I Z A Institute
of Labor Economics
Initiated by Deutsche Post Foundation

DISCUSSION PAPER SERIES

IZA DP No. 15963

ddml: Double/Debiased Machine Learning in Stata

Achim Ahrens
Christian B. Hansen
Mark E. Schaffer
Thomas Wiemann

FEBRUARY 2023

I Z A Institute
of Labor Economics

Initiated by Deutsche Post Foundation

DISCUSSION PAPER SERIES

IZA DP No. 15963

ddml: Double/Debiased Machine Learning in Stata

Achim Ahrens
ETH Zürich

Christian B. Hansen
University of Chicago

Mark E. Schaffer
Heriot-Watt University Edinburgh and IZA

Thomas Wiemann
University of Chicago

FEBRUARY 2023

ISSN: 2365-9793

IZA – Institute of Labor Economics

Schaumburg-Lippe-Straße 5–9
53113 Bonn, Germany

Phone: +49-228-3894-0
Email: publications@iza.org

www.iza.org

ABSTRACT

ddml: Double/Debiased Machine Learning in Stata

We introduce the package ddml for Double/Debiased Machine Learning (DDML) in Stata. Estimators of causal parameters for five different econometric models are supported, allowing for flexible estimation of causal effects of endogenous variables in settings with unknown functional forms and/or many exogenous variables. ddml is compatible with many existing supervised machine learning programs in Stata. We recommend using DDML in combination with stacking estimation which combines multiple machine learners into a final predictor. We provide Monte Carlo evidence to support our recommendation.

JEL Classification: C14, C21, C87

Keywords: st0001, causal inference, machine learning, doubly-robust estimation

Corresponding author:
Mark E. Schaffer
School of Social Sciences
Heriot-Watt University
Edinburgh EH14 4AS
Scotland

E-mail: m.e.schaffer@hw.ac.uk

1 Introduction

Identification of causal effects frequently relies on an unconfoundedness assumption, requiring that treatment or instrument assignment is sufficiently random given observed control covariates. Estimation of causal effects in these settings then involves conditioning on the controls. Unfortunately, estimators of causal effects that are insufficiently flexible to capture the effect of confounds generally do not produce consistent estimates of causal effects even when unconfoundedness holds. For example, Blandhol et al. (2022) highlight that TSLS estimands obtained after controlling linearly for confounds do not generally correspond to weakly causal effects even when instruments are valid conditional on controls. Even in the ideal scenario where theory provides a small number of relevant controls, theory rarely specifies the exact nature of confounding. Thus, applied empirical researchers wishing to exploit unconfoundedness assumptions to learn causal effects face a nonparametric estimation problem.

Traditional nonparametric estimators suffer greatly under the curse of dimensionality and are quickly impractical in the frequently encountered setting with multiple observed covariates.[1] These difficulties leave traditional nonparametric estimators essentially inapplicable in the presence of increasingly large and complex data sets, e.g. textual confounders as in Roberts et al. (2020) or digital trace data (Hangartner et al. 2021). Tools from supervised machine learning have been put forward as alternative estimators. These approaches are often more robust to the curse of dimensionality via the exploitation of regularization assumptions. A prominent example of a machine learning-based causal effects estimator is Post-Double Selection Lasso (PDS-Lasso) of Belloni et al. (2014), which fits auxiliary lasso regressions of the outcome and treatment(s), respectively, against a menu of transformed controls. Under an approximate sparsity assumption, which posits that the DGP can be approximated well by a relatively small number of terms included in the menu, this approach allows for precise treatment effect estimation. The lasso can also be used for approximating optimal instruments (Belloni et al. 2012). Lasso-based approaches for estimation of causal effects have become a popular strategy in applied econometrics (e.g. Gilchrist and Sands 2016; Dhar et al. 2022), partially facilitated by the availability of software programs in Stata (`pdslasso`, Ahrens et al. 2018; StataCorp 2019) and R (`hdm`, Chernozhukov et al. 2016).

Although approximate sparsity is a weaker regularization assumption than assuming a linear functional form that depends on a known low-dimensional set of variables, it may not be suitable in a wide range of applications. For example, Giannone et al. (2021) argue that approximate sparsity may provide a poor description in several economic examples. There is thus a potential benefit to expanding the set of regularization assumptions and correspondingly considering a larger set of machine learners including, for example, random forests, gradient boosting, and neural networks. While the theoretical properties of these estimators are an active research topic (see, e.g., Athey et al. 2019; Farrell et al. 2021), machine learning methods are widely adopted in industry and practice for their empirical performance. To facilitate their application for

1. For example, the number of coefficients in polynomial series regression with interaction terms increases exponentially in the number of covariates.

causal inference in common econometric models, Chernozhukov et al. (2018) propose Double/Debiased Machine Learning (DDML), which exploits Neyman orthogonality of estimating equations and cross-fitting to formally establish asymptotic normality of estimators of causal parameters under relatively mild convergence rate conditions on nonparametric estimators.

DDML increases the set of machine learners that researchers can leverage for estimation of causal effects. Deciding which learner is most suitable for a particular application is difficult, however, since researchers are rarely certain about the structure of the underlying data generating process. A practical solution is to construct combinations of a diverse set of machine learners using stacking (Wolpert 1992; Breiman 1996). Stacking is a meta learner given by a weighted sum of individual machine learners (the "base learners"). When the weights corresponding to the base learners are chosen to maximize out-of-sample predictive accuracy, this approach hedges against the risk of relying on any particular poorly suited or ill-tuned machine learner.

In this article, we introduce the Stata package `ddml`, which implements DDML for Stata.[2] `ddml` adds to a small number of programs for causal machine learning in Stata (Ahrens et al. 2018, StataCorp 2019, StataCorp 2021). We briefly summarize the four main features of the program:

1. `ddml` supports flexible estimators of causal parameters in five econometric models: (1) the Partially Linear Model, (2) the Interactive Model (for binary treatment), (3) the Partially Linear IV Model, (4) the Flexible Partially Linear IV Model, and (5) the Interactive IV Model (for binary treatment and instrument).

2. `ddml` supports data-driven combinations of multiple machine learners via stacking by leveraging `pystacked` (Ahrens et al. 2022), our complementary Stata frontend relying on the Python library *scikit-learn* (Pedregosa et al. 2011; Buitinck et al. 2013).

3. Aside from `pystacked`, `ddml` can be used in combination with many other existing supervised machine learning programs available in or via Stata. `ddml` has been tested with `lassopack` (Ahrens et al. 2020), `rforest` (Schonlau and Zou 2020), `svmachines` (Guenther and Schonlau 2018), and `parsnip` (Huntington-Klein 2021). Indeed, the requirements for compatibility with `ddml` are minimal: Any `eclass` program with the Stata-typical "`reg y x`" syntax, support for `if` conditions and post-estimation `predict` is compatible with `ddml`.

4. `ddml` provides flexible multi-line syntax and short one-line syntax. The multi-line syntax offers a wide range of options, guides the user through the DDML algorithm step-by-step, and includes auxiliary programs for storing, loading and displaying additional information. We also provide a complementary one-line version called `qddml` ('quick' `ddml`), which uses a similar syntax as `pdslasso` and `ivlasso` (Ahrens et al. 2018).

2. This article refers to version 1.2 of `ddml`.

The article proceeds as follows. Section 2 outlines DDML for the Partially Linear and Interactive Models under conditional unconfoundedness assumptions. Section 3 outlines DDML for Instrumental Variables (IV) models. Section 4 discusses how stacking can be combined with DDML and provides evidence from Monte Carlo simulations illustrating the advantages of DDML with stacking. Section 5 explains the features, syntax and options of the program. Section 6 demonstrates the program's usage with two applications.

2 DDML with Conditional Unconfoundedness

This section discusses DDML for the Partially Linear Model and the Interactive Model in turn. Both models are special cases of the general causal model

$$Y = f_0(D, \boldsymbol{X}, U), \tag{1}$$

where f_0 is a structural function, Y is the outcome, D is the variable of interest, $\boldsymbol{X}$ are observed covariates, and U are all unobserved determinants of Y (i.e., other than D and $\boldsymbol{X}$).[3] The key difference between the Partially Linear Model and the Interactive Model is their position in the trade-off between functional form restrictions on f_0 and restrictions on the joint distribution of observables $(D, \boldsymbol{X})$ and unobservables U. For both models, we highlight key parameters of interest, state sufficient identifying assumptions, and outline the corresponding DDML estimator. A random sample $\{(Y_i, D_i, \boldsymbol{X}_i)\}_{i=1}^n$ from $(Y, D, \boldsymbol{X})$ is considered throughout.

2.1 The Partially Linear Model (`partial`)

The Partially Linear Model imposes the estimation model

$$Y = \theta_0 D + g_0(\boldsymbol{X}) + U \tag{2}$$

where θ_0 is a fixed unknown parameter. The key feature of the model is that the controls $\boldsymbol{X}$ enter through the unknown and potentially nonlinear function g_0. Note that D is not restricted to be binary and may be discrete, continuous or mixed. For simplicity, we assume that D is a scalar, although `ddml` allows for multiple treatment variables in the Partially Linear Model.

The parameter of interest is θ_0, the causal effect of D on Y.[4] The key identifying assumption is given in Assumption 1.[5]

3. Since in (1), $(D, \boldsymbol{X}, U)$ jointly determine Y, the model is also dubbed the "all causes model" (see, e.g., Heckman and Vytlacil 2007). Note that the model can equivalently be put into potential outcome notation with potential outcomes defined as $Y(d) \equiv f_0(d, \boldsymbol{X}, U)$.
4. The interpretation of θ_0 can be generalized. For example, the results of Angrist and Krueger (1999) imply that in the general causal model (1), θ_0 is a positively weighted average of causal effects (e.g., conditional average treatment effects) under stronger identifying assumptions. The basic structure can also be used to obtain valid inference on objects of interest, such as projection coefficients, in the presence of high-dimensional data or nonparametric estimation without requiring a causal interpretation.
5. Discussions of Partially Linear Model typically show identification under the stronger assumption

Assumption 1 (Conditional Orthogonality) $E[Cov(U, D|\boldsymbol{X})] = 0$.

To show identification of θ_0, consider the score

$$\psi(\boldsymbol{W}; \theta, m, \ell) = \Big(Y - \ell(\boldsymbol{X}) - \theta\big(D - m(\boldsymbol{X})\big)\Big)\big(D - m(\boldsymbol{X})\big), \tag{3}$$

where $\boldsymbol{W} \equiv (Y, D, \boldsymbol{X})$, and ℓ and m are nuisance functions. Letting $m_0(\boldsymbol{X}) \equiv E[D|\boldsymbol{X}]$ and $\ell_0(\boldsymbol{X}) \equiv E[Y|\boldsymbol{X}]$, note that

$$E\left[\psi(\boldsymbol{W}; \theta_0, m_0, \ell_0)\right] = 0$$

by Assumption 1. When in addition $E[Var(D|\boldsymbol{X})] \neq 0$, we get

$$\theta_0 = \frac{E\left[\big(Y - \ell_0(\boldsymbol{X})\big)\big(D - m_0(\boldsymbol{X})\big)\right]}{E\left[(D - m_0(\boldsymbol{X}))^2\right]}. \tag{4}$$

Equation (4) is constructive in that it motivates estimation of θ_0 via a simple two-step procedure: First, estimate the conditional expectation of Y given $\boldsymbol{X}$ (i.e., ℓ_0) and of D given $\boldsymbol{X}$ (i.e., m_0) using appropriate nonparametric estimators (e.g., machine learners). Second, residualize Y and D by subtracting their respective conditional expectation function (CEF) estimates, and regress the resulting CEF residuals of Y on the CEF residuals of D. This approach is fruitful when the estimation error of the first step does not propagate excessively to the second step. DDML leverages two key ingredients to control the impact of the first step estimation error on the second step estimate: 1) second step estimation based on Neyman orthogonal scores and 2) cross-fitting. As shown in Chernozhukov et al. (2018), this combination facilitates the use of any nonparametric estimator that converges sufficiently quickly in the first and potentially opens the door for the use of many machine learners.

Neyman orthogonality refers to a property of score functions ψ that ensures local robustness to estimation errors in the first step. Formally, it requires that the Gateaux derivative with respect to the nuisance functions evaluated at the true values is mean-zero. In the context of the partially linear model, this condition is satisfied for the moment condition (3):

$$\partial_r \left\{E\left[\psi(W; \theta_0, m_0 + r(m - m_0), \ell_0 + r(\ell - \ell_0))\right]\right\}|_{r=0} = 0,$$

where the derivative is with respect to the scalar r and evaluated at $r = 0$. Heuristically, we can see that this condition alleviates the impact of noisy estimation of nuisance functions as local deviations of the nuisance functions away from their true values leave the moment condition unchanged. We refer to Chernozhukov et al. (2018) for a detailed discussion but highlight that all score functions discussed in this article are Neyman orthogonal.

---

that $E[U|D, \boldsymbol{X}] = 0$. We differentiate here to highlight differences between the Partially Linear Model and Interactive Model.

Cross-fitting ensures independence between the estimation error from the first step and the regression residual in the second step. To implement cross-fitting, we randomly split the sample into K evenly-sized folds, denoted as $I_1, \ldots, I_K$. For each fold k, the conditional expectations ℓ_0 and m_0 are estimated using only observations *not* in the kth fold – i.e., in $I_k^c \equiv I \setminus I_k$ – resulting in $\hat{\ell}_{I_k^c}$ and $\hat{m}_{I_k^c}$, respectively, where the subscript I_k^c indicates the subsample used for estimation. The out-of-sample predictions for an observation i in the kth fold are then computed via $\hat{\ell}_{I_k^c}(\boldsymbol{X}_i)$ and $\hat{m}_{I_k^c}(\boldsymbol{X}_i)$. Repeating this procedure for all K folds then allows for computation of the DDML estimator for θ_0:

$$\hat{\theta}_n = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{\ell}_{I_{k_i}^c}(\boldsymbol{X}_i)\right)\left(D_i - \hat{m}_{I_{k_i}^c}(\boldsymbol{X}_i)\right)}{\frac{1}{n}\sum_{i=i}^{n}\left(D_i - \hat{m}_{I_{k_i}^c}(\boldsymbol{X}_i)\right)^2}, \tag{5}$$

where k_i denotes the fold of the ith observation.[6]

We summarize the DDML algorithm for the Partially Linear Model in Algorithm 1:[7]

❑ **Algorithm 1. DDML for the Partially Linear Model.**

Split the sample $\{(Y_i, D_i, \boldsymbol{X}_i)\}_{i=1}^n$ randomly in K folds of approximately equal size. Denote I_k the set of observations included in fold k and I_k^c its complement.

1. For each $k \in \{1, \ldots, K\}$:
    a. Fit a CEF estimator to the sub-sample I_k^c using Y_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\hat{\ell}_{I_k^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
    b. Fit a CEF estimator to the sub-sample I_k^c using D_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\hat{m}_{I_k^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
2. Compute (5).

❑

Chernozhukov et al. (2018) give conditions on the joint distribution of the data, in particular on g_0 and m_0, and properties of the nonparametric estimators used for CEF estimation, such that $\hat{\theta}_n$ is consistent and asymptotically normal. Standard errors are equivalent to the conventional linear regression standard errors of $Y_i - \hat{\ell}_{I_{k_i}^c}(\boldsymbol{X}_i)$ on $D_i - \hat{m}_{I_{k_i}^c}(\boldsymbol{X}_i)$. `ddml` computes the DDML estimator for the Partially Linear Model using Stata's `regress`. All standard errors available for linear regression in Stata are also available in `ddml`, including different heteroskedasticity and cluster-robust standard errors.[8]

6. We here omit the constant from the estimation stage. Since the residualized outcome and treatment may not be exactly mean-zero in finite samples, `ddml` includes the constant by default in the estimation stage of partially linear models.
7. Algorithm 1 corresponds to the 'DML2' algorithm in Chernozhukov et al. (2018). Chernozhukov et al. (2018, Remark 3.1) recommend 'DML2' over the alternative 'DML1' algorithm, which fits the final estimator by fold.
8. See `help regress##vcetype` for available options.

Remark 1: Number of folds. The number of cross-fitting folds is a necessary tuning choice. Theoretically, any finite value is admissable. Chernozhukov et al. (2018, Remark 3.1) report that four or five folds perform better than only using $K = 2$. Based on our simulation experience, we find that more folds tends to lead to better performance as more data is used for estimation of conditional expectation functions, especially when the sample size is small. We believe that more work on setting the number of folds would be useful, but believe that setting $K = 5$ provides is likely a good baseline in many settings.

