The Stata Journal (2024)
24, Number 1, pp. 3–45

DOI: 10.1177/1536867X241233641

ddml: Double/debiased machine learning in Stata

Achim Ahrens
ETH Zürich
Zürich, Switzerland
achim.ahrens@gess.ethz.ch

Christian B. Hansen
University of Chicago
Chicago, IL
christian.hansen@chicagobooth.edu

Mark E. Schaffer
Heriot-Watt University
Edinburgh, U.K.
m.e.schaffer@hw.ac.uk

Thomas Wiemann
University of Chicago
Chicago, IL
wiemann@uchicago.edu

Abstract. In this article, we introduce a package, `ddml`, for double/debiased machine learning 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 or many exogenous variables. `ddml` is compatible with many existing supervised machine learning programs in Stata. We recommend using double/debiased machine learning in combination with stacking estimation, which combines multiple machine learners into a final predictor. We provide Monte Carlo evidence to support our recommendation.

Keywords: st0738, ddml, causal inference, machine learning, double/debiased machine learning, doubly robust estimation

1 Introduction

Identification of causal effects frequently relies on an unconfoundedness assumption, which is 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 two-stage least-squares 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 few 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 datasets, for example, textual confounders as in Roberts, Stewart, and Nielsen (2020) or digital trace data (Hangartner, Kopp, and Siegenthaler 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 the post double selection lasso (PDS-lasso) of Belloni, Chernozhukov, and Hansen (2014), which fits auxiliary lasso regressions of the outcome and treatments against a menu of transformed controls. Under an approximate sparsity assumption, which posits that the data-generating process (DGP) can be approximated well by relatively few terms included in the menu, this approach allows for precise treatment-effects 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 (for example, Gilchrist and Sands [2016]; Dhar, Jain, and Jayachandran [2022]), partially facilitated by the availability of software programs in Stata (`pdslasso`, Ahrens, Hansen, and Schaffer 2018) and R (`hdm`, Chernozhukov, Hansen, and Spindler 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, Lenza, and Primiceri (2021) argue that approximate sparsity may provide a poor description in several economic examples. Thus, there is 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, for example, Athey, Tibshirani, and Wager [2019] and Farrell, Liang, and Misra [2021]), machine learning methods are widely adopted in industry and practice for their empirical performance. To facilitate their application for 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, because researchers are rarely certain about the structure of the underlying DGP. 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

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

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 package `ddml`, which implements DDML for Stata.[2] `ddml` adds to a few programs for causal machine learning in Stata (Ahrens, Hansen, and Schaffer 2018). We briefly summarize the four main features of the program:

1. `ddml` supports flexible estimators of causal parameters in five econometric models: a) the partially linear model, b) the interactive model (for binary treatment), c) the partially linear instrumental-variables (IV) model, d) the flexible partially linear IV model, and e) 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, Hansen, and Schaffer 2023), our complementary Stata front-end relying on the Python library `scikit-learn` (Pedregosa et al. 2011; Buitinck et al. 2013). `ddml` also supports two novel approaches to pairing DDML with stacking introduced in Ahrens et al. (2024): Short-stacking takes a shortcut by leveraging the cross-fitted predicted values for estimating the stacking weights, and pooled stacking enforces common weights across cross-fitting folds.

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, Hansen, and Schaffer 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 "`regress y x`" syntax, support for `if` conditions, and a postestimation `predict` command is compatible with `ddml`.

4. `ddml` provides flexible multiline syntax and short one-line syntax. The multiline 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 to `pdslasso` and `ivlasso` (Ahrens, Hansen, and Schaffer 2018).

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 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 command. Section 6 demonstrates the command's usage with two applications.

2. This article refers to version 1.4.2 of `ddml`.

2 DDML with conditional unconfoundedness

This section discusses DDML for the partially linear model and the interactive model in turn. The exposition follows Chernozhukov et al. (2018). 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 (that is, other than D and $\boldsymbol{X}$).[3] The key difference between the partially linear model and the interactive model is their positions in the tradeoff 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]

Assumption 1 (Conditional orthogonality)**.** $E\{\mathrm{Cov}(U, D|\boldsymbol{X})\} = 0$.

To show identification of θ_0, consider the score

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

3. Because $(D, \boldsymbol{X}, U)$ jointly determine Y in (1), the model is also dubbed the "all causes model" (see, for example, 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 (for example, 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 models typically show identification under the stronger assumption that $E(U|D, \boldsymbol{X}) = 0$. We differentiate here to highlight differences between the partially linear model and interactive model.

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, $E[\mathrm{Var}(D|\boldsymbol{X})] \neq 0$, we get

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

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 $\boldsymbol{X}$ (that is, ℓ_0) and of D given $\boldsymbol{X}$ (that is, m_0) using appropriate nonparametric estimators (for example, 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 step 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),

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

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.

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—that is, in $I_k^c \equiv I \setminus I_k$—resulting in $\widehat{\ell}_{I_k^c}$ and $\widehat{m}_{I_k^c}$, 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 $\widehat{\ell}_{I_k^c}(\boldsymbol{X}_i)$ and $\widehat{m}_{I_k^c}(\boldsymbol{X}_i)$. Repeating this procedure for all K folds then allows for computation of the DDML estimator for θ_0:

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

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 subsample I_k^c using Y_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\widehat{\ell}_{I_k^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
   b. Fit a CEF estimator to the subsample I_k^c using D_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\widehat{m}_{I_k^c}(\boldsymbol{X}_i)$ for $i \in I_k$.
2. Compute (5).

❑

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

Remark 1: Number of folds. The number of cross-fitting folds is a necessary tuning choice. Theoretically, any finite value is admissible. Chernozhukov et al. (2018) report in remark 3.1 that four or five folds perform better than only using $K = 2$. Based on our simulation experience, we find that more folds tend to lead to better performance because more data are used for estimation of CEFs, 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 $K = 5$ is likely a good baseline in many settings.

6. We here omit the constant from the estimation stage. Because 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) in 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 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()` option, which automatically fits 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 $\widehat{\theta}_n^{(r)}$ denote the DDML estimate from the rth cross-fit repetition, and let $\widehat{s}_n^{(r)}$ denote its associated standard-error estimate with $r = 1, \ldots, R$. The aggregate median point estimate and associated standard error are defined as

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

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

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

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()`.

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 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.

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

Assumptions 2 and 3 below are sufficient for identification of the ATE and ATET. Note that 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 definitions.[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 (that is, 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,

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

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

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

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.

10. In the defined interactive model under assumption 2, the heterogeneity in treatment effects that the ATE and ATET average over is fully observed because 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, for example, Belloni et al. (2017).

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

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

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

where $\widehat{g}_{I^c_k}$ and $\widehat{m}_{I^c_k}$ are cross-fitted estimators for g_0 and m_0 as defined in section 2.1. Because D is binary, the cross-fitted values $\widehat{g}_{I^c_k}(1,\boldsymbol{X})$ and $\widehat{g}_{I^c_k}(0,\boldsymbol{X})$ are computed by using only treated and untreated observations, respectively. $\widehat{p}_{I^c_k}$ is a cross-fitted estimator of the unconditional treatment probability.

