Chapter 3: Machine Learning and Model Selection

Frontier Topics in Empirical Economics

Lecture notes based on slides by Zibin Huang
College of Business, Shanghai University of Finance and Economics

September 2025

Contents

1 Introduction

In Chapter 2 we assembled a powerful toolkit of non-parametric and semi-parametric methods: kernel regression, local polynomial regression, series estimation, and the partially linear model. Together with the linear regression from Chapter 1, we now have many tools in our toolbox. A natural and important question follows: *which tool should we choose?*

This is a model selection question, and it arises even when we restrict ourselves to a single tool. Take ordinary least squares (OLS) as an example. A linear specification says $y = x\beta+\varepsilon$. Why linear? Is it because linearity is simple? Why not $y = \ln x+x^3+\varepsilon$? Consider the Mincer wage equation, in which we regress *wage* on *edu*, *exp*, and exp^2. Why include the square of experience but not, say, edu^3? The honest answer is that most of the time we are making these choices by habit or convention rather than by any systematic criterion. The functional form is our assumption—and, as we saw in Chapter 2, if the assumption is wrong, everything collapses.

For a long time, model selection was largely ignored in applied economics. The main reason was data availability: when your dataset had a few hundred observations and a handful of variables, it was not feasible for you to further separate your small dataset to training (estimation) set and validation set. Thus, the scope for model selection was limited. Nowadays, the world has changed. More and more datasets are available with large sample sizes and high-dimensional covariate spaces. Big data brings more chances—but also more decisions to make. We should take model selection seriously.

There are broadly two approaches to choosing a model. The first is a **data-driven approach**: we use the data themselves to select the model, without injecting prior knowledge about causal relationships. This is the machine learning philosophy. The second is a **structure-based approach**: we use economic theory, causal logic, or a Directed Acyclic Graph (DAG) to determine which variables belong in the model. In this chapter we focus on the first approach. We select models using only data, without putting our economic knowledge or theory into the process. The structure-based approach will be the topic of Chapter 4.

Before introducing any specific machine learning algorithm, we need to understand a fundamental statistical concept that governs all of model selection: the *bias-variance tradeoff*. We encountered this idea briefly in Chapter 2 when we discussed the choice

of bandwidth in kernel regression. Here we develop it more generally and show that it is the central principle behind every model selection method.

2 The Bias-Variance Tradeoff

2.1 Why Not Always Use the Most Complicated Model?

Consider three models of increasing flexibility:

$$\text{A traditional linear model:} \qquad y = x\beta + \varepsilon, \tag{1}$$

$$\text{A model with a quadratic term:} \qquad y = x\beta + x^2\alpha + \varepsilon, \tag{2}$$

$$\text{A non-parametric model:} \qquad y = g(x) + \varepsilon. \tag{3}$$

Why not always use the second or the third one? More flexibility means less risk of misspecification, so it seems like a free lunch.

To sharpen the question, consider two nested linear models:

$$\text{Model A:} \qquad y = x_1'\beta + \varepsilon, \tag{4}$$

$$\text{Model B:} \qquad y = x_1'\beta_1 + x_2'\beta_2 + \varepsilon. \tag{5}$$

Model B includes everything in Model A plus additional regressors x_2. Is it always better to have the more complicated model?

The answer is *no*. The reason is that more flexibility helps with one problem but hurts with another. Adding x_2 reduces the risk of misspecification—this lowers *bias*. But it also makes the estimated model more sensitive to the particular sample at hand—this raises *variance*. This tension between bias and variance is the central theme of this chapter.

2.2 Decomposing Prediction Error

To make the tradeoff precise, assume the true data-generating process is

$$Y = f(X) + \varepsilon, \qquad E[\varepsilon \mid X] = 0, \qquad \text{Var}(\varepsilon \mid X) = \sigma_\varepsilon^2.$$

We train a model $\hat{f}(x)$ on some data. Because the training data are random, $\hat{f}(x)$ is itself a random object. If we drew a different sample, we would get a different fitted model: $\hat{f}^1(x)$, $\hat{f}^2(x)$, and so on. The expectation $E[\hat{f}(x)]$ is taken over these different samples—it is the average model we would get if we could repeat the estimation many times.

