ddml: Double/debiased machine learning in Stata

Achim Ahrens (ETH Zürich)

Mark E Schaffer (Heriot-Watt University, IZA)
Christian B Hansen (University of Chicago)
Thomas Wiemann (University of Chicago)

Package website: `https://statalasso.github.io/`

May 19, 2022

Italian Stata Group Meeting

Introduction

- A rich and growing literature exploits machine learning to facilitate causal inference.
- **A central focus:** *high-dimensional* controls and/or instruments, which can arise if
  - we observe many controls/instruments
  - controls/instruments enter through an unknown function
- Belloni, Chernozhukov, and Hansen (2014) and Belloni et al. (2012) propose estimators *relying on the Lasso* that allow for high-dimensional controls/instruments.

  $\Rightarrow$ Available via `pdslasso` in Stata (Ahrens, Hansen, and Schaffer, 2020)

Introduction

What if we don't want to use the lasso?

- The Lasso might not be the *best-performing machine learner* for a particular problem.
- The Lasso relies on the *approximate sparsity assumption*, which might not be appropriate in some settings.

Chernozhukov et al. (2018) propose *Double/Debiased Machine Learning* (DDML) which allow to exploit machine learners other than the Lasso.

Our contribution:

- We introduce `ddml`, which implements DDML for Stata.
- We provide simulation evidence on the finite sample performance of DDML.
- Our recommendation is to use DDML in combination with stacking.

Background

Motivating example. The partial linear model:

$$y_i = \underbrace{\theta d_i}_{\text{causal part}} + \underbrace{g(\boldsymbol{x}_i)}_{\text{nuisance}} + \varepsilon_i.$$

How do we account for confounding factors $\boldsymbol{x}_i$? — The standard approach is to assume linearity $g(\boldsymbol{x}_i) = \boldsymbol{x}_i'\beta$ and consider alternative combinations of controls.

Problems:

- Non-linearity & unknown interaction effects
- High-dimensionality: we might have “many” controls
- We don’t know which controls to include

Background

Motivating example. The partial linear model:

$$y_i = \underbrace{\theta d_i}_{\text{causal part}} + \underbrace{g(\boldsymbol{x}_i)}_{\text{nuisance}} + \varepsilon_i .$$

Post-double selection (Belloni, Chernozhukov, and Hansen, 2014) and *post-regularization* (Chernozhukov, Hansen, and Spindler, 2015) provide data-driven solutions for this setting.

Both "double" approaches rely on the *sparsity assumption* and use two auxiliary lasso regressions: $y_i \rightsquigarrow \boldsymbol{x}_i$ and $d_i \rightsquigarrow \boldsymbol{x}_i$. lasso PDS

Related approaches exist for *optimal IV* estimation (Belloni et al., 2012) and/or *IV with many controls* (Chernozhukov, Hansen, and Spindler, 2015).

Background

These methods have been implemented for Stata in `pdslasso` (Ahrens, Hansen, and Schaffer, 2020), `dsregress` (StataCorp) and R (`hdm`; Chernozhukov, Hansen, and Spindler, 2016).

A quick example using AJR (2001):

```
. clear

. use https://statalasso.github.io/dta/AJR.dta

.
. pdslasso logpgp95 avexpr ///
          (lat_abst edes1975 avelf temp* humid* steplow-oilres)
.
. ivlasso logpgp95 (avexpr=logem4) ///
          (lat_abst edes1975 avelf temp* humid* steplow-oilres), ///
          partial(logem4)
```

Example 1 (`pdslasso`) allows for high-dimensional controls.

Example 2 (`ivlasso`) treats `avexpr` as endogenous and exploits `logem4` as an instrument. (More details in the `pds/ivlasso` help file.)

Background

There are **advantages** of relying on lasso:

- intuitive assumption of (approximate) sparsity
- computationally relatively cheap (due to plugin lasso penalty; no cross-validation needed)
- Linearity has its advantages (e.g. extension to fixed effects; Belloni et al., 2016)

But there are also **drawbacks**:

- What if the sparsity assumption is not plausible?
- There is a wide set of machine learners at disposable—Lasso might not be the best choice.
- Lasso requires careful feature engineering to deal with non-linearity & interaction effects.