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

3 DDML with IV

This section outlines the partially linear IV model, the flexible partially linear IV model, and the interactive IV model. The discussion is again based on Chernozhukov et al. (2018). 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 IVs $\boldsymbol{Z}$. 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, Hansen, and Spindler (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 $(Y, D, \boldsymbol{X}, \boldsymbol{Z})$.

11. Optimality requires the assumption of homoskedasticity. The instruments are valid more generally but are not optimal under heteroskedasticity. Obtaining optimal instruments under heteroskedasticity would require estimating conditional variance functions.

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 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 IVs $\boldsymbol{Z}$ 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 IVs 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)**.** $E\{\text{Cov}(U, Z|\boldsymbol{X})\} = 0$.

Assumption 5 (Conditional linear IV relevance)**.** $E\{\text{Cov}(D, Z|\boldsymbol{X})\} \neq 0$.

To show identification, consider the score function

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

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 that $E\{\psi(\boldsymbol{W}; \theta_0, \ell_0, m_0, r_0)\} = 0$. 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

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

where assumption 5 is used to ensure a nonzero denominator.

The DDML estimator based on (9) is given by

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

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

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

12. As in section 2.1, the interpretation of θ_0 can be generalized under stronger identifying assumptions. See Angrist, Graddy, and Imbens (2000).

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. When 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, Hansen, and Spindler (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 $\widetilde{p}(\boldsymbol{Z},\boldsymbol{X})$, it holds that

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

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 $\widetilde{p}$ such that

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

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

Assumption 7 (Conditional IV relevance)**.** $E\{\mathrm{Var}(D|\boldsymbol{Z},\boldsymbol{X})|\boldsymbol{X}\} \neq 0$.

Now consider the score function

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

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

13. The choice $\widetilde{p}(\boldsymbol{Z},\boldsymbol{X}) = E(D|\boldsymbol{Z},\boldsymbol{X})$ results in the optimal instrument, in the sense of semiparametric efficiency, under homoskedasticity.

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

where assumption 7 ensures a nonzero denominator.

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

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

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

In simulations, we find that the finite-sample performance of the estimator in (14) improves when the LIE 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 $k \in \{1, \ldots, K\}$, do the following:
   a. Fit a CEF estimator to the subsample I_k^c using Y_i as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\widehat{\ell}_{I^c_k}(\boldsymbol{X}_i)$ for $i \in I_k$.
   b. Fit a CEF estimator to the subsample 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 $\widehat{p}_{I^c_k}(\boldsymbol{Z}_i, \boldsymbol{X}_i)$ for $i \in I_k$ and in-sample predicted values $\widehat{p}_{I^c_k}(\boldsymbol{Z}_i, \boldsymbol{X}_i)$ for $i \in I_k^c$.
   c. Fit a CEF estimator to the subsample I_k^c using the in-sample predicted values $\widehat{p}_{I^c_k}(\boldsymbol{Z}_i, \boldsymbol{X}_i)$ as the outcome and $\boldsymbol{X}_i$ as predictors. Obtain the out-of-sample predicted values $\widehat{m}_{I^c_k}(\boldsymbol{X}_i)$ for $i \in I_k$.
2. Compute (14).

❑

Standard errors corresponding to $\widehat{\theta}_n$ in (14) are the same as in section 3.1, where the instrument is now given by $\widehat{p}_{I^c_{k_i}}(\boldsymbol{Z}_i, \boldsymbol{X}_i) - \widehat{m}_{I^c_{k_i}}(\boldsymbol{X}_i)$. 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 representing assignment to treatment.

The parameter of interest we target is

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

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), we follow the exposition in Chernozhukov et al. (2018), where "local" does not strictly refer to compliers but instead to observations with a higher propensity score—that is, 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 1 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

$$\theta_0 = \frac{E\left\{\ell_0(1, \boldsymbol{X}) - \ell_0(0, \boldsymbol{X})\right\}}{E\left\{p_0(1, \boldsymbol{X}) - p_0(0, \boldsymbol{X})\right\}} \tag{15}$$

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 (15) because the estimating equations implicit in obtaining (15) do not satisfy Neyman orthogonality. Hence, a direct estimator of θ_0 obtained by plugging nonparametric estimators in for nuisance functions in (15) 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 not introduced in this article. 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 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

$$\widehat{\theta}_n = \frac{\frac{1}{n}\sum_i \left[\frac{Z_i\{Y_i - \widehat{\ell}_{I^c_{k_i}}(1, \boldsymbol{X}_i)\}}{\widehat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} - \frac{(1-Z_i)\{Y_i - \widehat{\ell}_{I^c_{k_i}}(0, \boldsymbol{X}_i)\}}{1-\widehat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} + \widehat{\ell}_{I^c_{k_i}}(1, \boldsymbol{X}_i) - \widehat{\ell}_{I^c_{k_i}}(0, \boldsymbol{X}_i)\right]}{\frac{1}{n}\sum_i \left[\frac{Z_i\{D_i - \widehat{p}_{I^c_{k_i}}(1, \boldsymbol{X}_i)\}}{\widehat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} - \frac{(1-Z_i)\{D_i - \widehat{p}_{I^c_{k_i}}(0, \boldsymbol{X}_i)\}}{1-\widehat{r}_{I^c_{k_i}}(\boldsymbol{X}_i)} + \widehat{p}_{I^c_{k_i}}(1, \boldsymbol{X}_i) - \widehat{p}_{I^c_{k_i}}(0, \boldsymbol{X}_i)\right]} \tag{16}$$

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

`ddml` supports heteroskedasticity and cluster–robust standard errors for $\widehat{\theta}_n$. 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 a highly active research area, recent advances provide convergence rates for special instances of many machine learners, including lasso (Bickel, Ritov, and Tsybakov 2009; Belloni et al. 2012), random forests (Wager and Walther 2015; Wager and Athey 2018; Athey, Tibshirani, and Wager 2019), neural networks (Schmidt-Hieber 2020; Farrell, Liang, and Misra 2021), and boosting (Luo, Spindler, and Kück 2016). 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 hypertuning parameters. It thus provides a convenient framework for combining and identifying suitable learners, thereby reducing the risk of misspecification. Ahrens et al. (2024) introduce short-stacking, which reduces the

computational cost of pairing DDML and stacking drastically, and pooled stacking, which enforces common weights across cross-fitting folds.

We discuss stacking approaches to DDML estimation in section 4.1. Section 4.2 demonstrates the performance of DDML in combination with stacking approaches using a simulation.