Remark 2: Cross-fitting repetitions. DDML relies on randomly splitting the sample into K folds. We recommend running the cross-fitting procedure more than once using different random folds to assess randomness introduced via the sample splitting. `ddml` facilitates this using the `rep(`*integer*`)` options, which automatically estimates the same model multiple times and combines the resulting estimates to obtain the final estimate. By default, `ddml` reports the median over cross-fitting repetitions. `ddml` also supports the average of estimates. Specifically, let $\hat{\theta}_n^{(r)}$ denote the DDML estimate from the rth cross-fit repetition and $\hat{s}_n^{(r)}$ its associated standard error estimate with $r = 1, \ldots, R$. The aggregate median point estimate and associated standard error are defined as

$$\breve{\hat{\theta}}_n = \text{median}\left(\left(\hat{\theta}_n^{(r)}\right)_{r=1}^{R}\right) \quad \text{and} \quad \breve{\hat{s}}_n = \sqrt{\text{median}\left(\left((\hat{s}_n^{(r)})^2 + (\hat{\theta}_n^{(r)} - \breve{\hat{\theta}}_n)^2\right)_{r=1}^{R}\right)}.$$

The aggregate mean point estimates and associated standard error are calculated as

$$\bar{\hat{\theta}}_n = \frac{1}{R}\sum_{r=1}^{R}\hat{\theta}_n^{(r)} \quad \text{and} \quad \bar{\hat{s}}_n = \sqrt{\text{hmean}\left(\left((\hat{s}_n^{(r)})^2 + (\hat{\theta}_n^{(r)} - \bar{\hat{\theta}}_n)^2\right)_{r=1}^{R}\right)},$$

where hmean() is the harmonic mean.[9]

Remark 3: Cluster-dependence and folds. Under cluster-dependence, we recommend randomly assigning folds by cluster; see `fcluster(`*varname*`)`.

2.2 The Interactive Model (`interactive`)

The Interactive Model is given by

$$Y = g_0(D, \boldsymbol{X}) + U \tag{6}$$

where D takes values in $\{0, 1\}$. The key deviations from the Partially Linear Model are that D must be a scalar binary variable and that D is not required to be additively

9. The harmonic mean of $x_1, \ldots, x_n$ is defined as $\text{hmean}(x_1, \ldots, x_n) = n\left(\sum_{i=1}^{n}\frac{1}{x_i}\right)^{-1}$. We use the harmonic mean as it is less sensitive to outlier values.

separable from the controls $\boldsymbol{X}$. In this setting, the parameters of interest we consider are

$$\theta_0^{\text{ATE}} \equiv E[g_0(1, \boldsymbol{X}) - g_0(0, \boldsymbol{X})]$$
$$\theta_0^{\text{ATET}} \equiv E[g_0(1, \boldsymbol{X}) - g_0(0, \boldsymbol{X})|D = 1],$$

which correspond to the average treatment effect (ATE) and average treatment effect on the treated (ATET), respectively.

Assumptions 2 and 3 below are sufficient for identification of the ATE and ATET. Note the conditional mean independence condition stated here is stronger than the conditional orthogonality assumption sufficient for identification of θ_0 in the Partially Linear Model.

Assumption 2 (Conditional Mean Independence) $E[U|D, \boldsymbol{X}] = 0$.

Assumption 3 (Overlap) $\Pr(D = 1|\boldsymbol{X}) \in (0, 1)$ *with probability 1.*

Under assumptions 2 and 3, we have

$$E[Y|D, \boldsymbol{X}] = E[g_0(D, \boldsymbol{X})|D, \boldsymbol{X}] + E[U|D, \boldsymbol{X}] = g_0(D, \boldsymbol{X}),$$

so that identification of the ATE and ATET immediately follows from their definition.[10]

In contrast to Section 2.1, second-step estimators are not directly based on the moment conditions used for identification. Additional care is needed to ensure local robustness to first-stage estimation errors (i.e., Neyman orthogonality). In particular, the Neyman orthogonal score for the ATE that Chernozhukov et al. (2018) consider is the efficient influence function of Hahn (1998)

$$\psi^{\text{ATE}}(\boldsymbol{W}; \theta, g, m) = \frac{D(Y - g(1, \boldsymbol{X}))}{m(\boldsymbol{X})} - \frac{(1 - D)(Y - g(0, \boldsymbol{X}))}{1 - m(\boldsymbol{X})} + g(1, \boldsymbol{X}) - g(0, \boldsymbol{X}) - \theta,$$

where $\boldsymbol{W} \equiv (Y, D, \boldsymbol{X})$. Similarly for the ATET,

$$\psi^{\text{ATET}}(\boldsymbol{W}; \theta, g, m, p) = \frac{D(Y - g(0, \boldsymbol{X}))}{p} - \frac{m(\boldsymbol{X})(1 - D)(Y - g(0, \boldsymbol{X}))}{p(1 - m(\boldsymbol{X}))} - \theta.$$

Importantly, for $g_0(D, \boldsymbol{X}) \equiv E[Y|D, \boldsymbol{X}]$, $m_0(\boldsymbol{X}) \equiv E[D|\boldsymbol{X}]$, and $p_0 \equiv E[D]$, Assumptions 2 and 3 imply

$$E[\psi^{\text{ATE}}(\boldsymbol{W}; \theta_0^{\text{ATE}}, g_0, m_0)] = 0$$
$$E[\psi^{\text{ATET}}(\boldsymbol{W}; \theta_0^{\text{ATET}}, g_0, m_0, p_0)] = 0;$$

10. In the defined Interactive Model under Assumption 2, the heterogeneity in treatment effects that the ATE and ATET average over is fully observed since U is additively separable. Under stronger identifying assumptions, the DDML ATE and ATET estimators outlined here also apply to the ATE and ATET in the general causal model (1) that average over both observed and unobserved heterogeneity. See, e.g., Belloni et al. (2017).

and we also have that the Gateaux derivative of each condition with respect to the nuisance parameters (g_0, m_0, p_0) is zero.

As before, the DDML estimators for the ATE and ATET leverage cross-fitting. The DDML estimators of the ATE and ATET based on ψ^{ATE} and ψ^{ATET} are

$$\hat{\theta}_n^{\text{ATE}} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{D_i(Y_i - \hat{g}_{I^c_{k_i}}(1, \boldsymbol{X}_i))}{\hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)} - \frac{(1-D_i)(Y_i - \hat{g}_{I^c_{k_i}}(0, \boldsymbol{X}_i))}{1 - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)} + \hat{g}_{I^c_{k_i}}(1, \boldsymbol{X}_i) - \hat{g}_{I^c_{k_i}}(0, \boldsymbol{X}_i)\right), \tag{7}$$

$$\hat{\theta}_n^{\text{ATET}} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{D_i(Y_i - \hat{g}_{I^c_{k_i}}(0, \boldsymbol{X}_i))}{\hat{p}} - \frac{\hat{g}_{I^c_{k_i}}(0, \boldsymbol{X}_i)(1-D_i)(Y_i - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i))}{\hat{p}(1 - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i))}\right), \tag{8}$$

where $\hat{g}_{I^c_k}$ and $\hat{m}_{I^c_k}$ are cross-fitted estimators for g_0 and m_0 as defined in Section 2.1. Since D is binary, the cross-fitted values $\hat{g}_{I^c_k}(1, \boldsymbol{X})$ and $\hat{g}_{I^c_k}(0, \boldsymbol{X})$ are computed by only using treated and untreated observations, respectively. $\hat{p} \equiv \frac{1}{n}\sum_{i=1}^{n} D_i$ is the sample share of treated observations.

`ddml` supports heteroskedasticity and cluster-robust standard errors for $\hat{\theta}_n^{\text{ATE}}$ and $\hat{\theta}_n^{\text{ATET}}$. The algorithm for estimating the ATE and ATET are conceptually similar to Algorithm 1. We delegate the detailed outline to Appendix A. Mean and median aggregation over cross-fitting repetitions are implemented as outlined in Remark 2.

3 DDML with Instrumental Variables

This section outlines the Partially Linear IV Model, the Flexible Partially Linear IV Model, and the Interactive IV Model. As in the previous section, each model is a special case of the general causal model (1). The discussion in this section differs from the preceding section in that identifying assumptions leverage instrumental variables $\boldsymbol{Z}$. The two partially linear IV models assume strong additive separability as in (2), while the Interactive IV Model allows for arbitrary interactions between the treatment D and the controls $\boldsymbol{X}$ as in (6). The Flexible Partially Linear IV Model allows for approximation of optimal instruments[11] as in Belloni et al. (2012) and Chernozhukov et al. (2015a), but relies on a stronger independence assumption than the Partially Linear IV Model. Throughout this discussion, we consider a random sample $\{(Y_i, D_i, \boldsymbol{X}_i, \boldsymbol{Z}_i)\}_{i=1}^n$ from $(Y, D, \boldsymbol{X}, \boldsymbol{Z})$.

3.1 Partially Linear IV Model (`iv`)

The Partially Linear IV Model considers the same functional form restriction on the causal model as the Partially Linear Model in Section 2.1. Specifically, the Partially

11. We only accommodate approximation of optimal instruments under homoskedasticity. The instruments are valid more generally but are not optimal under heteroskedasticity. Obtaining optimal instruments under heteroskedasticity would require estimating conditional variance functions.

Linear IV Model maintains

$$Y = \theta_0 D + g_0(\boldsymbol{X}) + U,$$

where θ_0 is the unknown parameter of interest.[12]

The key deviation from the Partially Linear Model is that the identifying assumptions leverage instrumental variables Z, instead of directly restricting the dependence of D and U. For ease of exposition, we focus on scalar-valued instruments in this section but we emphasize that `ddml` for Partially Linear IV supports multiple instrumental variables and multiple treatment variables.

Assumptions 4 and 5 below are sufficient orthogonality and relevance conditions, respectively, for identification of θ_0.

Assumption 4 (Conditional IV Orthogonality) $E[Cov(U, Z|\boldsymbol{X})] = 0$.

Assumption 5 (Conditional Linear IV Relevance) $E[Cov(D, Z|\boldsymbol{X})] \neq 0$.

To show identification, consider the score function

$$\psi(\boldsymbol{W}; \theta, \ell, m, r) = \Big(Y - \ell(\boldsymbol{X}) - \theta(D - m(\boldsymbol{X}))\Big)\Big(Z - r(\boldsymbol{X})\Big),$$

where $\boldsymbol{W} \equiv (Y, D, \boldsymbol{X}, Z)$. Note that for $\ell_0(\boldsymbol{X}) \equiv E[Y|\boldsymbol{X}]$, $m_0(\boldsymbol{X}) \equiv E[D|\boldsymbol{X}]$, and $r_0(\boldsymbol{X}) \equiv E[Z|\boldsymbol{X}]$, Assumption 4 implies $E[\psi(\boldsymbol{W}; \theta_0, \ell_0, m_0, r_0)] = 0$. We will also have that the Gateux derivative of $E[\psi(\boldsymbol{W}; \theta_0, \ell_0, m_0, r_0)]$ with respect to the nuisance functions (ℓ_0, m_0, r_0) will be zero. Rewriting $E[\psi(\boldsymbol{W}; \theta_0, \ell_0, m_0, r_0)] = 0$ then results in a Wald expression given by

$$\theta_0 = \frac{E\left[\big(Y - \ell_0(\boldsymbol{X})\big)\big(Z - r_0(\boldsymbol{X})\big)\right]}{E\left[\big(D - m_0(\boldsymbol{X})\big)\big(Z - r_0(\boldsymbol{X})\big)\right]}, \tag{9}$$

where Assumption 5 is used to ensure a non-zero denominator.

The DDML estimator based on Equation (9) is given by

$$\hat{\theta}_n = \frac{\frac{1}{n}\sum_{i=1}^{n}\Big(Y_i - \hat{\ell}_{I^c_{k_i}}(\boldsymbol{X}_i)\Big)\Big(Z_i - \hat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)\Big)}{\frac{1}{n}\sum_{i=i}^{n}\Big(D_i - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)\Big)\Big(Z_i - \hat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)\Big)}, \tag{10}$$

where $\hat{\ell}_{I^c_k}$, $\hat{m}_{I^c_k}$, and $\hat{r}_{I^c_k}$ are appropriate cross-fitted CEF estimators.

Standard errors corresponding to $\hat{\theta}_n$ are equivalent to the IV standard errors where $Y_i - \hat{\ell}_{I^c_{k_i}}(\boldsymbol{X}_i)$ is the outcome, $D_i - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)$ is the endogenous variable, and $Z_i - \hat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)$ is the instrument. `ddml` supports conventional standard errors available for

12. As in Section 2.1, the interpretation of θ_0 can be generalized under stronger identifying assumptions. See Angrist et al. (2000).

linear instrumental variable regression in Stata, including heteroskedasticity and cluster-robust standard errors. Mean and median aggregation over cross-fitting repetitions are implemented as outlined in Remark 2. In the case where we have multiple instruments or endogenous regressors, we adjust the algorithm by residualizing each instrument and endogenous variable as above and applying two-stage least squares with the residualized outcome, endogenous variables, and instruments.

3.2 Flexible Partially Linear IV Model (`fiv`)

The Flexible Partially Linear IV Model considers the same parameter of interest as the Partially Linear IV Model. The key difference here is that identification is based on a stronger independence assumption which allows for approximating optimal instruments using nonparametric estimation, including machine learning, akin to Belloni et al. (2012) and Chernozhukov et al. (2015a). In particular, the Flexible Partially Linear IV Model leverages a conditional mean independence assumption rather than an orthogonality assumption as in Section 3.1. As in Section 3.1, we state everything in the case of a scalar D.

Assumption 6 (Conditional IV Mean Independence) $E[U|\boldsymbol{Z},\boldsymbol{X}] = 0$.

Assumption 6 implies that for any function $\tilde{p}(\boldsymbol{Z},\boldsymbol{X})$, it holds that

$$E\left[\Big(Y - \ell_0(X) - \theta\big(D - m_0(\boldsymbol{X})\big)\Big)\Big(\tilde{p}(\boldsymbol{Z},\boldsymbol{X}) - E\left[\tilde{p}(\boldsymbol{Z},\boldsymbol{X})|\boldsymbol{X}\right]\Big)\right] = 0, \tag{11}$$

where $\ell_0(\boldsymbol{X}) = E[Y|\boldsymbol{X}]$ and $m_0(\boldsymbol{X}) = E[D|\boldsymbol{X}]$. Identification based on (11) requires that there exists some function $\tilde{p}$ such that

$$E\left[Cov(D,\tilde{p}(\boldsymbol{Z},\boldsymbol{X})|\boldsymbol{X})\right] \neq 0. \tag{12}$$

A sufficient assumption is that D and $\boldsymbol{Z}$ are not mean independent conditional on $\boldsymbol{X}$. This condition allows setting $\tilde{p}(\boldsymbol{Z},\boldsymbol{X}) = E[D|\boldsymbol{Z},\boldsymbol{X}]$ which will then satisfy (12).[13] Assumption 7 is a consequence of this non-mean independence.

Assumption 7 (Conditional IV Relevance) $E[Var(E[D|\boldsymbol{Z},\boldsymbol{X}]|\boldsymbol{X})] \neq 0$.

Consider now the score function

$$\psi(\boldsymbol{W};\theta,\ell,m,p) = \Big(Y - \ell(\boldsymbol{X}) - \theta(D - m(\boldsymbol{X}))\Big)\Big(p(\boldsymbol{Z},\boldsymbol{X}) - m(\boldsymbol{X})\Big),$$

where $\boldsymbol{W} \equiv (Y, D, \boldsymbol{X}, \boldsymbol{Z})$. Note that for $\ell_0(\boldsymbol{X}) \equiv E[Y|\boldsymbol{X}]$, $m_0(\boldsymbol{X}) \equiv E[D|\boldsymbol{X}]$, and $p_0(\boldsymbol{Z},\boldsymbol{X}) \equiv E[D|\boldsymbol{Z},\boldsymbol{X}]$, Assumption 6 and the law of iterated expectations imply

13. The choice $\tilde{p}(\boldsymbol{Z},\boldsymbol{X}) = E[D|\boldsymbol{Z},\boldsymbol{X}]$ results in the optimal instrument, in the sense of semi-parametric efficiency, under homoskedasticity.

$E[\psi(\boldsymbol{W}; \theta_0, \ell_0, m_0, p_0)] = 0$ and the Gateaux differentiability condition holds. Rewriting then results in a Wald expression given by

$$\theta_0 = \frac{E\left[\big(Y - \ell_0(\boldsymbol{X})\big)\big(p_0(\boldsymbol{Z}, \boldsymbol{X}) - m_0(\boldsymbol{X})\big)\right]}{E[\big(D - m_0(\boldsymbol{X})\big)\big(p_0(\boldsymbol{Z}, \boldsymbol{X}) - m_0(\boldsymbol{X})\big)]}, \tag{13}$$

where Assumption 7 ensures a non-zero denominator.

The DDML estimator based on the moment solution (13) is given by

$$\hat{\theta}_n = \frac{\frac{1}{n}\sum_{i=1}^{n}\big(Y_i - \hat{\ell}_{I^c_{k_i}}(\boldsymbol{X}_i)\big)\big(\hat{p}_{I^c_{k_i}}(\boldsymbol{Z}_i, \boldsymbol{X}_i) - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)\big)}{\frac{1}{n}\sum_{i=i}^{n}\big(D_i - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)\big)\big(\hat{p}_{I^c_{k_i}}(\boldsymbol{Z}_i, \boldsymbol{X}_i) - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)\big)}, \tag{14}$$

where $\hat{\ell}_{I^c_k}$, $\hat{m}_{I^c_k}$, and $\hat{p}_{I^c_k}$ are appropriate cross-fitted CEF estimators.

