Double-debiased machine learning in Stata

Introducing `ddml` and `pystacked`[†]

Achim Ahrens (ETH Zürich)

based on joint work with

Mark E Schaffer (Heriot-Watt University, CEPR & IZA)

Christian B Hansen (University of Chicago)

German Stata Group Meeting
25 June 2021

[†] Preliminary version. Subject to change.

Introduction

- In recent years, machine learning (ML) has increasingly been leveraged in social sciences and economics.
- One central question is how ML can be used for *causal inference*. Two major approaches:
  1. Exploring treatment effect heterogeneity (Causal Forests, GRF, GATES)
  2. Robust inference in the presence of high-dimensional controls and/or instruments

Introduction

- In recent years, machine learning (ML) has increasingly been leveraged in social sciences and economics.
- One central question is how ML can be used for *causal inference*. Two major approaches:
  1. Exploring treatment effect heterogeneity (Causal Forests, GRF, GATES)
  2. Robust inference in the presence of high-dimensional controls and/or instruments $\longleftarrow$ *Today's focus*

Introduction

- In recent years, machine learning (ML) has increasingly been leveraged in social sciences and economics.
- One central question is how ML can be used for *causal inference*. Two major approaches:
  1. Exploring treatment effect heterogeneity (Causal Forests, GRF, GATES)
  2. Robust inference in the presence of high-dimensional controls and/or instruments $\longleftarrow$ *Today's focus*
- We introduce `ddml` for Double-debiased machine learning and `pystacked` for Stacking (a meta-learning algorithm).
- Requirement for fast ML implementations: Stata's Python integration means that we can utilize Python's ML modules.

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.

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.

Background

Motivating example. The partial linear model:

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

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

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, 2019), `dsregress` (StataCorp) and R (`hdm`; Chernozhukov, Hansen, and Spindler, 2016).

Background

These methods have been implemented for Stata in `pdslasso` (Ahrens, Hansen, and Schaffer, 2019), `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.)

Motivation

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)

Motivation

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.

Our contribution

Double-debiased Machine Learning (DDML) due to Chernozhukov, Chetverikov, Demirer, Duflo, Hansen, Newey, and Robins (2018) has been suggested as an extension to Post-double selection. DDML allows for a broad set of machine learners.

Our contribution

Double-debiased Machine Learning (DDML) due to Chernozhukov, Chetverikov, Demirer, Duflo, Hansen, Newey, and Robins (2018) has been suggested as an extension to Post-double selection. DDML allows for a broad set of machine learners.

We introduce `ddml` for Stata:

- Compatible with various ML programs in Stata (e.g. `lassopack`, `pylearn`, `randomforest`).
  - → Any program with the classical "`reg y x`" syntax and post-estimation `predict` will work.
- Short (one-line) and flexible multi-line version
- Uses *Stacking Regression* as the default machine learner (implemented via separate program `pystacked`)
- 5 models supported: partial linear model, interactive model, LATE, partial IV model, optimal IV.

Review of DDML

The partial linear model:

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

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, there is a problem: 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.

Review of DDML

Yet, there is a problem: 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 with the partial linear model

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

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

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`):

Optimal 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`):

LATE 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 $E[D|\boldsymbol{x}_i, z_i = 1]$; $E[z_i|\boldsymbol{x}_i]$.

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; etc.

Stacking regression

Which machine learner should we use?

We suggest *Stacking regression* (Wolpert, 1992) as the *default* machine learner, which we have implemented in the separate program `pystacked` using Python's scikit learn.

Stacking is an ensemble method that combines multiple base learners into one model. As the default, we use *non-negative least squares*:

$$\boldsymbol{w} = \arg\min_{w_j \geq 0} \sum_{i=1}^{n} \left(y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{-i}(\boldsymbol{x}_i) \right)^2,$$

where $\hat{f}_m^{-i}(\boldsymbol{x}_i)$ are cross-validated predictions of base learner m.

pystacked

pystacked implements stacking regression (Wolpert, 1992) via scikit learn's StackingRegressor and StackingClassifier.

`pystacked`

`pystacked` implements stacking regression (Wolpert, 1992) via scikit learn's `StackingRegressor` and `StackingClassifier`.

Syntax:

`pystacked` *depvar* [*indepvars*] [*if*] [*in*] [`type(`*string*`)` `methods(`*string*`)` `finalest(`*string*`)` `folds(`*integer*`)` `voting ...`]

`methods()`	list of ML algorithms in any order separated by spaces
`type()`	*reg(ress)* for regression problems or *class(ify)* for classification problems.

pystacked

`pystacked` implements stacking regression (Wolpert, 1992) via scikit learn's `StackingRegressor` and `StackingClassifier`.

Syntax:

`pystacked` *depvar* [*indepvars*] [*if*] [*in*] [`type(`*string*`)` `methods(`*string*`)` `finalest(`*string*`)` `folds(`*integer*`)` `voting ...`]

`methods()`	list of ML algorithms in any order separated by spaces
`type()`	*reg(ress)* for regression problems or *class(ify)* for classification problems.