4.1 DDML and 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 (GB) trees with varying tree depths, and feed-forward neural nets with varying numbers 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 holdout 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 1. The stacking objective function for estimating $\ell_0(\boldsymbol{X})$ using the training sample T_k is then defined as

$$\min_{w_{k,1},\ldots,w_{k,J}} \sum_{i \in T_k} \left\{ Y_i - \sum_{j=1}^{J} w_{k,j} \widehat{\ell}^{(j)}_{T^c_{k,v(i)}}(\boldsymbol{X}_i) \right\}^2, \quad \text{s.t. } w_{k,j} \geq 0, \quad \sum_{j=1}^{J} |w_{k,j}| = 1 \tag{17}$$

where $w_{k,j}$ are referred to as stacking weights. We use $\widehat{\ell}^{(j)}_{T^c_{k,v(i)}}(\boldsymbol{X}_i)$ to denote the cross-validated predicted value for observation i, which is obtained from fitting learner j on the subsample $T^c_{k,v(i)} \equiv T_k \setminus T_{k,v(i)}$, that is, the subsample excluding the fold $v(i)$ into which observation i falls. The stacking predicted values are obtained as $\sum_j \widehat{w}_{k,j} \widehat{\ell}^{(j)}_k(\boldsymbol{X}_i)$, 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, Tibshirani, and Friedman 2009). However, it is not the only sensible choice of combining base learners. For example, stacking could instead select the single learner with the lowest quadratic loss 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, Hansen, and Schaffer 2023).

Pooled stacking. A variant of stacking specific to DDML is *pooled stacking*. Standard stacking fits the final learner K separate times, once in each cross-fitting step, yielding K separate sets of stacking weights $\widehat{w}_{k,j}$ for the J learners. With DDML pooled stacking, we can impose the additional constraint in (17) that the weights are the same across all cross-fitting folds, $\widehat{w}_{k,j} = \widehat{w}_j, \forall\, k$. By returning one set of stacking weights, pooled stacking imposes an additional degree of regularization and facilitates interpretation but suffers from the same high computational cost as pairing DDML with (regular) stacking.

Short-stacking. Stacking and pooled stacking rely on cross-validation. In the context of DDML, we can also exploit the cross-fitted predicted values directly for stacking. That is, we can directly apply CLS to the cross-fitted predicted values for estimating $\ell_0(\boldsymbol{X})$ [and similarly, $m_0(\boldsymbol{X})$]:

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

We refer to this form of stacking that uses the cross-fitted predicted values as *short-stacking* because it takes a shortcut. This is to contrast it to regular stacking, which estimates the stacking weights for each cross-fitting fold k. The main advantage of short-stacking relative to standard stacking is the lower computational cost because short-stacking does not require the fitting of the j learners on subsamples to obtain the cross-validated predicted values $\widehat{\ell}^{(j)}_{T^c_{k,v(i)}}(\boldsymbol{X}_i)$ needed for standard stacking. Furthermore, short-stacking (like pooled stacking) also produces one set of weights for the entire sample, which facilitates interpretation and implies a higher degree of regularization. A potential disadvantage of short-stacking is that it is more susceptible to overfitting issues because stacking weights and structural parameters are estimated using the same cross-fitted predicted values. We thus recommend only considering short-stacking in regular settings where the number of candidate learners is small relative to N (see also the discussion in Ahrens et al. [2024]). Algorithm A.4 in the online appendix summarizes the short-stacking algorithm for the partially linear model.[15]

15. While short-stacking can be applied similarly 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 online appendix; see algorithm A.5.

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 numbers of observed covariates:

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

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. DGPs 4 and 5 are identical, except that DGP 5 does not add nuisance covariates that 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 and ridge with untransformed base controls.
3.–4. Cross-validated lasso and ridge with fifth-order polynomials of base controls but no interactions (referred to as "Poly 5").
5.–6. Cross-validated lasso and ridge with second-order polynomials and all first-order interaction terms (referred to as "Poly 2 + Inter.").

16. All base learners have been implemented using `pystacked`. We use the defaults of `pystacked` for parameter values and settings not mentioned here.

7. Random forests with low regularization: base controls, maximum tree depth of 10, 500 trees, and approximately $\sqrt{p}$ features considered at each split.
8. Random forests with medium regularization: same as 7 but with maximum tree depth of 6.
9. Random forests with high regularization: same as 7 but with maximum tree depth of 2.
10. GB trees with low regularization: base controls, 1,000 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 five 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 2 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, short-stacking, and pooled stacking, which we combine with CLS and the single-best learner. We set the number of folds to $K = 20$ if $n = 100$ and to $K = 5$ if $n = 1000$. That is, we adapt the number of folds K to the total sample size n to ensure that the CEF estimators are trained on sufficiently large training samples.

For comparison, we report results for ordinary least squares (OLS) and PDS-lasso with base controls, PDS-lasso with Poly 5, PDS-lasso with Poly 2 + interactions, and an oracle estimator using the full sample.[18] The oracle 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 (CR) of 95% confidence intervals for DGPs 1–3 in table 1. We delegate results for DGPs 4 and 5, including a brief discussion, to online appendix B. DDML estimators leveraging stacking approaches perform favorably compared with 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 more precisely estimated in larger 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.

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`.

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	0.113	0.915	0.036	0.944	0.090	0.915	0.032	0.934	0.117	0.890	0.036	0.944
OLS (Base)	0.130	0.886	0.037	0.942	0.286	0.375	0.288	0	0.224	0.617	0.209	0.007
PDS-Lasso (Base)	0.133	0.840	0.037	0.941	0.295	0.276	0.287	0	0.227	0.588	0.209	0.006
PDS-Lasso (Poly 5)	0.148	0.768	0.038	0.935	0.219	0.465	0.106	0.327	0.228	0.583	0.198	0.016
PDS-Lasso (Poly 2 + Inter.)	0.158	0.742	0.039	0.931	0.200	0.549	0.033	0.922	0.231	0.591	0.176	0.058
DDML methods:												
Base learners												
OLS	0.134	0.710	0.038	0.930	0.286	0.226	0.288	0	0.226	0.424	0.209	0.006
Lasso with CV (Base)	0.116	0.893	0.037	0.950	0.286	0.312	0.288	0	0.221	0.617	0.210	0.008
Ridge with CV (Base)	0.123	0.888	0.038	0.941	0.291	0.288	0.289	0	0.246	0.542	0.219	0.002
Lasso with CV (Poly 5)	0.127	0.843	0.039	0.940	0.144	0.762	0.092	0.458	0.202	0.635	0.193	0.025
Ridge with CV (Poly 5)	0.139	0.794	0.050	0.864	0.178	0.620	0.125	0.188	0.265	0.478	0.221	0.002
Lasso with CV (Poly 2 + Inter.)	0.137	0.816	0.042	0.925	0.155	0.718	0.036	0.925	0.214	0.623	0.184	0.048
Ridge with CV (Poly 2 + Inter.)	0.158	0.695	0.095	0.505	0.152	0.685	0.112	0.308	0.191	0.645	0.265	0
Random forest (Low)	0.179	0.728	0.098	0.566	0.249	0.438	0.196	0.015	0.229	0.619	0.110	0.447
Random forest (Medium)	0.186	0.718	0.128	0.317	0.252	0.431	0.222	0.005	0.232	0.612	0.140	0.227
Random forest (High)	0.237	0.565	0.227	0.007	0.270	0.362	0.269	0	0.259	0.520	0.230	0.002
Gradient boosting (Low)	0.103	0.906	0.039	0.924	0.126	0.802	0.080	0.578	0.118	0.885	0.038	0.938
Gradient boosting (Medium)	0.126	0.860	0.048	0.881	0.197	0.598	0.144	0.125	0.137	0.838	0.045	0.906
Gradient boosting (High)	0.239	0.544	0.154	0.142	0.277	0.338	0.258	0	0.205	0.665	0.079	0.715
Neural net	0.137	0.773	0.152	0.131	0.106	0.843	0.075	0.534	0.107	0.882	0.037	0.865
Meta learners												
Stacking: CLS	0.117	0.898	0.037	0.949	0.169	0.683	0.037	0.918	0.169	0.742	0.048	0.892
Stacking: Single best	0.119	0.907	0.037	0.949	0.149	0.744	0.036	0.925	0.157	0.781	0.045	0.902
Short-stacking: CLS	0.116	0.892	0.037	0.948	0.170	0.671	0.037	0.920	0.165	0.754	0.046	0.897
Short-stacking: Single best	0.117	0.899	0.037	0.950	0.141	0.756	0.036	0.925	0.157	0.784	0.045	0.908
Pooled stacking: CLS	0.117	0.895	0.038	0.949	0.188	0.625	0.037	0.922	0.181	0.731	0.047	0.896
Pooled stacking: Single best	0.117	0.894	0.037	0.950	0.173	0.656	0.036	0.925	0.181	0.725	0.045	0.906