In simulations, we find that the finite sample performance of the estimator in (14) improves when the law of iterated expectations applied to $E[p_0(\boldsymbol{Z}, \boldsymbol{X})] = m_0(\boldsymbol{X})$ is explicitly approximately enforced in estimation. As a result, we propose an intermediate step to the previously considered two-step DDML algorithm: Rather than estimating the conditional expectation of D given $\boldsymbol{X}$ directly, we estimate it by projecting first-step estimates of the conditional expectation of $p_0(\boldsymbol{Z}, \boldsymbol{X})$ onto $\boldsymbol{X}$ instead. Algorithm 2 outlines the LIE-compliant DDML algorithm for computation of (14).

❏ **Algorithm 2. LIE-compliant DDML for the Flexible Partially Linear IV Model.**

Split the sample $\{(Y_i, D_i, \boldsymbol{X}_i, \boldsymbol{Z}_i)\}_{i=1}^n$ randomly in K folds of approximately equal size. Denote I_k the set of observations included in fold k and I_k^c its complement.

1. For each $k \in \{1, \ldots, K\}$:
   a. Fit a CEF estimator to the sub-sample I_k^c using Y_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\hat{\ell}_{I^c_k}(\boldsymbol{X}_i)$ for $i \in I_k$.
   b. Fit a CEF estimator to the sub-sample I_k^c using D_i as the outcome and $(\boldsymbol{Z}_i, \boldsymbol{X}_i)$ as predictors. Obtain the out-of-sample predicted values $\hat{p}_{I^c_k}(\boldsymbol{Z}_i, \boldsymbol{X}_i)$ for $i \in I_k$ and in-sample predicted values $\hat{p}_{I^c_k}(\boldsymbol{Z}_i, \boldsymbol{X}_i)$ for $i \in I_k^c$.
   c. Fit a CEF estimator to the sub-sample I_k^c using the in-sample predicted values $\hat{p}_{I^c_k}(\boldsymbol{Z}_i, \boldsymbol{X}_i)$ as the outcome and X_i as predictors. Obtain the out-of-sample predicted values $\hat{m}_{I^c_k}(\boldsymbol{X}_i)$ for $i \in I_k$.
2. Compute (14).

❏

Standard errors corresponding to $\hat{\theta}_n$ in (14) are the same as in Section 3.1 where the instrument is now given by $\hat{p}_{I^c_{k_i}}(\boldsymbol{Z}_i, \boldsymbol{X}_i) - \hat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)$. Mean and median aggregation over cross-fitting repetitions are as outlined in Remark 2.

3.3 Interactive IV Model (`interactiveiv`)

The Interactive IV Model considers the same causal model as in Section 2.2; specifically

$$Y = g_0(D, \boldsymbol{X}) + U$$

where D takes values in $\{0, 1\}$. The key difference from the Interactive Model is that this section considers identification via a binary instrument Z.

The parameter of interest we target is

$$\theta_0 = E\left[g_0(1, \boldsymbol{X}) - g_0(0, \boldsymbol{X}) | \, p_0(1, \boldsymbol{X}) > p_0(0, \boldsymbol{X})\right], \tag{15}$$

where $p_0(Z, \boldsymbol{X}) \equiv \Pr(D = 1|Z, \boldsymbol{X})$. Here, θ_0 is a local average treatment effect (LATE). Note that in contrast to the LATE developed in Imbens and Angrist (1994), "local" here does not strictly refer to compliers but instead observations with a higher propensity score – i.e., a higher probability of complying.[14]

Identification again leverages Assumptions 6 and 7 made in the context of the Flexible Partially Linear IV Model. In addition, we assume that the propensity score is weakly monotone with probability one, and that the support of the instrument is independent of the controls.

Assumption 8 (Monotonicity) $p_0(1, \boldsymbol{X}) \geq p_0(0, \boldsymbol{X})$ *with probability 1.*

Assumption 9 (IV Overlap) $\Pr(Z = 1|\boldsymbol{X}) \in (0, 1)$ *with probability 1.*

Assumptions 6-9 imply that

$$\theta_0 = \frac{E\left[\ell_0(1, \boldsymbol{X}) - \ell_0(0, \boldsymbol{X})\right]}{E\left[p_0(1, \boldsymbol{X}) - p_0(0, \boldsymbol{X})\right]}, \tag{16}$$

where $\ell_0(Z, \boldsymbol{X}) \equiv E[Y|Z, \boldsymbol{X}]$, verifying identification of the LATE θ_0. Akin to Section 6, however, estimators of θ_0 should not directly be based on Equation (16) because the estimating equations implicit in obtaining (16) do not satisfy Neyman-orthogonality. Hence, a direct estimator of θ_0 obtained by plugging nonparametric estimators in for nuisance functions in (16) will potentially be highly sensitive to the first step nonparametric estimation error. Rather, we base estimation on the Neyman orthogonal score function

$$\begin{aligned}\psi(\boldsymbol{W}; \theta, \ell, p, r) = &\frac{Z(Y - \ell(1, \boldsymbol{X}))}{r(\boldsymbol{X})} - \frac{(1 - Z)(Y - \ell(0, \boldsymbol{X}))}{1 - r(\boldsymbol{X})} + \ell(1, \boldsymbol{X}) - \ell(0, \boldsymbol{X}) \\ &+ \left[\frac{Z(D - p(1, \boldsymbol{X}))}{r(\boldsymbol{X})} - \frac{(1 - Z)(D - p(0, \boldsymbol{X}))}{1 - r(\boldsymbol{X})} + p(1, \boldsymbol{X}) - p(0, \boldsymbol{X})\right] \times \theta\end{aligned}$$

14. Identification of the conventional complier-focused LATE is achieved under stronger conditional independence and monotonicity assumptions. Under these stronger assumptions, the DDML LATE estimator outlined here targets the conventionally considered LATE parameter.

where $\boldsymbol{W} \equiv (Y, D, \boldsymbol{X}, Z)$. Note that under Assumptions 6-9 and for $\ell_0(Z, \boldsymbol{X}) \equiv E[Y|Z, \boldsymbol{X}]$, $p_0(Z, \boldsymbol{X}) \equiv E[D|Z, \boldsymbol{X}]$, and $r_0(\boldsymbol{X}) \equiv E[Z|\boldsymbol{X}]$, we have $E[\psi(\boldsymbol{W}; \theta_0, \ell_0, p_0, r_0)] = 0$ and can verify that its Gateaux derivative with respect to the nuisance functions local to their true values is also zero.

The DDML estimator based on the orthogonal score ψ is then

$$\hat{\theta}_n = \frac{\frac{1}{n}\sum_i \left(\frac{Z_i(Y_i - \hat{\ell}_{I^c_{k_i}}(1, \boldsymbol{X}_i))}{\hat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} - \frac{(1-Z_i)(Y_i - \hat{\ell}_{I^c_{k_i}}(0, \boldsymbol{X}_i))}{1-\hat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} + \hat{\ell}_{I^c_{k_i}}(1, \boldsymbol{X}_i) - \hat{\ell}_{I^c_{k_i}}(0, \boldsymbol{X}_i) \right)}{\frac{1}{n}\sum_i \left(\frac{Z_i(D_i - \hat{p}_{I^c_{k_i}}(1, \boldsymbol{X}_i))}{\hat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} - \frac{(1-Z_i)(D_i - \hat{p}_{I^c_{k_i}}(0, \boldsymbol{X}_i))}{1-\hat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} + \hat{p}_{I^c_{k_i}}(1, \boldsymbol{X}_i) - \hat{p}_{I^c_{k_i}}(0, \boldsymbol{X}_i) \right)}, \tag{17}$$

where $\hat{\ell}_{I^c_k}$, $\hat{p}_{I^c_k}$, and $\hat{r}_{I^c_k}$ are appropriate cross-fitted CEF estimators. Since Z is binary, the cross-fitted values $\hat{\ell}_{I^c_k}(1, \boldsymbol{X})$ and $\hat{p}_{I^c_k}(1, \boldsymbol{X})$, as well as $\hat{\ell}_{I^c_k}(0, \boldsymbol{X})$ and $\hat{p}_{I^c_k}(0, \boldsymbol{X})$ are computed by only using treated and untreated observations, respectively.

`ddml` supports heteroskedasticity and cluster-robust standard errors for $\hat{\theta}_n$. Mean and median aggregation over cross-fitting repetitions are implemented as outlined in Remark 2.

4 The choice of machine learner

Chernozhukov et al. (2018) show that DDML estimators are asymptotically normal when used in combination with a general class of machine learners satisfying a relatively weak convergence rate requirement for estimating the CEFs. While asymptotic properties of common machine learners remain an highly active research area, recent advances provide convergence rates for special instances of many machine learners, including lasso (Bickel et al. 2009; Belloni et al. 2012), random forests (Wager and Walther 2016; Wager and Athey 2018; Athey et al. 2019), neural networks (Schmidt-Hieber 2020; Farrell et al. 2021), and boosting (Luo et al. 2022). It seems likely that many popular learners will fall under the umbrella of suitable learners as theoretical results are further developed. However, we note that currently known asymptotic properties do not cover a wide range of learners, such as very deep and wide neural networks and deep random forests, as they are currently implemented in practice.

The relative robustness of DDML to the first-step learners leads to the question of which machine learner is the most appropriate for a given application. It is *ex ante* rarely obvious which learner will perform best. Further, rather than restricting ourselves to one learner, we might want to combine several learners into one final learner. This is the idea behind stacking generalization, or simply "stacking", due to Wolpert (1992) and Breiman (1996). Stacking allows one to accommodate a diverse set of base learners with varying tuning and hyper-tuning parameters. It thus provide a convenient framework for combining and identifying suitable learners, thereby reducing the risk of misspecification.

We introduce stacking for DDML in Section 4.1. Section 4.2 demonstrates the performance of DDML in combination with stacking using a simulation.

4.1 Stacking

Our discussion of stacking in the context of DDML focuses on the Partially Linear Model in (2), but we highlight that DDML and stacking can be combined in the same way for all other models supported in `ddml`. Suppose we consider J machine learners, referred to as base learners, to estimate the CEFs $\ell_0(\boldsymbol{X}) \equiv E[Y|\boldsymbol{X}]$ and $m_0(\boldsymbol{X}) \equiv E[D|\boldsymbol{X}]$. The set of base learners could, for example, include cross-validated lasso and ridge with alternative sets of predictors, gradient boosted trees with varying tree depth and feed-forward neural nets with varying number of hidden layers and neurons. Generally, we recommend considering a relatively large and diverse set of base learners, and including some learners with alternative tuning parameters.

We randomly split the sample into K cross-fitting folds, denoted as $I_1, \ldots, I_K$. In each cross-fitting step k, we define the training sample as $I_k^c \equiv T_k$, comprising all observations excluding the cross-fitting hold-out fold k. This training sample is further divided into V cross-validation folds, denoted as $T_{k,1}, \ldots, T_{k,V}$. The stacking regressor fits a final learner to the training sample T_k using the cross-validated predicted values of each base learner as inputs. A typical choice for the final learner is constrained least squares (CLS) which restricts the weights to be positive and sum to one. The stacking objective function for estimating $\ell_0(\boldsymbol{X})$ using the training sample T_k is then defined as:

$$\min_{w_{k,1},\ldots,w_{k,J}} \sum_{i \in T_k} \left(Y_i - \sum_{j=1}^{J} w_{k,j} \hat{\ell}^{(j)}_{T^c_{k,v(i)}}(\boldsymbol{X}_i) \right)^2, \qquad \text{s.t. } w_{k,j} \geq 0, \ \sum_{j=1}^{J} |w_{k,j}| = 1,$$

where $w_{k,j}$ are referred to as stacking weights. We use $\hat{\ell}^{(j)}_{T^c_{k,v(i)}}(\boldsymbol{X}_i)$ to denote the cross-validated predicted value for observation i, which is obtained from fitting learner j on the sub-sample $T^c_{k,v(i)} \equiv T_k \setminus T_{k,v(i)}$, i.e., the sub-sample excluding the fold $v(i)$ which observation i falls into. The stacking predicted values are obtained as $\sum_j \hat{w}_{k,j} \hat{\ell}^{(j)}_k(\boldsymbol{X}_i)$ where each learner j is fit on the step-k training sample T_k. The objective function for estimating $m_0(\boldsymbol{X})$ is defined accordingly.

CLS frequently performs well in practice and facilitates the interpretation of stacking as a weighted average of base learners (Hastie et al. 2009). It is, however, not the only sensible choice of combining base learners. For example, stacking could instead select the single learner with the lowest quadratic loss, i.e., by imposing the constraint $w_{k,j} \in \{0, 1\}$ and $\sum_{k,j} w_{k,j} = 1$. We refer to this choice as "single best" and include it in our simulation experiments. We implement stacking for DDML using `pystacked` (Ahrens et al. 2022).

Short-stacking. Stacking relies on cross-validation. In the context of DDML we can also exploit the cross-*fitted* predicted values directly for stacking. That is, we can di-

rectly apply CLS to the cross-fitted predicted values for estimating $\ell_0(\boldsymbol{X})$ (and similarly $m_0(\boldsymbol{X})$):

$$\min_{w_1,\ldots,w_J} \sum_{i=1}^{n} \left(Y_i - \sum_{j=1}^{J} w_j \hat{\ell}^{(j)}_{I^c_{k(i)}}(\boldsymbol{X}_i) \right)^2, \qquad \text{s.t. } w_j \geq 0, \ \sum_{j=1}^{J} |w_j| = 1$$

We refer to this form of stacking that utilizes the cross-fitted predicted values as *short-stacking* as it takes a short-cut. This is to contrast it with regular stacking which estimates the stacking weights for each cross-fitting fold k. The main advantage of short-stacking is the lower computational cost. Short-stacking also produces a single set of weights for the entire sample, rather than a different set of weights in each cross-fit fold and thus facilitates interpretation. Algorithm A.4 in the Appendix summarizes the short-stacking algorithm for the Partially Linear Model.[15]

4.2 Monte Carlo simulation

To illustrate the advantages of DDML with stacking, we generate artificial data based on the Partially Linear Model

$$Y_i = \theta_0 D_i + c_Y g(\boldsymbol{X}_i) + \sigma_Y(D_i, \boldsymbol{X}_i)\varepsilon_i \tag{18}$$
$$D_i = c_D g(\boldsymbol{X}_i) + \sigma_D(\boldsymbol{X}_i)u_i \tag{19}$$

where both ε_i and u_i are independently drawn from the standard normal distribution. We set the target parameter to $\theta_0 = 0.5$ and the sample size to either $n = 100$ or $n = 1000$. The controls $\boldsymbol{X}_i$ are drawn from the multivariate normal distribution with $N(0, \boldsymbol{\Sigma})$ where $\Sigma_{ij} = (0.5)^{|i-j|}$. The number of controls is set to $p = \dim(\boldsymbol{X}_i) = 50$, except in DGP 5 where $p = 7$. The constants c_Y and c_D are chosen such that the R^2 in (18) and (19) are approximately equal to 0.5. To induce heteroskedasticity, we set

$$\sigma_D(\boldsymbol{X}_i) = \sqrt{\frac{(1 + g(\boldsymbol{X}_i))^2}{\frac{1}{n}\sum_i (1 + g(\boldsymbol{X}_i))^2}} \quad \text{and} \quad \sigma_Y(D_i, \boldsymbol{X}_i) = \sqrt{\frac{(1 + \theta_0 D_i + g(\boldsymbol{X}_i))^2}{\frac{1}{n}\sum_i (1 + \theta_0 D_i + g(\boldsymbol{X}_i))^2}}$$

The nuisance function $g(\boldsymbol{X}_i)$ is generated using five exemplary DGPs, which cover linear and nonlinear processes with varying degrees of sparsity and varying number of observed

15. While short-stacking can be applied in a similar fashion to other conditional expectations, a complication arises in the Flexible Partially Linear IV Model where the cross-fitted predicted values of $E[D|\boldsymbol{X}]$ depend on $E[D|\boldsymbol{X}, \boldsymbol{Z}]$. We describe the algorithm that accounts for this in the Appendix; see Algorithm A.5.

covariates:

$$
\begin{aligned}
\text{DGP 1:} \quad & g(\boldsymbol{X}_i) && = \sum\nolimits_j 0.9^j X_{ij} \\
\text{DGP 2:} \quad & g(\boldsymbol{X}_i) && = X_{i1}X_{i2}+X_{i3}^2+X_{i4}X_{i5}+X_{i6}X_{i7}+X_{i8}X_{i9}+X_{i10}+X_{i11}^2+X_{i12}X_{i13} \\
\text{DGP 3:} \quad & g(\boldsymbol{X}_i) && = \mathbb{1}\{X_{i1} > 0.3\}\mathbb{1}\{X_{i2} > 0\}\mathbb{1}\{X_{i3} > -1\} \\
\text{DGP 4:} \quad & g(\boldsymbol{X}_i) && = X_{i1} + \sqrt{|X_{i2}|} + \sin(X_{i3}) + 0.3X_{i4}X_{i5} + X_{i6} + 0.3X_{i7}^2 \\
\text{DGP 5:} \quad & g(\boldsymbol{X}_i) && = \text{same as DGP 4 with } p = 7
\end{aligned}
$$

DGP 1 is a linear design involving many negligibly small parameters. While not exactly sparse, the design can be approximated well through a sparse representation. DGP 2 is linear in the parameters and exactly sparse, but includes interactions and second-order polynomials. DGPs 3-5 are also exactly sparse but involve complex nonlinear and interaction effects. DGP 4 and 5 are identical, except that DGP 5 does not add nuisance covariates which are unrelated to Y and D.