How good is the model at predicting a new observation? The standard measure is the *prediction mean squared error* (PMSE) at a point x_0:

$$E\Big[(Y - \hat{f}(x_0))^2 \mid X = x_0\Big].$$

This measures how far, on average, our prediction $\hat{f}(x_0)$ is from the actual outcome Y. The following decomposition is the cornerstone of model selection theory.

Bias-Variance Decomposition

$$E\Big[(Y - \hat{f}(x_0))^2 \mid X = x_0\Big] = \underbrace{\sigma_\varepsilon^2}_{\text{irreducible error}} + \underbrace{\big[E\hat{f}(x_0) - f(x_0)\big]^2}_{\text{Bias}^2} + \underbrace{E\big[\hat{f}(x_0) - E\hat{f}(x_0)\big]^2}_{\text{Variance}}. \tag{6}$$

Prediction error = Irreducible error + Bias^2+ Variance.

Let us understand each component.

Irreducible error (σ_ε^2). This is the noise inherent in the data. No matter how clever our model is, we can never predict better than σ_ε^2 because the outcome Y is genuinely random around $f(X)$. This term sets the floor for prediction error.

Bias $[E\hat{f}(x_0) - f(x_0)]$. Bias measures the systematic error—how far, on average, our model's prediction is from the truth. A simple model like linear OLS applied to a nonlinear DGP will systematically miss the curvature, producing large bias. A very flexible model can track the curvature closely, producing small bias.

Variance $E[\hat{f}(x_0) - E\hat{f}(x_0)]^2$. Variance measures how much the fitted model changes from sample to sample. A simple model like linear OLS is stable: it produces similar fitted lines regardless of the specific sample drawn. A very flexible model—say, a twentieth-order polynomial—is highly sensitive to the particular data points: adding

or removing a few observations can change the fitted curve dramatically.

The fundamental tension is now clear:

- Increasing model complexity $\Rightarrow$ Bias $\downarrow$ but Variance $\uparrow$.
- A super complicated model $\Rightarrow$ Variance $\uparrow\uparrow\uparrow$. The model becomes so sensitive to the training data that it changes wildly from one sample to the next.
- Such a model *overfits* the current data: it achieves excellent in-sample fit but poor out-of-sample prediction.

Good model selection means finding the sweet spot that minimizes the *total* prediction error—the sum of bias squared and variance.

2.3 An Example of Overfitting

Let us make the discussion concrete with a simple simulation. Consider the data-generating process

$$Y = 1 + 1.5X + \varepsilon, \qquad \varepsilon \sim N(0, 3600).$$

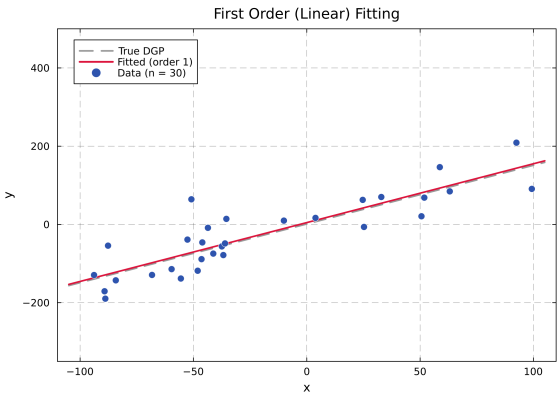
This is a linear relationship, but a very noisy one—the standard deviation of the noise is 60, which is large relative to the signal. Suppose we draw 30 observations from this process and try to fit polynomials of increasing order.

With a **first-order (linear)** polynomial, the fitted line lies close to the true DGP. There is some discrepancy—the line does not pass through every data point—but this is appropriate because the data points contain noise. The fit captures the signal and ignores the noise.

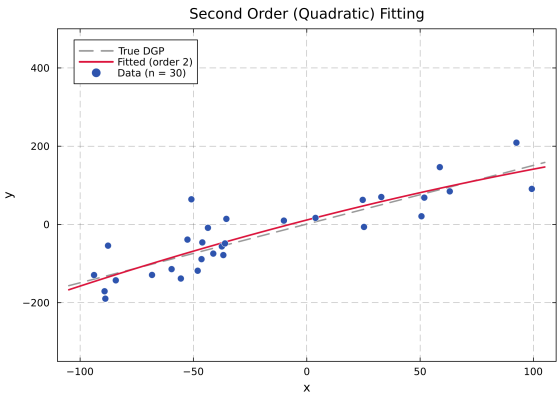
With a **second-order (quadratic)** polynomial, the fitted curve bends slightly, picking up a small amount of curvature that exists in the noise but not in the true DGP. The improvement in R^2 is marginal.