NOTES: The table reports median absolute bias (MAB) and coverage rates (CR) of 95% confidence intervals. We use 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 online appendix.

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 few of the individual learners but are still competitive and superior to most learners. Looking across all results, we see that stacking provides robustness to potentially very bad performance that could be obtained from using one poorly performing learner.

There are overall small performance differences among the six stacking estimators considered, suggesting that short-stacking has a substantial practical advantage because of its lower computational cost. Ahrens et al. (2024) report that short-stacking reduces the compute time by a factor of $1/V$, where V is the number of cross-validation folds. 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 online appendix, provide further insights into how stacking functions with CLS. CLS assigns large stacking weights to base learners with low MSPEs, which in turn are associated with low biases. 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 the highest MSPE and the 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 compares favorably with 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 with 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.

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`. See the help files for all available commands and options.

5.1 Syntax: ddml

The `ddml` estimation proceeds in four steps.

Step 1: Initialize ddml and select model.

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

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()`, in which model specifications and estimation results will be stored. The default is `mname(m0)`.

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

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

In step 2, 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.

Table 2. Conditional expectations that must be specified for each model

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

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 because it involves fitting the selected machine learners repeatedly.

```
ddml crossfit [, mname(name) shortstack poolstack nostdstack
   finalest(name)]
```

Step 4: Estimate causal effects.

Finally, we estimate the parameter of interest for all combinations of learners added in step 2.

```
ddml estimate [, mname(name) robust cluster(varname) vce(vcetype)
   noconstant showconstant atet ateu trim(real) shortstack poolstack
   finalest(name)]
```

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

```
ddml estimate [, mname(name) spec(string) rep(string) allcombos notable
   replay]
```

Utilities

`ddml describe` provides information about the model setup or results:

```
ddml describe [, mname(name) sample learners crossfit estimates all]
```

`ddml` stores many internal results on associate arrays, notably the various stacking weights. These can be retrieved using `ddml extract`:

`ddml extract` [*object_name*, `mname(`*name*`) vname(`*varname*`) ename(`*name*`) stata show(`*display_item*`) keys key1(`*string*`) key2(`*string*`) key3(`*string*`) subkey1(`*string*`) subkey2(`*string*`)`]

`ddml export` saves the estimated conditional expectations and other variables to a CSV file:

`ddml export` *filename* [`, mname(`*name*`) addvars(`*varlist*`)`]

5.2 Options

5.2.1 Step 1 options: Initialization.

`mname(`*name*`)` is the name of the DDML model. This option allows running of multiple DDML models simultaneously. The default is `mname(m0)`.

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

`fcluster(`*varname*`)` is the cluster identifier for cluster randomization of folds.

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

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

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

5.2.2 Step 2 options: Adding learners.

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

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

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

5.2.3 Step 3 options: Cross-fitting.

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

`poolstack` asks for pooled stacking to be used. This is available only if `pystacked` has been used for standard stacking in all equations.

`nostdstack` is used with `pystacked` and short-stacking; it tells `pystacked` to generate the base learner predictions without the computationally expensive additional step of obtaining the stacking weights.

`finalest(`*name*`)` sets the final estimator for all stacking methods; the default estimator, `finalest(nnls1)`, is least squares without a constant and with the constraints that weights are nonnegative and sum to 1. Alternative final estimators include `singlebest` (use the minimum mean squared error [MSE] base learner), `ols` (ordinary least squares), and `avg` (unweighted average of all base learners).

5.2.4 Step 4 options: Estimation.

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

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

`allcombos` estimates all possible specifications. By default, only the minimum mean squared error, short-stacking, or pooled stacking specification is estimated and displayed.

`notable` suppresses the summary table.

`replay` is used in combination with `spec()` and `rep()` to display and return estimation results.

`robust` reports standard errors that are robust to the presence of arbitrary heteroskedasticity.

`cluster(`*varname*`)` selects the cluster–robust variance–covariance estimator.

`vce(`*vcetype*`)` selects the variance–covariance estimator, for example, `vce(hc3)` or `vce(cluster id)`. See `help regress##vcetype` for available options.

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

`showconstant` displays the constant term in the summary estimation output table (`partial`, `iv`, and `fiv` models only).

`atet` reports the average treatment effect on the treated (the default is ATE).

`ateu` reports the average treatment effect on the untreated (the default is ATE).

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

`shortstack`, `poolstack`, `finalest(`*name*`)`: see above under `ddml crossfit`.

Refitting the final learner using `ddml estimate` with stacking options is generally very fast because it does not require cross-fitting again.

For descriptions of the utility commands `ddml describe`, `ddml extract`, and `ddml export` and of their options, see their corresponding help files.

5.3 Short syntax: qddml

The `ddml` package includes the wrapper program `qddml`, which provides a one-line syntax for fitting a `ddml` model. The one-line syntax follows the syntax of `pdslasso` and `ivlasso` (Ahrens, Hansen, and Schaffer 2018). The main restriction of `qddml` compared with the more flexible multiline syntax is that `qddml` allows for only one user-specified machine learner.

`qddml` has integrated support for `pystacked`, which is the default learner in all equations. The syntax for `qddml` options differs depending on whether `pystacked` is used as the learner in each equation.

Syntax when used with pystacked

`qddml` *depvar treatment_vars* [`(`*controls*`)`]
`(`*treatment_vars*`=`*excluded_instruments*`), model(`*name*`)` [`shortstack stdstack poolstack finalest(`*name*`) pystacked(`*string*`) pystacked_y(`*string*`) pystacked_d(`*string*`) pystacked_z(`*string*`)` *options*]

The `pystacked()` option sets the options for all the conditional expectations estimated by `pystacked`; the `_y`, `_d`, and `_z` variants control the options sent to the corresponding conditional expectation estimations. Other options are as in `ddml`.

