A Brief Review of Machine Learning

Part 1

Achim Ahrens

CERGE-EI

November 18, 2025

Part 1: A Brief Review of Machine Learning

Part 2: Double/Debiased Machine Learning

How is ML different?

Econometrics

- Focus on parameter estimation and *causal* inference.
- Prediction/forecasting is traditionally done in a parametric framework (e.g. ARIMA, VAR).
- *Typical question:* Does x have a causal effect on y?

How is ML different?

Econometrics

- Focus on parameter estimation and *causal* inference.
- Prediction/forecasting is traditionally done in a parametric framework (e.g. ARIMA, VAR).
- *Typical question:* Does x have a causal effect on y?

Supervised Machine Learning

- Focus on prediction & classification.
- *Wide set of methods:* regularized regression, random forest, regression trees, support vector machines, neural nets, etc.
- Shared properties of many methods: **Non-parametric** and **data-adaptive**.
- *Procedure:* Algorithm is first 'trained' on random data partition and then 'validated' using 'unseen' data.
- *Strengths:* *Out-of-sample* prediction, high-dimensional data

Let's talk motivation

Why do we need machine learning? — Use cases.

I. Prediction for its own sake:

- You observe data $\{Y_i, X_i\}_{i \in \mathcal{I}}$ and want to use X_j with $j \notin \mathcal{I}$ for *out-of-sample* prediction.
- *Applications:* predicting long-term employment, predicting crime

Let's talk motivation

Why do we need machine learning? — Use cases.

I. Prediction for its own sake:

- You observe data $\{Y_i, X_i\}_{i \in \mathcal{I}}$ and want to use X_j with $j \notin \mathcal{I}$ for *out-of-sample* prediction.
- *Applications:* predicting long-term employment, predicting crime

II. Prediction as an intermediate step for causal inference:

- *Example 1:* Use ML to condition on confounders.
- *Example 2:* IV estimation. The first stage of IV/2SLS is essentially a prediction problem.
- *Methods:* PDS-Lasso, DDML ← **Today's topic**

Let's talk motivation

Why do we need machine learning? — Use cases.

III. Estimating heterogeneous treatment effects:

- We are interested in the causal effect of treatment D on Y. The treatment effect may vary with observed characteristics X. We want to estimate the *conditional* average treatment effect.
- *Methods:* Causal forests, R learners & Meta learners

Let's talk motivation

Why do we need machine learning? — Use cases.

III. Estimating heterogeneous treatment effects:

- We are interested in the causal effect of treatment D on Y. The treatment effect may vary with observed characteristics X. We want to estimate the *conditional* average treatment effect.
- *Methods:* Causal forests, R learners & Meta learners

IV. Policy learning:

- Formulate a policy rule $\pi(X)$ designed to maximize Y where the policy rule assigns a treatment a from the choice set of treatment options $\mathcal{A} = \{1, 2, \ldots, D\}$ to each individual based on their observed covariates X.
- *Methods:* Policy trees

Prediction

The general setting

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i, \qquad i = 1, \ldots, n.$$

We observe sample $\{y_i, x_i\}_{i \in \mathcal{I}}$, $p := dim(x_i)$ and want to use x_j with $j \notin \mathcal{I}$ for *out-of-sample* prediction.

Why would we use a fitting procedure other than OLS?

We know that OLS is BLUE. — However, ...

- One obvious reason is that *we don't want to assume linearity*.
- Say we are willing to assume linearity:
  - If $p > n$, OLS is not identified.
  - If $p \leq n$, but p is "large," OLS tends to perform poorly.

Bias-variance trade-off

For the sake of argument, let's say we are happy to assume linearity $f(x_i) = x_i'\beta$.

Bias-variance trade-off

For the sake of argument, let's say we are happy to assume linearity $f(x_i) = x_i'\beta$.

Mean squared prediction error (MSPE): Expected value of the square of the prediction error for an out-of-sample observation $j \notin \mathcal{I}$:

$$\begin{aligned} MSPE &= E\left[(y_j - \widehat{f}(x_j))^2\right] \\ &= \sigma^2 + \left[bias(\widehat{f}(x_j))\right]^2 + var\left(\widehat{f}(x_j)\right) \end{aligned}$$

Bias-variance trade-off

For the sake of argument, let's say we are happy to assume linearity $f(x_i) = x_i'\beta$.

Mean squared prediction error (MSPE): Expected value of the square of the prediction error for an out-of-sample observation $j \notin \mathcal{I}$:

$$\begin{aligned} MSPE &= E\left[(y_j - \widehat{f}(x_j))^2\right] \\ &= \sigma^2 + \left[bias(\widehat{f}(x_j))\right]^2 + var\left(\widehat{f}(x_j)\right) \end{aligned}$$

- The first term is idiosyncratic.
- The second term is the predictor's bias.
- The third term is the variance of the predictor.

Bias-variance trade-off

For the sake of argument, let's say we are happy to assume linearity $f(x_i) = x_i'\beta$.

Mean squared prediction error (MSPE): Expected value of the square of the prediction error for an out-of-sample observation $j \notin \mathcal{I}$:

$$\begin{aligned} MSPE &= E\left[(y_j - \widehat{f}(x_j))^2\right] \\ &= \sigma^2 + \left[bias(\widehat{f}(x_j))\right]^2 + var\left(\widehat{f}(x_j)\right) \end{aligned}$$

Because OLS is unbiased, the bias term disappears. — But the variance term may be large.