We consider DDML with the following supervised machine learners for cross-fitting the CEFs:[16]

1.-2. Cross-validated lasso & ridge with untransformed base controls
3.-4. Cross-validated lasso & ridge with 5th-order polynomials of base controls but no interactions (referred to as 'Poly 5')
5.-6. Cross-validated lasso & ridge with second-order polynomials and all first order interaction terms (referred to as 'Poly 2 + Inter.')
7. Random forests (RF) with low regularization: base controls, maximum tree depth of 10, 500 trees and approximately $\sqrt{p}$ features considered at each split
8. RF with medium regularization: same as 7., but with maximum tree depth of 6
9. RF with high regularization: same as 7., but with maximum tree depth of 2
10. Gradient boosted trees (GB) with low regularization: base controls, 1000 trees and a learning rate of 0.3. We enable early stopping which uses a 20% validation sample to decide whether to stop the learning algorithm. Learning is terminated after 5 iterations with no meaningful improvement in the mean-squared loss of the validation sample.[17]
11. GB with medium regularization: same as 10., but with learning rate of 0.1
12. GB with high regularization: same as 10., but with learning rate of 0.01
13. Feed-forward neural net with base controls and two layers of size 20

We use the above set of learners as base learners for DDML with stacking approaches. Specifically, we estimate DDML using stacking regression with CLS, stacking with single-best learner, and short-stacking with CLS. We set the number of folds to $K = 20$ if $n = 100$, and $K = 5$ if $n = 1000$. For comparison, we report results for OLS using the base controls, PDS-Lasso with the base controls, PDS-Lasso with Poly 5, PDS-Lasso with Poly 2 + Interactions, and an oracle estimator using the full sample.[18] The oracle

16. All base learners have been implemented using `pystacked`. We use the defaults of `pystacked` for parameter values and settings not mentioned here.
17. We use a tolerance level of 0.01 to measure improvements.
18. The PDS-Lasso estimators set tuning parameters using the default in `pdslasso`.

estimator presumes knowledge of the function $g(\boldsymbol{X})$, and obtains estimates by regressing Y on the two variables D and $g(\boldsymbol{X})$.

We report simulation median absolute bias (MAB) and coverage rates of 95% confidence intervals (CR) for DGPs 1-3 in Table 1. We delegate results for DGPs 4 and 5, including a brief discussion, to Appendix B. DDML estimators leveraging stacking or short-stacking perform favorably in comparison to individual base learners in terms of bias and coverage. The relative performance of stacking approaches seems to improve as the sample size increases, likely reflecting that the stacking weights are imprecisely estimated in very small samples. For $n = 1000$, the bias of stacking with CLS is at least as low as the bias of the best-performing individual learner under DGP 1-2, while only gradient boosting and neural net yield a lower bias than stacking under DGP 3.

Results for coverage are similar with stacking based estimates being comparable with the best performing feasible estimates and the oracle when $n = 1000$. With $n = 100$, coverage of confidence intervals for stacking-based estimators are inferior to coverages for a small number of the individual learners but still competitive and superior than most learners. Looking across all results, we see that stacking provides robustness to potentially very bad performance that could be obtained from using a single poorly performing learner.

There are overall little performance differences among the four stacking approaches considered. There is some evidence that the single-best selector outperforms CLS in very small sample sizes in DGPs 2-3, but not in DGP 1 (and also not in DGPs 4-5, see Table B.1). We suspect that the single-best selector works better in scenarios where there is one base learner that clearly dominates.

The mean-squared prediction errors (MSPE) and the average stacking weights, which we report in Tables B.2 and B.3 in the Appendix, provide further insights into how stacking with CLS functions. CLS assigns large stacking weights to base learners with a low MSPE, which in turn are associated with a low bias. Importantly, stacking assigns zero or close-to-zero weights to poorly specified base learners such as the highly regularized random forest, which in all three DGPs ranks among the individual learners with highest MSPE and highest bias. The robustness to misspecified and ill-chosen machine learners, which could lead to misleading inference, is indeed one of our main motivations for advocating stacking approaches to DDML.

DDML with stacking approaches also compare favorably to conventional full-sample estimators. In the relatively simple linear DGP 1, DDML with stacking performs similarly to OLS and the infeasible oracle estimator—both in terms of bias and coverage—for $n = 100$ and $n = 1000$. In the more challenging DGPs 2 and 3, the bias of DDML with stacking is substantially lower than the biases of OLS and the PDS-Lasso estimators. While the bias and size distortions of DDML with stacking are still considerable in comparison to the infeasible oracle for $n = 100$, they are close to the oracle for $n = 1000$. The results overall highlight the flexibility of DDML with stacking to flexibly approximate a wide range of DGPs provided a diverse set of base learners is chosen.

Table 1: Bias and Coverage Rates in the Linear and Nonlinear DGPs

	DGP 1				DGP 2				DGP 3			
	$n = 100$		$n = 1000$		$n = 100$		$n = 1000$		$n = 100$		$n = 1000$	
	MAB	CR.	MAB	CR	MAB	CR	MAB	CR	MAB	CR	MAB	CR
Full sample:												
Oracle	0.109	0.909	0.041	0.927	0.101	0.889	0.032	0.942	0.114	0.905	0.038	0.933
OLS (Base)	0.126	0.879	0.040	0.925	0.276	0.392	0.289	0.	0.212	0.637	0.207	0.015
PDS-Lasso (Base)	0.119	0.850	0.042	0.918	0.289	0.309	0.289	0.	0.220	0.607	0.207	0.011
PDS-Lasso (Poly 5)	0.144	0.776	0.042	0.919	0.215	0.491	0.105	0.304	0.219	0.594	0.198	0.026
PDS-Lasso (Poly 2 + Inter.)	0.155	0.760	0.042	0.912	0.203	0.543	0.033	0.931	0.219	0.595	0.175	0.059
DDML methods:												
Base learners												
PDS-Lasso (Poly 2 + Inter.)	0.155	0.760	0.042	0.912	0.203	0.543	0.033	0.931	0.219	0.595	0.175	0.059
OLS	0.140	0.710	0.041	0.900	0.283	0.241	0.290	0.	0.215	0.423	0.207	0.013
Lasso with CV (Base)	0.116	0.890	0.042	0.932	0.278	0.338	0.289	0.	0.212	0.621	0.208	0.012
Ridge with CV (Base)	0.119	0.880	0.040	0.904	0.286	0.324	0.291	0.	0.234	0.573	0.218	0.006
Lasso with CV (Poly 5)	0.129	0.872	0.042	0.923	0.128	0.745	0.096	0.436	0.197	0.662	0.192	0.029
Ridge with CV (Poly 5)	0.137	0.805	0.050	0.824	0.180	0.621	0.129	0.173	0.256	0.516	0.220	0.008
Lasso with CV (Poly 2 + Inter.)	0.247	0.667	0.065	0.806	0.173	0.720	0.085	0.634	0.238	0.619	0.238	0.009
Ridge with CV (Poly 2 + Inter.)	0.318	0.356	0.076	0.633	0.137	0.751	0.083	0.458	0.134	0.762	0.068	0.684
Random forest (Low)	0.178	0.733	0.100	0.541	0.246	0.434	0.196	0.016	0.223	0.612	0.110	0.457
Random forest (Medium)	0.182	0.720	0.129	0.330	0.243	0.420	0.224	0.004	0.225	0.612	0.139	0.217
Random forest (High)	0.237	0.586	0.229	0.011	0.266	0.368	0.271	0.	0.251	0.542	0.230	0.005
Gradient boosting (Low)	0.107	0.932	0.040	0.901	0.129	0.783	0.082	0.560	0.113	0.885	0.036	0.936
Gradient boosting (Medium)	0.124	0.851	0.047	0.857	0.199	0.566	0.144	0.119	0.131	0.855	0.044	0.900
Gradient boosting (High)	0.233	0.553	0.154	0.150	0.272	0.346	0.259	0.002	0.189	0.693	0.078	0.707
Neural net	0.149	0.782	0.152	0.169	0.105	0.845	0.075	0.528	0.116	0.864	0.039	0.854
Meta learners												
Stacking: CLS	0.115	0.902	0.042	0.928	0.169	0.675	0.036	0.915	0.165	0.756	0.046	0.866
Stacking: Single best	0.116	0.896	0.042	0.933	0.147	0.745	0.036	0.918	0.154	0.792	0.044	0.894
Short-stacking: CLS	0.113	0.898	0.041	0.928	0.171	0.662	0.037	0.907	0.163	0.766	0.044	0.892
Short-stacking: Single best	0.110	0.902	0.042	0.932	0.134	0.734	0.035	0.917	0.148	0.825	0.042	0.910

Notes: The table reports median absolute bias (MAB) and coverage rate of a 95% confidence interval (CR). We employ standard errors robust to heteroskedasticity. For comparison, we report the following full sample estimators: infeasible Oracle, OLS, PDS-Lasso with base and two different expanded sets of covariates. DDML estimators use 20 folds for cross-fitting if $n = 100$, and 5 folds if $n = 1000$. Meta-learning approaches rely on all listed base learners. Results are based on 1,000 replications. Results for DGPs 4 and 5 can be found in Table B.1 in the Appendix.

5 The program

In this section, we provide an overview of the `ddml` package. We introduce the syntax and workflow for the main programs in Section 5.1. Section 5.2 lists the options. Section 5.3 covers the simplified one-line program `qddml`. We provide an overview of supported machine learning programs in Section 5.4. Finally, Section 5.5 adds a note on how to ensure replication with `ddml`.

5.1 Syntax: `ddml`

The `ddml` estimation proceeds in four steps.

Step 1: Initialize `ddml` and select model.

`ddml init` *model* [`,` `mname(`*name*`)` `vars(`*varlist*`)` `kfolds(`*integer*`)`
`fcluster(`*varname*`)` `foldvar(`*varlist*`)` `reps(`*integer*`)` `tabfold` `vars(`*varlist*`)`]

where *model* selects between the Partially Linear Model (`partial`), the Interactive Model (`interactive`), the Partially Linear IV Model (`iv`), the Flexible Partially Linear IV Model (`fiv`), and the Interactive IV Model (`interactiveiv`). This step creates a persistent Mata object with the name provided by `mname(`*name*`)` in which model specifications and estimation results will be stored. The default name is *m0*.

At this stage, the user-specified folds for cross-fitting can be set via integer-valued Stata variables (see `foldvar(`*varlist*`)`). By default, observations are randomly assigned to folds and `kfolds(`*integer*`)` determines the number of folds (the default is 5). Cluster-randomized fold splitting is supported (see `fcluster(`*varname*`)`). The user can also select the number of times to fully repeat the cross-fitting procedure (see `rep(`*integer*`)`).

Step 2: Add supervised machine learners for estimating conditional expectations.

In this second step, we select the machine learning programs for estimating CEFs.

`ddml` *cond_exp* [`,` `mname(`*name*`)` `vname(`*varname*`)` `learner(`*name*`)` `vtype(`*string*`)`
`predopt(`*string*`)`] `:` *command depvar vars* [`,` *cmdopt*]

where *cond_exp* selects the conditional expectation to be estimated by the machine learning program *command*. At least one learner is required for each conditional expectation. Table 2 provides an overview of which conditional expectations are required by each model. The program *command* is a supervised machine learning program such as `cvlasso` or `pystacked` (see compatible programs in Section 5.4). The options *cmdopt* are specific to that program.

cond_exp	partial	interactive	iv	fiv	late
E[Y|X]	✓		✓	✓	
E[Y|X,D]		✓			
E[Y|X,Z]					✓
E[D|X]	✓	✓	✓	✓	
E[D|Z,X]				✓	✓
E[Z|X]			✓		✓

Table 2: The table lists the conditional expectations which need to be specified for each model.

Step 3: Perform cross-fitting.

This step implements the cross-fitting algorithm. Each learner is fit iteratively on training folds and out-of-sample predicted values are obtained. Cross-fitting is the most time-consuming step, as it involves fitting the selected machine learners repeatedly.

`ddml crossfit` [`, mname(`*name*`) shortstack`]

Step 4: Estimate causal effects.

In the last step, we estimate the parameter of interest for all combination of learners added in Step 2.

`ddml estimate` [`, mname(`*name*`) robust cluster(`*varname*`) vce(`*vcetype*`) att trim spec(`*string*`) rep(`*string*`)`]

To report and post selected results, we can use `ddml estimate` with the `replay` option:

`ddml estimate` [`, replay mname(`*name*`) spec(`*integer or string*`) rep(`*integer or string*`) fulltable notable allest`]

5.2 Options

Step 1 options: Initialization.

`mname(`*name*`)` name of the DDML model. Allows to run multiple DDML models simultaneously. Defaults to *m0*.

`kfolds(`*integer*`)` number of cross-fitting folds. The default is 5.

`fcluster(`*varname*`)` cluster identifiers for cluster randomization of folds.

`foldvar(`*varlist*`)` integer variables to specify custom folds (one per cross-fitting repetition).

`reps(`*integer*`)` number of cross-fitting repetitions, i.e., how often the cross-fitting procedure is repeated on randomly generated folds.

`tabfold` prints a table with frequency of observations by fold.

Step 2 options: Adding learners.

`vname(`*varname*`)` name of the dependent variable in the reduced form estimation. This is usually inferred from the *command* line but is mandatory for the `fiv` model.

`learner(`*varname*`)` optional name of the variable to be created.

`vtype(`*string*`)` optional variable type of the variable to be created. Defaults to *double*. *none* can be used to leave the type field blank. (Setting `vtype(`*none*`)` is required when using `ddml` with `rforest`.)

`predopt(`*varname*`)` `predict` option to be used to get predicted values. Typical values could be `xb` or `pr`. Default is blank.

Step 3 options: Cross-fitting.

`shortstack` asks for short-stacking to be used. Short-stacking runs constrained least squares on the cross-fitted predicted values to obtain a weighted average of several base learners.

Step 4 options: Estimation.

`spec(`*string*`)` select specification. This can either be the specification number, *mse* for minimum-MSE specification (the default) or *ss* for short-stacking.

`rep(`*string*`)` select cross-fitting repetitions. This can either be the cross-fit repetition number, *mn* for mean aggregation or *md* for median aggregation (the default). See Remark 2 for more information.

`robust` report SEs that are robust to the presence of arbitrary heteroskedasticity.

`cluster(`*varname*`)` select cluster-robust variance-covariance estimator.

`vce(`*type*`)` select variance-covariance estimator, e.g. `vce(hc3)` or `vce(cluster id)`. See `help regress##vcetype` for available options.

`trim(`*real*`)` trimming of propensity scores for the Interactive and Interactive IV models. The default is 0.01 (that is, values below 0.01 and above 0.99 are set to 0.01 and 0.99, respectively).

`atet` report average treatment effect of the treated (default is ATE).

`noconstant` suppress constant term in the estimation stage (only relevant for partially linear models).

5.3 Short syntax: `qddml`

The `ddml` package includes the wrapper program `qddml` which provides a one-line syntax for estimating a `ddml` model. The one-line syntax follows the syntax of `pdslasso` and `ivlasso` (Ahrens et al. 2018). The main restriction of `qddml` compared to the more flexible multi-line syntax is that `qddml` only allows for one user-specified machine learner (in addition to `regress`, which is added by default).

Syntax for Partially Linear and Interactive Model

`qddml` *depvar treatment_vars* (*controls*)`, model(partial|interactive)` [*options*]

Syntax for IV models

`qddml` *depvar* (*controls*) (*treatment_vars=excluded_instruments*) `,`
`model(iv|late|fiv)` [`cmd(`*string*`) cmdopt(`*string*`)` *ddml_options* `noreg`]

where *ddml_options* options are listed in Section 5.2 and are internally passed on to the `ddml` sub-routines. `cmd(`*string*`)` selects the machine learning command to be used and `cmdopt(`*string*`)` allows to pass options to the estimation command. `qddml` adds by default `regress` to the user-specified learner; this behavior can be disabled with `noreg`. See the `qddml` help file for a full list of options.