Syntax when used with other learners

`qddml` *depvar treatment_vars* [`(`*controls*`)`]
`(`*treatment_vars*`=excluded_instruments), model(`*name*`)` [`cmd(`*string*`) ycmd(`*string*`) dcmd(`*string*`) zcmd(`*string*`) *cmdopt(`*string*`) *vtype(`*name*`) *predopt(`*name*`)` *options*]

The cmd() option sets the options for all the conditional expectations estimated by pystacked; the ycmd(), dcmd(), and zcmd() variants control the options sent to the corresponding conditional expectation estimations. The cmdopt() option can be used to either set the options for all equations or, if the asterisk is replaced with y, d, or z, set the options for the corresponding conditional expectation estimation. Other options are as in ddml.

5.4 Supported machine learning programs

ddml is compatible with any supervised machine learning program in Stata that supports the typical "regress y x" syntax, comes with a postestimation predict command, and supports if statements. We have tested ddml with the following programs:

- pystacked facilitates the stacking of a wide range of machine learners, including regularized regression, random forests, support vector machines, GB trees, and feed-forward neural nets using Python's scikit-learn (Ahrens, Hansen, and Schaffer 2023; Pedregosa et al. 2011; Buitinck et al. 2013). In addition, pystacked can also be used as a front-end to fit individual machine learners. ddml has integrated support for pystacked and is the recommended default learner.
- lassopack implements regularized regression, for example, lasso, ridge, and elastic net (Ahrens, Hansen, and Schaffer 2020).
- 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 as usual. Fold ID variables are also stored as Stata variables (named m0_fid_r by default, where m0 is the default model name and r is the cross-fitting repetition). ddml models are stored on Mata structs and using Mata's associative arrays. Specifically, the ddml model created by

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

`ddml init` is an `mStruct`, and information relating to the estimation of conditional expectations is 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 ensures that the same random fold variable is used for a given dataset.
- Using the same fold variable alone is usually not sufficient to ensure replication, because 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()` option.
- Replication of `ddml` results may require additional steps with some programs that rely on randomization in other software environments, for example, 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 individual learner on the entire sample to see what settings need to be passed to them for their results to replicate.
- Beware of changing samples. Fold splits or learner idiosyncrasies may mean that sample sizes vary slightly across learners, estimation samples, or cross-fitting repetitions. `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, Venti, and Wise (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 online appendix C. Additional examples can also be found in the help file. Based on Berry, Levinsohn, and Pakes (1995), we show in section 6.2 how to use `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 IVs.

6.1 401(k) and financial wealth

The data consist of $n = 9915$ households from the 1991 Survey of Income and Program Participation. The application is originally due to Poterba, Venti, and Wise (1995) but has been revisited by Belloni et al. (2017), Chernozhukov et al. (2018), and Wüthrich and Zhu (2023), among others. Following previous studies, we include the control variables age, income, years of education, and family size, as well as indicators for marital status, two-earner status, benefit pension status, individual retirement account 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
. 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 more 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, `pystacked` with the `m(lassocv)` option for cross-validated lasso (as an example of how to use `pystacked` to estimate one learner), and `rforest` for random forests. When using `rforest`, we need to add the option `vtype(none)` because the postestimation `predict` command of `rforest` does not support variable types.

```
. *** add learners for E[Y|X]
. ddml E[Y|X]: regress $Y $X
Learner Y1_regress added successfully.
. ddml E[Y|X]: pystacked $Y c.($X)##c.($X), type(reg) method(lassocv)
Learner Y2_pystacked 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]: regress $D $X
Learner D1_regress added successfully.
. ddml E[D|X]: pystacked $D c.($X)##c.($X), type(reg) method(lassocv)
Learner D2_pystacked added successfully.
. ddml E[D|X], vtype(none): rforest $D $X, type(reg)
Learner D3_rforest added successfully.
```

The flexible `ddml` syntax allows specification of different sets of covariates for different learners. This flexibility can be useful because, for example, linear learners such as the lasso might perform better if 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 the cross-validated lasso 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` subcommand `describe` allows us to verify that the learners were correctly registered:

```
. ddml describe
Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_regress Y2_pystacked Y3_rforest
D equations (1):        e401
 e401 learners:         D1_regress D2_pystacked D3_rforest
```

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 that correspond to the six learners specified in the previous step. These variables are called `Y1_reg_1`, `Y2_pystacked_1_1`, `Y3_rforest_1`, `D1_reg_1`, `D2_pystacked_1_1`, and `D3_rforest_1`. `Y` and `D` indicate the outcome and the treatment variable. Indexes `1` to `3` are learner counters. `regress`, `pystacked`, and `rforest` correspond to the names of the commands used. The `_1` suffix indicates the cross-fitting repetition. The additional `_1` in the case of `D2_pystacked_1_1` indicates the learner number (there is only one `pystacked` learner).

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

```
. ddml describe, crossfit
Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_regress Y2_pystacked Y3_rforest
D equations (1):        e401
 e401 learners:         D1_regress D2_pystacked D3_rforest
Crossfit results (detail):
                                    All    By fold:
Cond. exp.  Learner      rep         MSE        1         2         3         4
net_tfa     shortstack     1       1.5e+09   1.8e+09   1.4e+09   1.4e+09   1.5e+09
e401        shortstack     1          0.18      0.17      0.17      0.18      0.17
```

Step 4: Estimate causal effects.

In this final step, we obtain the causal effect estimates. Because we requested short-stacking in step 3, `ddml` shows the 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.

```
. ddml estimate, robust
Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_regress Y2_pystacked Y3_rforest
D equations (1):        e401
 e401 learners:         D1_regress D2_pystacked D3_rforest
DDML estimation results:
spec  r     Y learner     D learner         b        SE
 mse  1  Y2_pystacked  D2_pystacked  9788.291 (1339.797)
  ss  1  [shortstack]          [ss]  9713.861 (1334.252)
mse = minimum MSE specification for that resample.
Shortstack DDML model
y-E[y|X]  = y-Y_net_tfa_ss_1                    Number of obs   =      9915
D-E[D|X]  = D-D_e401_ss_1
------------------------------------------------------------------------------
             |               Robust
     net_tfa | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        e401 |   9713.861   1334.252     7.28   0.000     7098.774    12328.95
       _cons |   94.84956   534.8125     0.18   0.859    -953.3637    1143.063
------------------------------------------------------------------------------
Stacking final estimator: nnls1
```