Crucial insight: A biased ("regularized") estimator may be able to outperform OLS in terms of prediction performance even though it's biased.

Bias-variance trade-off

For the sake of argument, let's say we are happy to assume linearity $f(x_i) = x_i'\beta$.

Mean squared prediction error (MSPE): Expected value of the square of the prediction error for an out-of-sample observation $j \notin \mathcal{I}$:

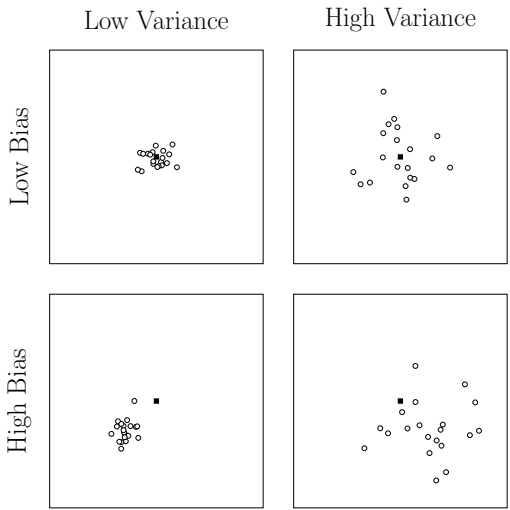
$$\begin{aligned} MSPE &= E\left[(y_j - \widehat{f}(x_j))^2\right] \\ &= \sigma^2 + \left[bias(\widehat{f}(x_j))\right]^2 + var\left(\widehat{f}(x_j)\right) \end{aligned}$$

This formula is the foundation for the **bias-variance trade-off**.

Why is this a trade-off?

- More flexibility is associated with more variance, but lower bias.
- In the context of OLS: more predictors imply lower bias, but higher variance.

Bias-variance trade-off



Suppose we have many potentially relevant predictors.

A low-bias/ high-variance estimator (top-right; e.g., OLS with all predictors) may be outperformed by a high-bias/ low-variance predictor (bottom-left; e.g., OLS with only highly correlated predictors).

Refugee prediction example

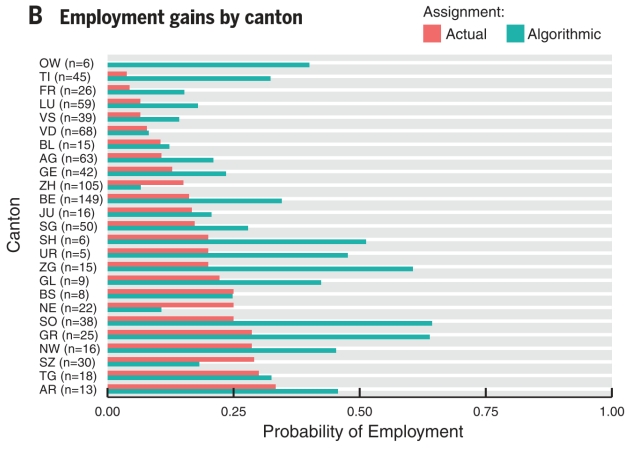
There are cases where ML methods can be applied 'off-the-shelf' to policy questions (Kleinberg et al., 2015; Athey, 2017).

Example: Bansak et al. (2018), 'Improving refugee integration through data-driven algorithmic assignment', *Science*.

- Refugee integration on settlement location, personal characteristics and synergies between the two.
- For example, the ability to speak French results is expected to lead to higher employment chances in French-speaking cantons of Switzerland.
- Host countries rarely take these synergies into account. Assignment procedures are usually based on capacity considerations (US) or random (Switzerland).

Refugee prediction example

The refugee allocation algorithm has the potential to lead to employment gains.
Predicted vs actual employment shares for Swiss cantons:



Regularization

Bias-variance trade-off:

$$\begin{aligned} MSPE &= E\left[(y_j - \widehat{f}(x_j))^2\right] \\ &= \sigma^2 + \left[bias(\widehat{f}(x_j))\right]^2 + var\left(\widehat{f}(x_j)\right) \end{aligned}$$

The bias-variance trade-off tells us that inducing some bias might actually improve out-of-sample prediction performance.

Regularization is used to impose additional constraints and thus reduce variance. Examples:

- *Shrinkage:* reducing the (absolute) size of coefficients
- *Subsetting:* removing some predictors from the model

Over-fitting

But how much regularization do we need?

A common theme in ML are (hyper-)tuning parameters that govern the degree of regularization.

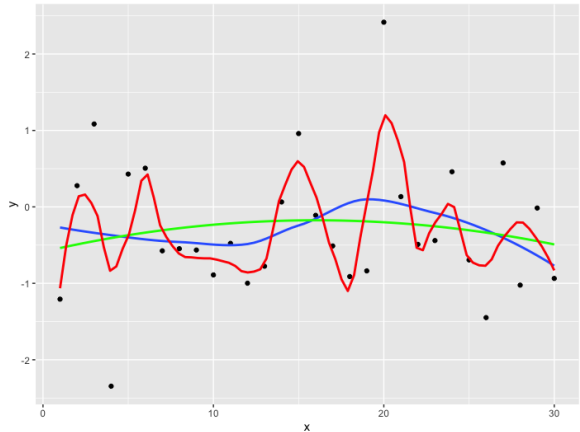
If we regularize too little, we run into the risk of **over-fitting**:

$\rightarrow$ high in-sample performance, but poor performance on unseen data

If we regularize too much, the bias is too large for the predictor to perform well.