Review of DDML

The partial linear model:

$$\begin{aligned} y_i &= \theta d_i + g(\boldsymbol{x}_i) + \varepsilon_i \\ d_i &= m(\boldsymbol{x}_i) + v_i \end{aligned}$$

Naive idea: We estimate conditional expectations $\ell(\boldsymbol{x}_i) = E[y_i|\boldsymbol{x}_i]$ and $m(\boldsymbol{x}_i) = E[d_i|\boldsymbol{x}_i]$ using ML and partial out the effect of $\boldsymbol{x}_i$ (in the style of Frisch-Waugh-Lovell):

$$\hat{\theta}_{DDML} = \left(\frac{1}{n} \sum_i \hat{v}_i^2 \right)^{-1} \frac{1}{n} \sum_i \hat{v}_i (y_i - \hat{\ell}),$$

where $\hat{v}_i = d_i - \hat{m}_i$.

Review of DDML

Yet, this approach is flawed: The estimation error $\ell(\boldsymbol{x}_i) - \hat{\ell}$ and v_i may be correlated due to **over-fitting**, leading to poor performance.

DDML, thus, relies on **cross-fitting**. Cross-fitting is sample splitting with swapped samples.

DDML for the partial linear model (DML 2)

We split the sample in K random folds of equal size denoted by I_k:

- For $k = 1, \ldots, K$, estimate $\ell(\boldsymbol{x}_i)$ and $m(\boldsymbol{x}_i)$ using sample I_k^c and form out-of-sample predictions $\hat{\ell}_i$ and $\hat{m}_i$ for all i in I_k.
- Construct estimator $\hat{\theta}$ as

$$\left(\frac{1}{n}\sum_i \hat{v}_i^2\right)^{-1} \frac{1}{n}\sum_i \hat{v}_i(y_i - \hat{\ell}),$$

where $\hat{v}_i = d_i - \hat{m}_i$. $\hat{m}_i$ and $\hat{\ell}_i$ are the cross-fitted predicted values.

DDML models

The DDML framework can be applied to other models (all implemented in `ddml`):

Interactive model

$$y_i = g(d_i, \boldsymbol{x}_i) + u_i \qquad E[u_i|\boldsymbol{x_i}, d_i] = 0$$
$$z_i = m(\boldsymbol{x}_i) + v_i \qquad E[u_i|\boldsymbol{x_i}] = 0$$

As in the Partial Linear Model, we are interested in the ATE, but do not assume that d_i (a binary treatment variable) and $\boldsymbol{x}_i$ are separable.

We estimate the conditional expectations $E[y_i|\boldsymbol{x}_i, d_i = 0]$ and $E[y_i|\boldsymbol{x}_i, d_i = 1]$ as well as $E[d_i|\boldsymbol{x}_i]$ using a supervised machine learner.

DDML models

The DDML framework can be applied to other models (all implemented in `ddml`):

Partial linear IV model

$$\begin{aligned} y_i &= d_i\theta + g(\boldsymbol{x}_i) + u_i & E[u_i|\boldsymbol{x_i}, z_i] &= 0 \\ z_i &= m(\boldsymbol{x}_i) + v_i & E[v_i|\boldsymbol{x_i}] &= 0 \end{aligned}$$

where the aim is to estimate the average treatment effect θ using observed instrument z_i in the presence of controls $\boldsymbol{x}_i$. We estimate the conditional expectations $E[y_i|\boldsymbol{x_i}]$, $E[d_i|\boldsymbol{x}_i]$ and $E[z_i|\boldsymbol{x}_i]$ using a supervised machine learner.

DDML models

The DDML framework can be applied to other models (all implemented in `ddml`):

High-dimensional IV model

$$
\begin{aligned}
y_i &= d_i\theta + g(\boldsymbol{x}_i) + u_i \\
d_i &= h(\boldsymbol{z}_i) + m(\boldsymbol{x}_i) + v_i
\end{aligned}
$$

where the estimand of interest is θ. The instruments and controls enter the model through unknown functions $g()$, $h()$ and $f()$.

We estimate the conditional expectations $E[y_i|\boldsymbol{x}_i]$, $E[\hat{d}_i|\boldsymbol{x}_i]$ and $\hat{d}_i := E[d_i|\boldsymbol{z}_i, \boldsymbol{x}_i]$ using a supervised machine learner. The instrument is then formed as $\hat{d}_i - \hat{E}[\hat{d}_i|\boldsymbol{x}_i]$ where $\hat{E}[\hat{d}_i|\boldsymbol{x}_i]$ denotes the estimate of $E[\hat{d}_i|\boldsymbol{x}_i]$.

