pystacked: Stacking generalization and machine learning in Stata

Achim Ahrens (ETH Zürich)

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

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

May 19, 2022

Italian Stata Group Meeting

Introduction: Stacking

- The machine leaning (ML) toolbox includes a rich set of flexible methods: regularized regression, random forests, SVM, boosting, neural nets.
- When faced with a new prediction or classification task, it is *a priori* rarely obvious which machine learner is best suited for a particular task.
- *Typical approach:*
  - Validating learner based on hold-out sample
  - Cross-validation (K-fold, Leave-one-out, One-step ahead)

  The underlying idea: Select *one* learner as the best.

Introduction: Stacking

This approach seems incomplete: combining several different learners could improve performance.

The idea of *stacking generalization*, or simply *stacking*, is to combine learners (Wolpert, 1992; Breiman, 1996).

General idea:

- Combine a set of "base" (or "level-0") learners using a "final" (or "level-1") estimator.
- It is advisable to include a relatively large and diverse set of base learners to capture different types of pattern in the data.
- Stacking also provides an effective framework for hyper-parameter tuning.

Introduction: Stata's ML tools

There is a growing number of programs for ML in Stata:

- `lassopack` for regularized regression (Ahrens, Hansen, and Schaffer, 2020)
- `rforest` for random forests (Schonlau and Zou, 2020)
- `svm` for support vector machines (Guenther, 2016)
- Cerulli (2021) and Droste (2020) provide an interface to *scikit-learn* (Pedregosa et al., 2011; Buitinck et al., 2013)
- `mlrtime` allows Stata users to make use of R's *parsnip* machine learning library (Huntington-Klein, 2021)

Our contribution: We complement these programs by offering a package that can be used to fit a wide range of machine learners, and for *stacking*.

Introducing pystacked

We introduce `pystacked` for stacking regression and binary classification in Stata.

- `pystacked` allows to fit multiple machine learning algorithms via Python's *scikit-learn* (Pedregosa et al., 2011; Buitinck et al., 2013)[1] and *combine these into one final prediction* as a weighted average of individual predictions.
- `pystacked` can also be used to *fit a single machine learner* and thus provides an easy-to-use and versatile API to *scikit-learn*'s machine learning algorithms.
- Our main motivation for developing `pystacked`: Use it in combination with Double-Debiased Machine Learning (Chernozhukov et al., 2018)

  $\Rightarrow$ **Second talk**

[1] We stress that `pystacked` relies on *scikit-learn* and the on-going work of the *scikit-learn* contributors. We thus suggest that users cite *scikit-learn* along with this article when using `pystacked`.

Stacking regression

Which machine learner should we use?

We don't know whether we have a sparse or dense problem; linear or non-linear; etc.

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

$$\boldsymbol{w} = \arg\min_{w_j \geq 0} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{J} w_j \hat{y}_i^{(j)} \right)^2 ,$$

where $\hat{y}_i^{(j)}$ are cross-validated predictions of base learner j.

Voting regression is a special case with unweighted (or user-specified) weights.

Stacking regression

1. *Cross-validation:*
   1.1 Split the data randomly into K partitions of approximately equal size. These partitions are referred to as *folds*. Denote the set of observations in fold k as I_k, and its complement as I_K^c such that $I_K^c = \{1, ..., n\}/I_k$. I_k constitutes the validation set and I_k^c the training sample.
   1.2 For each fold $k = 1, \ldots, K$ and each base learner $j = 1, \ldots, J$, fit machine learner j to the training data I_k^c and obtain out-of-sample predicted values $\hat{y}_i^{(j)}$ for $i \in I_k$.
2. *Final learner:* Fit the final learner to the full sample. The default choice is non-negative least squares (NNLS):

$$\min_{w_1,\ldots,w_J} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{J} w_j \hat{y}_i^{(j)} \right)^2 \qquad \text{s.t.} \quad w_j \geq 0.$$

   The weights are standardized to sum to 1 after estimation, i.e., $\hat{\hat{w}}_j = \hat{w}_j / \sum_j \hat{w}_j$. The stacking predicted values are defined as $\hat{y}_i^{\star} = \sum_j \hat{\hat{w}}_j \hat{y}_i^{(j)}$.