`pystacked` supports lasso, elastic net, random forest, gradient boosting and support vector machines.

A brief `pystacked` example

```
. insheet using https://statalasso.github.io/dta/housing.csv, comma
(14 vars, 506 obs)

.
. pystacked medv crim-lstat, ///
>                 type(regress) pyseed(243) method(lassoic rf gradboost)
```

Method	Weight
lassoic	0.0768684
rf	0.0000000
gradboost	0.9231316

A brief pystacked example

```
. insheet using https://statalasso.github.io/dta/housing.csv, comma
(14 vars, 506 obs)

.
. pystacked medv crim-lstat, ///
>                  type(regress) pyseed(243) method(lassoic rf gradboost)
```

Method	Weight
lassoic	0.0768684
rf	0.0000000
gradboost	0.9231316

The `method()` argument can be used to specify the base learners. Here, we use lasso with AIC (`lassoic`), random forest (`rf`) and gradient boosting (gradboost).

A brief `pystacked` example

```
. insheet using https://statalasso.github.io/dta/housing.csv, comma
(14 vars, 506 obs)

.
. pystacked medv crim-lstat, ///
>                 type(regress) pyseed(243) method(lassoic rf gradboost)
```

Method	Weight
lassoic	0.0768684
rf	0.0000000
gradboost	0.9231316

The `method()` argument can be used to specify the base learners. Here, we use lasso with AIC (`lassoic`), random forest (`rf`) and gradient boosting (gradboost).

Due to the non-negativity constraint some base learners are assigned a weight of zero. Also note that weights are standardized to sum to 1.

Back to DDML

The one-line syntax:

`qddml` *depvar* [*indepvars*] [(*controls*)] [(*endog*=*instruments*)] [*if*] [*in*] [*weight*], *model(string)* [`cmd(`*string*`)` `cmdopt(`*string*`)` `kfolds(`*integer*`)` ...]

`cmd()` selects the machine learner (default `pystacked`). `cmdopt()` allows to pass options to the ML program.

Examples:

Partial linear Interactive model IV model Optimal IV model LATE

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*

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

Extended ddml syntax

Step 1: Initialise `ddml` and select model:

`ddml init` *model*

where *model* is either 'partial', 'iv', 'interactive', 'optimaliv', '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` [`, kfolds(`*integer*`)` ...]

This allows to set the number of folds.

Extended `ddml` syntax

Step 3: Cross-fitting

`ddml crossfit` [`, kfolds(`*integer*`)` ...]

This allows to set the number of folds.

Step 4: Estimation of causal effects

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

Extended `ddml` syntax

Step 3: Cross-fitting

`ddml crossfit` [, `kfolds(`*integer*`)` ...]

This allows to set the number of folds.

Step 4: Estimation of causal effects

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

Additional auxiliary commands:

`ddml describe` (describe current model set up), `ddml save`, `ddml use`, `ddml export` (export in csv format).

Extended ddml syntax: Example

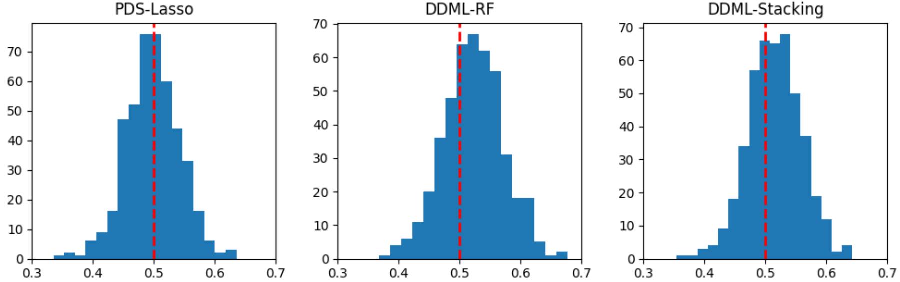
```
. global Y logpgp95
. global X lat_abst edes1975 avelf temp* humid* steplow-oilres
. global D avexpr
.
. *** initialise ddml and select model;
. ddml init partial
.
. *** specify supervised machine learners for E[Y|X] ("yeq") and E[D|X] ("deq")
. * y-equation:
. ddml yeq, gen(pyy): pystacked $Y $X, 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 example

How does DDML perform compared to PDS-lasso?



The simulation is based on an *approximate sparse* design with $\beta_j = (1/j)^2$ and $p = 100$ (which should favor the lasso). DGP

Replication & transparency

Stata's Python integration allows for fast computing and to make use of existing ML programs.

But there are also additional set-up cost and, in particular, *challenges for replications*.

We have to set the seed in both Stata (due to randomization of folds) and Python (due to randomization specific to ML algorithm).

Randomization is involved in the estimation of conditional expectations. To facilitate transparency and replication, estimated conditional expectations can be saved for later inspection or use.

We also need to keep track of package versions of all programs involved (`ddml`, `pystacked`, Python, scikit-learn, etc.)

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.
- More testing & de-bugging needed; 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 (Jan. 16, 2019). *Lassopack: Model Selection and Prediction with Regularized Regression in Stata*. URL: http://arxiv.org/abs/1901.05397 (visited on 01/17/2019).

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.

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

References II

- 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, pp. 590–605. URL: https://doi.org/10.1080/07350015.2015.1102733 (visited on 02/14/2015).
- Chernozhukov, Victor, Chris Hansen, and Martin Spindler (2016). *High-Dimensional Metrics in R*.
- 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.
- Chernozhukov, Victor et al. (2018). “Double/Debiased Machine Learning for Treatment and Structural Parameters”. In: *The Econometrics Journal* 21.1, pp. C1–C68. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ectj.12097.

References III

Tibshirani, Robert (Jan. 1996). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288.

Wolpert, David H. (1992). "Stacked Generalization". In: *Neural Networks* 5.2, pp. 241–259. URL: https://www.sciencedirect.com/science/article/pii/S0893608005800231.

qddml example: Interactive model

