

UK Stata Conference 2023
London, 7-8 September

A review of machine learning commands in Stata

Performance and usability evaluation

Giovanni Cerulli

IRCrES-CNR

Research Institute on Sustainable Economic Growth

National Research Council of Italy

Machine Learning

Definition, relevance, applications

What is **Machine Learning** ?

Machine Learning

A relatively new approach to **data analytics**, which places itself in the intersection between **statistics**, **computer science**, and **artificial intelligence**

ML objective

Turning **information** into **knowledge** and **value** by “letting the data speak”

ML purposes

Limiting
prior assumptions

Model-free
philosophy

Based on **algorithm**
computation, graphics

Mostly focused on
prediction than
inference

Targeted to
Big Data

Targeted to **complexity**
reduction

ML analyses

Prediction

**Feature-importance
detection**

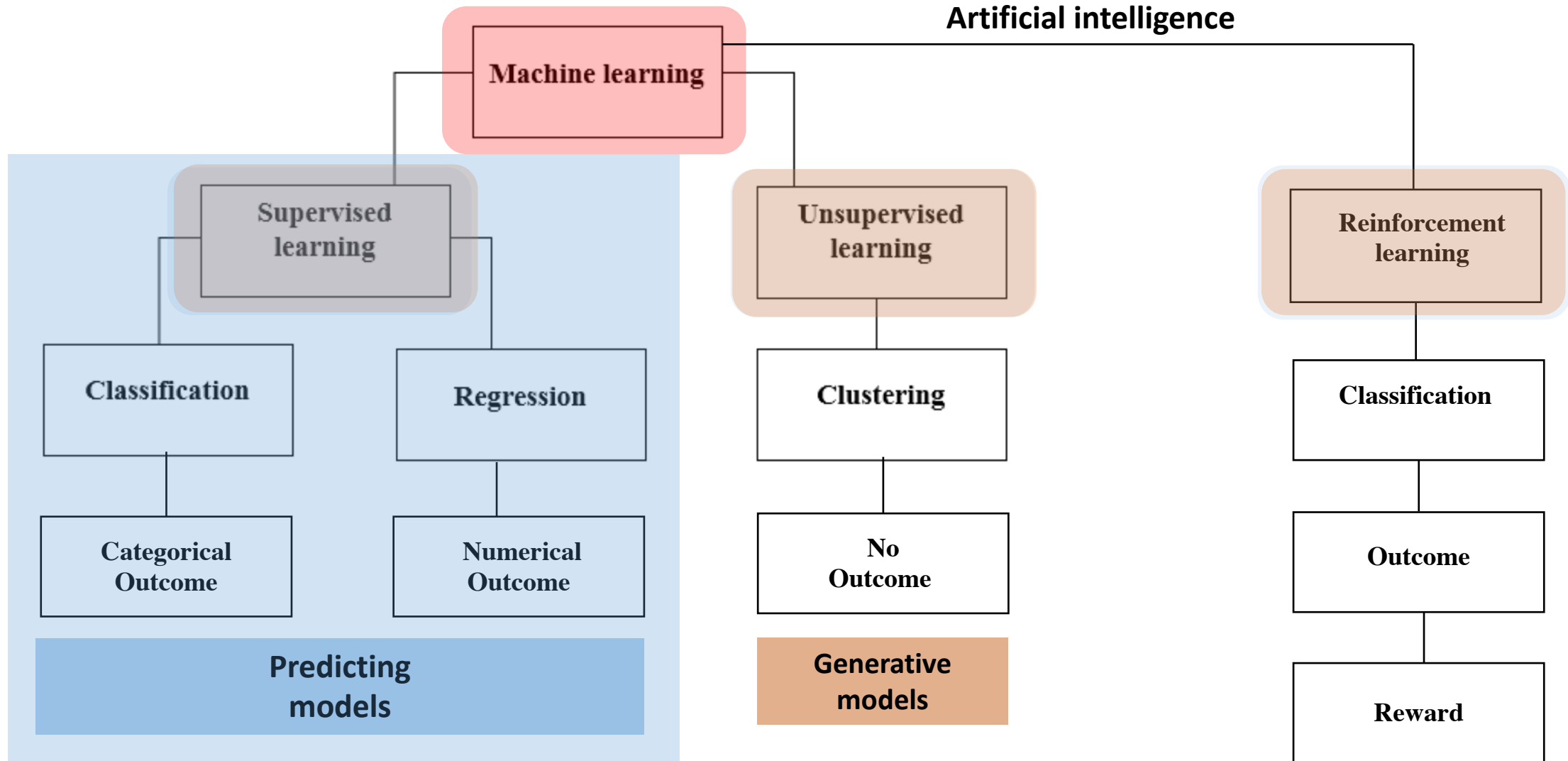
**Signal-from-noise
extraction**

**Correct specification
via model selection**

**Model-free
classification**

**Model-free
clustering**

Supervised, Unsupervised, Reinforcement Learning



Machine Learning application examples

Identifying
risk factors
for prostate
cancer

Predicting heart
attack by
demographic,
diet and clinical
measurements

Customizing
email spam
detection
system

Predict stock
market price
variation

Self-driving
cars

Classifying
pixels in a
land-satellite
images

Establishing the
relationship
between salary
and many of its
determinants

Pattern
recognition
of
handwritten
symbols

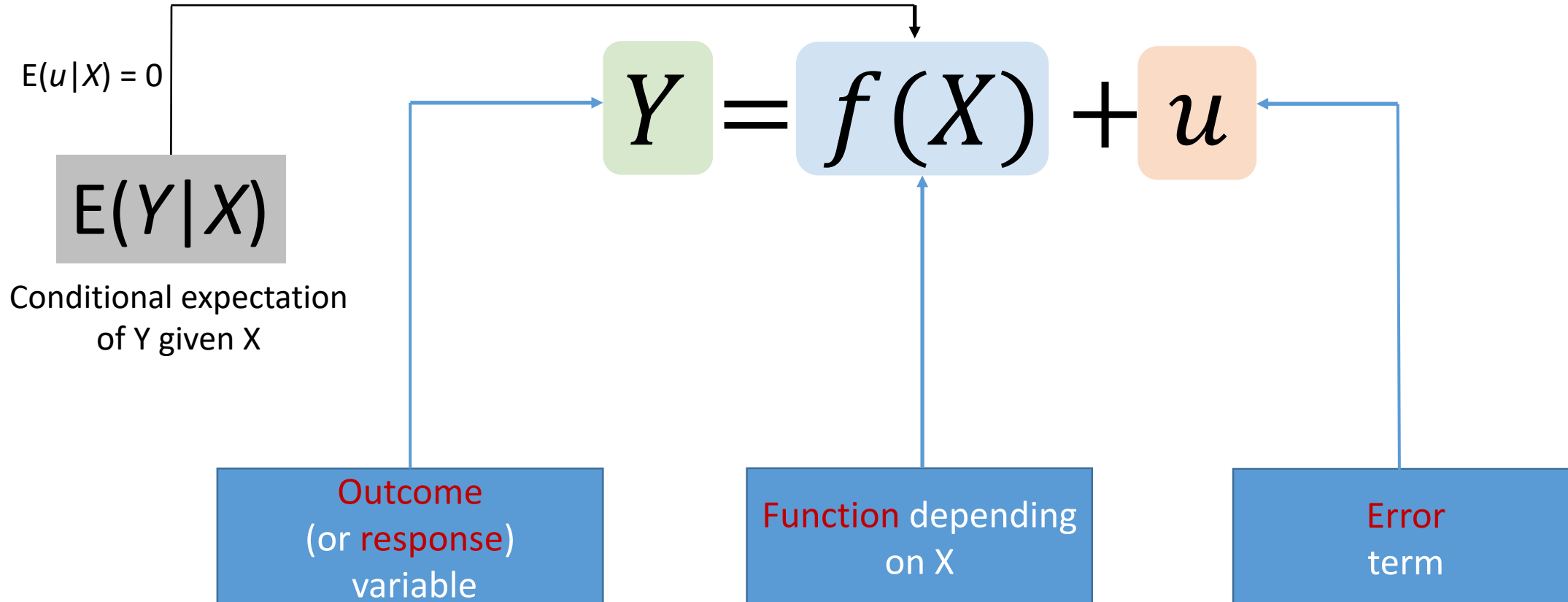
Automated
languages
translators
(Google
Translate)