Because we have specified three learners per conditional expectation, there are nine specifications relying on the base learners in total (because we can combine `Y1_reg_1`, `Y2_pystacked_1`, and `Y3_rforest_1` with `D1_reg_1`, `D2_pystacked_1`, and `D3_rforest_1`). To get all results, we add the `allcombos` option:

```
. ddml estimate, robust allcombos

Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_regress Y2_pystacked Y3_rforest
D equations (1):        e401
 e401 learners:         D1_regress D2_pystacked D3_rforest

DDML estimation results:
spec  r     Y learner     D learner         b        SE
   1  1    Y1_regress    D1_regress  5986.657 (1523.694)
   2  1    Y1_regress  D2_pystacked  9563.875 (1389.172)
   3  1    Y1_regress    D3_rforest  8138.218 (1278.134)
   4  1  Y2_pystacked    D1_regress  9175.519 (1371.065)
*  5  1  Y2_pystacked  D2_pystacked  9788.291 (1339.797)
   6  1  Y2_pystacked    D3_rforest  8023.298 (1218.538)
   7  1    Y3_rforest    D1_regress  8166.968 (1488.753)
   8  1    Y3_rforest  D2_pystacked  9887.655 (1407.031)
   9  1    Y3_rforest    D3_rforest  8968.295 (1298.973)
  ss  1  [shortstack]          [ss]  9713.861 (1334.252)
* = minimum MSE specification for that resample.

Shortstack DDML model
y-E[y|X]  = y-Y_net_tfa_ss_1                    Number of obs   =      9915
D-E[D|X]  = D-D_e401_ss_1

------------------------------------------------------------------------------
             |               Robust
     net_tfa | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        e401 |   9713.861   1334.252     7.28   0.000     7098.774    12328.95
       _cons |   94.84956   534.8125     0.18   0.859    -953.3637    1143.063
------------------------------------------------------------------------------

Stacking final estimator: nnls1
```

We can use the `spec(`*string*`)` option to select among the listed specifications. *string* is either the specification number—`ss`, `st`, or `ps` to get the short-stacking, standard stacking, or pooled stacking specification, respectively—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()` option can be provided either in combination with `allcombos` or after estimation in combination with the `replay` option, for example,

```
. ddml estimate, spec(1) replay

Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_regress Y2_pystacked Y3_rforest
D equations (1):        e401
 e401 learners:         D1_regress D2_pystacked D3_rforest

DDML estimation results:
spec  r     Y learner     D learner         b        SE
   1  1  Y2_pystacked  D2_pystacked  5986.657 (1523.694)
  ss  1  [shortstack]          [ss]  9713.861 (1334.252)
mse = minimum MSE specification for that resample.

DDML model, specification 1
y-E[y|X]  = y-Y1_regress_1                      Number of obs   =      9915
D-E[D|X]  = D-D1_regress_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	10.74705	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:

```
. generate double Y1_resid = $Y - Y1_reg
. generate double D1_resid = $D - D1_reg
. regress Y1_resid D1_resid, 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_resid	Coefficient	Robust std. err.	t	P>|t|	[95% conf.	interval]
D1_resid	5986.657	1523.694	3.93	0.000	2999.906	8973.407
_cons	10.74705	561.2911	0.02	0.985	-1089.498	1110.992

Manual estimation using `regress` allows the use of `regress`'s postestimation tools.

Stacking

We next demonstrate DDML with stacking. To this end, we exploit the stacking regressor implemented in `pystacked`. `pystacked` allows combining multiple base learners with learner-specific settings and covariates into a final meta-learner. The learners are separated by `||`. `method()` selects the learner, `xvars()` specifies learner-specific covariates

(overwriting the default covariates `$X`), and `opt()` 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 five cores. A detailed explanation of the `pystacked` syntax can be found in Ahrens, Hansen, and Schaffer (2023).

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

After cross-fitting, we retrieve the cross-fitted MSPE using the `ddml extract` command with `show(mse)` or examine the stacking weights using `stweights`:

```
. quietly ddml crossfit
. ddml extract, show(stweights)
mean stacking weights across folds/resamples for D1_pystacked (e401)
final stacking estimator: nnls1
              learner  mean_weight        rep_1
      ols           1    .00915031    .00915031
  lassocv           2    .19106656    .19106656
  ridgecv           3    .34720737    .34720737
       rf           4    .03258878    .03258878
gradboost           5    .41998697    .41998697
mean stacking weights across folds/resamples for Y1_pystacked (net_tfa)
final stacking estimator: nnls1
              learner  mean_weight        rep_1
      ols           1    .03589102    .03589102
  lassocv           2    .33837078    .33837078
  ridgecv           3    .49997977    .49997977
       rf           4    .04156376    .04156376
gradboost           5    .09033751    .09033751
```

Finally, in the estimation stage, we retrieve the results of DDML with stacking:

```
. ddml estimate, robust

Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_pystacked
D equations (1):        e401
 e401 learners:         D1_pystacked

DDML estimation results:
spec  r     Y learner     D learner         b        SE
  st  1  Y1_pystacked  D1_pystacked  9544.626 (1303.897)

Stacking DDML model
y-E[y|X]  = y-Y1_pystacked_1                      Number of obs   =      9915
D-E[D|X]  = D-D1_pystacked_1
------------------------------------------------------------------------------
             |               Robust
     net_tfa | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        e401 |   9544.626   1303.897     7.32   0.000     6989.035    12100.22
       _cons |  -100.0744   535.5587    -0.19   0.852     -1149.75    949.6013
------------------------------------------------------------------------------

Stacking final estimator: nnls1
```

The DDML-specific stacking approaches of short-stacking and pooled stacking can be requested at either the cross-fitting or the estimation step. Refitting the final learner at the estimation step allows us to avoid repeating the computationally intensive cross-fitting. Here we request short-stacking and pooled stacking but using the single-best base learner.

```
. ddml estimate, robust shortstack poolstack finalest(singlebest)

Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_pystacked
D equations (1):        e401
 e401 learners:         D1_pystacked

DDML estimation results:
spec  r     Y learner     D learner         b        SE
  st  1  Y1_pystacked  D1_pystacked  9544.626 (1303.897)
  ss  1  [shortstack]          [ss]  9812.624 (1336.534)
  ps  1   [poolstack]          [ps]  9766.171 (1335.864)

Shortstack DDML model
y-E[y|X]  = y-Y_net_tfa_ss_1                      Number of obs   =      9915
D-E[D|X]  = D-D_e401_ss_1
------------------------------------------------------------------------------
             |               Robust
     net_tfa | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        e401 |   9812.624   1336.534     7.34   0.000     7193.065    12432.18
       _cons |   75.17922   534.6284     0.14   0.888    -972.6732    1123.032
------------------------------------------------------------------------------

Stacking final estimator: singlebest
```

Short-stacking is computationally much faster than regular stacking (or pooled stacking) because it avoids the cross-validation within cross-fitting folds. Below, we disable regular stacking with the `nostdstack` option in the cross-fitting stage. In this example, where we use parallelization with 5 cores, the run time is only 50.7 seconds for DDML with short-stacking compared with 93.0 seconds for DDML with regular stacking.

```
. quietly ddml crossfit, shortstack nostdstack
. ddml estimate, robust

Model:                  partial, crossfit folds k=4, resamples r=1
Mata global (mname):    m0
Dependent variable (Y): net_tfa
 net_tfa learners:      Y1_pystacked
D equations (1):        e401
 e401 learners:         D1_pystacked

DDML estimation results:
spec  r     Y learner     D learner         b        SE
  ss  1  [shortstack]          [ss]  9601.109 (1302.116)

Shortstack DDML model
y-E[y|X]  = y-Y_net_tfa_ss_1                      Number of obs   =      9915
D-E[D|X]  = D-D_e401_ss_1
```

net_tfa	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
e401	9601.109	1302.116	7.37	0.000	7049.009	12153.21
_cons	83.47775	533.8799	0.16	0.876	-962.9075	1129.863

```
Stacking final estimator:
```

One-line syntax.