Over-fitting

Classical example: selecting the bandwidth in local linear regression.



In this data, the y and x are uncorrelated. The red line follows the points closely, but will perform poorly on unseen data.

Validation

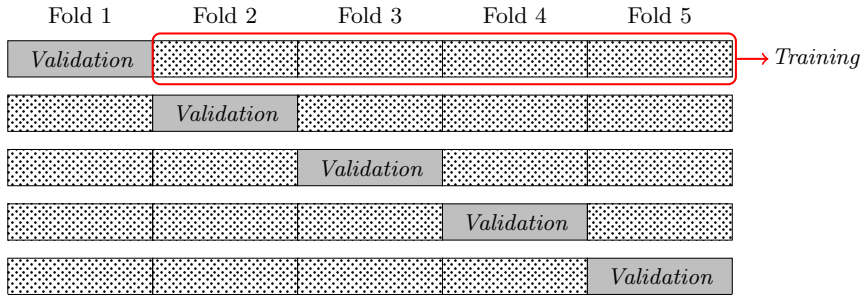
How do we select (hyper-)tuning parameters?

Basic idea: set aside part of your data for validation.

Some terminology:

- **Training sample**: Observations used to fit the model.
- **Test sample**, **validation sample**, or **holdout sample**: Observations used to evaluate the performance of the 'trained' model.

K-fold cross-validation



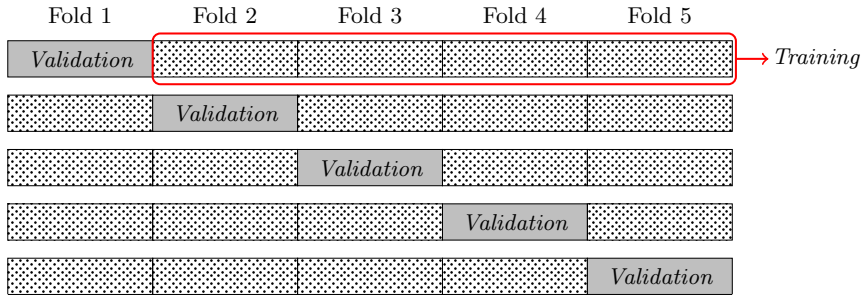
STEP 1 Divide data set into K groups (folds) of approximately equal size.

STEP 2 Treat fold 1 as the validation data set. Fold 2-5 constitute the training data.

STEP 3 Estimate the candidate models using the training data. Assess predictive performance using the validation data.

STEP 4 Identify the model that shows best out-of-sample predictive performance.

K-fold cross-validation



STEP 1 Divide data set into K groups (folds) of approximately equal size.

STEP 2 Treat fold 1 as the validation data set. Fold 2-5 constitute the training data.

STEP 3 Estimate the candidate models using the training data. Assess predictive performance using the validation data.

STEP 4 Identify the model that shows best out-of-sample predictive performance.

Common mistake: Pre-processing of predictors (e.g. standardization, imputation) should happen as part of the estimation step. $\rightarrow$ risk of *data leakage*.

Key concepts

Supervised vs unsupervised ML: Supervised ML has a well-defined outcome that we want to predict, while unsupervised ML does not have a distinction between independent/dependent variable.

Bias-variance-tradeoff: Model complexity (e.g., more predictors) implies less bias, but higher variance.

Over-fitting describes the poor performance of a complex model on out-of-sample data. In the regression context: high in-sample R^2, but low out-of-sample R^2.

Validation: The model is assessed using unseen data and some loss function (e.g. mean-squared error).

Cross-validation is a generalisation where we the data is iteratively split in training and validation sample. Typical forms are *K-fold* or *leave-one-out* CV

Overview of selected ML algorithms

High-dimensionality

The standard linear model

$$y_i = \boldsymbol{x}_i'\boldsymbol{\beta} + \varepsilon_i$$

We have up to $p = \dim(\beta)$ potential regressors. p can be very large, potentially even larger than the number of observations n.

The *high-dimensional model* accommodates situations where

- we observe many predictors;
- we only observe a few explanatory variables, but the number of potential regressors is large when accounting for model uncertainty, non-linearity, temporal & spatial effects, etc.

Penalized regression: General approach

Idea: put a 'price' on the use of regressors.

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\|\widehat{\beta}\|_L$ is some norm. λ governs the strength of the penalty.

Penalized regression: General approach

Idea: put a 'price' on the use of regressors.

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\|\widehat{\beta}\|_L$ is some norm. λ governs the strength of the penalty.

There is a cost to including lots of regressors, and we can reduce the objective function by shrinking the coefficients of the ones that contribute little to the fit. – *Some options:*

Penalized regression: General approach

Idea: put a 'price' on the use of regressors.

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\|\widehat{\beta}\|_L$ is some norm. λ governs the strength of the penalty.

There is a cost to including lots of regressors, and we can reduce the objective function by shrinking the coefficients of the ones that contribute little to the fit. – *Some options:*

- $L0$ 'norm' ($L = 0$): the penalty term is a function of the *number of regressors*.
- $L1$ norm ($L = 1$, Lasso): the penalty term is a function of the *sum of the absolute value of the coefficients on the regressors*.
- $L2$ norm ($L = 2$, Ridge): the penalty term is a function of the *sum of the squares of the coefficients on the regressors*.

