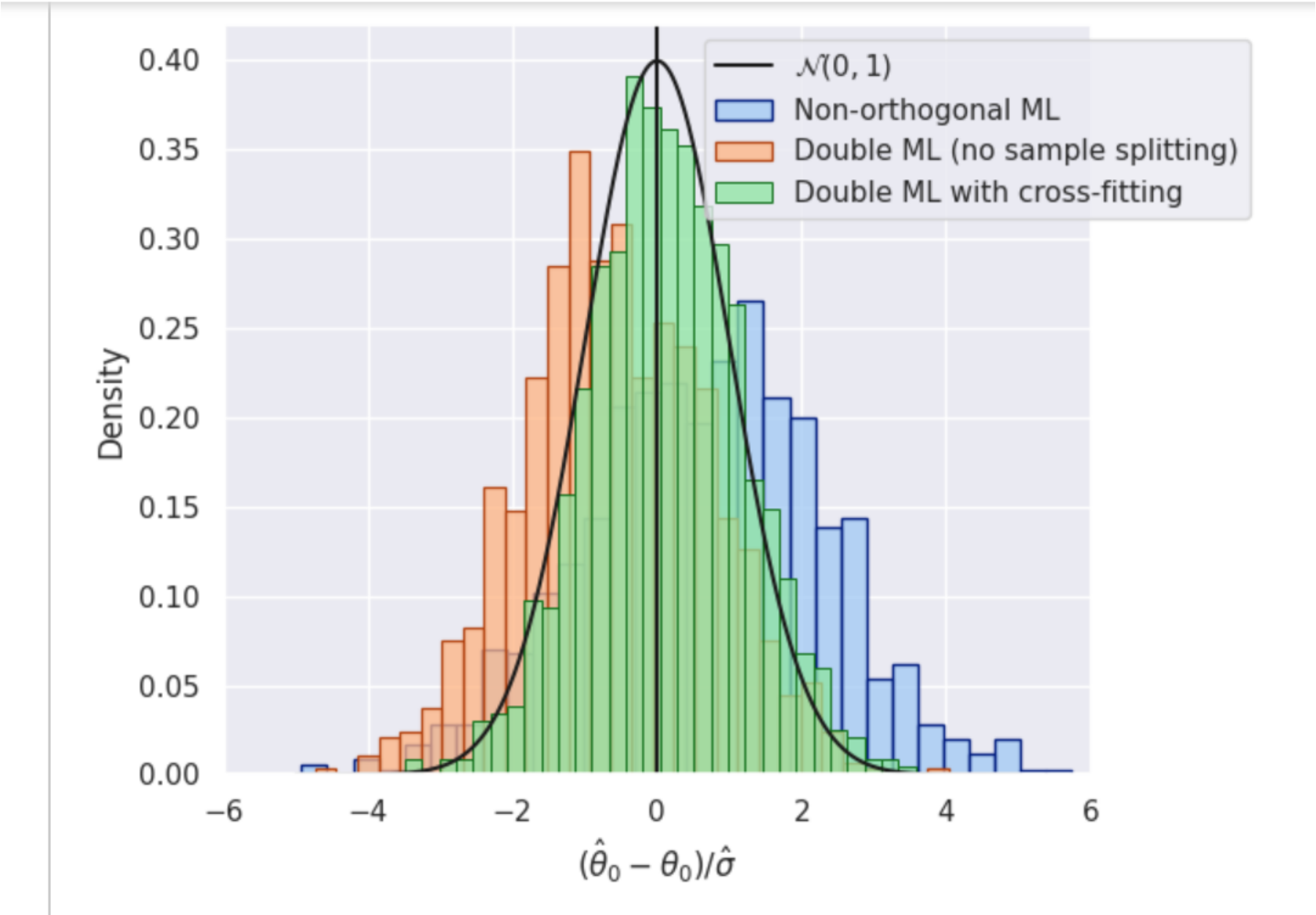
There is a nice figure (Figure 20.4) on the `DoubleML` website which puts together the effects of partialing out and cross-fitting.

- The blue histogram does everything wrong. β and $g(x)$ in (1) are learned together, and the slow convergence of the ML procedure leads to (in this case) upward bias.
- The orange histogram uses double machine learning (partialing out), but no sample splitting. Now there is downward bias.
- The green histogram uses both DML and cross-fitting with just two folds. It seems to be almost unbiased and a bit less spread out than the orange one.

The `DoubleML` website actually provides the code to generate these three histograms. The ML procedure they use is a random forest.

There are 500 observations, 20 inputs (regressors), and 1000 replications.

Figure 20.4 — Benefits of partialing out and cross-fitting



Inference about Causal Parameters

It is asymptotically valid to base inferences about β (or the parameters of interest for other models) on conventional standard errors.