Vocal
recognition
systems
(Amazon
Alexa)

The **basics** of Machine Learning

Modelling as “learning”

More generally, suppose that we observe a quantitative response Y and p different predictors, X_1, X_2, \dots, X_p . We assume that there is some relationship between Y and $X = (X_1, X_2, \dots, X_p)$, which can be written in the very general form



Reducible and irreducible prediction errors

Consider a given estimate \hat{f} and a set of predictors X , which yields the prediction $\hat{Y} = \hat{f}(X)$. Assume for a moment that both \hat{f} and X are fixed. Then, it is easy to show that

$$\begin{aligned} E(Y - \hat{Y})^2 &= E[f(X) + \epsilon - \hat{f}(X)]^2 \\ &= \underbrace{[f(X) - \hat{f}(X)]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}}, \end{aligned}$$

where $E(Y - \hat{Y})^2$ represents the average, or *expected value*, of the squared difference between the predicted and actual value of Y , and $\text{Var}(\epsilon)$ represents the *variance* associated with the error term ϵ .

Machine Learning



Techniques for estimating f with the aim of
minimizing the **reducible error**



$$f(X) = E(Y|X)$$

The ML jargon

| STATISTICS | MACHINE LEARNING |
|-----------------------------------|-------------------------|
| <i>Statistical model</i> | <i>Learner</i> |
| <i>Estimation sample</i> | <i>Training dataset</i> |
| <i>Out-of-sample observations</i> | <i>Test dataset</i> |
| <i>Estimation method</i> | <i>Algorithm</i> |
| <i>Observation</i> | <i>Instance</i> |
| <i>Predictor</i> | <i>Feature</i> |
| <i>Dependent variable</i> | <i>Target</i> |

Assessing model **predictive accuracy**

Evaluating the **performance** of a **statistical learning method** on a given dataset

Quantifying whether the **predicted response** value for a given observation is close to the **true response** value for that observation

Commonly-used measure is the **Mean Squared Error (MSE)**, given by:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2,$$

Training error vs. Test error

- The *test error* is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the *training error* can be easily calculated by applying the statistical learning method to the observations used in its training.
- But the training error rate often is quite different from the test error rate, and in particular the former can *dramatically underestimate* the latter.

Train-MSE vs Test-MSE

Training dataset

N in-sample available observations



$$\text{Tr} = \{x_i, y_i\}_1^N$$



$$\text{MSE}_{\text{Tr}} = \text{Ave}_{i \in \text{Tr}} [y_i - \hat{f}(x_i)]^2$$



Overfitting as flexibility increases

Testing dataset

M out-of-sample observations



$$\text{Te} = \{x_i, y_i\}_1^M$$

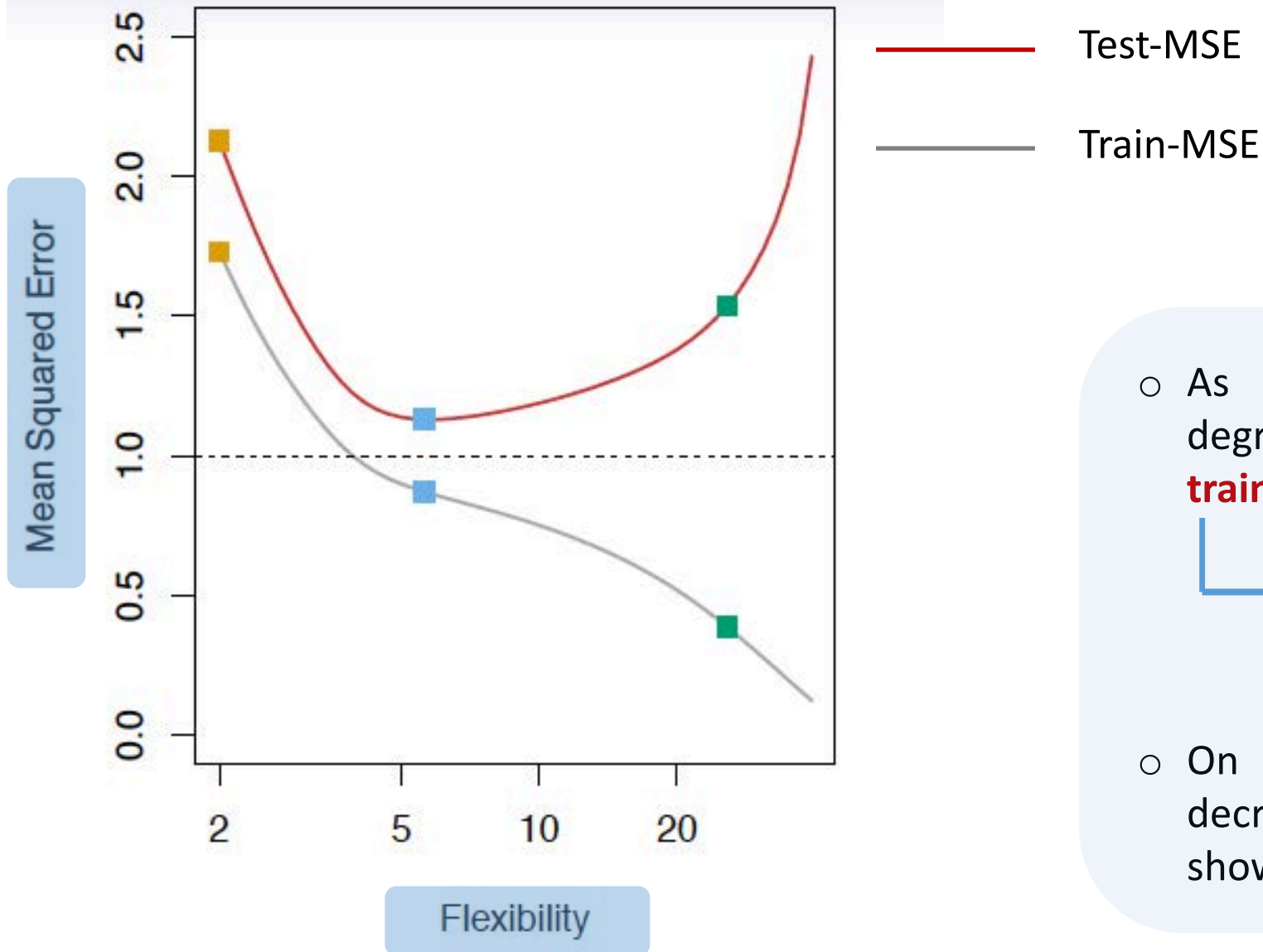


$$\text{MSE}_{\text{Te}} = \text{Ave}_{i \in \text{Te}} [y_i - \hat{f}(x_i)]^2$$



True fitting accuracy

Train-MSE overfitting



- As long as model **flexibility** (i.e., degree-of-freedom) increases, the **train-MSE decreases** monotonically.