```
. webuse cattaneo2, clear
(Excerpt from Cattaneo (2010) Journal of Econometrics 155: 138-154)

.
. global Y bweight

. global D mbsmoke

. global X c.(mmarried mage fbaby medu)#c.(mmarried mage fbaby medu)

.
. qddml $Y $D ($X), model(interactive) cmdopt(method(rf gradboost))
DML with Y0=m0_y, Y1=m0_y and D=m0_d:
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
mbsmoke	-1533.568	45.91291	-33.40	0.000	-1623.556	-1443.581

Back

qddml example: IV model

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

.
. global Y logpgp95

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

. global D avexpr

. global Z logem4

.
. *** now, using one-line command:
. qddml $Y ($X) ($D = $Z), model(iv) cmdopt(method(rf gradboost))
DML with Y=m0_y and D=m0_d1, Z=m0_z1:
```

m0_y	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
m0_d1	.8975094	.2329957	3.85	0.000	.4408462	1.354173

Back

qddml example: LATE

```
. use "http://fmwww.bc.edu/repec/bocode/j/jtpa.dta",clear

.
. gen lnearnings = log(earnings)
(1,332 missing values generated)

. global Y lnearnings

. global D training

. global Z assignmt

. global X sex-age4554

.
. *** now, do the same using one-line command
. qddml $Y ($X) ($D=$Z), model(late) cmdopt(method(rf gradboost))
DML with Y0=m0_y, Y1=m0_y, D0=m0_d, D1=m0_d:
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
training	13.17984	.1269613	103.81	0.000	12.931	13.42868

Back

Simulation example

Approximate sparsity:

$$
\begin{aligned}
y_i &= 0.5 d_i + \boldsymbol{x}_i'(c_y \boldsymbol{\beta}) + \varepsilon_i, & \varepsilon_i &\sim N(0,1) \\
d_i &= \boldsymbol{x}_i'(c_d \boldsymbol{\beta}) + \nu_i, & \nu_i &\sim N(0,1)
\end{aligned}
$$

where β_j is approximately sparse: $\beta_j = (1/j)^2$. c_y and c_d are chosen such that R_y^2 and R_d^2 are 0.8 and 0.2, respectively.
$N = 500$, number of cross-fitting folds=5, $p = 100$.

Back

The LASSO

The LASSO (Least Absolute Shrinkage and Selection Operator, Tibshirani, 1996), "ℓ_1 norm".

Minimize: $$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \boldsymbol{x}_i'\boldsymbol{\beta}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

There's a cost to including lots of regressors, and we can reduce the objective function by throwing out the ones that contribute little to the fit.

The effect of the penalization is that LASSO sets the $\hat{\beta}_j$s for some variables to zero. In other words, it does the *model selection* for us.

In contrast to ℓ_0-norm penalization (AIC, BIC) computationally feasible. Path-wise coordinate descent ('shooting') algorithm allows for fast estimation. Back

Choosing controls: Post-Double-Selection LASSO

Our model is $$y_i = \underbrace{\boxed{\alpha d_i}}_{\text{aim}} + \underbrace{\boxed{\beta_1 x_{i,1} + \ldots + \beta_p x_{i,p}}}_{\text{nuisance}} + \varepsilon_i.$$

- Step 1: Use the LASSO to estimate

$$y_i = \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_j x_{i,j} + \ldots + \beta_p x_{i,p} + \varepsilon_i,$$

  i.e., without d_i as a regressor. Denote the set of LASSO-selected controls by A.

- Step 2: Use the LASSO to estimate

$$d_i = \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_j x_{i,j} + \ldots + \beta_p x_{i,p} + \varepsilon_i,$$

  i.e., where the causal variable of interest is the dependent variable. Denote the set of LASSO-selected controls by B.

- Step 3: Estimate using OLS

$$y_i = \alpha d_i + \mathbf{w}_i' \beta + \varepsilon_i$$

  where $\mathbf{w}_i = A \cup B$, i.e., the union of the selected controls from Steps 1 and 2.

Back