`pystacked` overview

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

Main features:

- Two alternatives syntaxes
- 10+ different machine learners supported that can be used stand-alone or as base learners in combination with stacking
- Regression+classification
- Graphing and plotting features
- Supports central *scikit-learn* learn pipelines
- Supports sparse matrices and parallelization

(Base) Machine learners

`method()`	`type()`	*Machine learner description*
ols	*regress*	Linear regression
logit	*class*	Logistic regression
lassoic	*regress*	Lasso with AIC/BIC penalty
lassocv	*regress*	Lasso with CV penalty
	class	Logistic lasso with CV penalt
ridgecv	*regress*	Ridge with CV penalty
	class	Logistic ridge with CV penalty
elasticcv	*regress*	Elastic net with CV penalty
	class	Logistic elastic net with CV
svm	*regress*	Support vector regression
	class	Support vector classification
gradboost	*regress*	Gradient boosting regressor
	class	Gradient boosting classifier
rf	*regress*	Random forest regressor
	class	Random forest classifier
linsvm	*class*	Linear SVC
nnet	*regress*	Neural net
	class	Neural net

Note: The first two columns list all allowed combinations of `method(`*string*`)` and `type(`*string*`)`, which are used to select base learners. Column 3 provides a description of each machine learner. 'CV penalty' indicates that the penalty level is chosen to minimize the cross-validated MSPE. 'AIC/BIC penalty' indicates that the penalty level minimizes either either the Akaike or Bayesian information criterion. SVC refers to support vector classification.

Main syntax

Syntax 1:

`pystacked` *depvar predictors* [*if*] [*in*] [, `methods(`*string*`)`
`cmdopt1(`*string*`)` `cmdopt2(`*string*`)` ... `cmdopt10(`*string*`)`
`pipe1(`*string*`)` `pipe2(`*string*`)` ... `pipe10(`*string*`)`
`xvars1(`*predictors*`)` `xvars2(`*predictors*`)` ... `xvars10(`*predictors*`)`
general_options]

Notes:

- `methods(`*string*`)` is used to select base learners, where *string* is a list of base learners.
- Options are passed on to base learners via `cmdopt1(`*string*`)`, `cmdopt2(`*string*`)` to `cmdopt10(`*string*`)`.
- `pipe*(`*string*`)` are for pipelines; `xvars*(`*predictors*`)` allows to specify a learner-specific variable lists of predictors.
- *Limitation:* only 10 base learners supported.

Main syntax

Syntax 2:

`pystacked` *depvar* [*indepvars*] `||` `method`(*string*) `opt`(*string*) `pipe`(*string*) `xvars`(*predictors*) [`||` `method`(*string*) `opt`(*string*) `pipe`(*string*) `xvars`(*predictors*) ... `||`] [*if*] [*in*] [, *general_options*]

Notes:

Base learners are added before the comma using `method`(*string*) along with further learner-specific settings and separated by '`||`'.

Pipelines and learner-specific predictors

Pipelines

scikit-learn uses pipelines to pre-preprocess input data on the fly. In `pystacked`, pipelines can be used to impute missing values, create polynomials and interactions, and to standardize predictors.