This phenomenon is called **overfitting**

- On the contrary, the **test-MSE** first decreases, and then increases, thus showing a **minimum**

Decomposition of the Test-MSE

Suppose we have fit a model $\hat{f}(x)$ to some training data Tr , and let (x_0, y_0) be a test observation drawn from the population. If the true model is $Y = f(X) + \epsilon$ (with $f(x) = E(Y|X = x)$), then

$$E \left(y_0 - \hat{f}(x_0) \right)^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

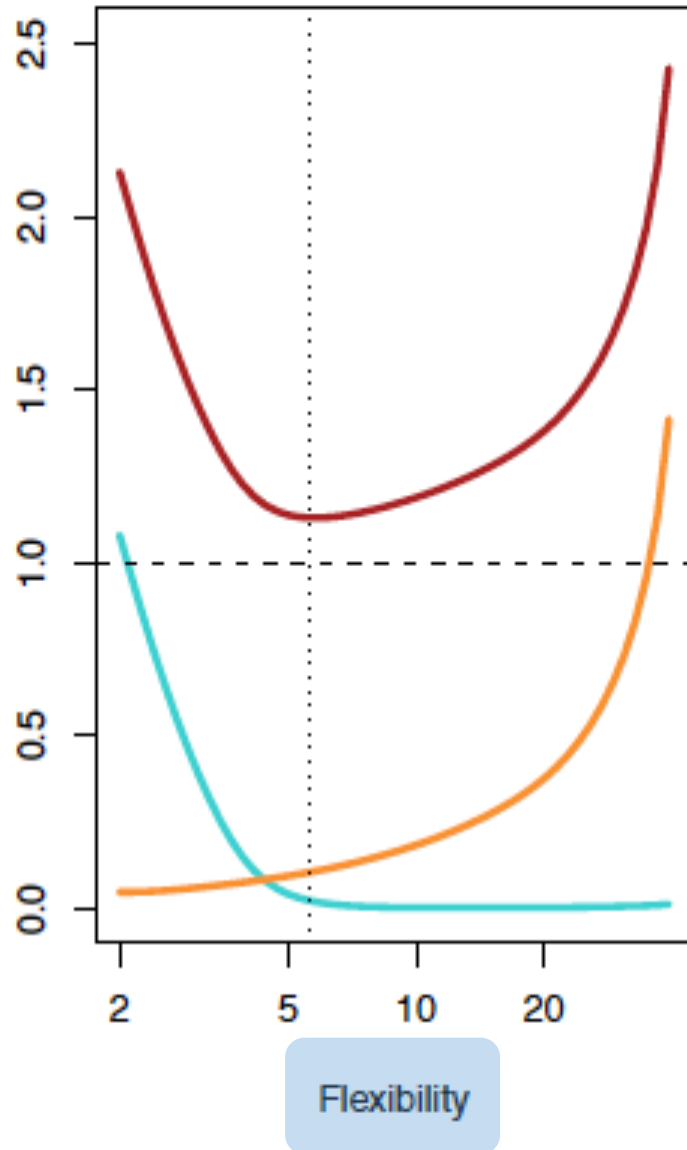
Test-MSE

Variance
of the specific
ML method

Bias square
of the specific
ML method

Variance of the
Irreducible
error term

The variance-bias trade-off



Typically as the *flexibility* of \hat{f} increases, its variance increases, and its bias decreases. So choosing the flexibility based on average test error amounts to a *bias-variance trade-off*.

- MSE
- Bias
- Var
- Error variance

Observe that the error variance represents a **lower bound** for the **Test-MSE**

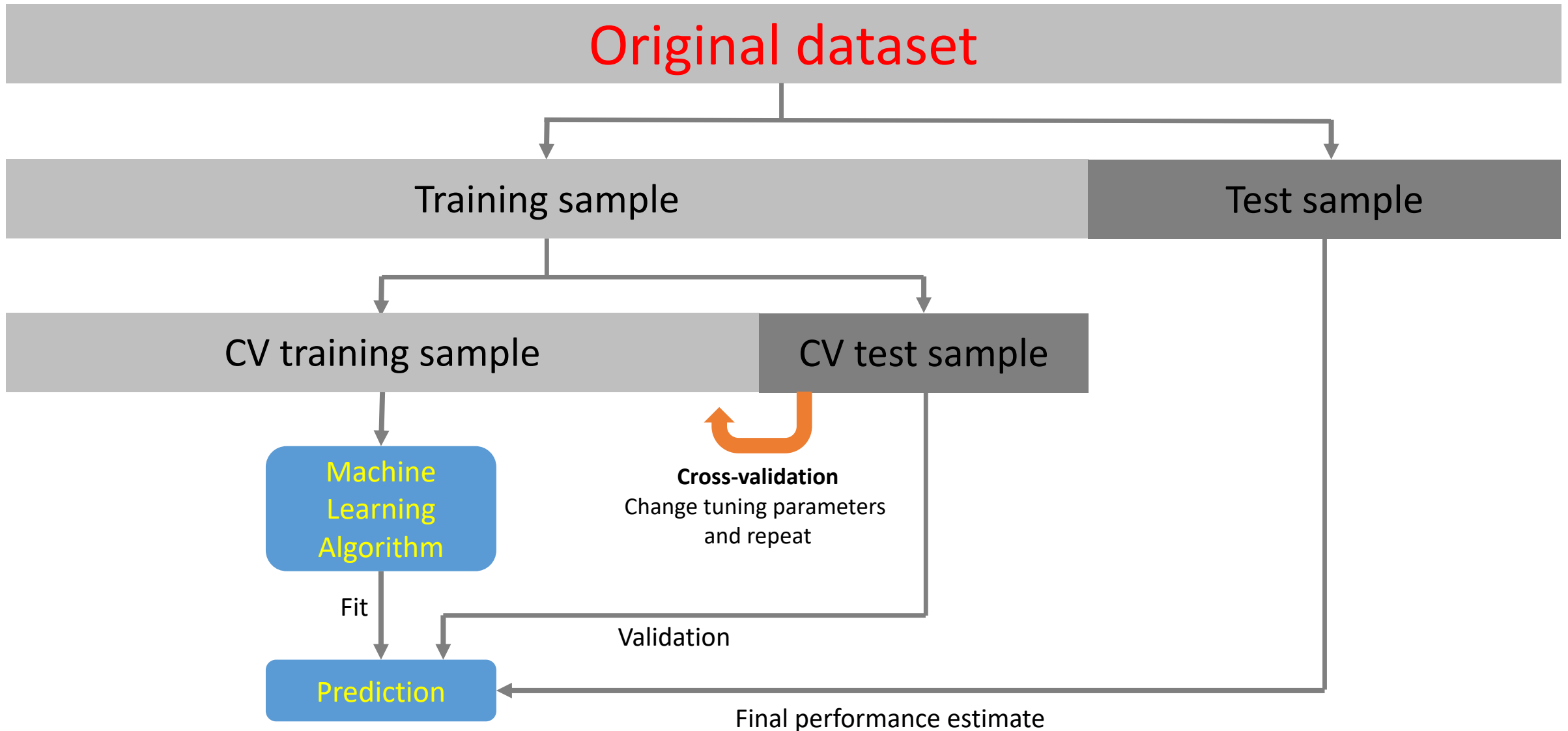
What is **optimal tuning**?