5.4 Supported machine learning programs

`ddml` is compatible with any supervised ML program in Stata that supports the typical "`reg y x`" syntax, comes with a post-estimation `predict` and supports `if` statements. We have tested `ddml` with the following programs:

- `lassopack` implements regularized regression, e.g. lasso, ridge, elastic net (Ahrens et al. 2020).
- `pystacked` facilitates the stacking of a wide range of machine learners including regularized regression, random forests, support vector machines, gradient boosted trees and feed-forward neural nets using Python's *scikit-learn* (Ahrens et al. 2022; Pedregosa et al. 2011; Buitinck et al. 2013). In addition, `pystacked` can also be used as a front-end to fit individual machine learners.
- `rforest` is a random forest wrapper for WEKA (Schonlau and Zou 2020; Frank et al. 2009).

- `svmachines` allows for the estimation of support vector machines using `libsvm` (Chang and Lin 2011; Guenther and Schonlau 2018).
- The program `parsnip` of the package `mlrtime` provides access to R's *parsnip* machine learning library through `rcall` (Huntington-Klein 2021; Haghish 2019). Using `parsnip` requires the installation of the supplementary wrapper program `parsnip2`.[19]

Stata programs that are currently not supported can be added relatively easily using wrapper programs (see `parsnip2` for an example).

5.5 Inspecting results and replication

In this section we discuss how to ensure replicability when using `ddml`. We also discuss some tools available for tracing replication failures. First, however, we briefly describe how `ddml` stores results.

`ddml` stores estimated conditional expectations in Stata's memory as Stata variables. These variables can be inspected, graphed and summarized in the usual way. Fold ID variables are also stored as Stata variables (by default named `m0_fid_`r, where `m0` is the default model name and r is the cross-fitting repetition). `ddml` models are stored on Mata `struct`s and using Mata's associative arrays. Specifically, the `ddml` model created by `ddml init` is an `mStruct`, and information relating to the estimation of conditional expectations are stored in `eStruct`s. Results relating to the overall model estimation are stored in associative arrays that live in the `mStruct`, and results relating to the estimation of conditional expectations are stored in associative arrays that live in the corresponding `eStruct`s.

Replication tips:

- Set the Stata seed before `ddml init`. This ensure that the same random fold variable is used for a given data set.
- Using the same fold variable alone is usually not sufficient to ensure replication, since many machine learning algorithms involve randomization. That said, note that the fold variable is stored in memory and can be reused for subsequent estimations via the `foldvar(`*varlist*`)` option.
- Replication of `ddml` results may require additional steps with some programs that rely on randomization in other software environments, e.g., R or Python. `pystacked` uses a Python seed generated in Stata. Thus, when `ddml` is used with `pystacked`, setting the seed before `ddml init` also guarantees that the same Python seed underlies the stacking estimation. Other programs relying on randomization outside of Stata might not behave in the same way. Thus, when using other programs, check the help files for options to set external random seeds. Try estimating each of the individual learners on the entire sample to see what settings need to be passed to them for their results to replicate.

19. Available from `https://github.com/aahrens1/parsnip2`.

- The `ddml extract` utility can be used to retrieve and inspect a wide range of intermediate results and statistics from Mata structures. This can be useful for trying to track down estimation errors and replication anomalies.

- Beware of changing samples. Fold splits or learner idiosyncracies may mean that sample sizes vary slightly across learners, estimation samples and/or cross-fitting repetitions. Note that `ddml` stores sample indicators for cross-fitting repetitions as Stata variables; `ddml extract` with the `show(n)` option will report sample sizes by learner and fold. See the `ddml extract` help file for more information.

- The `ddml export` utility can be used to export the estimated conditional expectations, fold variables and sample indicators to a CSV format file for examination and comparison in other software environments.

6 Applications

We demonstrate the `ddml` workflow using two applications. In Section 6.1, we apply the DDML estimator to estimate the effect of 401(k) eligibility on financial wealth following Poterba et al. (1995). We focus on the Partially Linear Model for the sake of brevity, but provide code that demonstrates the use of `ddml` with the Interactive Model, Partially Linear IV Model and Interactive IV Model using the same application in Appendix C. Additional examples can also be found in the help file. Based on Berry et al. (1995), we show in Section 6.2 how to employ `ddml` for the estimation of the Flexible Partially Linear IV Model which allows both for flexibly controlling for confounding factors using high-dimensional function approximation of confounding factors and for estimation of optimal instrumental variables.

6.1 401(k) and financial wealth

The data consists of $n = 9915$ households from the 1991 SIPP. The application is originally due to Poterba et al. (1995), but has been revisited by Belloni et al. (2017), Chernozhukov et al. (2018), and Wüthrich and Zhu (2021), among others. Following previous studies, we include the control variables age, income, years of education, family size, as well as indicators for martial status, two-earner status, benefit pension status, IRA participation, and home ownership. The outcome is net financial assets and the treatment is eligibility to enroll for the 401(k) pension plan.

We load the data and define three globals for outcome, treatment and control variables. We then proceed in the four steps outlined in Section 5.1.

```
. use "sipp1991.dta", clear
. global Y net_tfa
. global X age inc educ fsize marr twoearn db pira hown
. global D e401
```

Step 1: Initialize `ddml` model.

We initialize the `ddml` model and select the Partially Linear Model in (2). Before initialization, we set the seed to ensure replication. This should always be done before `ddml init`, which executes the random fold assignment. In this example, we opt for four folds to ensure the readability of some of the output shown below, although we recommend considering a larger number of folds in practice.

```
. set seed 123
. ddml init partial, kfolds(4)
```

Step 2: Add supervised machine learners for estimating conditional expectations.

In this step, we specify which machine learning programs should be used for the estimation of the conditional expectations $E[Y|\boldsymbol{X}]$ and $E[D|\boldsymbol{X}]$. For each conditional expectation, at least one learner is required. For illustrative purposes, we consider `regress` for linear regression, `cvlasso` for cross-validated lasso and `rforest` for random forests. When using `rforest`, we need to add the option `vtype(none)` since the post-estimation `predict` command of `rforest` does not support variable types.

```
. *** add learners for E[Y|X]
. ddml E[Y|X]: reg $Y $X
Learner Y1_reg added successfully.
. ddml E[Y|X]: cvlasso $Y c.($X)##c.($X), lopt postresults
Learner Y2_cvlasso added successfully.
. ddml E[Y|X], vtype(none): rforest $Y $X, type(reg)
Learner Y3_rforest added successfully.
. *** add learners for E[D|X]
. ddml E[D|X]: reg $D $X
Learner D1_reg added successfully.
. ddml E[D|X]: cvlasso $D c.($X)##c.($X), lopt postresults
Learner D2_cvlasso added successfully.
. ddml E[D|X], vtype(none): rforest $D $X, type(reg)
Learner D3_rforest added successfully.
```

The flexible `ddml` syntax allows specifying different sets of covariates for different learners. This flexibility can be useful as, for example, linear learners such as the lasso might perform better if, for example, interactions are provided as inputs, whereas tree-based methods such as random forests may detect certain interactions in a data-driven way. Here, we use interactions and second-order polynomials for `cvlasso`, but not for the other learners.

This application has only one treatment variable, but `ddml` does support multiple treatment variables. To add a second treatment variable, we would simply add a statement such as `ddml E[D|X]: reg D2 $X` where `D2` would be the name of the second treatment variable. An example with two treatments is provided in the help file.

The auxiliary `ddml` sub-command `describe` allows to verify that the learners were

correctly registered:

```
. ddml describe
Model:                  partial, crossfit folds k=4, resamples r=1
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_reg Y2_cvlasso Y3_rforest
D equations (1):        e401
 e401 learners:         D1_reg D2_cvlasso D3_rforest
Specifications:         9 possible specs
```

Step 3: Perform cross-fitting.

The third step performs cross-fitting, which is the most time-intensive process. The `shortstack` option enables the short-stacking algorithm of Section 4.1.

```
. ddml crossfit, shortstack
Cross-fitting E[y|X] equation: net_tfa
Cross-fitting fold 1 2 3 4 ...completed cross-fitting...completed short-stacking
Cross-fitting E[D|X] equation: e401
Cross-fitting fold 1 2 3 4 ...completed cross-fitting...completed short-stacking
```

Six variables are created and stored in memory which correspond to the six learners specified in the previous step. These variables are called `Y1_reg_1`, `Y2_cvlasso_1`, `Y3_rforest_1`, `D1_reg_1`, `D2_cvlasso_1` and `D3_rforest_1`. `Y` and `D` indicate outcome and treatment variable. The index `1` to `3` is a learner counter. `reg`, `cvlasso` and `rforest` correspond to the name of the commands used. The `_1` suffix indicates the cross-fitting repetition.

After cross-fitting, we can inspect the mean-squared prediction errors by fold and learner:

```
. ddml desc, crossfit
Model:                  partial, crossfit folds k=4, resamples r=1
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_reg Y2_cvlasso Y3_rforest
D equations (1):        e401
 e401 learners:         D1_reg D2_cvlasso D3_rforest
Specifications:         9 possible specs
Crossfit results (detail):
                                    All    By fold:
Cond. exp.  Learner      rep         MSE        1         2         3         4
net_tfa     Y1_reg         1     3.1e+09   3.0e+09   3.5e+09   4.0e+09   2.0e+09
            Y2_cvlasso     1     2.9e+09   2.5e+09   3.0e+09   4.0e+09   1.9e+09
            Y3_rforest     1     3.3e+09   3.3e+09   3.7e+09   4.0e+09   2.0e+09
            shortstack     1     2.9e+09   2.5e+09   3.0e+09   4.0e+09   1.9e+09
e401        D1_reg         1        0.20      0.21      0.19      0.20      0.20
            D2_cvlasso     1        0.20      0.20      0.19      0.20      0.19
            D3_rforest     1        0.22      0.22      0.21      0.22      0.21
            shortstack     1        0.20      0.20      0.19      0.20      0.19
```

Step 4: Estimate causal effects.

In this final step, we obtain the causal effect estimates. Since we requested short-stacking in Step 3, `ddml` shows short-stacking result which relies on the cross-fitted values of each base learner. In addition, the specification that corresponds to the minimum-MSE learners is listed at the beginning of the output (denoted as `opt`).

```
. ddml estimate, robust
DDML estimation results:
spec  r     Y learner     D learner         b        SE
 opt  1    Y2_cvlasso    D2_cvlasso  9788.185(1343.972)
  ss  1  [shortstack]          [ss]  9739.060(1330.268)
opt = minimum MSE specification for that resample.
Shortstack DDML model
y-E[y|X]  = net_tfa_ss_1                        Number of obs   =      9915
D-E[D|X,Z]= e401_ss_1
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
e401	9739.06	1330.268	7.32	0.000	7131.782	12346.34
_cons	85.81494	534.6987	0.16	0.872	-962.1753	1133.805

Since we have specified three learners per conditional expectation, there are in total 9 specifications relying on the base learners (since we can combine `Y1_reg_1`, `Y2_cvlasso_1` and `Y3_rforest_1` with `D1_reg_1`, `D2_cvlasso_1` and `D3_rforest_1`). To get all results, we add the `allcombos` option:

```
. ddml estimate, robust allcombos
DDML estimation results:
spec  r     Y learner     D learner         b        SE
   1  1       Y1_reg        D1_reg  5986.657(1523.694)
   2  1       Y1_reg    D2_cvlasso  9700.519(1393.963)
   3  1       Y1_reg    D3_rforest  8659.141(1258.982)
   4  1    Y2_cvlasso        D1_reg  9189.396(1370.593)
*  5  1    Y2_cvlasso    D2_cvlasso  9788.185(1343.972)
   6  1    Y2_cvlasso    D3_rforest  8496.965(1199.668)
   7  1    Y3_rforest        D1_reg  9044.071(1485.073)
   8  1    Y3_rforest    D2_cvlasso 10081.930(1430.001)
   9  1    Y3_rforest    D3_rforest  9528.450(1293.485)
  ss  1  [shortstack]          [ss]  9739.060(1330.268)
* = minimum MSE specification for that resample.
Shortstack DDML model
y-E[y|X]  = net_tfa_ss_1                        Number of obs   =      9915
D-E[D|X,Z]= e401_ss_1
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
e401	9739.06	1330.268	7.32	0.000	7131.782	12346.34
_cons	85.81494	534.6987	0.16	0.872	-962.1753	1133.805

We can use the `spec(`*string*`)` option to select among the listed specifications. *string* is either the specification number, `ss` to get the short-stacking specification or `mse` for the specification corresponding to the minimal MSPE. In the example above, `spec(1)` reports in full the specification using `regress` for estimating both $E[Y|\boldsymbol{X}]$ and $E[D|\boldsymbol{X}]$. The `spec(`*string*`)` option can be provided either in combinations with `allcombos`, or after estimation in combination with the `replay` option, for example:

```
. ddml estimate, spec(1) replay
DDML estimation results:
spec  r     Y learner     D learner         b        SE
 opt  1    Y2_cvlasso    D2_cvlasso  9788.185(1343.972)
  ss  1  [shortstack]          [ss]  9739.060(1330.268)
opt = minimum MSE specification for that resample.
DDML model, specification 1
y-E[y|X]  = Y1_reg_1                            Number of obs   =      9915
D-E[D|X,Z]= D1_reg_1
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
e401	5986.657	1523.694	3.93	0.000	3000.271	8973.042
_cons	10.74706	561.2911	0.02	0.985	-1089.363	1110.857

Manual final estimation.

In the background, `ddml estimate` regresses `Y1_reg_1` against `D1_reg_1` with a constant. We can verify this manually:

```
. reg Y1_reg D1_reg,  robust
Linear regression                               Number of obs     =      9,915
                                                F(1, 9913)        =      15.44
                                                Prob > F          =     0.0001
                                                R-squared         =     0.0023
                                                Root MSE          =      55891
```

Y1_reg_1	Coefficient	Robust std. err.	t	P>|t|	[95% conf.	interval]
D1_reg_1	5986.657	1523.694	3.93	0.000	2999.906	8973.407
_cons	10.74706	561.2911	0.02	0.985	-1089.498	1110.992

Manual estimation using `regress` allows the use of postestimation regression tools such as `avplot`.

One-line syntax.

`qddml` provides a simplified and convenient one-line syntax. The main constraint of `qddml` is that it only allows for one user-specified learner. For demonstration, we use

`cvlasso`:

```
. set seed 123
. qddml $Y $D (c.($X)##c.($X)), model(partial) ///
>                               cmd(cvlasso) cmdopt(lopt postresults) ///
>                                                          kfolds(4) ///
>                               robust ///
>                                                          noreg
DDML estimation results:
spec  r     Y learner     D learner         b        SE
 opt  1    Y1_cvlasso    D1_cvlasso  9788.185(1343.972)
opt = minimum MSE specification for that resample.
Min MSE DDML model
y-E[y|X]  = Y1_cvlasso_1                          Number of obs   =      9915
D-E[D|X,Z]= D1_cvlasso_1
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
e401	9788.185	1343.972	7.28	0.000	7154.048	12422.32
_cons	84.7435	534.6942	0.16	0.874	-963.2378	1132.725

`qddml` always includes linear regression (`regress`) as an additional learner. The option `noreg` disables this default behavior.

Using `pystacked` with single learners

The above example relying on `cvlasso` and `rforest` runs relatively slowly at 380 seconds. We can substantially improve the speed by using the lasso and random forests implementation offered by `pystacked`, which calls the Python library *scikit-learn*. This reduces the total computational time to only 29 seconds.

```
. *** add learners for E[Y|X]
. ddml E[Y|X]: reg $Y $X
Learner Y1_reg added successfully.
. ddml E[Y|X]: pystacked $Y c.($X)##c.($X), type(reg) m(lassocv)
Learner Y2_pystacked added successfully.
. ddml E[Y|X]: pystacked $Y $X, type(reg) m(rf)
Learner Y3_pystacked added successfully.
. *** add learners for E[D|X]
. ddml E[D|X]: reg $D $X
Learner D1_reg added successfully.
. ddml E[D|X]: pystacked $D c.($X)##c.($X), type(reg) m(lassocv)
Learner D2_pystacked added successfully.
. ddml E[D|X]: pystacked $D $X, type(reg) m(rf)
Learner D3_pystacked added successfully.
```

Note that, despite its name, `pystacked` does not execute stacking regression in this example, since only one learner is specified in each call to `pystacked`.