DDML models

The DDML framework can be applied to other models (all implemented in `ddml`):

Interactive IV model

$$\begin{aligned} y_i &= \mu(\boldsymbol{x}_i, \boldsymbol{z}_i) + u_i & E[u_i|\boldsymbol{x}_i, z_i] &= 0 \\ d_i &= m(z_i, \boldsymbol{x}_i) + v_i & E[v_i|\boldsymbol{x}_i, z_i] &= 0 \\ z_i &= p(\boldsymbol{x}_i) + \xi_i & E[\xi_i|\boldsymbol{x}_i] &= 0 \end{aligned}$$

where the aim is to estimate the local average treatment effect.

We estimate, using a supervised machine learner, the following conditional expectations: $E[y_i|\boldsymbol{x}_i, z_i = 0]$ and $E[y_i|\boldsymbol{x}_i, z_i = 1]$; $E[D|\boldsymbol{x}_i, z_i = 0]$ and $E[D|\boldsymbol{x}_i, z_i = 1]$; $E[z_i|\boldsymbol{x}_i]$.

The `ddml` package

We introduce `ddml` for Stata:

- Compatible with various ML programs in Stata (e.g. `lassopack`, `pylearn`, `randomforest`).
  - $\rightarrow$ *Any* program with the classical "`reg y x`" syntax and post-estimation `predict` will work.
- Short (one-line) and flexible multi-line version
- 5 models supported: partial linear model, interactive model, interactive IV model, partial IV model, optimal IV.

Stacking regression

Which machine learner should we use?

`ddml` supports a range of ML programs: `pylearn`, `lassopack`, `randomforest`. — Which one should we use?

We don't know whether we have a sparse or dense problem; linear or non-linear. We don't know whether, e.g., lasso or random forests will perform better.

Stacking, as implemented in `pystacked`, provides a solution: We use an 'optimal' combination of base learners.

qddml example: Partial linear model

qddml is the one-line ('quick') version of ddml and uses a syntax similar to pds/ivlasso.

```
. use https://statalasso.github.io/dta/AJR.dta, clear

.
. global Y logpgp95

. global X lat_abst edes1975 avelf temp* humid* steplow-oilres

. global D avexpr

.
. qddml $Y $D ($X), model(partial) cmdopt(method(rf gradboost))

DML with Y=m0_y and D=m0_d1:
------------------------------------------------------------------------------
        m0_y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       m0_d1 |   .3391897   .0621291     5.46   0.000      .217419    .4609604
------------------------------------------------------------------------------
```

Extended ddml syntax

Step 1: Initialise `ddml` and select model:

`ddml init` *model* [, `kfolds(`*integer*`)` `reps(`*integer*`)` ...]

where *model* is either 'partial', 'iv', 'interactive', 'ivhd', 'late'.

Step 2: Add supervised ML programs for estimating conditional expectations:

`ddml` *eq newvarname* [, *eqopt*]: *command depvar indepvars* [, *cmdopt*]

where *eq* selects the conditional expectations to be estimated. *command* is a ML program that supports the standard reg y x-type syntax. *cmdopt* are specific to that program.

Multiple estimation commands per equation are allowed.

Extended `ddml` syntax

Step 3: Cross-fitting

```
ddml crossfit
```

Step 4: Estimation of causal effects

```
ddml estimate [, robust ...]
```

Additional auxiliary commands:

`ddml describe` (describe current model set up),`ddml save` & `ddml use` (to import/save `ddml` objects), `ddml extract` (to retrieve objects), `ddml export` (export in csv format).