Any learner is characterized by one or more **hyper-parameters** λ controlling for model **flexibility** :

$$Y = f(X , \lambda)$$

Optimal tuning means to find the λ^* that **minimizes the test error** among all possible λ

Model optimal tuning for prediction



Tuning parameters of ML methods

| ML method | Tuning parameter 1 | Tuning parameter 2 | Tuning parameter 3 |
|------------------------|-------------------------------------|---------------------------------|--------------------|
| Linear Models and GLS | <i>N. of covariates</i> | | |
| Lasso | <i>Penalization coefficient</i> | | |
| Elastic-Net | <i>Penalization coefficient</i> | <i>Elastic parameter</i> | |
| Nearest-Neighbor | <i>N. of neighbors</i> | | |
| Neural Network | <i>N. of hidden layers</i> | <i>N. of neurons</i> | |
| Trees | <i>N. of leaves (or tree-depth)</i> | | |
| Boosting | <i>Learning parameter</i> | <i>N. of sequential trees</i> | <i>Tree-depth</i> |
| Random Forest | <i>N. of features for splitting</i> | <i>N. of bootstrapped trees</i> | <i>Tree-depth</i> |
| Bagging | <i>Tree-depth</i> | <i>N. of bootstrapped trees</i> | |
| Support Vector Machine | <i>C</i> | <i>Gamma</i> | |
| Kernel regression | <i>Bandwidth</i> | <i>Kernel type</i> | |
| Piecewise regression | <i>N. of knots</i> | | |
| Series regression | <i>N. of series terms</i> | | |

Software

Software



General purpose
ML platform



Deep Learning
platform



Deep Learning
platform



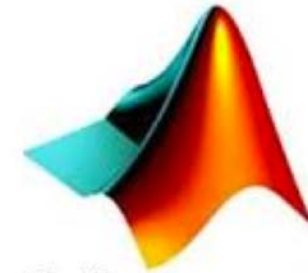
Software



**Python/Stata fully
integrated platform via
the SFI environment**



**Various ML packages but
poor deep learning
libraries
(CARET library)**



MATLAB



**Statistics and Machine Learning Toolbox
Deep Learning Toolbox**

Supervised Machine learning in Stata

First-generation stand-alone commands

- rforest
- boost (only for Windows)
- svm
- sctree, srtree
- subset
- mlp2

Regularized regression/classification in Stata

- Stata Corp LASSO package
- LASSOPACK

Second-generation general purpose ML commands (based on *Python's Scikit-learn*)

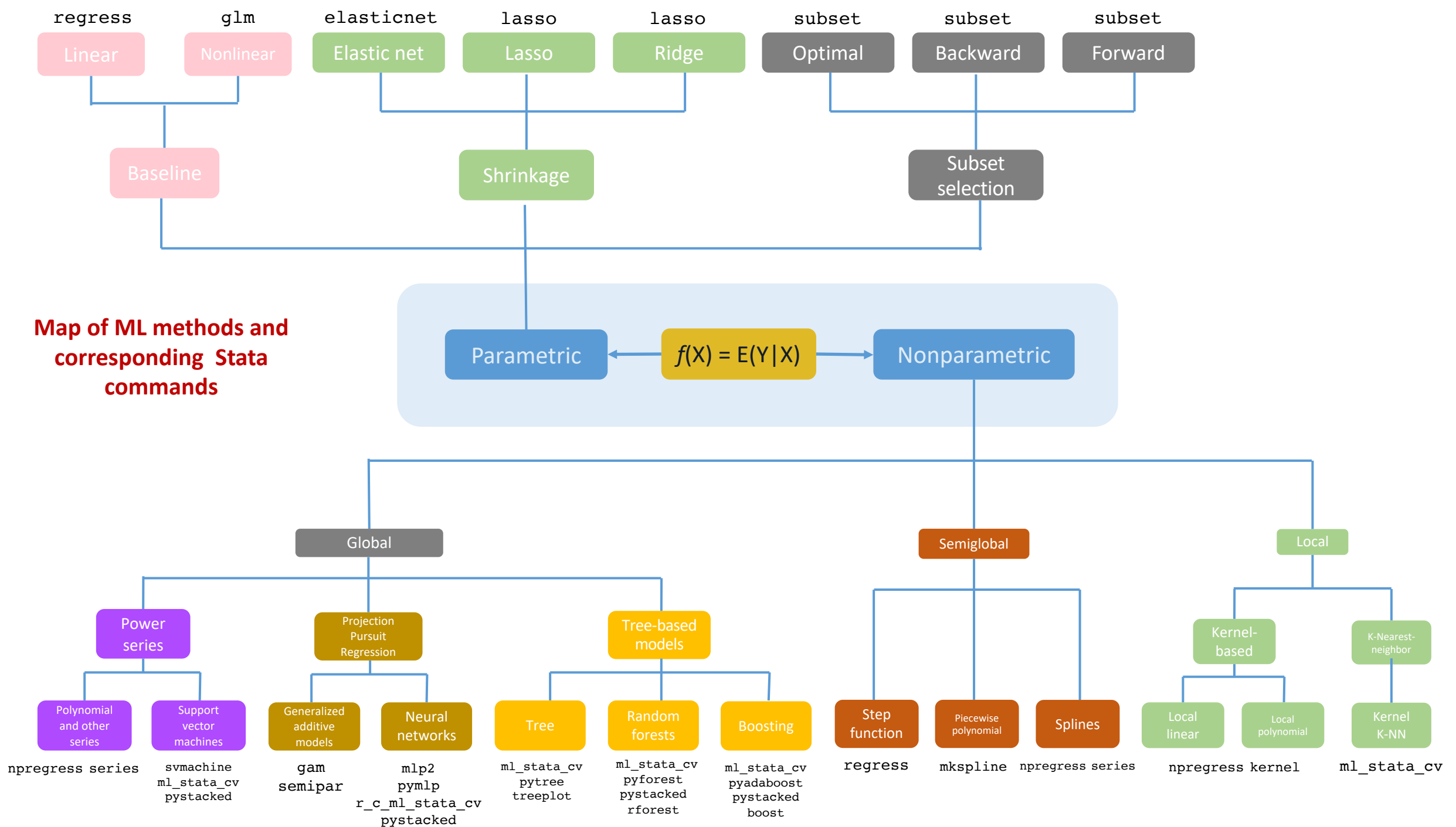
- pylearn
- pystacked
- r_ml_stata_cv and c_ml_stata_cv

ML for causal inference

- Lasso-based causal inference
- General ML causal inference (double-debiased ML)

ML hyperparameters' tuning

- gridsearch
- r_ml_stata_cv and c_ml_stata_cv



First-generation stand-alone commands

- `rforest`
- `svm`
- `boost` (only for Windows)
- `sctree`, `srtree`
- `subset`
- `mlp2`

The syntax to fit a random forest model is

```
rforest depvar indepvars [if] [in] [, type(string) iterations(int)
    numvars(int) depth(int) lsize(int) variance(real) seed(int)
    numdecimalplaces(int) ]
```

with the following postestimation command:

```
predict newvar|varlist|stub* [if] [in] [, pr]
```

type(str)

The type of decision tree. Must be one of "class" (classification) or "reg" (regression).

iterations(int)

Set the number of iterations (trees), default to 100 if not specified.

numvars(int)

Set the number of variables to randomly investigate, default to sqrt(number of indepvars).

depth(int)

Set the maximum depth of the random forest, default to 0 for unlimited, if not specified.

The software development in Stata was built on top of the Weka Java implementation, which was developed by the University of Waikato.

svmachines

(Schonlau and Guenther, 2016)

The full syntax of the command to fit a SVM model is as follows:

```
svmachines depvar indepvars [if] [in] [, type(type) kernel(kernel) c(#)
  epsilon(#) nu(#) gamma(#) coef0(#) degree(#) shrinking probability
  sv(newvar) tolerance(#) verbose cache_size(#)]
```

The most interesting thing a fitted machine-learning model can do is predict response values. To that end, the standard `predict` command may be used during postestimation as follows:

```
predict newvar [if] [in] [, probability scores verbose]
```

This command is a wrapper for Python's `libsvm`

```
boost varlist [if] [in], distribution(string) maxiter(#) [influence  
predict(varname) shrink(#) bag(#) trainfraction(#) interaction(#)  
seed(#)]
```

boost is implemented as a Windows C++ plugin.

sctree and srtree (Cerulli, 2019)

sctree— Implementing classification trees via optimal pruning, bagging, random forests, and boosting methods

Syntax

```
sctree outcome [varlist] [if] [in] model(modeltype) rversion(R_version) [prune(integer) cv_tree prediction(new_data_filename)  
  in_samp_data(filename) out_samp_data(filename) ntree(integer) mtry(integer) inter_depth(integer) shrinkage(number)  
  pdp(string) seed(integer)
```

Description

sctree is a Stata wrapper for the R functions "tree()", "randomForest()", and "gbm()". It allows to implement the following classification tree models: (1) classification tree with optimal pruning, (2) bagging, (3) random forests, and (4) boosting.