By the **fifth-** and **sixth-order**, the fitted curve is clearly oscillating. It tries to pass more data points by twisting the curve, but the wiggles have nothing to do with the true DGP—they are artifacts of the noise.

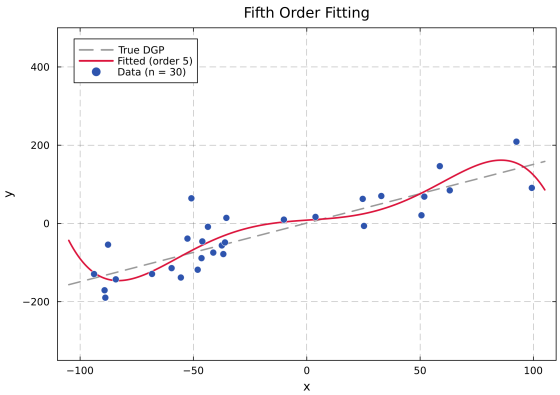
Figure 1 illustrates this progression. In the first-order panel the fitted line (red) nearly coincides with the true DGP (gray dashed). As the polynomial order rises to 5 and 6, the fitted curve begins to deviate from the straight line, bending to chase individual data points.



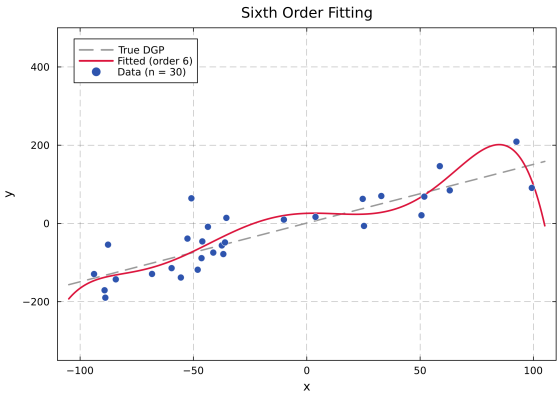
(a) First order (linear)



(b) Second order (quadratic)



(c) Fifth order



(d) Sixth order

Figure 1: Polynomial fitting of orders 1–6. The true DGP is $Y = 1 + 1.5X + \varepsilon$ with $\sigma = 60$ and $n = 30$. Gray dashed: true relationship. Red solid: fitted polynomial. As the order increases, the fitted curve begins to deviate from the straight-line truth, tracking noise rather than signal.

Overfitting

High-order polynomials pick up **noise, not signals**. They achieve excellent in-sample fit but terrible out-of-sample prediction. This is the defining symptom of overfitting. The model's bed has been carved into the exact shape of one night's sleeper at a specific time point—it fits perfectly for him right at that moment, but it will be definitely uncomfortable to sleep on such a bed without a possibility to move your body.

Figure 2: The bias-variance tradeoff and overfitting. As model complexity increases, training error decreases monotonically, but test error eventually rises—indicating overfitting. The optimal model minimizes total prediction error (test error).

This example echoes our experience from Chapter 2. Recall Runge's phenomenon: a high-degree global polynomial oscillates violently at the boundary, even though the function it tries to approximate is smooth. And the Gibbs phenomenon: a Fourier series overshoots at a discontinuity no matter how many terms we add. Both are instances of the same underlying principle: *more parameters do not always improve approximation*. The extra flexibility can be wasted on fitting noise rather than signal.

3 Goodness of Fit and Cross-Validation

The overfitting example teaches us that we cannot simply maximize in-sample fit. We need a criterion that rewards good fit but penalizes unnecessary complexity. In this section we review several such criteria.

3.1 Traditional Information Criteria

The most familiar approach is to modify the in-sample fit measure with a penalty for the number of parameters.