The DML2 estimate $\tilde{\beta}$ for the PLR model is $(\check{\boldsymbol{v}}^\top \check{\boldsymbol{v}})^{-1} \check{\boldsymbol{v}}^\top \check{\boldsymbol{u}}$. This is just the OLS estimate obtained by regressing $\check{\boldsymbol{u}}$ on $\check{\boldsymbol{v}}$.

- In many cases, we would have to worry about the fact that $\check{\boldsymbol{u}}$ and $\check{\boldsymbol{v}}$ are generated regressors. But, for DDML procedures, we can just pretend that they are observed variables.
- Thus the usual hetero-robust (or perhaps cluster-robust) standard errors for $\tilde{\beta}$ are asymptotically valid.
- Note that we are only making inferences about β, or whatever causal parameters we care about.
- We are not attempting to make inferences about $g(\boldsymbol{x}_i)$ or $m(\boldsymbol{x}_i)$ in either the PLR or PLIV models.

Instead of just using a standard error and the standard normal distribution, we can use the **multiplier bootstrap**.

The multiplier bootstrap is similar to the wild bootstrap, but it involves multiplying scores instead of residuals by random variables. It is also called the **score bootstrap**.

Because there is no nonlinear estimation for the bootstrap samples, it is very cheap, but it may not be very accurate.

In the case of $\tilde{\beta}$ for the PLR model, recall that

$$\begin{aligned} \tilde{\beta} &= (\check{\boldsymbol{v}}^\top \check{\boldsymbol{v}})^{-1} \check{\boldsymbol{v}}^\top \check{\boldsymbol{u}} \\ &= (\check{\boldsymbol{v}}^\top \check{\boldsymbol{v}})^{-1} \sum_{i=1}^{n} \check{v}_i \check{u}_i. \end{aligned} \tag{37}$$

As we have seen, the product $\check{\boldsymbol{v}}^\top \check{\boldsymbol{u}}$ is a particular case of a Neyman orthogonal score.

Asymptotically,

$$n^{1/2}(\tilde{\beta} - \beta_0) \stackrel{a}{=} (n^{-1}\check{\boldsymbol{v}}^\top\check{\boldsymbol{v}})^{-1}n^{-1/2}\check{\boldsymbol{v}}^\top\check{\boldsymbol{u}}. \tag{38}$$

The first factor on the right-hand side of (38) is asymptotically nonstochastic.

It has a positive mean, and the division by n asymptotically eliminates all the random variation around that mean.

Thus, asymptotically, all of the randomness in $\tilde{\beta}$ comes from the random variation in the $\check{v}_i\check{u}_i$, that is, the scores.

For the score bootstrap, we simply multiply each of the scores by a random variable ζ_i^b with mean 0 and variance 1.

For the b^{th} bootstrap sample (out of B samples), we obtain

$$\tilde{\beta}^{*b} = (\check{\boldsymbol{v}}^\top\check{\boldsymbol{v}})^{-1}\sum_{i=1}^{n}\zeta_i^b\check{v}_i\check{u}_i. \tag{39}$$

We can then use the distribution of the $\tilde{\beta}^{*b}$ in various ways.

- Find the $1-\alpha$ quantile of the $|\tilde{\beta}^{*b}|$ and use it instead of the usual critical value from the normal distribution to obtain a symmetric confidence interval.
- Find the $\alpha/2$ and $1-\alpha/2$ quantiles of the $\tilde{\beta}^{*b}$ and use them to obtain an asymmetric confidence interval.
- Calculate the standard deviation of the $\tilde{\beta}^{*b}$ and use it together with an $\mathrm{N}(0,1)$ critical value to obtain a symmetric interval.
- Calculate bootstrap t-statistics instead of bootstrap coefficients, and use them to construct some sort of studentized bootstrap interval. These would use the asymptotic standard error.

The `DoubleML` package uses the last of these, apparently imposing symmetry. If we are worried about bias, this seems like a bad idea.

There are three choices for the distribution of the ζ_i^b, all of them dubious. These include standard normal and Mammen (misleadingly called wild). Rademacher is not an option.

None of the proposed methods imposes a null hypothesis, which often makes bootstrap methods work better in finite samples.

They are all inexpensive because the scores and other needed quantities are only computed once.

This could also be true for bootstrap tests if a null hypothesis were imposed, but not for bootstrap confidence intervals.

- For the PLR model, the $\check{u}_i$ and $\check{v}_i$ are only computed once, and they are always based on unrestricted estimates.
- This means that they ignore the effects of nonlinearity in the ML procedures on the distribution of $\tilde{\beta}$.
- If we actually re-estimated the model for each bootstrap sample, we might obtain quite different bootstrap distributions, but at much higher cost.

Even without doing that, I suspect that we could do better than the procedures currently implemented in DoubleML.