Based on R

modeltype_options

Description

Model

tree

Simple classification tree model

randomforests

bagging and random forest models

boosting1

Boosting model with a binary outcome (i.e., $y=0,1$)

boosting2

Boosting model with a multinomial outcome (e.g., $y=A,B,C$)

subset

(Cerulli, 2019)

subset— Implementing covariates best and stepwise subset selection

Syntax

```
subset outcome [varlist] [if] [in], model(modeltype) rversion(R_version) [nvmax(number) index_values(filename)  
      matrix_results(filename) optimal_vars(filename)
```

Description

subset is a Stata wrapper for the R function "regsubsets()", providing "best", "backward", and "forward" stepwise subset covariates selection, a Machine Learning approach to select the optimal number of features (covariates) in a supervised linear learning approach (i.e. a linear regression model) with many covariates. The "forward" model can be also used when p (the number of covariates) is larger than N (the sample size). This method provides both the optimal subset of covariates for each specific size of the model (i.e., size=1 covariates, size=2 covariates, etc.), and the overall optimal size. The latter one is found using three criteria as validation approaches: Adjusted R², CP, and BIC.

Based on R

| <i>modeltype_options</i> | Description |
|--------------------------|-----------------------------|
| Model | |
| best_subset | Best subset selection |
| backward | Backward stepwise selection |
| forward | Forward stepwise selection |

mlp2 — Multilayer perceptron with 2 hidden layers

```
mlp2 fit depvar indepvars [if] [in] [, fit_options]
```

Programmed in Mata

depvar is a categorical or continuous variable. The list *indepvars* cannot be empty.

| <i>options</i> | Description |
|---------------------------------|---|
| layer1(#) | numbers of neurons in the 1-st hidden layer; default is the number of levels of <i>depvar</i> |
| layer2(#) | numbers of neurons in the 2-nd hidden layer; default is level1 |
| nobias | no bias terms are used |
| <u>optimizer</u>(string) | optimizer; default is optimizer(gd) |
| loss(string) | loss function; default depends on <i>depvar</i> |
| initvar(#) | initializing variance factor; default is initvar(1) |
| <u>restarts</u>(#) | maximum number of restarts; default is restarts(10) |
| <u>lrate</u>(#) | learning rate of the optimizer; default is lrate(0.1) |
| <u>friction</u>(#) | target friction for momentum optimizers; default is friction(0.9) |
| fricrate(#) | friction rate for momentum optimizers; default is fricrate(0.5) |
| <u>epsilon</u>(#) | gradient smoothing term; default is epsilon(1e-8) |
| decay(#) | decay parameter of RMSProp optimizer; default is decay(0.9) |
| <u>losstol</u>(#) | stopping loss tolerance; default is losstol(1e-4) |
| dropout1(#) | 1st hidden layer dropout probability; default is dropout1(0) |
| dropout2(#) | 2nd hidden layer dropout probability; default is dropout2(0) |
| batch(#) | training batch size; default is batch(50) or entire sample |
| epochs(#) | maximum number of iterations; default is epochs(100) |
| echo(#) | report loss values at every # number of iterations; default is echo(0) |

PROS

- All these commands are valuable commands for implementing in Stata specific ML methods
- **rforest** and **boost** allow also for *factor importance*
- **sctree** and **srtree** produce a tree plot (also with optimal pruning)
- **mlp2** is the directly programmed in Mata

CONS

- Mainly wrappers for R, Java, C++, and Python (not SFI)
- All these commands are not very well suited for the optimal tuning of the hyper-parameters
- For optimal tuning, **rforest** and **boost** can use **gridsearch** which has however limitations
- **Boost** runs only under Windows
- **Subset** only consider linear models (no GLM implemented)
- **mlp2** considers only 2 layers and is not suited for the optimal tuning of the hyper-parameters

Regularized regression/classification in Stata

- LASSOPACK
- Stata Corp LASSO package

Stata implementation via LASSOPACK

(Ahrens, Hansen, and Schaffer, 2020)

LASSOPACK includes three commands: `lasso2` implements LASSO and related estimators. `cvlasso` supports cross-validation, and `rlasso` offers the 'rigorous' (theory-driven) approach to penalization.

Basic syntax

```
lasso2 depvar indepvars [if] [in] [, ...]
```

```
cvlasso depvar indepvars [if] [in] [, ...]
```

```
rlasso depvar indepvars [if] [in] [, ...]
```

Stata 18 built-in commands **lasso/elasticnet**

| <i>Basic regularized regression commands</i> | | | | | |
|--|--------------------|----------------------|-------------------------|----------------------|-----------------------|
| Model | Lasso | | Elasticnet | | Square-root Lasso |
| Linear | <code>lasso</code> | <code>linear</code> | <code>elasticnet</code> | <code>linear</code> | <code>sqrlasso</code> |
| Probit | <code>lasso</code> | <code>probit</code> | <code>elasticnet</code> | <code>probit</code> | |
| Logit | <code>lasso</code> | <code>logit</code> | <code>elasticnet</code> | <code>logit</code> | |
| Poisson | <code>lasso</code> | <code>poisson</code> | <code>elasticnet</code> | <code>poisson</code> | |

Lasso for Cox proportional hazards models

lasso cox and **elasticnet cox** expand the existing LASSO suite for prediction and model selection to include a high-dimensional semiparametric Cox proportional hazards model.

Stata 18 built-in command **lasso**

```
lasso model depvar [ (alwaysvars) ] othervars [if] [in] [weight] [ , options ]
```

model is one of linear, logit, probit, or poisson.

alwaysvars are variables that are always included in the model.

othervars are variables that lasso will choose to include in or exclude from the model.

Examples



```
lasso linear y1 (x1 x2) x3-x100
```

```
lasso logit y2 x1-x100 rseed(1234)
```

Lasso in Stata (post-estimation commands)

| Command | Description |
|--------------------------|--|
| <code>bicplot</code> | plot Bayesian information criterion function |
| <code>coefpath</code> | plot path of coefficients |
| <code>cvplot</code> | plot cross-validation function |
| <code>lassocoef</code> | display selected coefficients |
| <code>lassogof</code> | goodness of fit after lasso for prediction |
| <code>lassoinfo</code> | information about lasso estimation results |
| <code>lassoknots</code> | knot table of coefficient selection and measures of fit |
| <code>lassoselect</code> | select alternative λ^* (and α^* for <code>elasticnet</code>) |

PROS

- Both LASSOPACK and LASSO are flexible packages to implement regularized regression
- Both use three optimal-tuning strategies:
 - Information criteria
 - Plug-in
 - Cross-validation
- Both have useful post-estimation commands (including `predict`)
- Both have useful graphical representations of results
 - Lasso coefficient-path plot
 - Cross-validation optimal tuning plot

CONS

- Both do not estimate `multinomial` lasso/elasticnet
- Absent or not flexible time-series cross-validation for optimal tuning
 - LASSOPACK has time-series/panel-data cross-validation available, but it is poorly flexible and computationally *slow*

Second-generation general-purpose ML commands

- `pylearn`
- `pystacked`
- `r_ml_stata_cv` and `c_ml_stata_cv`

pylearn - Supervised learning algorithms in Stata based on the **Scikit-learn** library of Python.

pylearn is a set of Stata commands to perform supervised learning in Stata. These commands all exhibit a common Stata-like syntax for model estimation and post-estimation (i.e., they look very similar to `regress`). **pylearn** currently includes these models:

[R] **pytree** estimates decision trees.