Learner-specific predictors

- By default, `pystacked` uses the same set of predictors for each base learner.
- This is often not desirable: For example, when using linear machine learners such as the lasso adding polynomials, interactions and other transformations of the base set of predictors might greatly improve out-of-sample prediction performance.
- *Solution:* Use pipelines or `xvars*(`*predictors*`)`

Demonstration 1: Single base learner

We import the California house price data from Pace and Barry (1997), and split the sample randomly into training and validation partition using a 75/25 split. The aim of the prediction task is to predict median house prices (`medhousevalue`) using a set of house price characteristics

```
. clear all

. use https://statalasso.github.io/dta/cal_housing.dta, clear

. set seed 42

. gen train=runiform()

. replace train=train<.75
(20,640 real changes made)

. replace medh = medh/10e3
variable medhousevalue was long now double
(20,640 real changes made)

. label var medh
```

Demonstration 1: Single base learner

The option `method(gradboost)` selects gradient boosting. We will later see that we can specify more than one learner in `methods()`, and that we can also fit gradient boosted classification trees.

```
. pystacked medh longi-medi if train, type(reg) methods(gradboost)
Single base learner: no stacking done.

Stacking weights:
─────────────────────────────────────────
  Method          │       Weight
──────────────────┼──────────────────────
  gradboost       │       1.0000000
. predict double yhat_gb1 if !train
```

The output shows the stacking weights associated with each base learner. Since we only consider one method, the output is not particularly informative and simply shows a weight of one for gradient boosting. Yet, `pystacked` has fitted 100 boosted trees (the default) in the background!

Demonstration 1: Single base learner

Here, we compare lasso with and without the *poly2* pipeline:

```
. pystacked medh longi-medi if train, type(reg) methods(lassocv)
Single base learner: no stacking done.

Stacking weights:
```

Method	Weight
lassocv	1.0000000

```
. predict double yhat_lasso1 if !train

.
. pystacked medh longi-medi if train, type(reg) methods(lassocv) ///
>     pipe1(poly2)
Single base learner: no stacking done.

Stacking weights:
```

Method	Weight
lassocv	1.0000000

```
. predict double yhat_lasso2 if !train
```

Demonstration 2: Stacking regression

We now consider a stacking regression application with five base learners:

1. linear regression,
2. lasso with penalty chosen by cross-validation,
3. lasso with second order polynomials and interactions,
4. random forest with default settings,
5. gradient boosting with a learning rate of 0.01 and 1000 trees.

Demonstration 2: Stacking regression

Syntax 1:

```
. set seed 42

. pystacked medh longi-medi if train,                              ///
>     type(regress)                                                ///
>     methods(ols lassocv lassocv rf gradboost)                    ///
>     pipe3(poly2) cmdopt5(learning_rate(0.01)                     ///
>     n_estimators(1000))

Stacking weights:
```

Method	Weight
ols	0.0000000
lassocv	0.0000000
lassocv	0.4687747
rf	0.2508847
gradboost	0.2803406

Demonstration 2: Stacking regression

Syntax 2:

```
. set seed 42

. pystacked medh longi-medi                                            || ///
>     m(ols)                                                           || ///
>     m(lassocv)                                                       || ///
>     m(lassocv) pipe(poly2)                                           || ///
>     m(rf)                                                            || ///
>     m(gradboost) opt(learning_rate(0.01) n_estimators(1000))            ///
>     if train, type(regress)

Stacking weights:
```

Method	Weight
ols	0.0000000
lassocv	0.0000000
lassocv	0.4687747
rf	0.2508847
gradboost	0.2803406