Stacking

We next demonstrate DDML with stacking. To this end, we exploit the stacking regressor implemented in `pystacked`. `pystacked` allows to combine multiple base learners with learner-specific settings and covariates into a final meta learner. The learners are separated by `||`. `method(`*name*`)` selects the learner, `xvars(`*varlist*`)` specifies learner-specific covariates (overwritting the default covariates `$X`) and `opt(`*string*`)` passes options to the learners. In this example, we use OLS, cross-validated lasso and ridge, random forests and gradient boosting. We furthermore use parallelization with 5 cores. A detailed explanation of the `pystacked` syntax can be found in Ahrens et al. (2022).

```
. *** add learners for E[Y|X]
. ddml E[Y|X]: pystacked $Y $X                                              || ///
>     method(ols)                                                           || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                       || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                       || ///
>     m(rf) pipe(sparse) opt(max_features(5))                               || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(5)
Learner Y1_pystacked added successfully.
. *** add learners for E[D|X]
. ddml E[D|X]: pystacked $D $X                                              || ///
>     method(ols)                                                           || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                       || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                       || ///
>     m(rf) pipe(sparse) opt(max_features(5))                               || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(5)
Learner D1_pystacked added successfully.
```

After cross-fitting, we can obtain the MSE and stacking weights averaged over folds:

```
. qui ddml crossfit
. ddml extract, show(pystacked)
mean pystacked weights across folds/resamples for D1_pystacked (e401)
              learner  mean_weight
      ols           1    .01244342
  lassocv           2    .11013228
  ridgecv           3    .41958367
       rf           4     .0309631
gradboost           5    .42687753
mean pystacked MSEs across folds/resamples for D1_pystacked (e401)
             learner   mean_MSE
      ols          1  .20041698
  lassocv          2  .19618448
  ridgecv          3  .19647461
       rf          4  .21506883
gradboost          5  .19659169
mean pystacked weights across folds/resamples for Y1_pystacked (net_tfa)
              learner  mean_weight
      ols           1     .1001147
  lassocv           2    .61233119
  ridgecv           3    .16437072
       rf           4    .09763384
gradboost           5    .02910359
```

```
mean pystacked MSEs across folds/resamples for Y1_pystacked (net_tfa)
            learner   mean_MSE
      ols         1  3.342e+09
  lassocv         2  3.081e+09
  ridgecv         3  3.095e+09
       rf         4  3.428e+09
gradboost         5  3.325e+09
```

By adding the `detail` option, the user can display the stacking weights for each fold. Finally, we retrieve the results of DDML with stacking:

```
. ddml estimate, robust

DDML estimation results:
spec  r     Y learner     D learner         b        SE
 opt  1  Y1_pystacked  D1_pystacked  9396.124(1300.402)
opt = minimum MSE specification for that resample.

Min MSE DDML model
y-E[y|X]  = Y1_pystacked_1                      Number of obs   =      9915
D-E[D|X,Z]= D1_pystacked_1
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
e401	9396.124	1300.402	7.23	0.000	6847.384	11944.86
_cons	139.9994	535.8003	0.26	0.794	-910.1499	1190.149

The run time of this example is 83 seconds, which despite its computational complexity is faster than using `ddml` with `cvlasso` and `rforest`.

6.2 The market for automobiles

For this demonstration, we follow Chernozhukov et al. (2015b) who estimate a stylized demand model using instrumental variables based on the data from Berry et al. (1995). The authors of the original study estimate the effect of prices on the market share of automobile models in a given year ($n = 2217$). The controls are product characteristics (a constant, air conditioning dummy, horsepower divided by weight, miles per dollar, vehicle size). To account for endogenous prices, Berry et al. (1995) suggest exploiting other products' characteristics as instruments. Following Chernozhukov et al. (2015b), we define the baseline set of instruments as the sum over all other products' characteristics, calculated separately for own-firm and other-firm products, which yields 10 baseline instruments. Chernozhukov et al. (2015b) also construct an augmented set of instruments, including first-order interactions, squared and cubic terms. In the analysis below, we extend Chernozhukov et al. (2015b) by applying DDML with stacking and a diverse set of learners including OLS, lasso, ridge, random forest and gradient boosted trees. We use the augmented set of controls for all base learners and OLS, which we include for reference.

We load and prepare the data:

```
. use BLP_CHS.dta, clear
. global Y y
. global D price
. global Xbase hpwt air mpd space
. global Xaug augX*
. global Zbase Zbase*
. global Zaug Zaug*
```

Step 1: Initialize `ddml` model.

```
. set seed 123
. ddml init fiv, kfolds(4) reps(5)
```

Note that in the `ddml init` step, we include the option `reps(5)` which will result in running the full cross-fitting procedure five times, each with a different random split of the data. Replicating the procedure multiple times allows us to gauge the impact of randomness due to the random splitting of the data into subsamples.

Step 2: Add supervised machine learners for estimating conditional expectations. Estimation of a `fiv` model requires us to add learners for $E[Y|\boldsymbol{X}]$, $E[D|\boldsymbol{X},\boldsymbol{Z}]$ and $E[D|\boldsymbol{X}]$. Compared to the other models supported by `ddml`, there is one complication that arises because, in order to estimate $E[D|\boldsymbol{X}]$, we exploit fitted values of $E[D|\boldsymbol{X},\boldsymbol{Z}]$ to impose LIE-compliance. Since these fitted values have not yet been generated, we use the placeholder `{D}` that in the cross-fitting stage will be internally replaced with estimates of $E[D|\boldsymbol{X},\boldsymbol{Z}]$. We use the `learner(`*string*`)` option to match one learner for $E[D|\boldsymbol{X}]$ with a learner for $E[D|\boldsymbol{X},\boldsymbol{Z}]$, and `vname(`*varname*`)` to indicate the name of the treatment variable.

```
. *** add learners for E[Y|X]
. ddml E[Y|X], learner(Ypystacked): pystacked $Y $Xaug                  || ///
>     method(ols) xvars($Xbase)                                         || ///
>     m(lassocv)                                                        || ///
>     m(ridgecv)                                                        || ///
>     m(rf) opt(n_estimators(200) max_features(None))                   || ///
>     m(rf) opt(n_estimators(200) max_features(10))                     || ///
>     m(rf) opt(n_estimators(200) max_features(5))                      || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.01))           || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.1))            || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.3))            , ///
>     njobs(4)
Learner Ypystacked added successfully.
. ddml E[D|X,Z], learner(Dpystacked): pystacked $D $Xaug $Zaug          || ///
>     method(ols) xvars($Xbase $Zbase)                                  || ///
>     m(lassocv)                                                        || ///
>     m(ridgecv)                                                        || ///
>     m(rf) opt(n_estimators(200) max_features(None))                   || ///
>     m(rf) opt(n_estimators(200) max_features(10))                     || ///
>     m(rf) opt(n_estimators(200) max_features(5))                      || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.01))           || ///
```

```
>     m(gradboost) opt(n_estimators(800) learning_rate(0.1))             || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.3))             ,  ///
>     njobs(4)
Learner Dpystacked added successfully.
. ddml E[D|X], mname(m0) learner(Dpystacked) vname($D):                   ///
>                                              pystacked {D} $Xaug      || ///
>     method(ols) xvars($Xaug)                                           || ///
>     m(lassocv)                                                         || ///
>     m(ridgecv)                                                         || ///
>     m(rf) opt(n_estimators(200) max_features(None))                    || ///
>     m(rf) opt(n_estimators(200) max_features(10))                      || ///
>     m(rf) opt(n_estimators(200) max_features(5))                       || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.01))            || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.1))             || ///
>     m(gradboost) opt(n_estimators(800) learning_rate(0.3))             ,  ///
>     njobs(4)
Learner Dpystacked_h added successfully.
```

Steps 3-4: Perform cross-fitting (output omitted) and estimate causal effects.

```
. qui ddml crossfit
. ddml estimate, robust
DDML estimation results:
spec  r     Y learner     D learner         b        SE    DH learner
 opt  1    Ypystacked    Dpystacked    -0.108  ( 0.010)  Dpystacked_h
 opt  2    Ypystacked    Dpystacked    -0.123  ( 0.010)  Dpystacked_h
 opt  3    Ypystacked    Dpystacked    -0.110  ( 0.011)  Dpystacked_h
 opt  4    Ypystacked    Dpystacked    -0.127  ( 0.011)  Dpystacked_h
 opt  5    Ypystacked    Dpystacked    -0.127  ( 0.012)  Dpystacked_h
opt = minimum MSE specification for that resample.
Mean/med.   Y learner     D learner         b        SE    DH learner
 mse mn     [min-mse]         [mse]    -0.119  ( 0.013)         [mse]
 mse md     [min-mse]         [mse]    -0.123  ( 0.013)         [mse]
Median over min-mse specifications
y-E[y|X]  = Ypystacked                              Number of obs   =      2217
E[D|X,Z]  = Dpystacked
E[D|X]    = Dpystacked_h
Orthogonalised D = D - E[D|X]; optimal IV = E[D|X,Z] - E[D|X].
```

y	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
price	-.1227476	.0125741	-9.76	0.000	-.1473924	-.0981028

```
Summary over 5 resamples:
       D eqn      mean       min       p25       p50       p75       max
       price   -0.1190   -0.1271   -0.1269   -0.1227   -0.1102   -0.1079
```

Manual final estimation. We can obtain the final estimate manually. To this end, we construct the instrument as $\hat{E}[D|X,Z] - \hat{E}[D|X]$ and the residualized endogenous regressor as $D - \hat{E}[D|X]$. The residualized dependent variable is saved in memory. Here we obtain the estimate from the first cross-fitting replication. We could obtain the estimate for replication r by changing the "_1" to "_r".

```
. gen optiv = Dpystacked_1 - Dpystacked_h_1
. gen dtilde = $D - Dpystacked_h_1
. ivreg Ypystacked_1 (dtilde=optiv), robust
Instrumental variables 2SLS regression          Number of obs     =      2,217
                                                F(1, 2215)        =     110.22
                                                Prob > F          =     0.0000
                                                R-squared         =     0.0888
                                                Root MSE          =     .96503
------------------------------------------------------------------------------
             |               Robust
Ypystacked_1 | Coefficient  std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      dtilde |  -.1078928   .0102771   -10.50   0.000    -.1280465    -.087739
       _cons |    .006487   .0205013     0.32   0.752    -.0337168    .0466908
------------------------------------------------------------------------------
Instrumented: dtilde
 Instruments: optiv
```

One-line syntax (output omitted).