[R] **pyforest** estimates random forests.

[R] **pymlp** estimates multi-layer perceptrons (feed-forward neural networks).

[R] **pyadaboost** estimates adaptive boosted trees/regressions (AdaBoost).

[R] **pygradboost** estimates gradient boosted trees.

pytree - example

pytree — Decision tree regression and classification with Python and scikit-learn

Syntax

```
pytree depvar indepvars [if] [in], type(string) [options]
```

| <i>options</i> | Description |
|--|--|
| <hr/> | |
| Main | |
| type (<i>string</i>) | <i>string</i> may be regress or classify . |
| Pre-processing | |
| training (<i>varname</i>) | <i>varname</i> is an indicator for the training sample |
| Decision tree options | |
| criterion (<i>string</i>) | Criterion for splitting nodes (see details below) |
| max_depth (#) | Maximum tree depth |
| min_samples_split (#) | Minimum observations per node |
| min_weight_fraction_leaf (#) | Min fraction at leaf |
| max_features (<i>numeric</i>) | Maximum number of features to consider per tree |
| max_leaf_nodes (#) | Maximum leaf nodes |
| min_impurity_decrease (#) | Propensity to split |

pystacked -- Stata program for Stacking Regression

- pystacked implements stacking regression (Wolpert, 1992) via Scikit-learn's modules:
`sklearn.ensemble.StackingRegressor`
`sklearn.ensemble.StackingClassifier`
- Stacking is a way of combining multiple supervised machine learners (the "base" or "level-0 learners) into a meta learner.
- The currently supported base learners are: *linear regression, logit, lasso, ridge, elastic-net, (linear) support vector machines, gradient boosting, and neural-nets (MLP)*
- **pystacked** can also be used with a single base learner and, thus, provides an easy-to-use API for Scikit-learn's machine learning algorithms

pystacked - syntax

```
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.

pystacked - learners

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

`r_ml_stata_cv` and `c_ml_stata_cv` (Cerulli, 2022)

`r_ml_stata_cv` and `c_ml_stata_cv` are two commands for implementing machine learning regression and classification algorithms respectively in Stata 16

- They use the Stata/Python integration (sfi) capability of Stata 16 and allows to implement the following ML algorithms:

`r_ml_stata_cv`

ordinary least squares, elastic-net, tree, boosting, random forest, neural network, nearest neighbor, support vector machine.

`c_ml_stata_cv`

tree, boosting, random forest, regularized multinomial, neural network, naive Bayes, nearest neighbor, support vector machine, standard (unregularized) multinomial.

- They provides hyper-parameters' optimal tuning via K-fold cross-validation using greed search
- These commands make use of the Python **Scikit-learn** API to carry out both cross-validation and prediction

r_ml_stata_cv

```
r_ml_stata_cv depvar varlist , mlmodel(modeltype) data_test(filename)
seed(integer) [ learner_options cv_options other_options ]
```

modeltype_options

Description

Model

| | |
|------------------------|----------------------------|
| ols | Ordinary least squares |
| elasticnet | Elastic net |
| tree | Tree regression |
| randomforest | Bagging and random forests |
| boost | Boosting |
| nearestneighbor | Nearest neighbor |
| neuralnet | Neural network |
| svm | Support vector machine |

Regression

c_ml_stata_cv

```
c_ml_stata_cv depvar varlist , mlmodel(modeltype) data_test(filename)
seed(integer) [ learner_options cv_options other_options ]
```

| <i>modeltype_options</i> | Description |
|--------------------------|----------------------------|
| <hr/> | |
| Model | |
| tree | Classification tree |
| randomforest | Bagging and random forests |
| boost | Boosting |
| regmult | Regularized multinomial |
| nearestneighbor | Nearest Neighbor |
| neuralnet | Neural network |
| naivebayes | Naive Bayes |
| svm | Support vector machine |
| multinomial | Standard multinomial |
| <hr/> | |

Classification

Account

PROS

- All three commands are valuable and flexible commands for implementing in Stata many ML methods
- **pylearn** is very flexible, as it is a perfect duplication in Stata of the Scikit-learn API of Python
- **pystacked** is also very flexible as pretty all the Scikit-learn's modules options are implemented. Also, it allows for stacking regression and classification
- **r_ml_stata_cv** and **c_ml_stata_cv** allow for a larger set of learners to implement (for example, the *nearest-neighbor* and the *regularized multinomial*!). Also, they allow for *grid-search* for optimal tuning using *cross-validation* using **sklearn.model_selection.GridSearchCV**. This is not carried out by neither **pylearn**, nor **pystacked**

CONS

- **pylearn** implements only a few learners and does not provide for grid-search for optimal tuning using cross-validation
- **pystacked** does not provide for grid-search for optimal tuning using cross-validation and does not provide stacking for classification when the outcome is multinomial
- **r_ml_stata_cv** and **c_ml_stata_cv** are a little less flexible - as only the most important options (main hyperparameters) of the Scikit-learn's modules are implemented. Also, it does not have a **predict** post-estimation command (as predictions are automatically generated)

ML and Causal Inference with Stata

Introduction

- Growing literature exploits **machine learning (ML)** to improve **causal inference (CI)**
- In applications, we may have **high-dimensional** controls and/or instruments
- Also, controls and/or instruments can enter through an **unknown** functions
- Two approaches for integrating ML and CI:
 1. **Lasso-based** approach
(Belloni, Chernozhukov, and Hansen, 2014; Belloni et al., 2012)
 2. **Double-debiased ML (DDML)**
(Chernozhukov et al., 2018; Chernozhukov et al., 2021).

Lasso causal inference with Stata

Lasso HD for treatment effects

The diagram illustrates the Lasso HD regression model for treatment effects. It features a central equation $y_i = \alpha d_i + \beta_1 X_{i,1} + \dots + \beta_p X_{i,p} + \epsilon_i$ enclosed in a light gray rounded rectangle. The term αd_i is highlighted with a green background and labeled 'aim' with a bracket underneath. The sum of the nuisance variables $\beta_1 X_{i,1} + \dots + \beta_p X_{i,p}$ is highlighted with a yellow background and labeled 'nuisance' with a bracket underneath. Above the equation, a horizontal line connects the treatment variable d_i to the error term ϵ_i , with the label $Cov(d; \epsilon)$ centered above it. Below the equation, a horizontal line branches into two paths leading to the conditions for exogeneity and endogeneity.

$$y_i = \underbrace{\alpha d_i}_{\text{aim}} + \underbrace{\beta_1 X_{i,1} + \dots + \beta_p X_{i,p}}_{\text{nuisance}} + \epsilon_i$$

$Cov(d; \epsilon)$

Exogeneity
 $Cov(d; \epsilon) = 0$

Endogeneity
 $Cov(d; \epsilon) \neq 0$

LASSO for TE: **exogeneity** case [$Cov(d; \epsilon) = 0$]

Our model is

$$y_i = \underbrace{\alpha d_i}_{\text{aim}} + \underbrace{\beta_1 x_{i,1} + \dots + \beta_p x_{i,p}}_{\text{nuisance}} + \epsilon_i.$$

The causal variable of interest or “treatment” is d_i . The x s are the set of potential controls and not directly of interest. We want to obtain an estimate of the parameter α .



- How to infer correctly on α ?
- Which controls to select?
- What if $p \gg N$

Strategies to estimate α

- **Naïve approach**
- **Partialing-out**
- **Double-selection**

Lasso in Stata

Lasso commands for causal inference

| Model | Partialing-out | Double-selection | Cross-fit partialing-out |
|-----------|----------------|------------------|--------------------------|
| Linear | poregress | dsregress | xporegress |
| Logit | pologit | dslogit | xpologit |
| Poisson | popoisson | dspoisson | xpopoisson |
| Linear IV | poivregress | | xpoivregress |

Stata implementation via **pdslasso**

Basic syntax