Adjusted R^2. Recall from introductory econometrics that R^2 never decreases when we add a regressor—even an irrelevant one. This is because OLS can always use an additional coefficient to reduce the residual sum of squares, even if only by fitting noise. The adjusted R^2 corrects for this by dividing both the residual and total sums of squares by their respective degrees of freedom. It can decrease when a useless regressor is added, because the penalty for losing a degree of freedom exceeds the gain in fit. This is the simplest form of complexity penalization.

AIC (Akaike Information Criterion). The AIC provides a more principled penalty:

$$\text{AIC} = 2k + n \ln(\text{RSS}/n),$$

where k is the number of estimated parameters and RSS is the residual sum of squares. The term $n \ln(\text{RSS}/n)$ measures goodness of fit (smaller RSS means better fit), while $2k$ penalizes complexity. We select the model with the smallest AIC. The factor of 2 in the penalty term reflects an asymptotic correction for the optimism of the maximized log-likelihood as an estimator of the expected Kullback–Leibler divergence from the true model.

BIC (Bayesian Information Criterion). The BIC is similar in spirit but uses a stronger penalty:

$$\text{BIC} = k \ln(n) + n \ln(\text{RSS}/n).$$

The penalty $k \ln(n)$ grows with the sample size, so BIC penalizes additional parameters more heavily in large samples. BIC is motivated by the Bayesian approach to model comparison. In practice, BIC tends to select simpler models than AIC, which is desirable when the true model is sparse.

3.2 Cross-Validation

AIC and BIC penalize complexity analytically. Cross-validation (CV) takes a different approach: it directly measures out-of-sample prediction performance by splitting the data into pieces.

The idea is beautifully simple. We want to know how well a model predicts on data it has never seen. So we deliberately set aside some observations, train the model on the rest, and then check how well the model predicts the held-out observations. If the model has overfit the training data, it will perform poorly on the held-out data, because the noise patterns it memorized are different in the new observations.

The most common implementation is **K-fold cross-validation**:

1. Divide the sample randomly into K equal-sized groups, called "folds."
2. Pick fold k (one fold in K) as the *validation sample*. Use the remaining $K-1$ folds as the *training sample* to estimate the model.
3. Compute the mean squared prediction error MSE_k on fold k—the average squared difference between the actual Y values in fold k and the model's predictions for those observations.
4. Repeat for each fold $k = 1, 2, \ldots, K$, so that every fold serves as the validation sample exactly once.
5. Average the K error measures:

$$\text{CV} = \frac{1}{K} \sum_{k=1}^{K} \text{MSE}_k.$$

Common choices are $K = 5$ or $K = 10$. The extreme case $K = n$ (each observation is its own fold) is called *leave-one-out* cross-validation.

Cross-Validation

CV measures the quality of *out-of-sample* prediction. We use data that were not involved in the estimation to evaluate the model. A smaller CV value means a model that generalizes better to new data. CV is particularly valuable because it directly targets what we care about—prediction on unseen observations—rather than relying on asymptotic approximations.

3.3 From Goodness of Fit to Machine Learning

We now have several standards for what constitutes a "good" model: adjusted R^2, AIC, BIC, and cross-validation. These criteria tell us *how to evaluate* a model, but they do not tell us *how to search* for one. If we have 20 potential regressors, there are $2^{20} \approx 1$ million possible subsets to consider. Evaluating all of them is computationally expensive and statistically problematic.

This is where machine learning enters the picture. Machine learning algorithms provide *automatic, efficient search procedures* for finding good models. They take the goodness-of-fit criteria we have just discussed and embed them into algorithms that explore the model space intelligently.

3.4 What Is Machine Learning?

What is machine learning? The Wikipedia definition is instructive:

> "Machine learning (ML) is an umbrella term for solving problems for which development of algorithms by human programmers would be cost-prohibitive, and instead *the problems are solved by helping machines 'discover' their own 'algorithms'*, without needing to be explicitly told what to do by any human-developed algorithms."

In economics, the use of machine learning is more specific. The main question is: *how complicated should the model be?* Given a set of potential predictors X, how should we predict Y? When Y is discrete (e.g., default vs. no default), this is a **classification** problem. When Y is continuous (e.g., house prices), this is a **prediction (regression)** problem.

There are many machine learning algorithms. We will introduce three families that are most relevant for economists: *penalized regression*, *tree-based methods*, and *neural networks*.