Penalization: L0 'norm'

One option: penalize something related to **the number** of regressors, $L0$ 'norm'.

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\|\widehat{\beta}\|_0 = \sum_{j=1}^{p} \mathbb{1}\left\{\hat{\beta}_j \neq 0\right\}$, i.e., the number of regressors.

There is 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.

This is in fact the idea underlying **AIC/BIC**, which have attractive theoretical properties.

Penalization: L0 'norm'

One option: penalize something related to **the number** of regressors, $L0$ 'norm'.

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\|\widehat{\beta}\|_0 = \sum_{j=1}^{p} \mathbb{1}\left\{\hat{\beta}_j \neq 0\right\}$, i.e., the number of regressors.

There is 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.

This is in fact the idea underlying **AIC/BIC**, which have attractive theoretical properties.

Problem: computationally infeasible if p is at all large (NP-hard).

Penalization: Lasso (L1 norm)

The Lasso (Least Absolute Shrinkage and Selection Operator, Tibshirani (1996)) penalizes the sum of the **absolute values** of the coefficients ($L1$ norm):

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\|\widehat{\beta}\|_1 = \sum_{j=1}^{p}|\widehat{\beta}_j|$, i.e., the sum of the absolute value of the (normalized) coefficients.

There's a cost to including lots of regressors, and we reduce Q by shrinking most the ones that contribute little to the fit. Note that if $\lambda = 0$ (zero penalty) we get OLS.

Penalization: Lasso (L1 norm)

The Lasso (Least Absolute Shrinkage and Selection Operator, Tibshirani (1996)) penalizes the sum of the **absolute values** of the coefficients ($L1$ norm):

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\|\widehat{\beta}\|_1 = \sum_{j=1}^{p}|\widehat{\beta}_j|$, i.e., the sum of the absolute value of the (normalized) coefficients.

There's a cost to including lots of regressors, and we reduce Q by shrinking most the ones that contribute little to the fit. Note that if $\lambda = 0$ (zero penalty) we get OLS.

Computationally feasible using efficient algorithms.

Key feature: Lasso solution *shrinks the βs for (many) variables to zero*.

Penalization: Ridge (L2 norm)

Ridge regression (also called Tikhonov regularization) penalizes the sum of the **squares** of $\widehat{\beta}_j$s ($L2$ norm):

$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\left\|\widehat{\beta}\right\|_2 = \sum_{j=1}^{p}\widehat{\beta}_j^2$, i.e., the square of the (normalized) coefficients.

There's a cost to a large weight ($\widehat{\beta}_j$) on a regressor, and we reduce the objective function Q by shrinking most the $\widehat{\beta}_j$s that contribute little to the fit. Note that if $\lambda = 0$ (zero penalty) we get OLS.

Penalization: Ridge (L2 norm)

Ridge regression (also called Tikhonov regularization) penalizes the sum of the **squares** of $\widehat{\beta}_j$s ($L2$ norm):

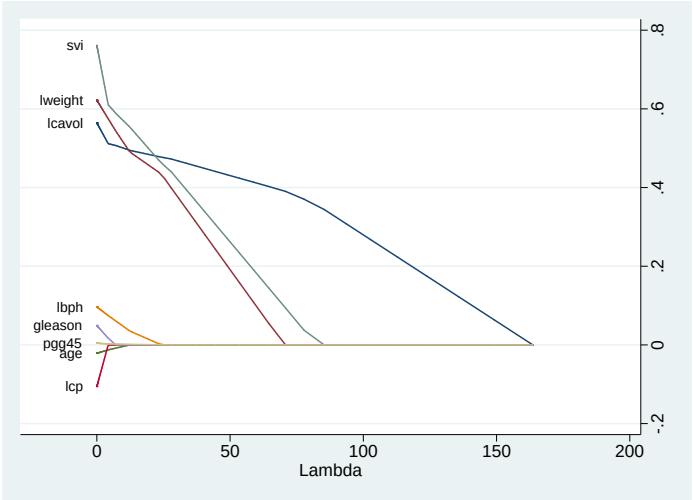
$$\min_{\widehat{\beta}} Q = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - \mathbf{x}_i'\widehat{\beta}\right]^2 + \frac{\lambda}{n}\left\|\widehat{\beta}\right\|_L$$

where $\left\|\widehat{\beta}\right\|_2 = \sum_{j=1}^{p}\widehat{\beta}_j^2$, i.e., the square of the (normalized) coefficients.

There's a cost to a large weight ($\widehat{\beta}_j$) on a regressor, and we reduce the objective function Q by shrinking most the $\widehat{\beta}_j$s that contribute little to the fit. Note that if $\lambda = 0$ (zero penalty) we get OLS.

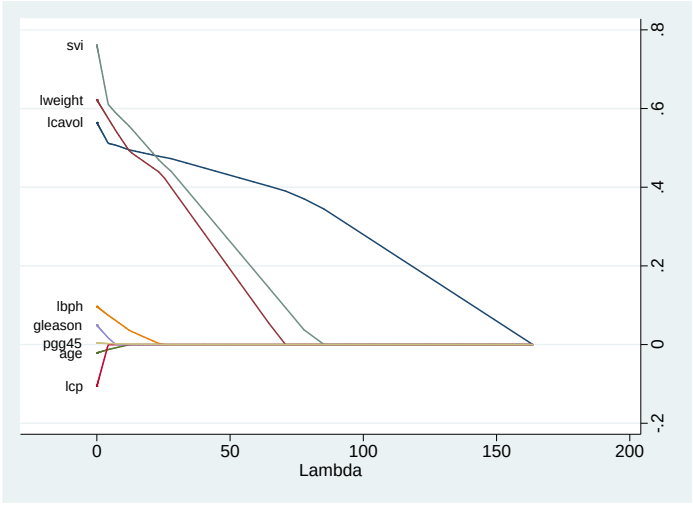
An *important feature*: All elements of $\widehat{\beta}_{Ridge}$ appear in the final model. ('Dense' solution.)