```
pdslasso depvar d_varlist (hd_controls_varlist) [if] [in] [, ...]
```

with many options and features, including:

- heteroskedastic- and cluster-robust penalty loadings.
- LASSO or Sqrt-LASSO
- support for Stata time-series and factor-variables
- pweights and aweights
- fixed effects and partialling-out unpenalized regressors



IMPORTANT: **pdslasso** provides 3 estimates of the effect:

Partialling-out (PO) approach:

- OLS using CHS lasso-orthogonalized vars
- OLS using CHS post-lasso-orthogonalized vars

Double-selection (DS) approach:

- OLS with PDS-selected variables and full regressor set

Stata implementation via **ivlasso**

Basic syntax

```
ivlasso depvar d_varlist (hd_controls_varlist) (endog_d_varlist =  
high_dimensional_IVs) [if][in][, ...]
```

IMPORTANT: **ivlasso** provides 3 estimates of the effect:

Partialling-out (PO) approach:

- *IV using CHS lasso-orthogonalized vars*
- *IV using CHS post-lasso-orthogonalized vars*

Double-selection (DS) approach:

- *IV with PDS-selected variables and full regressor set*

IMPORTANT: Compared to the Stata built-in **poivregress**, the user-written command **ivlasso** performs two additional effect estimates:

- *IV using CHS lasso-orthogonalized vars*
- *IV with PDS-selected variables and full regressor set*

The IV procedure used is however the same, that is: **Lasso IV-2**. The difference is in the last step, where **ivlasso** uses the DS approach or the PO with lasso coefficients as alternatives.

In sum, **poivregress** uses the **ivlasso** PO type:

- *IV using CHS post-lasso-orthogonalized vars*

Double-debiased ML

Why relaying on DDML?

- The **Lasso** learner might not be the best-performing machine learner in specific settings (parametric model)
- The Lasso relies on the **approximate sparsity** assumption, which might not be appropriate in some settings
- **Double-Debiased Machine Learning (DDML)** allows to exploit various machine learners other than the Lasso. So, it is a more general approach for integrating ML and CI

Three sources of bias when estimating ATEs by ML

1. Bias due to absence of orthogonalization

Easily solved using the **Frisch-Waugh-Lovell** orthogonalization (equivalent to the Robinson's partially linear model)

2. Bias due to learner's low rate of convergence

Fortunately, most ML methods have sufficiently **fast rate of convergence**, including neural nets, random forests, lasso and boosting

3. Bias due to learner's over-fitting

Easily solved by **cross-fitting** estimation

Treatment models to estimate

A. Model with **homogenous** treatment effect ($ATE = ATET = ATENT$)

$$y = \theta \cdot d + g(\mathbf{x}) + \epsilon$$

| | |
|---------------|---|
| [partial] | Model 1: $(d \perp \epsilon) \mathbf{x}$ |
| [interactive] | Model 3: $(d \text{ correlated to } \epsilon) \mathbf{x}$ |

B. Model with **heterogenous** treatment effect ($ATE \neq ATET \neq ATENT$)

$$y = g(d, \mathbf{x}) + \epsilon$$

| | |
|-----------------|---|
| [iv] | Model 2: $(d \perp \epsilon) \mathbf{x}$ |
| [interactiveiv] | Model 4: $(d \text{ correlated to } \epsilon) \mathbf{x}$ |

The **ddml** command

ddml -- *Stata package for Double Debiased Machine Learning*

ddml implements algorithms for causal inference aided by supervised machine learning as proposed in **Double/debiased machine learning** for treatment and structural parameters (Econometrics Journal, 2018).

Five different models are supported, allowing for binary or continuous treatment variables and endogeneity, high-dimensional controls and/or instrumental variables. **ddml** supports a variety of different ML programs, including but not limited to **lassopack** and **pystacked**.

ddml - syntax

Estimation with **ddml** 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) norandom tabfold vars(varlist) ]
```

where *model* is either *partial*, *iv*, *interactive*, *fiv*, *interactiveiv*; see [model descriptions](#).

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

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

where, depending on model chosen in Step 1, *eq* is either $E[Y|X]$ $E[Y|D,X]$ $E[Y|X,Z]$ $E[D|X]$ $E[D|X,Z]$ $E[Z|X]$. *command* is a supported supervised ML program (e.g. [pystacked](#) or [cvlasso](#)). See [supported programs](#).

Note: Options before ":" and after the first comma refer to **ddml**. Options that come after the final comma refer to the estimation command.

Step 3. Cross-fitting:

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

This step implements the cross-fitting algorithm. Each learner is fitted iteratively on training folds and out-of-sample predicted values are obtained.

Step 4. Estimate causal effects:

```
ddml estimate [ , mname(name) robust cluster(varname) vce(type) atet ateu trim(real) ]
```

The **ddml estimate** command returns treatment effect estimates for all combination of learners added in Step 2.

ML hyperparameters' tuning

- gridsearch
- `r_ml_stata_cv` and `c_ml_stata_cv`

gridsearch (Schonlau, 2021)

gridsearch runs a user-specified statistical learning algorithm repeatedly with a **grid of values** corresponding to **one or two tuning parameters**. This facilitates the tuning of statistical learning algorithms.

After evaluating all combinations of values according to criterion, **gridsearch** lists the best combination and the corresponding value of the criterion.

Only estimation commands that allow the use of **predict** after the estimation command can be used.

The program does not currently support the prediction of multiple variables as would be needed, for example, for *multinomial logistic regression*

gridsearch - syntax

gridsearch — Optimizing tuning parameter levels with a grid search

Syntax

```
gridsearch command depvar indepvars [if] [in] , method(str1 str2) par1name(str) par1list(numlist) criterion(str) [options ]
```

```
gridsearch discrim subcommand indepvars [if] [in] , method(str1 str2) par1name(str) par1list(numlist) criterion(str) group(depvar) [options ]
```

options

Description

| | |
|-----------------------------|--|
| par1name (string) | Name of the a tuning parameter of command |
| par1list (numlist) | Values to explore for tuning parameter |
| par2name (string) | Name of the an optional second tuning parameter of command |
| par2list (numlist) | Values to explore for the second tuning parameter |
| criterion (string) | Evaluation criterion |
| method (str1 str2) | str1 specifies train-validation method; str2 specifies corresponding option. |
| nogrid | Explore all parameter values as a list (do not form a grid) |
| options | Additional options are passed to the estimation command |
| predoptions (string) | Any prediction options are passed to the prediction command |