Extended `ddml` syntax: Example

```
. global Y logpgp95

. global X lat_abst edes1975 avelf temp* humid* steplow-oilres

. global D avexpr

.
. *** initialise ddml and select model;
. ddml init partial

.
. *** specify supervised machine learners for E[Y|X] ("yeq") and E[D|X] ("deq")
. * y-equation:
. ddml yeq, gen(pyy): pystacked $Y $X, type(reg) method(rf gradboost)
Equation successfully added.

.
. * d-equation:
. ddml deq, gen(pyd): pystacked $D $X, type(reg) method(rf gradboost)
Equation successfully added.
```

Extended `ddml` syntax: Example (cont'd.)

```
. *** cross-fitting and display mean-squared prediction error
. ddml crossfit
Model: partial
Number of Y estimating equations: 1
Number of D estimating equations: 1

Cross-fitting equation 1 2

Mean-squared error for y|X:
 Name                 Orthogonalized       Command        N           MSPE
--------------------------------------------------------------------------------

 logpgp95             m0_pyy               pystacked      64       0.573751
Mean-squared error for D|X:
 Name                 Orthogonalized       Command        N           MSPE
--------------------------------------------------------------------------------

 avexpr               m0_pyd               pystacked      64       1.648804

.
. *** estimation of parameter of interest
. ddml estimate

DML with Y=m0_pyy and D=m0_pyd (N=):
------------------------------------------------------------------------------
      m0_pyy |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      m0_pyd |   .3794184   .0569073     6.67   0.000     .2678821    .4909546
------------------------------------------------------------------------------
```

Simulation I: Advantages of Stacking

We demonstrate the use of `ddml` using the partially linear model by extending the analysis of 401(k) eligibility and total financial wealth of Poterba, Venti, and Wise (1995). The data consists of $n = 9915$ households from the 1991 SIPP.

Simulation set-up: We consider a *linear DGP* and a *non-linear DGP*, and compare performance of OLS, PDS-Lasso and various machine learners, including meta learners.

We would expect that stacking performs well under both settings, while linear approaches only perform well if the DGP is linear.

Simulation I: Advantages of Stacking

1. Use the full sample OLS estimate to obtain $\hat{\tau}_{OLS}$. Construct the partial residuals $y_i^{(r)} = y_i - \hat{\tau}_{OLS} d_i$.
2. Fit a supervised learning estimator that aims to predict $y_i^{(r)}$ with the controls x_i and d_i with x_i, respectively. Denote the fitted values as $\tilde{g}$ and $\tilde{h}$. We either use
   - linear regression (*Linear DGP*)
   - gradient boosting (*Non-linear DGP*)
3. Sample from the empirical distribution of x_i by bootstrapping n_s observations from the original data. Denote the bootstrapped sample by $\mathcal{D}_b$.
4. Generate the treatment and outcome variable over the bootstrap sample:

$$\tilde{d}_i^{(b)} = \mathbb{1}\{\tilde{h}(x_i) + \nu_i \geq 0.5\} \tag{1}$$

$$\tilde{y}_i^{(b)} = \tau_0 \tilde{d}_i^{(b)} + \tilde{g}(x_i) + \varepsilon_i, \forall i \in \mathcal{D}_b \tag{2}$$

where $\nu_i \overset{iid}{\sim} \mathcal{N}(0, \kappa_1)$ and $\varepsilon_i \overset{iid}{\sim} \mathcal{N}(0, \kappa_2)$, $\tau_0 = 6,000$.

Simulation I: Advantages of Stacking

We consider the following base learners:

- CV-Lasso with interactions and 2nd order polynomials
- CV-Ridge with interactions and 2nd order polynomials
- CV-Lasso with 10th order polynomials and no interactions
- CV-Ridge with 10th order polynomials and no interactions
- Random forest with low regularization: 8 predictors considered at each leaf split, no limit on the number of observation per node, bootstrap sample size of 70% (`max_features(8) min_samples_leaf(1) max_samples(.7)`)
- Random forest with high regularization: 5 predictors considered at each leaf split, at least 10 observation per node, bootstrap sample size of 70% (`max_features(5) min_samples_leaf(10) max_samples(.7)`)
- Gradient boosted trees with low regularization: 500 trees and a learnings rate of 0.01 (`n_estimators(500) learning_rate(0.01)`)
- Gradient boosted trees with high regularization: 250 trees and a learnings rate of 0.01 (`n_estimators(250) learning_rate(0.01)`)
- Feed-forward neural nets with 3 hidden layers of size 5 (`hidden_layer_sizes(5 5 5)`)