The Lasso: The solution path



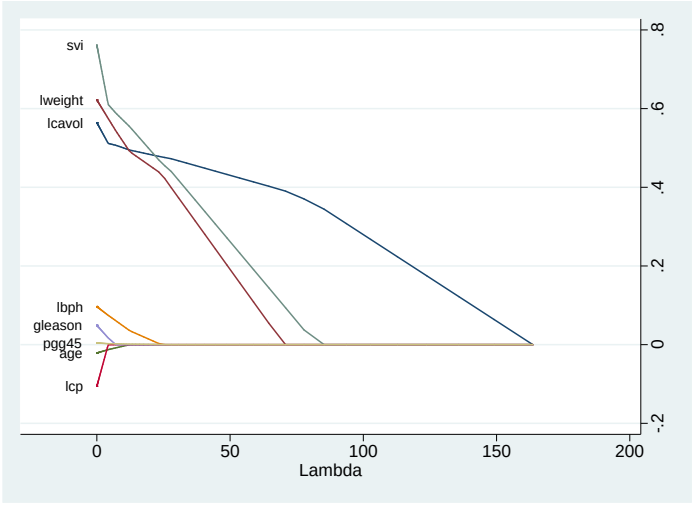
The Lasso coefficient path is a continuous and piecewise linear function of λ, with changes in slope where variables enter/leave the active set.

The Lasso: The solution path



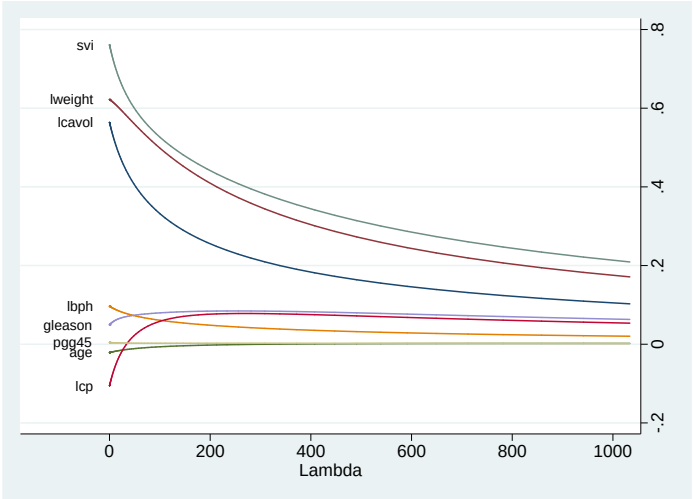
The Lasso yields sparse solutions. As λ increases, variables are being removed from the model. Thus, the Lasso can be used for model selection.

The Lasso: The solution path



We have reduced a complex model selection problem into a one-dimensional problem. We 'only' need to choose the 'right' penalty level, i.e., λ.

Lasso vs Ridge solution path



Ridge regression: No sparse solutions. The Ridge is not a model selection technique.

Choosing the tuning parameter(s)

Lasso & Co fundamentally rely on the tuning parameter λ.

How do we select λ?

Choosing the tuning parameter(s)

Lasso & Co fundamentally rely on the tuning parameter λ.

How do we select λ?— Three approaches:

- *Cross-validation:* Resample the data and find the λ that optimizes out-of-sample prediction.
- *Information criteria:* select the value of λ that minimizes information criterion (AICc, BIC or EBIC$_\gamma$).
- *'Rigorous' or 'plug-in' penalization:* Belloni et al. (2012) develop theory and feasible algorithms for the optimal λ under heteroskedastic and non-Gaussian errors. Feasible algorithms are available for Lasso and square-root Lasso.

Key concepts

Regularized regression minimizes (like OLS) the squared error, but applies a penalty to control model complexity and thus control the degree of over-fitting.

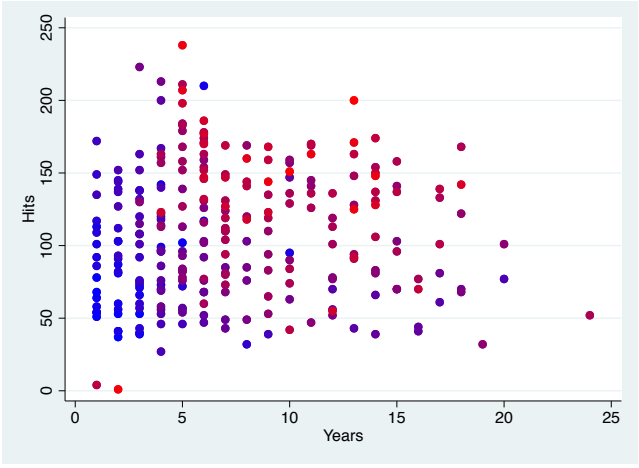
The Lasso penalizes the sum of absolute coefficients. By setting some predictors to zero, it simultaneously performs estimation and selection of predictors.