4 Penalized Regression

4.1 The Problem of Too Many Regressors

Let us start with a setting that is close to what economists already know: linear regression. Consider the model $y_i = x_i'\beta + \varepsilon_i$. What happens when the number of potential regressors is very large?

This is not a hypothetical concern. Imagine you have a household survey with 1,000 questions. Each answer is a potential regressor. Which ones should you include in your wage equation? OLS estimates

$$\hat{\beta}^{\text{OLS}} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i'\beta)^2.$$

Because OLS minimizes the sum of squared residuals (SSR), every additional regressor helps: more β's mean more flexibility, which means a smaller SSR. But we know that a smaller SSR does not mean a better model—it may just mean more overfitting. We need a mechanism to *penalize* the use of additional regressors.

4.2 The Penalized Objective Function

The key idea is to modify the OLS objective by adding a *penalty term* that discourages large or numerous coefficients:

Penalized Regression

$$\hat{\beta}^{\text{Pen}} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - x_i'\beta\right)^2 + \lambda \left(\|\beta\|_p\right)^p, \tag{7}$$

where $\|\beta\|_p = \left(\sum_j |\beta_j|^p\right)^{1/p}$ is the ℓ_p-norm, $\left(\|\beta\|_p\right)^p = \sum_j |\beta_j|^p$, and $\lambda \geq 0$ is the *tuning parameter*.

The objective function now has two parts. The first part, $\sum_i (y_i - x_i'\beta)^2$, is the usual SSR—it wants β to be large enough to fit the data well. The second part, $\lambda\|\beta\|_p^p$, is the penalty—it wants β to be small. The estimator must balance these two competing goals. The tuning parameter λ controls the tradeoff: larger λ means more shrinkage (simpler model, lower variance, higher bias); $\lambda = 0$ recovers OLS.

The two most important special cases correspond to $p = 1$ and $p = 2$.

4.2.1 LASSO Regression ($p = 1$)

When $p = 1$, the penalty is $\lambda \sum_j |\beta_j|$. This is called the **Least Absolute Shrinkage and Selection Operator** (LASSO), introduced by Tibshirani (1996).

The ℓ_1 penalty has a remarkable property: it can shrink some coefficients *exactly to zero*. This means that LASSO does not just shrink coefficients—it performs *variable selection*. Regressors with small predictive power are dropped entirely. In our 1,000-question survey example, LASSO might select, say, 15 variables and set the remaining 985 coefficients to exactly zero.

Why does ℓ_1 produce exact zeros while ℓ_2 does not? The geometric intuition is instructive. The ℓ_1 constraint set (the "diamond" $\sum |\beta_j| \leq c$) has sharp corners along the axes. The OLS contour is an ellipsoid. When the ellipsoid first touches the diamond, it is very likely to touch at a corner—where one or more coordinates are exactly zero. The ℓ_2 constraint set (the "ball" $\sum \beta_j^2 \leq c$) is smooth, so tangency almost surely occurs in the interior, where no coordinate is exactly zero.

4.2.2 Ridge Regression ($p = 2$)

When $p = 2$, the penalty is $\lambda \sum_j \beta_j^2$. This is called **Ridge regression**.

Ridge regression shrinks all coefficients toward zero but never sets any exactly to zero. In our survey example, Ridge would keep all 1,000 regressors but give small coefficients to the irrelevant ones. Ridge is particularly useful when regressors are highly correlated: in such settings, OLS estimates are unstable (small changes in the data produce large swings in $\hat{\beta}$), while Ridge stabilizes them through shrinkage.

4.2.3 Choosing λ

How do we choose the tuning parameter λ? Cross-validation. We try a grid of λ values, compute the CV score for each, and pick the λ that minimizes CV. This directly targets out-of-sample prediction quality, which is exactly what we want.

4.3 The Elastic Net

In practice, a popular choice is the **Elastic Net**, which combines the ℓ_1 and ℓ_2 penalties:

$$\hat{\beta}^{\text{EN}} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i'\beta)^2 + \lambda\left[\alpha\|\beta\|_1 + (1-\alpha)\left(\|\beta\|_2\right)^2\right], \tag{8}$$

where $\alpha \in [0,1]$ controls the mix between LASSO ($\alpha = 1$) and Ridge ($\alpha = 0$). The Elastic Net inherits the variable selection property of LASSO while retaining the stability of Ridge when regressors are correlated.