`qddml` provides a simplified and convenient one-line syntax. The main constraint of `qddml` is that it allows only for one user-specified learner. The default learner is `pystacked`, which by default uses OLS, cross-validated lasso, and gradient boosting as default learners. The `pystacked` base learners and non-`pystacked` commands can be modified via various options. Below, we use `qddml` with `pystacked`'s default base learners. We omit the output for the sake of brevity.

```
. quietly qddml $Y $D (c.($X)##c.($X)), model(partial) kfolds(4) robust
```

6.2 The market for automobiles

For this demonstration, we follow Chernozhukov, Hansen, and Spindler (2015b), who fit a stylized demand model using IVs based on the data from Berry, Levinsohn, and Pakes (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, Levinsohn, and Pakes (1995) suggest exploiting other products' characteristics as instruments. Fol-

lowing Chernozhukov, Hansen, and Spindler (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, Hansen, and Spindler (2015b) also construct an augmented set of instruments, including first-order interactions, squared terms, and cubic terms. In the analysis below, we extend Chernozhukov, Hansen, and Spindler (2015b) by applying DDML with stacking and a diverse set of learners, including OLS, lasso, ridge, random forest, and GB 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, 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 with the other models supported by `ddml`, there is one complication that arises because, to estimate $E(D|\boldsymbol{X})$, we exploit fitted values of $E(D|\boldsymbol{X},\boldsymbol{Z})$ to impose LIE compliance. Because these fitted values have not yet been generated, we use the placeholder `{D}`, which in the cross-fitting stage will be internally replaced with estimates of $E(D|\boldsymbol{X},\boldsymbol{Z})$. We use the `learner()` option to match one learner for $E(D|\boldsymbol{X})$ with a learner for $E(D|\boldsymbol{X},\boldsymbol{Z})$, and we use `vname()` 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.
. *** add learners for E[D|X,Z]
. ddml E[D|X,Z], learner(Dpystacked): pystacked $D $Xaug $Zaug            ||
>     method(ols) xvars($Xbase $Zbase)                                    ||
>     m(lassocv)                                                          ||
>     m(ridgecv)                                                          ||
>     m(rf) opt(n_estimators(200) max_features(None))                     ||
>     m(rf) opt(n_estimators(200) max_features(10))                       ||
>     m(rf) opt(n_estimators(200) max_features(5))                        ||
>     m(gradboost) opt(n_estimators(800) learning_rate(0.01))             ||
>     m(gradboost) opt(n_estimators(800) learning_rate(0.1))              ||
>     m(gradboost) opt(n_estimators(800) learning_rate(0.3)),
>     njobs(4)
Learner Dpystacked added successfully.
. *** add learners for E[D|X]
. ddml E[D|X], mname(m0) learner(Dpystacked) vname($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.

```
. quietly ddml crossfit
. ddml estimate, robust

Model:                  fiv, crossfit folds k=4, resamples r=5
Mata global (mname):    m0
Dependent variable (Y): y
 y learners:            Ypystacked
D equations (1):        price
 price learners:        Dpystacked

DDML estimation results:
spec  r     Y learner     D learner         b        SE     DH learner
  st  1    Ypystacked    Dpystacked    -0.112    (0.010)  Dpystacked_h
  st  2    Ypystacked    Dpystacked    -0.124    (0.010)  Dpystacked_h
  st  3    Ypystacked    Dpystacked    -0.110    (0.011)  Dpystacked_h
  st  4    Ypystacked    Dpystacked    -0.123    (0.011)  Dpystacked_h
  st  5    Ypystacked    Dpystacked    -0.130    (0.013)  Dpystacked_h
Mean/med    Y learner     D learner         b        SE     DH learner
  st mn    Ypystacked    Dpystacked    -0.120    (0.013)  Dpystacked_h
  st md    Ypystacked    Dpystacked    -0.123    (0.014)  Dpystacked_h
Median over 5 stacking resamples
y-E[y|X]  = y-Ypystacked                          Number of obs   =      2217
E[D|X,Z]  = D-Dpystacked
E[D^|X]   = Dpystacked_h
Orthogonalized D = D - E[D^|X]; optimal IV = E[D|X,Z] - E[D^|X].
------------------------------------------------------------------------------
             |               Robust
           y |  Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       price |   -.1234241   .0144181    -8.56   0.000     -.151683   -.0951652
------------------------------------------------------------------------------
Stacking final estimator: nnls1
Summary over 5 resamples:
       D eqn      mean       min       p25       p50       p75       max
       price   -0.1200   -0.1305   -0.1242   -0.1234   -0.1117   -0.1104
```

Manual final estimation. We can obtain the final estimate manually. To this end, we construct the instrument as $\widehat{E}(D|\boldsymbol{X},\boldsymbol{Z}) - \widehat{E}(D|\boldsymbol{X})$ and the residualized endogenous regressor as $D - \widehat{E}(D|\boldsymbol{X})$. 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".

```
. generate double Y1_resid = $Y - Ypystacked_1
. generate double dtilde = $D - Dpystacked_h_1
. generate double optiv = Dpystacked_1 - Dpystacked_h_1
. ivregress 2sls Y1_resid (dtilde=optiv), robust
Instrumental variables 2SLS regression            Number of obs   =      2,217
                                                  Wald chi2(1)    =     129.46
                                                  Prob > chi2     =     0.0000
                                                  R-squared       =     0.0939
                                                  Root MSE        =      .9654
------------------------------------------------------------------------------
             |               Robust
    Y1_resid |  Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
      dtilde |   -.1117023   .0098173   -11.38   0.000    -.1309438   -.0924607
       _cons |    .0040233   .0205103     0.20   0.844    -.0361762    .0442228
------------------------------------------------------------------------------
Endogenous: dtilde
Exogenous:  optiv
```

7 Conclusion

This article introduced the command `ddml`, which implements double/debiased machine learning. It allows for flexible estimation of structural parameters in five econometric models, leveraging a wide range of supervised machine learners. While `ddml` is compatible with many existing machine learning programs in Stata, it is specifically designed to be used with `pystacked`, which allows combining several learners into a meta-learner via stacking. We see several avenues for extensions: First, `ddml` primarily focuses on cross-sectional models. Some panel models are readily implementable in `ddml`, but expanding its capabilities for seamless use with a wide range of panel models would increase its practical relevance. Second, researchers and policymakers are frequently interested in learning treatment effects for specific subpopulations sharing observable characteristics. The estimation of conditional average treatment effects would be a natural extension to the existing `ddml` program. Third, `ddml` currently lacks underidentification diagnostics and weak-identification robust inference for IV regressions, which we hope to add in future releases.