The Lasso fundamentally relies in **sparsity**: the true model should have only a small number of non-zero coefficients. If sparsity fails, the data-generating process is referred to as **dense**.

The Ridge also shrinks coefficients towards zero, but does not perform model selection. **Elastic net** is a mix of lasso and ridge.

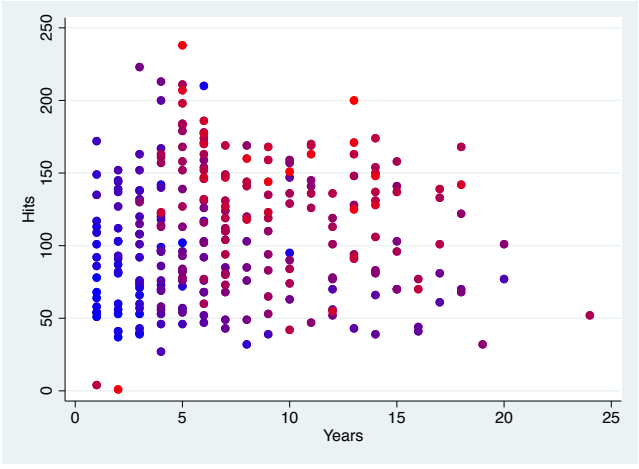
There are three options for selecting the **penalty level**: 1) cross-validation (typical for ML), 2) information criteria, 3) theory-grounded plugin-penalty.

Regression Trees



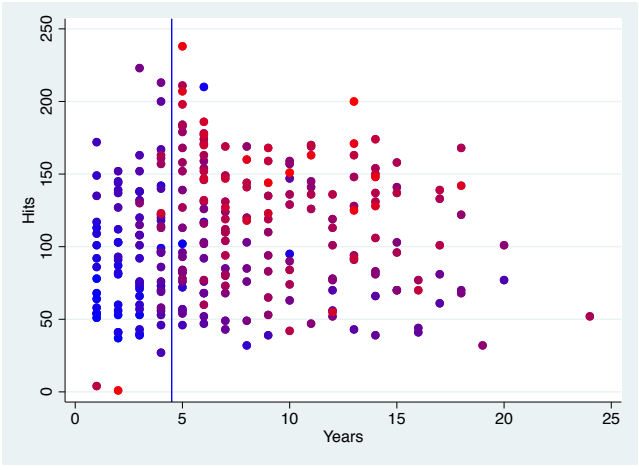
We plot the relationship between three variables: two predictors on the vertical/horizontal axis, and the outcome (salary). Blue (low) to red (high) indicates the outcome level.

Regression Trees



Tree-based approach: split the data in two in order to separate 'high' from 'low' salaries.

Regression Trees



The first split assigns observations with Years$<$4.5 into one branch and observations with Years$\geq$4.5 into another branch.

Regression Trees

How do we determine the first split?

We define the two regions (referred to as *leaves* or *terminal nodes*):

$$R_1(j, s) = \{X | X_j < s\} \text{ and } R_2(j, s) = \{X | X_j \geq s\}$$

and we seek the values of j and s that minimise residual sum of squares:

$$\sum_{i: x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 \quad + \quad \sum_{i: x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2.$$

$\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses across all observations that fall into leaf 1 and 2, respectively.

Regression Trees

How do we determine the first split?

We define the two regions (referred to as *leaves* or *terminal nodes*):

$$R_1(j, s) = \{X | X_j < s\} \text{ and } R_2(j, s) = \{X | X_j \geq s\}$$

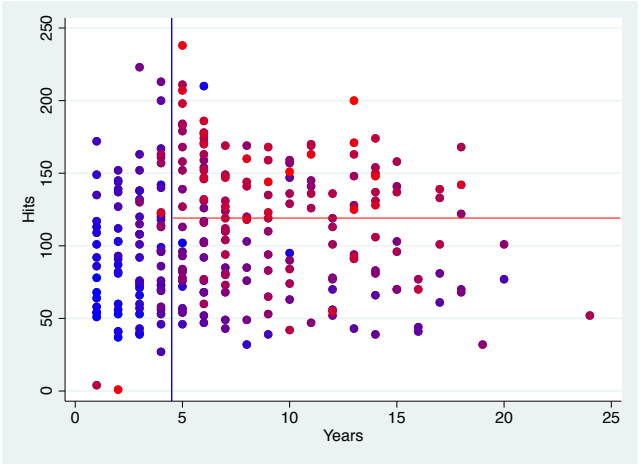
and we seek the values of j and s that minimise residual sum of squares:

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 \quad + \quad \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2.$$

$\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses across all observations that fall into leaf 1 and 2, respectively.

In words: we search for a threshold s over predictor j such that the squared error is minimized.

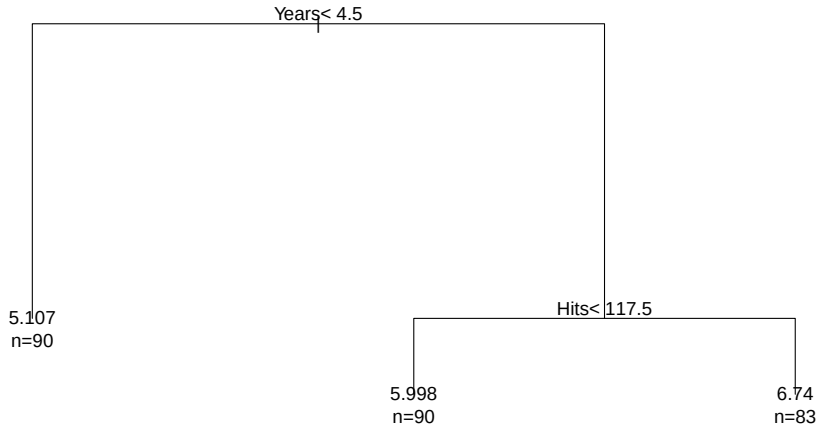
Regression Trees



We continue the process of splitting the data. Here we consider the best split for the region to the right of 4.5 years.