```
. qui qddml $Y ($Xaug) ($D = $Zaug), model(fiv) cmd(pystacked)
```

7 Acknowledgments

We thank users who tested earlier versions of the program. We also thank Jan Ditzen, Ben Jann, Eroll Kuhn and Di Liu for helpful comments, as well as participants at the German Stata Conference 2021, Italian Stata Conference 2022 and Swiss Stata Conference 2022. All remaining errors are our own.

8 References

Ahrens, A., C. B. Hansen, and M. E. Schaffer. 2018. PDSLASSO: Stata module for post-selection and post-regularization OLS or IV estimation and inference. Medium: Statistical Software Components, Boston College Department of Economics. https://ideas.repec.org/c/boc/bocode/s458459.html.

———. 2020. lassopack: Model selection and prediction with regularized regression in Stata. *The Stata Journal* 20(1): 176–235. https://doi.org/10.1177/1536867X20909697.

———. 2022. pystacked: Stacking generalization and machine learning in Stata. https://arxiv.org/abs/2208.10896.

Angrist, J. D., K. Graddy, and G. W. Imbens. 2000. The interpretation of instrumental variables estimators in simultaneous equations models with an application to the demand for fish. *The Review of Economic Studies* 67(3): 499–527.

Angrist, J. D., and A. B. Krueger. 1999. Empirical strategies in labor economics. In *Handbook of Labor Economics*, vol. 3, 1277–1366. Elsevier.

Athey, S., J. Tibshirani, and S. Wager. 2019. Generalized random forests. *Annals of Statistics* 47(2): 1148–1178. https://doi.org/10.1214/18-AOS1709.

Belloni, A., D. Chen, V. Chernozhukov, and C. Hansen. 2012. Sparse Models and Methods for Optimal Instruments With an Application to Eminent Domain. *Econometrica* 80(6): 2369–2429. Publisher: Blackwell Publishing Ltd. http://dx.doi.org/10.3982/ECTA9626.

Belloni, A., V. Chernozhukov, I. Fernández-Val, and C. Hansen. 2017. Program Evaluation and Causal Inference With High-Dimensional Data. *Econometrica* 85(1): 233–298. https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA12723.

Belloni, A., V. Chernozhukov, and C. Hansen. 2014. Inference on treatment effects after selection among high-dimensional controls. *Review of Economic Studies* 81: 608–650. https://doi.org/10.1093/restud/rdt044.

Berry, S., J. Levinsohn, and A. Pakes. 1995. Automobile prices in market equilibrium. *Econometrica: Journal of the Econometric Society* 841–890.

Bickel, P. J., Y. Ritov, and A. B. Tsybakov. 2009. Simultaneous analysis of Lasso and Dantzig selector. *Annals of statistics* 37(4): 1705–1732.

Blandhol, C., J. Bonney, M. Mogstad, and A. Torgovitsky. 2022. When is TSLS Actually LATE? *BFI Working Paper* (2022-16).

Breiman, L. 1996. Stacked regressions. *Machine Learning* 24(1): 49–64. http://link.springer.com/10.1007/BF00117832.

Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108–122.

Chang, C.-C., and C.-J. Lin. 2011. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* 2(3): 1–27.

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. Hansen, W. Newey, and J. Robins. 2018. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal* 21(1): C1–C68. Tex.ids= Chernozhukov2018a publisher: John Wiley & Sons, Ltd (10.1111). https://onlinelibrary.wiley.com/doi/abs/10.1111/ectj.12097.

Chernozhukov, V., C. Hansen, and M. Spindler. 2015a. Post-Selection and Post-Regularization Inference in Linear Models with Many Controls and Instruments. *American Economic Review* 105(5): 486–490. https://doi.org/10.1257/aer.p20151022.

———. 2015b. Valid Post-Selection and Post-Regularization Inference: An Elementary, General Approach. *Annual Review of Economics* 7(1). ArXiv: 1501.03430.

———. 2016. High-dimensional metrics in R 401: 1–32.

Dhar, D., T. Jain, and S. Jayachandran. 2022. Reshaping adolescents' gender attitudes: Evidence from a school-based experiment in India. *American Economic Review* 112(3): 899–927.

Farrell, M. H., T. Liang, and S. Misra. 2021. Deep neural networks for estimation and inference. *Econometrica* 89(1): 181–213.

Frank, E., M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. H. Witten, and L. Trigg. 2009. Weka-a machine learning workbench for data mining. In *Data mining and knowledge discovery handbook*, 1269–1277. Springer.

Giannone, D., M. Lenza, and G. E. Primiceri. 2021. Economic predictions with big data: The illusion of sparsity. *Econometrica* 89(5): 2409–2437.

Gilchrist, D. S., and E. G. Sands. 2016. Something to talk about: Social spillovers in movie consumption. *Journal of Political Economy* 124(5): 1339–1382.

Guenther, N., and M. Schonlau. 2018. SVMACHINES: Stata module providing Support Vector Machines for both Classification and Regression. Statistical Software Components, Boston College Department of Economics. https://ideas.repec.org/c/boc/bocode/s458564.html.

Haghish, E. 2019. Seamless interactive language interfacing between R and Stata. *The Stata Journal* 19(1): 61–82.

Hahn, J. 1998. On the role of the propensity score in efficient semiparametric estimation of average treatment effects. *Econometrica* 66(2): 315–331.

Hangartner, D., D. Kopp, and M. Siegenthaler. 2021. Monitoring hiring discrimination through online recruitment platforms. *Nature* 589(7843): 572–576. https://doi.org/10.1038/s41586-020-03136-0.

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning*. 2nd ed. New York: Springer-Verlag.

Heckman, J. J., and E. J. Vytlacil. 2007. Econometric evaluation of social programs, part I: Causal models, structural models and econometric policy evaluation. In *Handbook of Econometrics*, ed. J. J. Heckman and E. Leamer, vol. 6, chap. 70, 4779–4874. Amsterdam: Elsevier.

Huntington-Klein, N. C. 2021. mlrtime. https://github.com/NickCH-K/MLRtime/. [Online; accessed 02-December-2021].

Imbens, G. W., and J. D. Angrist. 1994. Identification and Estimation of Local Average Treatment Effects. *Econometrica* 62(2): 467–475.

Luo, Y., M. Spindler, and J. Kück. 2022. High-Dimensional L_2 Boosting: Rate of Convergence. *arXiv preprint arXiv:1602.08927* .

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12: 2825–2830.

Poterba, J. M., S. F. Venti, and D. A. Wise. 1995. Do 401 (k) contributions crowd out other personal saving? *Journal of Public Economics* 58(1): 1–32.

Roberts, M. E., B. M. Stewart, and R. A. Nielsen. 2020. Adjusting for Confounding with Text Matching. *American Journal of Political Science* 64(4): 887–903. https://onlinelibrary.wiley.com/doi/10.1111/ajps.12526.

Schmidt-Hieber, J. 2020. Nonparametric regression using deep neural networks with ReLU activation function. *Annals of Statistics* 48(4): 1875–1897.

Schonlau, M., and R. Y. Zou. 2020. The random forest algorithm for statistical learning. *The Stata Journal* 20(1): 3–29. https://doi.org/10.1177/1536867X20909688.

StataCorp. 2019. Stata Statistical Software: Release 16.

———. 2021. Stata Statistical Software: Release 17.

Wager, S., and S. Athey. 2018. Estimation and Inference of Heterogeneous Treatment Effects using Random Forests. *Journal of the American Statistical Association* 113(523): 1228–1242. Publisher: Taylor & Francis. https://doi.org/10.1080/01621459.2017.1319839.

Wager, S., and G. Walther. 2016. Adaptive concentration of regression trees, with application to random forests. *arXiv preprint arXiv:1503.06388* .

Wolpert, D. H. 1992. Stacked generalization. *Neural Networks* 5(2): 241–259. https://www.sciencedirect.com/science/article/pii/S0893608005800231.

Wüthrich, K., and Y. Zhu. 2021. Omitted variable bias of Lasso-based inference methods: A finite sample analysis. *Review of Economics and Statistics* 0((0)): 1–47.

About the authors

Achim Ahrens is Post-Doctoral Researcher and Senior Data Scientist at the Public Policy Group and Immigration Policy Lab, ETH Zürich.

Christian B. Hansen is the Wallace W. Booth Professor of Econometrics and Statistics at the University of Chicago Booth School of Business.

Mark E. Schaffer is Professor of Econonomics in Edinburgh Business School at Heriot-Watt University, Edinburgh, UK, and a Research Fellow at the Institute for the Study of Labour (IZA), Bonn.

Thomas Wiemann is an economics PhD student at the University of Chicago.

A Algorithms

❏ **Algorithm A.1. DDML for the Interactive Model.**

Split the sample $\{(Y_i, D_i, \boldsymbol{X}_i)\}_{i \in I}$ with $I = \{1, \ldots, n\}$ in K folds of approximately equal size. Denote I_k the set of observations included in fold k and $I_k^c = I \setminus I_k$ its complement.

1. Estimate conditional expectations. For each k:
   a. Fit the CEF estimator to observations in the sub-sample I_k^c for which $D_i = 1$ using Y_i as the outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $g_0(1, \boldsymbol{X}) = E[Y|\boldsymbol{X}, D = 1]$. Obtain the out-of-sample predicted values $\hat{g}_{I_{k_i}^c}(1, \boldsymbol{X}_i)$ for $i \in I_k$. Proceed in the same way to obtain $\hat{g}(0, \boldsymbol{X})$.
   b. For each k, fit the CEF estimator to the sub-sample I_k^c using D_i as outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $m(\boldsymbol{X}) = E[D|\boldsymbol{X}]$. Obtain the out-of-sample predicted values $\hat{m}_{I_{k_i}^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
2. Compute the ATE and ATET using (7) and (8).

❏

❏ **Algorithm A.2. DDML for the Partially Linear IV Model.**

Split the sample $\{(Y_i, D_i, \boldsymbol{X}_i)\}_{i \in I}$ with $I = \{1, \ldots, n\}$ in K folds of approximately equal size. Denote I_k the set of observations included in fold k and $I_k^c = I \setminus I_k$ its complement.

1. Estimate conditional expectations. For each k:
   a. Fit the CEF estimator to the sub-sample I_k^c using Y_i as outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $\ell_0(\boldsymbol{X}) = E[Y|\boldsymbol{X}]$. Obtain the out-of-sample predicted values $\hat{\ell}_{I_{k_i}^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
   b. Fit the CEF estimator to the sub-sample I_k^c using D_i as outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $m_0(\boldsymbol{X}) = E[D|\boldsymbol{X}]$. Obtain the out-of-sample predicted values $\hat{m}_{I_{k_i}^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
   c. Fit the CEF estimator to the sub-sample I_k^c using Z_i as outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $r_0(\boldsymbol{X}) = E[Z|\boldsymbol{X}]$. Obtain the out-of-sample predicted values $\hat{r}_{I_{k_i}^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
2. Compute (10).

❏

❏ **Algorithm A.3. DDML for the Interactive IV Model.**

Split the sample $\{(Y_i, D_i, \boldsymbol{X}_i)\}_{i \in I}$ with $I = \{1, \ldots, n\}$ in K folds of approximately equal size. Denote I_k the set of observations included in fold k and $I_k^c = I \setminus I_k$ its complement.

1. Estimate conditional expectations. For each k:

a. Fit the CEF estimator to observations in the sub-sample I_k^c for which $Z_i = 1$ using Y_i as outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $\ell_0(1, \boldsymbol{X}) = E[Y|\boldsymbol{X}, Z = 1]$. Obtain the out-of-sample predicted values $\hat{\ell}_{I_{k_i}^c}(1, \boldsymbol{X}_i)$ for $i \in I_k$. Proceed in the same way for the estimation of $\ell_0(0, \boldsymbol{X})$.

b. Fit the CEF estimator to observations in the sub-sample I_k^c for which $Z_i = 1$ using D_i as outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $p_0(1, \boldsymbol{X}) = \Pr(D = 1|\boldsymbol{X}, Z = 1)$. Obtain the out-of-sample predicted values $\hat{p}_{I_{k_i}^c}(1, \boldsymbol{X}_i)$ for $i \in I_k$. Proceed in the same way for the estimation of $p_0(0, \boldsymbol{X})$.

c. Fit the CEF estimator to the sub-sample I_k^c using D_i as outcome and $\boldsymbol{X}_i$ as predictors to estimate the conditional expectation $r(\boldsymbol{X}) = E[Z|\boldsymbol{X}]$. Obtain the out-of-sample predicted values $\hat{r}_{I_{k_i}^c}(\boldsymbol{X}_i)$ for $i \in I_k$.

2. Compute (17).

❑

❑ **Algorithm A.4. DDML with short-stacking for the Partially Linear Model.**

Split the sample $\{(Y_i, D_i, \boldsymbol{X}_i)\}_{i \in I}$ with $I = \{1, \ldots, n\}$ in K folds of approximately equal size. Denote I_k the set of observations included in fold k and $I_k^c = I \setminus I_k$ its complement. Select a set of J base learners with $J \geq 2$.

1. Estimate conditional expectations. For each k and base learner j:

a. Fit a CEF estimator j to the sub-sample I_k^c using Y_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\hat{\ell}_{I_k^c}^{(j)}(\boldsymbol{X}_i)$ for $i \in I_k$.

b. Fit a CEF estimator j to the sub-sample I_k^c using D_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\hat{m}_{I_k^c}^{(j)}(\boldsymbol{X}_i)$ for $i \in I_k$.

2. Short-stacking:

a. Apply constrained regression of Y_i against $\hat{\ell}_{I_k^c}^{(1)}(\boldsymbol{X}_i), \ldots, \hat{\ell}_{I_k^c}^{(J)}(\boldsymbol{X}_i)$ using the full sample I, which yields the short-stacked predicted values $\hat{\ell}^*(\boldsymbol{X}_i)$.

b. Apply constrained regression of D_i against $\hat{m}_{I_k^c}^{(1)}(\boldsymbol{X}_i), \ldots, \hat{m}_{I_k^c}^{(J)}(\boldsymbol{X}_i)$ using the full sample I, which yields the short-stacked predicted values $\hat{m}^*(\boldsymbol{X}_i)$.

3. Compute the short-stacked DDML esitmator using

$$\hat{\theta}_n = \frac{\frac{1}{n}\sum_{i=1}^n \big(Y_i - \hat{\ell}^*(\boldsymbol{X}_i)\big)\big(D_i - \hat{m}^*(\boldsymbol{X}_i)\big)}{\frac{1}{n}\sum_{i=i}^n \big(D_i - \hat{m}^*(\boldsymbol{X}_i)\big)^2}.$$

❑

❑ **Algorithm A.5. DDML with short-stacking for the Flexible IV Model.**

Split the sample $\{(Y_i, D_i, \boldsymbol{X}_i)\}_{i \in I}$ with $I = \{1, \ldots, n\}$ in K folds of approximately equal size. Denote I_k the set of observations included in fold k and $I_k^c = I \setminus I_k$ its complement. Select a set of J base learners with $J \geq 2$.

1. Estimating conditional expectations $\ell_0(\boldsymbol{X}) = E[Y|\boldsymbol{X}]$:
   a. For each k and base learner j, fit the CEF estimator to the sub-sample I_k^c using Y_i as outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\hat{\ell}_{I_k^c}^{(j)}(\boldsymbol{X}_i)$ for $i \in I_k$.
   b. Fit a constrained regression of Y_i against $\hat{\ell}_{I_k^c}^{(j)}(\boldsymbol{X}_i), \ldots, \hat{\ell}_{I_k^c}^{(J)}(\boldsymbol{X}_i)$ over the full sample I. The fitted values are the short-stacking estimates $\hat{\ell}^{\star}(\boldsymbol{X}_i)$.
2. Estimating conditional expectations $p_0(\boldsymbol{X}, \boldsymbol{Z}) = E[D|\boldsymbol{X}, \boldsymbol{Z}]$:
   a. For each k and base learner j, fit the CEF estimator to the sub-sample I_k^c using D_i as outcome and $(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ as predictors. Obtain the out-of-sample predicted values $\hat{p}_{I_k^c}^{(j)}(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ for $i \in I_k$, and the in-sample predicted values $\tilde{p}_k^{(j)}(\boldsymbol{X}_i, \boldsymbol{Z}_i) \equiv \hat{p}_{I_k^c}^{(j)}(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ for $i \in I_k^c$.
   b. For each k, fit a constrained regression of D_i against in-sample predicted values $\tilde{p}_k^{(1)}(\boldsymbol{X}_i, \boldsymbol{Z}_i), \ldots, \tilde{p}_k^{(J)}(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ over the sample I_k^c to obtain the out-of-sample short-stack predicted values $\hat{p}_{I_k^c}^{\star}(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ for $i \in I_k$.
   c. Fit a constrained regression of D_i against $\hat{p}_{I_k^c}^{(1)}(\boldsymbol{X}_i, \boldsymbol{Z}_i), \ldots, \hat{p}_{I_k^c}^{(J)}(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ over the full sample I. The fitted values are the short-stacking estimates $\hat{p}^{\star}(\boldsymbol{Z}_i, \boldsymbol{X}_i)$.
3. Estimating conditional expectations $m_0(\boldsymbol{X}) = E[D|\boldsymbol{X}]$:
   a. For each k and base learner j, fit the CEF estimator to the sub-sample I_k^c using in-sample fitted values $\tilde{p}_k^{(j)}(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\hat{m}_{I_k^c}^{(j)}(\boldsymbol{X}_i)$ for $i \in I_k$.
   b. Apply a constrained regression of $\hat{p}_{I_k^c}^{\star}(\boldsymbol{X}_i, \boldsymbol{Z}_i)$ against $\hat{m}_{I_k^c}^{(1)}(\boldsymbol{X}_i), \ldots, \hat{m}_{I_k^c}^{(J)}(\boldsymbol{X}_i)$ over the full sample I. The fitted values are the short-stacking estimates $\hat{m}^{\star}(\boldsymbol{X}_i)$.
4. Compute

$$\hat{\theta}_n = \frac{\frac{1}{n}\sum_{i=1}^{n}\big(Y_i - \hat{\ell}^{\star}(\boldsymbol{X}_i)\big)\big(\hat{p}^{\star}(\boldsymbol{Z}_i, \boldsymbol{X}_i) - \hat{m}^{\star}(\boldsymbol{X}_i)\big)}{\frac{1}{n}\sum_{i=i}^{n}\big(D_i - \hat{m}^{\star}(\boldsymbol{X}_i)\big)\big(\hat{p}^{\star}(\boldsymbol{Z}_i, \boldsymbol{X}_i) - \hat{m}^{\star}(\boldsymbol{X}_i)\big)}. \tag{20}$$

❑

B Additional simulation results

We briefly summarize the results for DGPs 4 and 5 shown in Table B.1. The stacking weights and MSPEs of the individual learners are reported in Tables B.2 and B.3, respectively.

The bias of DDML with stacking is relatively robust to the inclusion of additional noisy covariates. For $n = 100$, DDML with stacking performs at least as well as feasible full-sample estimators. For $n = 1000$, DDML with stacking outperforms OLS and PDS-Lasso, and exhibits a bias that is only slightly above the infeasible oracle estimator.

Table B.1: Bias and Coverage Rates in the Linear and Nonlinear DGPs

	DGP 4				DGP 5			
	$n = 100$		$n = 1000$		$n = 100$		$n = 1000$	
	MAB	Cov.	MAB	Cov.	MAB	Cov.	MAB	Cov.
Full sample:								
Oracle	0.104	0.903	0.031	0.953	0.109	0.912	0.035	0.945
OLS (Base)	0.137	0.847	0.076	0.613	0.118	0.859	0.080	0.602
PDS-Lasso (Base)	0.116	0.839	0.077	0.624	0.120	0.837	0.080	0.605
PDS-Lasso (Poly 5)	0.116	0.811	0.046	0.835	0.116	0.833	0.044	0.851
DDML methods:								
Base learners								
PDS-Lasso (Poly 2 + Inter.)	0.123	0.799	0.047	0.818	0.118	0.840	0.038	0.899
OLS	0.145	0.637	0.078	0.596	0.120	0.827	0.080	0.605
Lasso with CV (Base)	0.113	0.829	0.080	0.614	0.121	0.841	0.081	0.605
Ridge with CV (Base)	0.154	0.730	0.078	0.597	0.113	0.855	0.080	0.605
Lasso with CV (Poly 5)	0.118	0.838	0.037	0.902	0.130	0.846	0.039	0.920
Ridge with CV (Poly 5)	0.161	0.667	0.102	0.403	0.139	0.796	0.044	0.873
Lasso with CV (Poly 2 + Inter.)	0.176	0.811	0.094	0.598	0.121	0.872	0.038	0.921
Ridge with CV (Poly 2 + Inter.)	0.188	0.547	0.100	0.265	0.111	0.886	0.035	0.932
Random forest (Low)	0.187	0.639	0.118	0.304	0.116	0.852	0.048	0.816
Random forest (Medium)	0.191	0.628	0.140	0.138	0.119	0.845	0.068	0.682
Random forest (High)	0.232	0.519	0.223	0.002	0.167	0.729	0.169	0.044
Gradient boosting (Low)	0.096	0.897	0.031	0.939	0.107	0.886	0.038	0.936
Gradient boosting (Medium)	0.120	0.850	0.049	0.853	0.106	0.898	0.042	0.866
Gradient boosting (High)	0.200	0.586	0.124	0.230	0.163	0.746	0.115	0.319
Neural net	0.100	0.879	0.125	0.177	0.119	0.868	0.036	0.932
Meta learners								
Stacking: CLS	0.115	0.850	0.036	0.902	0.115	0.860	0.037	0.932
Stacking: Single best	0.112	0.851	0.036	0.906	0.117	0.873	0.036	0.929
Short-stacking: CLS	0.117	0.867	0.037	0.903	0.106	0.882	0.037	0.934
Short-stacking: Single best	0.107	0.867	0.035	0.911	0.109	0.884	0.037	0.939

Notes: The table reports median absolute bias (MAB) and coverage rate of a 95% confidence interval (CR). We employ standard errors robust to heteroskedasticity. For comparison, we report the following full sample estimators: infeasible Oracle, OLS, PDS-Lasso with base and two different expanded sets of covariates. DDML estimators use 20 folds for cross-fitting if $n = 100$, and 5 folds if $n = 1000$. Meta-learning approaches rely on all listed base learners. Results are based on 1,000 replications. Results for DGPs 1-3 can be found in Table 1 in the main text.

Table B.2: Stacking weights for the estimation of conditional expectation functions

	DGP 1		DGP 2		DGP 3		DGP 4		DGP 5	
Observations n	100	1000	100	1000	100	1000	100	1000	100	1000
Estimation of $E[Y\|\boldsymbol{X}]$:										
OLS	0.031	0.120	0.018	0.	0.034	0.001	0.045	0.015	0.154	0.012
Lasso with CV (Base)	0.183	0.632	0.015	0.	0.212	0.	0.422	0.017	0.131	0.014
Ridge with CV (Base)	0.347	0.039	0.002	0.	0.016	0.	0.027	0.013	0.052	0.013
Lasso with CV (Poly 5)	0.052	0.058	0.221	0.041	0.061	0.	0.134	0.553	0.097	0.274
Ridge with CV (Poly 5)	0.044	0.044	0.116	0.007	0.014	0.	0.025	0.001	0.077	0.052
Lasso with CV (Poly 2 + Inter.)	0.039	0.025	0.227	0.944	0.065	0.009	0.080	0.307	0.087	0.344
Ridge with CV (Poly 2 + Inter.)	0.096	0.011	0.060	0.001	0.038	0.	0.054	0.001	0.043	0.094
Random forest (Low regularization)	0.033	0.005	0.100	0.	0.086	0.062	0.028	0.	0.070	0.054
Random forest (Medium regularization)	0.018	0.	0.064	0.	0.077	0.	0.024	0.	0.043	0.003
Random forest (High regularization)	0.001	0.	0.015	0.	0.022	0.	0.001	0.	0.013	0.
Gradient boosting (Low regularization)	0.057	0.023	0.054	0.002	0.186	0.156	0.077	0.049	0.053	0.031
Gradient boosting (Medium regularization)	0.020	0.012	0.041	0.	0.119	0.689	0.037	0.020	0.022	0.013
Gradient boosting (High regularization)	0.005	0.	0.021	0.	0.026	0.071	0.003	0.	0.007	0.
Neural net	0.072	0.029	0.045	0.004	0.045	0.012	0.044	0.024	0.150	0.095
Estimation of $E[D\|\boldsymbol{X}]$:										
OLS	0.026	0.112	0.017	0.	0.033	0.001	0.044	0.014	0.179	0.011
Lasso with CV (Base)	0.168	0.649	0.015	0.	0.183	0.	0.420	0.015	0.117	0.014
Ridge with CV (Base)	0.360	0.031	0.003	0.	0.015	0.	0.024	0.013	0.053	0.011
Lasso with CV (Poly 5)	0.055	0.061	0.222	0.044	0.066	0.	0.142	0.558	0.095	0.282
Ridge with CV (Poly 5)	0.047	0.045	0.124	0.005	0.015	0.	0.021	0.001	0.076	0.054
Lasso with CV (Poly 2 + Inter.)	0.037	0.021	0.230	0.945	0.071	0.007	0.078	0.301	0.090	0.332
Ridge with CV (Poly 2 + Inter.)	0.094	0.012	0.071	0.001	0.036	0.	0.060	0.001	0.043	0.096
Random forest (Low regularization)	0.034	0.005	0.078	0.	0.090	0.046	0.021	0.001	0.074	0.050
Random forest (Medium regularization)	0.025	0.	0.054	0.	0.077	0.	0.027	0.	0.038	0.002
Random forest (High regularization)	0.003	0.	0.022	0.	0.027	0.	0.003	0.	0.017	0.
Gradient boosting (Low regularization)	0.055	0.024	0.056	0.002	0.191	0.167	0.076	0.050	0.055	0.034
Gradient boosting (Medium regularization)	0.021	0.010	0.042	0.	0.132	0.735	0.033	0.022	0.022	0.011
Gradient boosting (High regularization)	0.005	0.	0.021	0.	0.020	0.033	0.003	0.	0.006	0.
Neural net	0.068	0.030	0.045	0.003	0.043	0.011	0.047	0.024	0.134	0.104

Notes: This table shows the stacking weights averaged over bootstrap and cross-fitting iterations for each base learner. We report the stacking weights for the estimation of $E[Y|\boldsymbol{X}]$ and $E[D|\boldsymbol{X}]$, and for sample sizes of $n = 100$ and $n = 1000$.

Table B.3: Mean-squared prediction error the estimation of conditional expectation functions

	DGP 1		DGP 2		DGP 3		DGP 4		DGP 5	
Observations n	100	1000	100	1000	100	1000	100	1000	100	1000
Estimation of $E[Y\|\boldsymbol{X}]$*:*										
OLS	2.724	1.336	5.162	2.548	4.138	2.020	3.117	1.533	1.610	1.453
Lasso with CV (Base)	1.615	1.304	2.519	2.401	2.087	1.916	1.668	1.457	1.590	1.451
Ridge with CV (Base)	1.551	1.345	2.514	2.442	2.278	2.017	1.881	1.533	1.641	1.453
Lasso with CV (Poly 5)	2.149	1.346	2.526	1.582	2.293	1.875	1.960	1.361	1.791	1.343
Ridge with CV (Poly 5)	2.183	1.450	2.449	1.713	2.517	2.087	2.422	1.613	1.832	1.479
Lasso with CV (Poly 2 + Inter.)	3.049	1.824	3.131	1.906	2.874	2.358	2.998	2.038	1.748	1.370
Ridge with CV (Poly 2 + Inter.)	2.047	1.370	2.232	1.364	2.247	1.809	1.905	1.369	1.627	1.330
Random forest (Low regularization)	1.854	1.529	2.247	1.905	2.062	1.563	1.934	1.587	1.641	1.428
Random forest (Medium regularization)	1.873	1.611	2.254	2.016	2.064	1.644	1.946	1.651	1.649	1.461
Random forest (High regularization)	2.082	1.981	2.335	2.264	2.169	2.008	2.106	1.979	1.808	1.758
Gradient boosting (Low regularization)	2.173	1.638	2.654	1.951	2.262	1.489	2.128	1.588	1.935	1.529
Gradient boosting (Medium regularization)	2.123	1.563	2.468	1.940	2.110	1.438	2.082	1.531	1.894	1.495
Gradient boosting (High regularization)	2.311	1.805	2.450	2.287	2.189	1.497	2.246	1.683	2.073	1.651
Neural net	2.277	2.075	3.187	2.702	3.110	2.917	2.578	2.304	1.674	1.409
Stacking: CLS	1.631	1.312	2.210	1.367	2.049	1.446	1.763	1.362	1.594	1.330
Stacking: Single-best	1.678	1.306	2.322	1.364	2.111	1.442	1.806	1.364	1.698	1.338
Estimation of $E[D\|\boldsymbol{X}]$*:*										
OLS	2.175	1.067	4.249	2.080	3.317	1.617	2.540	1.230	1.276	1.163
Lasso with CV (Base)	1.287	1.042	2.073	1.962	1.672	1.534	1.352	1.169	1.262	1.161
Ridge with CV (Base)	1.235	1.076	2.071	1.996	1.828	1.614	1.523	1.230	1.304	1.163
Lasso with CV (Poly 5)	1.705	1.072	2.048	1.272	1.830	1.500	1.569	1.091	1.536	1.073
Ridge with CV (Poly 5)	1.757	1.155	2.012	1.381	2.015	1.668	1.973	1.294	1.459	1.176
Lasso with CV (Poly 2 + Inter.)	1.606	1.094	1.825	1.093	1.791	1.448	1.539	1.097	1.297	1.063
Ridge with CV (Poly 2 + Inter.)	2.436	1.450	2.604	1.533	2.341	1.885	2.403	1.635	1.394	1.095
Random forest (Low regularization)	1.459	1.215	1.855	1.550	1.649	1.249	1.571	1.274	1.315	1.144
Random forest (Medium regularization)	1.470	1.274	1.859	1.640	1.649	1.311	1.578	1.323	1.320	1.169
Random forest (High regularization)	1.628	1.561	1.920	1.847	1.731	1.604	1.706	1.590	1.441	1.408
Gradient boosting (Low regularization)	1.686	1.292	2.187	1.574	1.779	1.178	1.713	1.265	1.553	1.217
Gradient boosting (Medium regularization)	1.658	1.240	2.034	1.573	1.672	1.141	1.669	1.225	1.510	1.198
Gradient boosting (High regularization)	1.821	1.465	2.019	1.882	1.747	1.205	1.831	1.373	1.670	1.345
Neural net	1.831	1.673	2.665	2.221	2.519	2.365	2.139	1.861	1.347	1.129
Stacking: CLS	1.297	1.048	1.824	1.096	1.636	1.146	1.410	1.091	1.270	1.062
Stacking: Single-best	1.374	1.043	1.913	1.093	1.692	1.142	1.477	1.094	1.348	1.069

Notes: This table shows the cross-fitted mean-squared prediction error averaged over bootstrap iterations for the listed conditional expectation function estimators. We report the mean-squared predictor error for the estimation of $E[Y|\boldsymbol{X}]$ and $E[D|\boldsymbol{X}]$, and for sample sizes of $n = 100$ and $n = 1000$.

C Applications

Here we continue the 401(k) application from the main text to illustrate estimation of the interactive model and IV models. We use the same data and variables as outlined in the main text. For the IV models, we use eligibility to enroll for the 401(k) pension plan as the instrument and treat participation in a 401(k) as the endogenous variable.

C.1 Interactive Model (`interactive`)

We allow for heterogenous treatment effects using the `interactive` model. To this end, the conditional expectation of `$Y` given `$X` is fit separately for $D = 1$ and $D = 0$. We also use `reps(5)`. This will execute the `ddml` estimation three times using different random folds. This reduces dependence on a specific fold.