Account

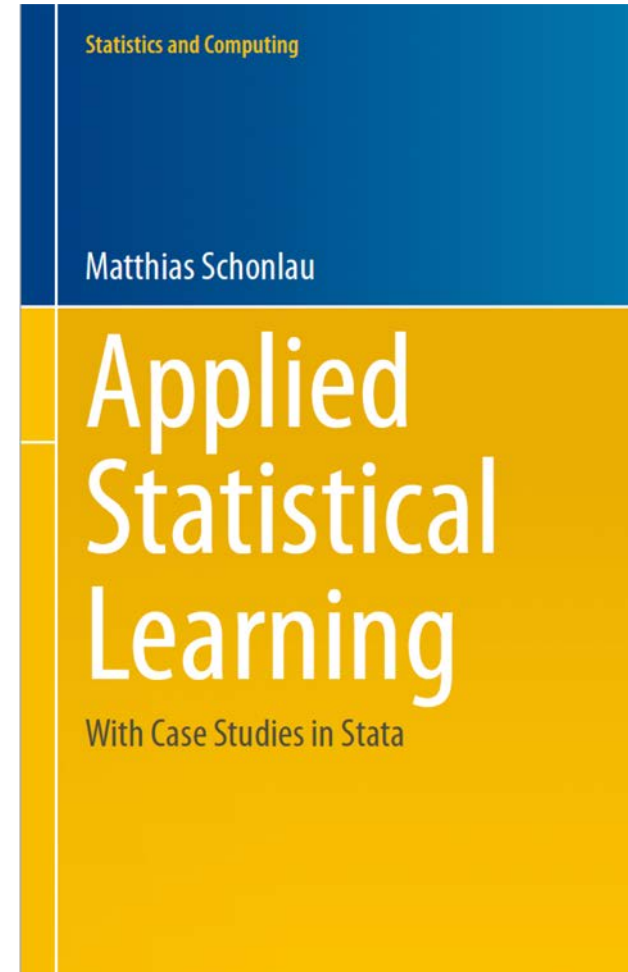
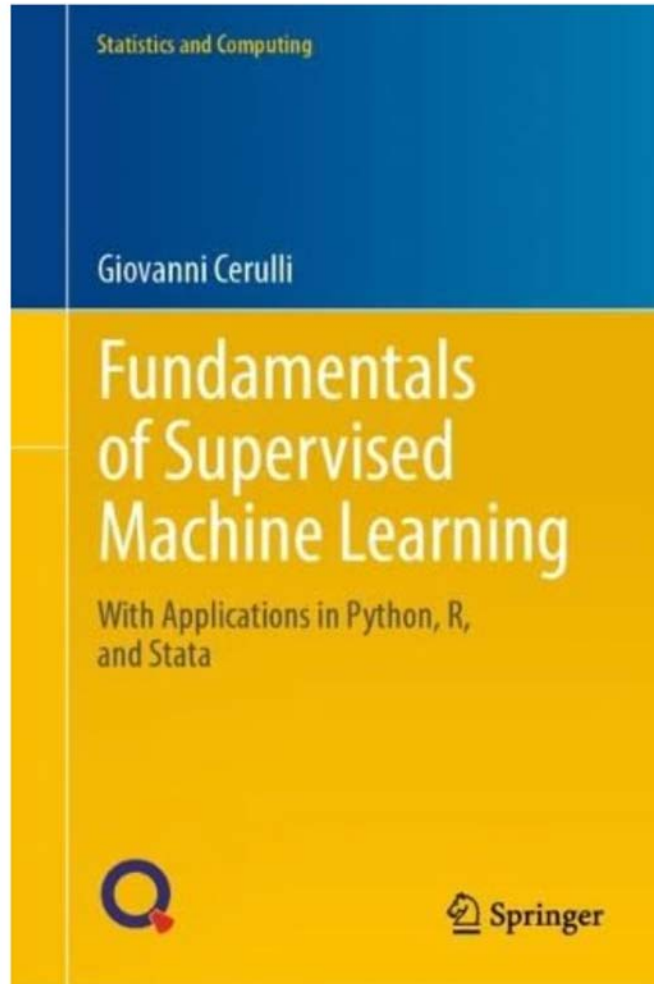
PROS

- All commands are valuable commands for hyper-parameter optimal tuning
- **gridsearch** allows for hyper-parameter optimal tuning using Stata native code
- **gridsearch** allows to use whatever learner having a **predict** post-estimation
- **r_ml_stata_cv** and **c_ml_stata_cv** allow for grid-search for optimal tuning using cross-validation using **sklearn.model_selection.GridSearchCV**. Also, they allow for optimal tuning of the *regularized multinomial*

CONS

- **gridsearch** allows only for the tuning of only **two hyper-parameters**.
- **gridsearch** is rather slow and does not allow for optimal tuning of the *regularized multinomial*
- **r_ml_stata_cv** and **c_ml_stata_cv** do not have a **predict** post-estimation command (as predictions are automatically generated). They allow for only a subset of hyper-parameters tuning (the most relevant, though!)

Books on Machine Learning using Stata



Conclusions

- Stata has many **valuable ML commands**, both native and based on other software
- The integration with **Python** is key for ML implementation
- However, there is **poor development for grid-search** for hyper-parameters optimal tuning. I would suggest the Stata Corp to develop an improvement of the **GRIDSEARCH** command using an architecture similar to the **CARET** package in R
- Stata users can provide **deep-learning** implementations by integrating into Stata the **KERAS** package of Python. Useful also for **advanced unsupervised learning**
- Stata has poor implementations of **reinforcement learning** (excluding the **OPL** command for “optimal policy learning” provided by Cerulli (2023) presented in Palo Alto at the US Stata Conference)

References 1/2

- **Ahrens, A., Hansen, C., Schaffer, M. E. & Wiemann, T.** (2023, January 23). ddml: Double/debiased machine learning in Stata. Statistical Software Components S459175, Boston College Department of Economics, revised 30 Apr 2023.
- **Ahrens, A., Hansen, C.B., Schaffer, M.E. (2018).** pdlasso and ivlasso: Programs for post-selection and post-regularization OLS or IV estimation and inference. <http://ideas.repec.org/c/boc/bocode/s458459.html>
- **Ahrens, A., Hansen, C. B. & Schaffer, M. E.** (March 2020). lassopack: Model selection and prediction with regularized regression. Stata Journal, 20(1), 176-235.
- **Ahrens, A., Hansen, C. B., & Schaffer, M. E.** (Accepted/In press). pystacked: Stacking generalization and machine learning in Stata. Stata Journal. <http://10.48550/arXiv.2208.10896>.
- **Balov N.** (2018). Multilayer perceptrons in Stata. <https://nbalov.github.io/posts/mlp2/mlp2.html>
- **Cerulli, G.** (2019). "SUBSET: Stata module to implement best covariates and stepwise subset selection." Statistical Software Components S458647, Boston College Department of Economics, revised December 6, 2022.
- **Cerulli, G.** (2019). "SRTREE: Stata module to implement regression trees via optimal pruning, bagging, random forests, and boosting methods." Statistical Software Components S458646, Boston College Department of Economics, revised January 26, 2022.

References 2/2

- **Cerulli, G.** (2019). "SCTREE: Stata module to implement classification trees via optimal pruning, bagging, random forests, and boosting methods." Statistical Software Components S458645, Boston College Department of Economics, revised July 11, 2022.
- **Cerulli, G.** (2022). Machine Learning using Stata/Python. The Stata Journal, 22(4), 1-39.
- **Droste, M.** (2020). pylearn. <https://github.com/NickCH-K/MLRtime/>. [Online; accessed 02-December-2021].
- **Guenther, N.** and **Schonlau, M.** (2016). "Support vector machines". In: Stata Journal, 16.4, 917–937(21).
- **Schonlau, M.** (2005). Boosted Regression (Boosting): An introductory tutorial and a Stata plugin. The Stata Journal, 5(3), 330-354.
- **Schonlau, M.** (2020, March). The Random Decision Forest Algorithm for Statistical Learning. The Stata Journal, 20(1), 3-29.
- **Schonlau, M.** (2021). GRIDSEARCH: Stata module to optimize tuning parameter levels with a grid search. Retrieved from <https://EconPapers.repec.org/RePEc:boc:bocode:s458859>.