Regression Trees

We can represent the three regions in tree form:

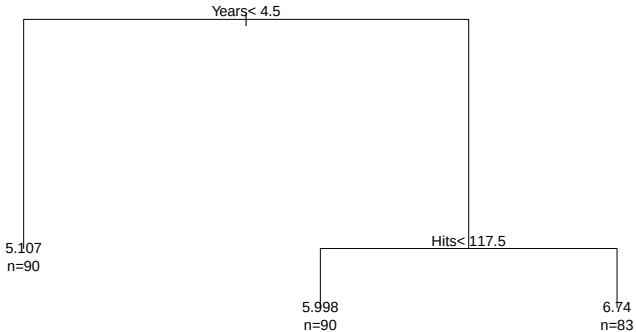


Regression Trees

How deep should the tree be?

Which tree is better for prediction?

A *shallow* tree. . .

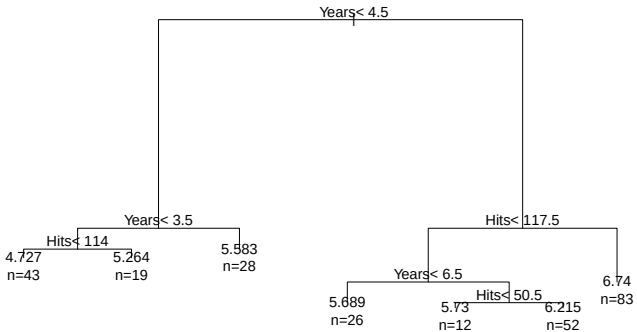


Regression Trees

How deep should the tree be?

Which tree is better for prediction?

. . . or a *deep* tree?

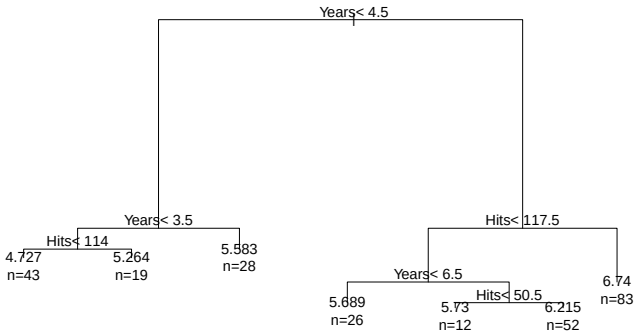


Regression Trees

How deep should the tree be?

Which tree is better for prediction?

. . . or a *deep* tree?



Regression Trees

How deep should the tree be?

Same problem as in a liner model: is a model with 3 predictors better than a model with 10 predictors for prediction?

A big tree with many regions is likely to *overfit* the data; high in-sample fit, but poor out-of-sample prediction performance.

Bias-variance trade-off: A tree with fewer splits (fewer regions $R_1, \ldots, R_J$) might lead to lower variance (as well as better interpretation).

This is similar to least squares: more predictors increases in-sample fit (R^2), but also increases variance.

Regression Trees

Advantages and disadvantages

- Easy to explain, easy to interpret (*with caution!*), easy to illustrate graphically.
- Trees can deal with non-linear relationships much better than linear regression without the need to generate polynomials and *ad hoc* dummy variables.
- High variance: *small* changes in the sample, can lead to *very* different trees.
- Predictive performance is often relatively poor due to over-fitting.

Regression Trees

Advantages and disadvantages

- Easy to explain, easy to interpret (*with caution!*), easy to illustrate graphically.
- Trees can deal with non-linear relationships much better than linear regression without the need to generate polynomials and *ad hoc* dummy variables.
- High variance: *small* changes in the sample, can lead to *very* different trees.
- Predictive performance is often relatively poor due to over-fitting.

Why are regression trees so popular? — Regression trees provide the basis for more sophisticated methods...

Bagging

We generate B bootstrap samples (i.e., we sample *with* replacement) from the full sample, fit trees on each bootstrap sample, and derive predictions as

$$\hat{f}_{\text{bag}} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

Note that, on average, each bootstrap sample uses $2/3$ of the data. The remaining $1/3$ are referred to as the *out-of-bag* (OOB) observations.

This is referred to as **bootstrap aggregation** or **bagging**.

Random forests

Random forests (due to Breiman 2001) is bagging with a small tweak: at each split, a random selection of m predictors is considered for making the next split (typical choice is $m \approx \sqrt{p}$).

The purpose of restricting the set of predictors is to de-correlate the individual trees in order to have sufficient variation across trees.

Random forests

Random forests (due to Breiman 2001) is bagging with a small tweak: at each split, a random selection of m predictors is considered for making the next split (typical choice is $m \approx \sqrt{p}$).

The purpose of restricting the set of predictors is to de-correlate the individual trees in order to have sufficient variation across trees.

- Random forest perform very well for prediction tasks.
- They require little tuning (compared to neural networks).
- Out-of-bag observations are suitable for assessing variable importance.
- They are not efficient at dealing with linear, quadratic or smooth processes.