Simulation I: Advantages of Stacking

Table: Average Stacking Weights

	Stacking		Single-Best	
Panel (A): Linear DGP	$Y\|X$	$D\|X$	$Y\|X$	$D\|X$
OLS	.646	.492	.813	.643
Lasso with CV (2nd order poly)	.111	.158	.161	.267
Ridge with CV (2nd order poly)	.063	.061	.018	.019
Lasso with CV (10th order poly)	.032	.08	.003	.049
Ridge with CV (10th order poly)	.03	.047	.005	.016
Random forest (low regularization)	.013	.012	0	0
Random forest (high regularization)	.017	.027	0	0
Gradient boosting (low regularization)	.028	.043	0	.006
Gradient boosting (high regularization)	.024	.074	0	.002
Neural net	.036	.005	0	0

Stacking assigns the highest weight to OLS if *the DGP is linear*. . .

Simulation I: Advantages of Stacking

Table: Average Stacking Weights

	Stacking		Single-Best	
	(1)	(2)	(3)	(4)
Panel (B): Non-Linear DGP	$Y \mid X$	$D \mid X$	$Y \mid X$	$D \mid X$
OLS	.037	.021	0	0
Lasso with CV (2nd order poly)	.039	.067	.083	.149
Ridge with CV (2nd order poly)	.177	.23	.12	.125
Lasso with CV (10th order poly)	.052	.077	.088	.061
Ridge with CV (10th order poly)	.078	.068	.019	.044
Random forest (low regularization)	.041	.01	0	0
Random forest (high regularization)	.028	.069	.001	0
Gradient boosting (low regularization)	.517	.213	.678	.359
Gradient boosting (high regularization)	.02	.239	.011	.262
Neural net	.012	.005	0	0

. . . and the highest weight to gradient boosting if *the DGP is non-linear*.

Simulation I: Advantages of Stacking

Table: Bias and Coverage Rates in the Linear and Non-Linear DGP

	$n_s = 9915$			$n_s = 99150$		
Panel (A): Linear DGP	Bias	MAB	Rate	Bias	MAB	Rate
Full sample:						
OLS	100.99	918.03	.95	-22.61	255.52	.94
PDS-Lasso	101.83	913.18	.95	-19.9	257.29	.94
DDML methods:						
Base learners						
OLS	105.07	906.96	.94	-23.05	256.51	.94
Lasso with CV (2nd order poly)	104.33	907.84	.94	-22.45	257.23	.94
Ridge with CV (2nd order poly)	103.22	898.56	.94	-23.27	255.54	.94
Lasso with CV (10th order poly)	49.56	1120.59	.93	37.98	260.53	.95
Ridge with CV (10th order poly)	1066	1342.38	.9	15.85	260.41	.95
Random forest (low regularization)	-59.63	1083.64	.91	-59.29	343.46	.86
Random forest (high regularization)	105.58	952.35	.94	-46.54	275.56	.91
Gradient boosting (low regularization)	53.97	930.93	.94	-41.84	252.14	.94
Gradient boosting (high regularization)	162.75	923.08	.95	48.31	259.12	.95
Neural net	-3594.99	5380.31	.17	-2165.41	3212.12	.16
Meta learners						
Stacking: NNLS	100.01	935.27	.94	-22.7	254.01	.94
Stacking: Single best	92.79	944.07	.95	-25.03	255.75	.94
Stacking: NNLS interactive	690.88	1245.02	.95	-51.29	359.4	.96
Short-stacking	100.51	912.88	.94	-24.04	252.17	.94
Single best	103.31	917.74	.94	-24.88	252.16	.94