```
. *** initialize
. set seed 123
. ddml init interactive, kfolds(5) reps(5)


. *** add learners for E[Y|X,D=0] and E[Y|X,D=0]
. ddml E[Y|X,D]: pystacked $Y $X                                          || ///
>     method(ols)                                                         || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                     || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                     || ///
>     m(rf) pipe(sparse) opt(max_features(5))                             || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(5)
Learner Y1_pystacked added successfully.
. *** add learners for E[D|X]
. ddml E[D|X]: pystacked $D $X                                            || ///
>     method(ols)                                                         || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                     || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                     || ///
>     m(rf) pipe(sparse) opt(max_features(5))                             || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(5)
Learner D1_pystacked added successfully.


. ddml estimate
DDML estimation results (ATE):
spec  r    Y0 learner    Y1 learner     D learner         b        SE
 opt  1  Y1_pystacked  Y1_pystacked  D1_pystacked  8026.896(1126.459)
 opt  2  Y1_pystacked  Y1_pystacked  D1_pystacked  7879.894(1122.244)
 opt  3  Y1_pystacked  Y1_pystacked  D1_pystacked  8049.652(1119.661)
 opt  4  Y1_pystacked  Y1_pystacked  D1_pystacked  8157.735(1113.299)
 opt  5  Y1_pystacked  Y1_pystacked  D1_pystacked  7753.944(1138.377)
opt = minimum MSE specification for that resample.
Mean/med.  Y0 learner    Y1 learner     D learner         b        SE
 mse mn     [min-mse]         [mse]         [mse]  7973.624(1132.547)
 mse md     [min-mse]         [mse]         [mse]  8026.896(1126.459)
Median over 5 min-mse specifications (ATE)
E[y|X,D=0]   = Y1_pystacked                        Number of obs   =      9915
E[y|X,D=1]   = Y1_pystacked
```

```
E[D|X]       = D1_pystacked
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
e401	8026.896	1126.459	7.13	0.000	5819.077	10234.72

```
Warning: 5 resamples had propensity scores trimmed to lower limit .01.
Summary over 5 resamples:
       D eqn      mean       min       p25       p50       p75       max
        e401   7973.6244 7753.9438 7879.8940 8026.8965 8049.6523 8157.7354
```

One-line syntax (output omitted).

```
. qui qddml $Y $D ($X), model(interactive) cmd(pystacked)
```

C.2 IV model (`iv`)

```
. use "sipp1991.dta", clear
. global Y net_tfa
. global X age inc educ fsize marr twoearn db pira hown
. global Z e401
. global D p401
```

Step 1: Initialize `ddml` model.

```
. set seed 123
. ddml init iv
```

Step 2: Add supervised machine learners for estimating conditional expectations.

```
. *** E[Y|X]
. ddml E[Y|X]: pystacked $Y $X                                              || ///
>     method(ols)                                                           || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                       || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                       || ///
>     m(rf) pipe(sparse) opt(max_features(5))                               || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(4)
Learner Y1_pystacked added successfully.
. *** E[D|X]
. ddml E[D|X]: pystacked $D $X                                              || ///
>     method(ols)                                                           || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                       || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                       || ///
>     m(rf) pipe(sparse) opt(max_features(5))                               || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
```

```
>     njobs(4)
Learner D1_pystacked added successfully.
. *** E[Z|X]
. ddml E[Z|X]: pystacked $Z $X                                          || ///
>     method(ols)                                                       || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                   || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                   || ///
>     m(rf) pipe(sparse) opt(max_features(5))                           || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(4)
Learner Z1_pystacked added successfully.
```

Step 3: Perform cross-fitting.

```
. ddml crossfit
Cross-fitting E[y|X] equation: net_tfa
Cross-fitting fold 1 2 3 4 5 ...completed cross-fitting
Cross-fitting E[D|X] equation: p401
Cross-fitting fold 1 2 3 4 5 ...completed cross-fitting
Cross-fitting E[Z|X]: e401
Cross-fitting fold 1 2 3 4 5 ...completed cross-fitting
. ddml extract, show(pystacked)
mean pystacked weights across folds/resamples for Z1_pystacked (e401)
             learner  mean_weight
      ols          1    .02435374
  lassocv          2    5.963e-19
  ridgecv          3    .48160664
       rf          4    .03685229
gradboost          5    .45718733
mean pystacked MSEs across folds/resamples for Z1_pystacked (e401)
            learner   mean_MSE
      ols         1  .20042018
  lassocv         2  .19661349
  ridgecv         3  .19661916
       rf         4  .21495839
gradboost         5  .19662694
mean pystacked weights across folds/resamples for Y1_pystacked (net_tfa)
             learner  mean_weight
      ols          1     .0948191
  lassocv          2    .13620045
  ridgecv          3    .68323138
       rf          4    .07307122
gradboost          5    .00888029
mean pystacked MSEs across folds/resamples for Y1_pystacked (net_tfa)
            learner   mean_MSE
      ols         1  3.256e+09
  lassocv         2  2.991e+09
  ridgecv         3  2.988e+09
       rf         4  3.418e+09
gradboost         5  3.280e+09
mean pystacked weights across folds/resamples for D1_pystacked (p401)
             learner  mean_weight
      ols          1    .06282466
  lassocv          2    .13679104
  ridgecv          3     .4428182
```

```
       rf            4    .06142842
gradboost            5    .29613768
mean pystacked MSEs across folds/resamples for D1_pystacked (p401)
            learner   mean_MSE
      ols           1  .17196874
  lassocv           2  .17048203
  ridgecv           3  .17061646
       rf           4  .18560235
gradboost           5  .17094582
```

Step 4: Estimate causal effects.

```
. ddml estimate
DDML estimation results:
spec  r     Y learner     D learner         b        SE     Z learner
 opt  1  Y1_pystacked  D1_pystacked 13528.643(1726.023)
opt = minimum MSE specification for that resample.
Min MSE DDML model
y-E[y|X]  = Y1_pystacked_1                      Number of obs   =      9915
D-E[D|X,Z]= D1_pystacked_1
Z-E[Z|X]  = Z1_pystacked_1
```

net_tfa	Coefficient	Std. err.	z	P>|z|	[95% conf.	interval]
p401	13528.64	1726.023	7.84	0.000	10145.7	16911.59
_cons	-42.68924	531.132	-0.08	0.936	-1083.689	998.3104

One-line syntax.

```
. qui qddml $Y ($X) ($D = $Z), model(iv) cmd(pystacked)
```

C.3 Interactive IV Model `interactiveiv`

```
. use "sipp1991.dta", clear
. global Y net_tfa
. global X age inc educ fsize marr twoearn db pira hown
. global Z e401
. global D p401
```

Step 1: Initialize `ddml` model.

```
. set seed 123
. ddml init interactiveiv
```

Step 2: Add supervised machine learners for estimating conditional expectations.

```
. * add learners for E[Y|X,Z=0] and E[Y|X,Z=0]
. ddml E[Y|X,Z]: pystacked $Y $X                                          || ///
>     method(ols)                                                         || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                    || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                    || ///
>     m(rf) pipe(sparse) opt(max_features(5))                             || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(4)
Learner Y1_pystacked added successfully.
. * add learners for E[D|X,Z=0] and E[D|X,Z=0]
. ddml E[D|X,Z]: pystacked $D $X                                          || ///
>     method(ols)                                                         || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                    || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                    || ///
>     m(rf) pipe(sparse) opt(max_features(5))                             || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(4)
Learner D1_pystacked added successfully.
. * add learners for E[Z|X,]
. ddml E[Z|X]: pystacked $Z $X                                            || ///
>     method(ols)                                                         || ///
>     m(lassocv) xvars(c.($X)##c.($X))                                    || ///
>     m(ridgecv) xvars(c.($X)##c.($X))                                    || ///
>     m(rf) pipe(sparse) opt(max_features(5))                             || ///
>     m(gradboost) pipe(sparse) opt(n_estimators(250) learning_rate(0.01)) ,  ///
>     njobs(4)
Learner Z1_pystacked added successfully.
```

Step 3: Perform cross-fitting.

```
. ddml crossfit
Cross-fitting E[y|X,Z] equation: net_tfa
Cross-fitting fold 1 2 3 4 5 ...completed cross-fitting
Cross-fitting E[D|X,Z] equation: p401
Cross-fitting fold 1 2 3 4 5 ...completed cross-fitting
Cross-fitting E[Z|X]: e401
Cross-fitting fold 1 2 3 4 5 ...completed cross-fitting
```

Step 4: Estimate causal effects.

```
. ddml estimate
DDML estimation results (LATE):
spec  r    Y0 learner    Y1 learner    D0 learner    D1 learner         b        SE     Z learner
 opt  1  Y1_pystacked  Y1_pystacked  D1_pystacked  D1_pystacked 11568.523(1613.594)  Z1_pystacked
opt = minimum MSE specification for that resample.

E[y|X,D=0]   = Y1_pystacked0_1                    Number of obs   =      9915
E[y|X,D=1]   = Y1_pystacked1_1
E[D|X,Z=0]   = D1_pystacked0_1
E[D|X,Z=1]   = D1_pystacked1_1
E[Z|X]       = Z1_pystacked_1
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]

```
        p401 |   11568.52   1613.594     7.17   0.000     8405.938    14731.11
```

```
Warning: 13 propensity scores trimmed to lower limit .01.
```

One-line syntax.

```
. qui qddml $Y ($X) ($D=$Z), model(late) cmd(pystacked)
```