8 Acknowledgments

We thank users who tested earlier versions of the program. We also thank Jan Ditzen, Ben Jann, Eroll Kuhn, Di Liu, and Moritz Marbach for helpful comments, as well as participants at Stata User Conferences in Germany (2021), Italy (2022), Switzerland (2022), and the U.K. (2023). All remaining errors are our own.

9 Programs and supplemental material

To install the software files as they existed at the time of publication of this article, type

```
. net sj 24-1
. net install st0738     (to install program files, if available)
. net get st0738         (to install ancillary files, if available)
```

To get the latest stable versions of `ddml` and `qddml` from our website, check the installation instructions at https://statalasso.github.io/docs/ddml/installation/. We update the stable website version more frequently than the Statistical Software Components version.

10 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. Statistical Software Components S458459, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458459.html.

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

———. 2023. pystacked: Stacking generalization and machine learning in Stata. *Stata Journal* 23: 909–931. https://doi.org/10.1177/1536867X231212426.

Ahrens, A., C. B. Hansen, M. E. Schaffer, and T. Wiemann. 2024. Model averaging and double machine learning. ArXiv:2401.01645 [econ.EM]. https://doi.org/10.48550/arXiv.2401.01645.

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. *Review of Economic Studies* 67: 499–527. https://doi.org/10.1111/1467-937X.00141.

Angrist, J. D., and A. B. Krueger. 1999. Empirical strategies in labor economics. In Vol. 3A of *Handbook of Labor Economics*, ed. O. Ashenfelter and D. Card, 1277–1366. Amsterdam: Elsevier. https://doi.org/10.1016/S1573-4463(99)03004-7.

Athey, S., J. Tibshirani, and S. Wager. 2019. Generalized random forests. *Annals of Statistics* 47: 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: 2369–2429. https://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: 233–298. https://doi.org/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* 63: 841–890. https://doi.org/10.2307/2171802.

Bickel, P. J., Y. Ritov, and A. B. Tsybakov. 2009. Simultaneous analysis of Lasso and Dantzig selector. *Annals of Statistics* 37: 1705–1732. https://doi.org/10.1214/08-AOS620.

Blandhol, C., J. Bonney, M. Mogstad, and A. Torgovitsky. 2022. When is TSLS actually LATE? Working paper, University of Chicago, Becker Friedman Institute (BFI)Working Paper No. 2022-16. https://doi.org/10.2139/ssrn.4014707.

Breiman, L. 1996. Stacked regressions. *Machine Learning* 24: 49–64. https://doi.org/10.1007/BF00117832.

Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Müller, O. Grisel, V. Niculae, et al. 2013. API design for machine learning software: Experiences from the scikit-learn project. European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases Workshop: Languages for Data Mining and Machine Learning.

Chang, C.-C., and C.-J. Lin. 2011. libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2: 1–27. https://doi.org/10.1145/1961189.1961199.

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. *Econometrics Journal* 21: C1–C68. https://doi.org/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: 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: 649–688. https://doi.org/10.1146/annurev-economics-012315-015826.

———. 2016. High-dimensional metrics in R. arXiv:1603.01700 [stat.ML]. https://doi.org/10.48550/arXiv.1603.01700.

Dhar, D., T. Jain, and S. Jayachandran. 2022. Reshaping adolescents' gender attitudes: Evidence from a school-based experiment in India. *American Economic Review* 112: 899–927. https://doi.org/10.1257/aer.20201112.

Farrell, M. H., T. Liang, and S. Misra. 2021. Deep neural networks for estimation and inference. *Econometrica* 89: 181–213. https://doi.org/10.3982/ECTA16901.

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*, ed. O. Maimon and L. Rokach, 1269–1277. Boston, MA: Springer. https://doi.org/10.1007/978-0-387-09823-4_66.

Giannone, D., M. Lenza, and G. E. Primiceri. 2021. Economic predictions with big data: The illusion of sparsity. *Econometrica* 89: 2409–2437. https://doi.org/10.3982/ECTA17842.

Gilchrist, D. S., and E. G. Sands. 2016. Something to talk about: Social spillovers in movie consumption. *Journal of Political Economy* 124: 1339–1382. https://doi.org/10.1086/688177.

Guenther, N., and M. Schonlau. 2018. svmachines: Stata module providing support vector machines for both classification and regression. Statistical Software Components S458564, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458564.html.

Haghish, E. F. 2019. Seamless interactive language interfacing between R and Stata. *Stata Journal* 19: 61–82. https://doi.org/10.1177/1536867X19830891.

Hahn, J. 1998. On the role of the propensity score in efficient semiparametric estimation of average treatment effects. *Econometrica* 66: 315–331. https://doi.org/10.2307/2998560.

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

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer. https://doi.org/10.1007/978-0-387-84858-7.

Heckman, J. J., and E. J. Vytlacil. 2007. Econometric evaluation of social programs. Part 1, Causal models, structural models and econometric policy evaluation. In Vol. 6B of *Handbook of Econometrics*, ed. J. J. Heckman and E. E. Leamer, 4779–4874. Amsterdam: Elsevier. https://doi.org/10.1016/S1573-4412(07)06070-9.

Huntington-Klein, N. C. 2021. mlrtime. GitHub. https://github.com/NickCH-K/MLRtime/.

Imbens, G. W., and J. D. Angrist. 1994. Identification and estimation of local average treatment effects. *Econometrica* 62: 467–475. https://doi.org/10.2307/2951620.

Luo, Y., M. Spindler, and J. Kück. 2016. High-dimensional L_2 boosting: Rate of convergence. arXiv:1602.08927 [stat.ML]. https://doi.org/10.48550/arXiv.1602.08927.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12: 2825–2830. https://doi.org/10.48550/arXiv.1201.0490.

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: 1–32. https://doi.org/10.1016/0047-2727(94)01462-W.

Roberts, M. E., B. M. Stewart, and R. A. Nielsen. 2020. Adjusting for confounding with text matching. *American Journal of Political Science* 64: 887–903. https://doi.org/10.1111/ajps.12526.

Schmidt-Hieber, J. 2020. Nonparametric regression using deep neural networks with ReLU activation function. *Annals of Statistics* 48: 1875–1897. https://doi.org/10.1214/19-AOS1875.

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

Wager, S., and S. Athey. 2018. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association* 113: 1228–1242. https://doi.org/10.1080/01621459.2017.1319839.

Wager, S., and G. Walther. 2015. Adaptive concentration of regression trees, with application to random forests. arXiv:1503.06388 [math.ST]. https://doi.org/10.48550/arXiv.1503.06388.

Wolpert, D. H. 1992. Stacked generalization. *Neural Networks* 5: 241–259. https://doi.org/10.1016/S0893-6080(05)80023-1.

Wüthrich, K., and Y. Zhu. 2023. Omitted variable bias of Lasso-based inference methods: A finite sample analysis. *Review of Economics and Statistics* 105: 982–997. https://doi.org/10.1162/rest_a_01128.

About the authors

Achim Ahrens is a postdoctoral 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 a professor of economics in Edinburgh Business School at Heriot-Watt University, Edinburgh, U.K., 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.