Demonstration 2: Stacking regression

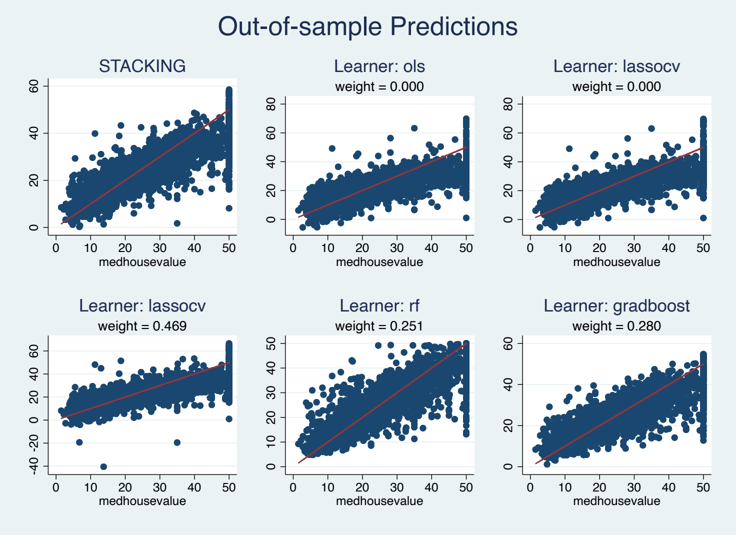
Predicted values. In addition to the stacking predicted values, we can also get the predicted values of each base learner using the `transform` option:

```
. predict double yhat, xb
. predict double ytrans, transf
. list yhat ytrans* if _n <= 5
```

	yhat	ytrans1	ytrans2	ytrans3	ytrans4	ytrans5
1.	41.357332	41.315834	41.24048	40.36963	43.16083	41.394926
2.	42.876817	41.45306	41.434102	46.420851	38.289107	41.056291
3.	39.222298	38.212036	38.176811	39.068403	41.268902	37.648071
4.	33.686191	32.332498	32.291791	33.645113	33.4869	33.933232
5.	25.253149	25.382839	25.374455	25.608601	23.8597	25.905815

Demonstration 2: Stacking regression

Plotting. The `graph` option creates a scatter plot of predicted versus observed values for stacking and each base learner:



Demonstration 2: Stacking regression

MSPE table. The `table` option allows to compare stacking weights with in-sample and out-of-sample MSPE. As with the `graph` option, we can use `table` as a post-estimation command:

```
. pystacked, table holdout
Number of holdout observations: 5192

MSPE: In-Sample and Out-of-Sample
```

Method	Weight	In-Sample	Out-of-Sample
STACKING	.	4.793	5.472
ols	0.000	6.986	6.853
lassocv	0.000	6.986	6.855
lassocv	0.469	6.613	6.564
rf	0.251	1.847	4.963
gradboost	0.280	5.312	5.511

Summary

- `pystacked` implements *stacked generalization* (Wolpert, 1992) for regression and binary classification via Python's *scikit-learn*.
- Stacking combines multiple supervised machine learners—the "base" or "level-0" learners—into a single learner.
- The currently *supported (base) machine learners* include regularized regression, random forest, gradient boosting, support vector machines and feed-forward neural nets (multi-layer perceptron).
- `pystacked` can also be used with as a 'regular' machine learning program to fit a single base learner and, thus, provides an easy-to-use API for *scikit-learn*'s machine learning algorithms.

References I

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.

Breiman, Leo (July 1996). "Stacked regressions". en. In: *Machine Learning* 24.1, pp. 49–64. URL: http://link.springer.com/10.1007/BF00117832 (visited on 12/04/2021).

Buitinck, Lars et al. (2013). "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122.

Cerulli, Giovanni (2021). *Machine Learning using Stata/Python*.

References II

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.

Droste, Michael (2020). *pylearn*. https://github.com/NickCH-K/MLRtime/. [Online; accessed 02-December-2021].

Guenther, N. (2016). "Support vector machines". In: *Stata Journal* 16.4, 917–937(21). URL: www.stata-journal.com/article.html?article=st0461.

Huntington-Klein, Nick C. (2021). *mlrtime*. https://github.com/mdroste/stata-pylearn/. [Online; accessed 02-December-2021].

References III

Pace, R. Kelley and Ronald Barry (1997). "Sparse spatial autoregressions". In: *Statistics & Probability Letters* 33.3, pp. 291–297. URL: https://www.sciencedirect.com/science/article/pii/S016771529600140X.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Schonlau, Matthias and Rosie Yuyan Zou (2020). "The random forest algorithm for statistical learning". In: *The Stata Journal* 20.1, pp. 3–29. URL: https://doi.org/10.1177/1536867X20909688.

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