Simulation I: Advantages of Stacking

Table: Bias and Coverage Rates in the Linear and Non-Linear DGP

	$n_s = 9915$			$n_s = 99150$		
Panel (B): Non-Linear DGP	Bias	MAB	Rate	Bias	MAB	Rate
Full sample:						
OLS	-2496.16	2477.19	.63	-2658.04	2636.31	0
PDS-Lasso	-2507.47	2489.77	.62	-2657.5	2635.94	0
DDML methods:						
Base learners						
OLS	-2522.98	2540.36	.62	-2660.54	2640.98	0
Lasso with CV (2nd order poly)	767.2	1078.29	.91	691.67	695.3	.64
Ridge with CV (2nd order poly)	825.21	1091.19	.9	702.55	707.28	.64
Lasso with CV (10th order poly)	-4214.09	1895.22	.92	-10.06	294.34	.94
Ridge with CV (10th order poly)	-2123.59	2095.56	.91	4.42	288.37	.94
Random forest (low regularization)	-104.54	1019.55	.92	-28.83	332.87	.87
Random forest (high regularization)	-110.06	959.96	.95	-21.52	280.36	.94
Gradient boosting (low regularization)	69.44	890.94	.95	7.28	263.62	.95
Gradient boosting (high regularization)	213.04	895.47	.95	174.14	291.63	.93
Neural net	-4706.76	5831.79	.17	-3216.85	3837.37	.15
Meta learners						
Stacking: NNLS	-62.97	1068.87	.84	18.36	269.02	.95
Stacking: Single best	-135.15	1035.41	.89	7.94	263.06	.95
Stacking: NNLS interactive	118.88	1102.99	.95	10.27	280.24	.95
Short-stacking	209.14	915.02	.95	13.67	261.73	.95
Single best	125.64	914.56	.95	7.28	263.62	.95

Simulation II: Small Sample Performance

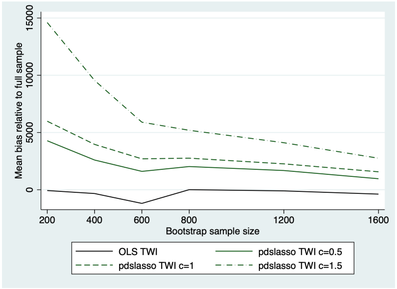
Wüthrich and Zhu (2021, henceforth WZ) consider two applications to demonstrate that PDS-Lasso suffers from a large finite sample bias and tends to underselect; again using the application of (Poterba, Venti, and Wise, 1995; Belloni et al., 2017).

They use two specifications:

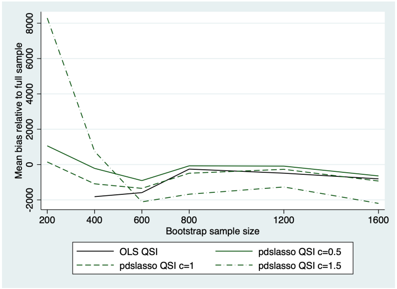
- two-way interactions (TWI) (as in Chernozhukov and Hansen, 2004); $p = 167$
- quadratic splines & interactions (QSI) (as in Belloni et al., 2017); $p = 272$

WZ run their simulations on bootstrap samples of the data ($n_b = \{200, 400, 800, 1600\}$) and calculate the bias as the mean difference to the full sample estimate ($N = 9915$).

Simulation II: Small Sample Performance



(a) Bias (TWI specification)

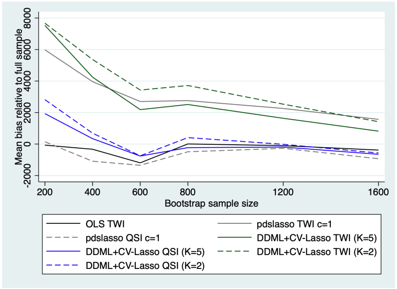


(b) Bias (QSI specification)

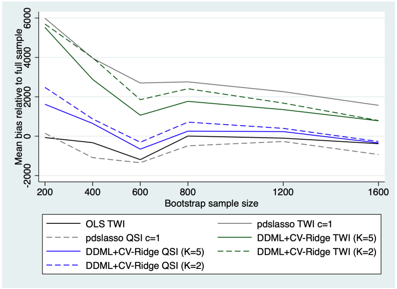
Notes: The figures report the mean bias calculated as the mean difference to the full sample estimates. Following WZ, we draw 600 bootstrap samples of size $n_b = \{200, 400, 600, 800, 1200, 1600\}$. 'TWI' indicates that the predictors have been expanded by two-way interactions. 'QSI' refers to the quadratic spline & interactions specification of Belloni et al. (2017).