Boosting

Recall: **Bagging** fits many trees and combines them to create one prediction.

Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Given the current model, we fit a tree to the residuals from the model. That is, we fit a tree to the current residuals, rather than the outcome Y.

Kearns/Valiant (1988/1989) posed the question: *'can a set of weak learners create a single strong learner?'*

$\rightarrow$ a general concept that can use not just trees but other methods as weak learner (least squares, logit)

Boosting

A simple boosting algorithm using regression trees

Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

For $b = 1, \ldots, B$:

1. Fit a tree $\hat{f}^b(x)$ with d splits (implying $d + 1$ nodes)
2. Update $\hat{f}(x)$ by adding a shrunken version of the new tree
$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$$
3. Update the residuals $r_i \leftarrow r_i - \lambda \hat{f}^b(x)$

The prediction of the boosting model is calculated as: $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$

The boosting approach learns *slowly*. The speed of learning is determined by λ (a shrinkage parameter).

Boosting

The important tuning parameters:

- B – *the number of trees*.
- *Shrinkage parameter* λ, which controls the learning rate. Typical values are 0.01 or 0.001. Small λ might require larger B for good performance.
- The number of *splits in each tree*. Even $d = 1$ (where each tree is a stump) can work well.

Boosting

History and advancements:

- `AdaBoost` (Freund/Schapire 1997): at each modelling step, weights are applied to the training data whereby missclassified observations receive higher *weights*; the algorithm is thus forced to focus on problematic/difficult decisions
- The boosting idea was re-interpreted and generalised in the *Gradient boosting* framework, allowing for a general class of loss-functions
- Modern gradient boosting implementations such as `XGBoost` also add a *regularization term* to penalise model complexity

Popular algorithms:

- `XGBoost`: A popular and fast implementation for gradient tree boosting; has been the default boosting method for years
- `CatBoost` & `LightGBM` (originally developed by Microsoft) improve on `xgboost`, among others, by adding better support for categorical variables

The choice of the learner

The machine leaning (ML) toolbox includes a rich set of flexible methods:

- We have *briefly* discussed regularized regression, random forests, boosting.
- We haven't covered other important methods: SVM, neural nets.

The choice of the learner

The machine leaning (ML) toolbox includes a rich set of flexible methods:

- We have *briefly* discussed regularized regression, random forests, boosting.
- We haven't covered other important methods: SVM, 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.

The optimal choice is a matter of finding the optimal match between (unobserved) DGP and the learner's regularization assumptions.

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.

Stacking regression

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). Goes under various names: super learner, model averaging, etc.

Stacking regression

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). Goes under various names: super learner, model averaging, etc.

General idea:

- Combine a set of "base" (or "level-0", "candidate") 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.

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*:

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

where $\hat{y}_{(j),i}^{-k(i)}$ are the cross-validated predictions of base learner j.

Stacking regression

1. *Cross-validation:*

   1.1 Split the sample $I = \{1, \ldots, n\}$ randomly into K partitions ('folds') of approximately equal size. Denote the set of observations in fold $k = 1, \ldots, K$ as I_k, and its complement as I_k^c such that $I_k^c = I \setminus I_k$. I_k constitutes the step-k validation set and I_k^c the step-k training sample.

Stacking regression

1. *Cross-validation:*
   1.1 Split the sample $I = \{1, \ldots, n\}$ randomly into K partitions ('folds') of approximately equal size. Denote the set of observations in fold $k = 1, \ldots, K$ as I_k, and its complement as I_k^c such that $I_k^c = I \setminus I_k$. I_k constitutes the step-k validation set and I_k^c the step-k training sample.
   1.2 For each fold $k = 1, \ldots, K$ and each base learner $j = 1, \ldots, J$, fit the supervised machine learner j to the training data I_k^c and obtain out-of-sample predicted values $\hat{y}_{(j),i}^{-k}$ for $i \in I_k$.

Stacking regression

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

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

The stacking predicted values are defined as $\hat{y}_i^\star = \sum_j \hat{w}_j \hat{y}_{(j),i}$ where $\hat{w}_j$ is the estimated stacking weight corresponding to learner j and $\hat{y}_{(j),i}$ are the predicted values from re-fitting learner j on the full sample I.

Key concepts

Ensemble methods combine several machine learners to improve performance. Random forests rely on trees; boosting may use trees or other simple base learners.

Model averaging or **stacking** Stacking goes a step further: it can combine cross-validated predictions from different types of machine learners, e.g., lasso and random forests.

While very flexible, ensemble methods are 'black box' methods and make **model interpretation** more challenging.

References I

- Athey, Susan (2017). "The Impact of Machine Learning on Economics". URL: https://www.nber.org/chapters/c14009.pdf.
- Bansak, Kirk et al. (2018). "Refugees using algorithmic assignment". In: 329.January, pp. 325–329.
- 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.
- 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).
- Kleinberg, Jon et al. (2015). "Prediction Policy Problems". In: *American Economic Review* 105.5, pp. 491–495. URL: http://www.aeaweb.org/articles?id=10.1257/aer.p20151023.

References II

- Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288. URL: http://www.jstor.org/stable/2346178.
- Wolpert, David H. (1992). "Stacked generalization". In: *Neural Networks* 5.2, pp. 241–259. URL: https://www.sciencedirect.com/science/article/pii/S0893608005800231.