Figure: Replication of Figure 8 in WZ

Simulation II: Small Sample Performance



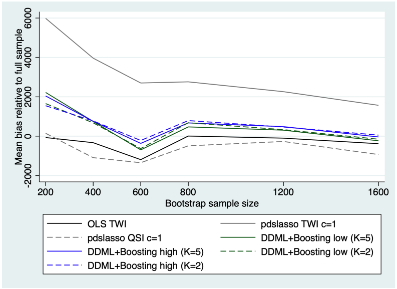
(a) CV-Lasso



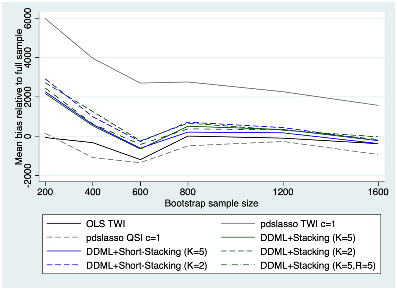
(b) CV-Ridge

Figure: Mean bias relative to full sample

Simulation II: Small Sample Performance



(a) Boosted trees



(b) Stacking

Figure: Mean bias relative to full sample

Main result: The finite sample bias of stacking stabilizes for $n_b > 600$, but in contrast to OLS/PDS-Lasso, stacking doesn't assume linearity.

Summary

- `ddml` implements Double/Debiased Machine Learning for Stata:
    - Compatible with various ML programs in Stata
    - Short (one-line) and flexible multi-line version
    - Uses Stacking Regression as the default machine learner; implemented via separate program `pystacked`
    - 5 models supported
- The advantage to `pdslasso` is that we can make use of almost any machine learner.
- But which machine learner should we use? – We suggest Stacking as it combines multiple ML methods into one prediction.
- We are in the final phase of development; hopefully we can make `ddml` available soon (following your feedback)

References I

Acemoglu, Daron, Simon Johnson, and James A Robinson (Dec. 2001). "The Colonial Origins of Comparative Development: An Empirical Investigation". In: *American Economic Review* 91.5, pp. 1369–1401. URL: http://www.aeaweb.org/articles?id=10.1257/aer.91.5.1369.

Ahrens, Achim, Christian B. Hansen, and Mark E. Schaffer (2020). "lassopack: Model selection and prediction with regularized regression in Stata". In: *The Stata Journal* 20.1, pp. 176–235. URL: https://doi.org/10.1177/1536867X20909697.

Belloni, A et al. (2017). "Program Evaluation and Causal Inference With High-Dimensional Data". In: *Econometrica* 85.1, pp. 233–298. URL: https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA12723.

Belloni, Alexandre, Victor Chernozhukov, and Christian Hansen (2014). "Inference on treatment effects after selection among high-dimensional controls". In: *Review of Economic Studies* 81, pp. 608–650. URL: https://doi.org/10.1093/restud/rdt044.

References II

Belloni, Alexandre et al. (2012). “Sparse Models and Methods for Optimal Instruments With an Application to Eminent Domain”. In: *Econometrica* 80.6. Publisher: Blackwell Publishing Ltd, pp. 2369–2429. URL: http://dx.doi.org/10.3982/ECTA9626.

Belloni, Alexandre et al. (2016). “Inference in High Dimensional Panel Models with an Application to Gun Control”. In: *Journal of Business & Economic Statistics* 34.4. Genre: Methodology, pp. 590–605. URL: https://doi.org/10.1080/07350015.2015.1102733 (visited on 02/14/2015).

Chernozhukov, Victor and Christian Hansen (Aug. 2004). “The effects of 401(K) participation on the wealth distribution: An instrumental quantile regression analysis”. In: *The Review of Economics and Statistics* 86.3. tex.eprint: https://direct.mit.edu/rest/article-pdf/86/3/735/1614135/0034653041811734.pdf, pp. 735–751. URL: https://doi.org/10.1162/0034653041811734.

References III

Chernozhukov, Victor, Christian Hansen, and Martin Spindler (May 2015). "Post-Selection and Post-Regularization Inference in Linear Models with Many Controls and Instruments". In: *American Economic Review* 105.5, pp. 486–490. URL: https://doi.org/10.1257/aer.p20151022.

— (2016). "High-dimensional metrics in r". In: 401, pp. 1–32.

Chernozhukov, Victor et al. (2018). "Double/debiased machine learning for treatment and structural parameters". In: *The Econometrics Journal* 21.1. tex.ids= Chernozhukov2018a publisher: John Wiley & Sons, Ltd (10.1111), pp. C1–C68. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ectj.12097.

Poterba, James M, Steven F Venti, and David A Wise (1995). "Do 401 (k) contributions crowd out other personal saving?" In: *Journal of Public Economics* 58.1, pp. 1–32.

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