5 Tree-Based Methods

5.1 The Idea: Partition and Average

Penalized regression stays within the linear model and controls complexity by shrinking coefficients. Tree-based methods take a completely different approach. The idea is disarmingly simple:

1. Partition the covariate space X into a set of rectangular regions.
2. Within each region, predict Y by the average of the Y values in that region.

This is a **Classification and Regression Tree** (CART). The name reflects the fact that the partition can be represented as a binary tree: each internal node is a yes/no question about one covariate ("Is $X_1 \leq 3.5$?"), and each terminal node (leaf) contains a predicted value.

Formally, we partition the space into M disjoint regions $R_1, \ldots, R_M$ and define the predicted value within each region as the sample mean:

$$\hat{f}(x_i) = \sum_{m=1}^{M} \hat{c}_m \, \mathbf{1}(x \in R_m), \qquad \hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i,$$

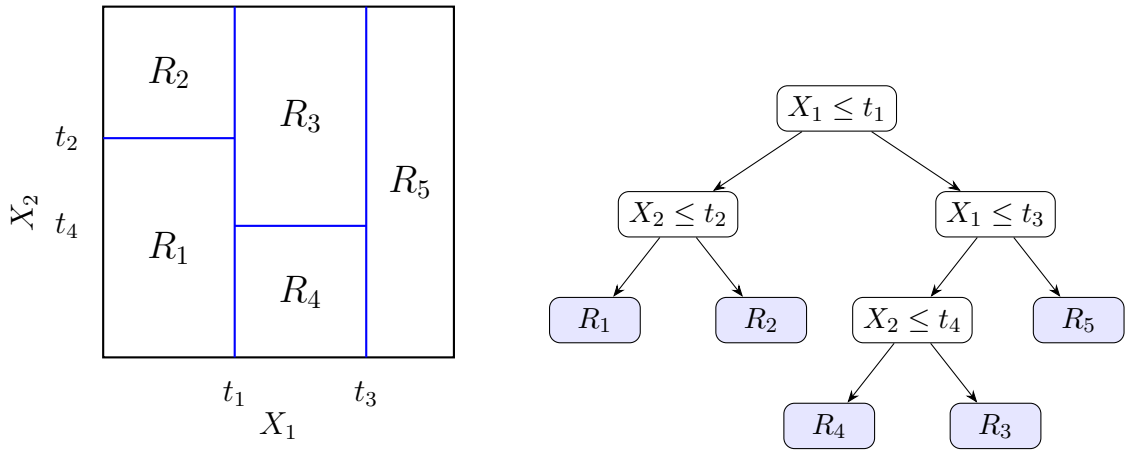
where $N_m = |\{x_i \in R_m\}|$ is the number of observations in region m.

5.2 Recursive Binary Partitions

How do we find a good partition? The answer is **recursive binary partitions**. At each step we split one existing region into two halves by choosing a variable and a cut

point. The process produces a binary tree: each internal node is a yes/no question about one covariate, and each terminal node (leaf) contains a predicted value.

Figure 3 illustrates a concrete example with two covariates X_1 and X_2. The first split is at $X_1 = t_1$; the left region is then split at $X_2 = t_2$; and so on. The resulting five regions $R_1, \ldots, R_5$ correspond exactly to the five leaves of the binary tree shown alongside.



(a) Partition of the (X_1, X_2) space

(b) Corresponding binary tree

Figure 3: Recursive binary partitioning. Each split divides one region into two by choosing a variable and a cut point. The five terminal nodes (leaves) of the tree correspond to the five rectangular regions.

5.3 Two Choices at Each Step

At each step of the recursive partitioning, we face two decisions:

1. **Continue or stop?** Should we keep splitting the current region, or is it small enough (or pure enough) to become a leaf?
2. **Where to partition?** If we continue, which variable X_j should we split on, and at what cut point s?

We use a **greedy algorithm** to answer both questions. "Greedy" means that at each step we make the locally best choice without looking ahead at future splits.

For each leaf R_m, we define the following quantities:

- **Size**: $N_m = |\{x_i \in R_m\}|$ (the number of observations in the leaf).
- **Fitted value** (mean as fit): $\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$.
- **SSE** (error in leaf): $Q_m(\mathcal{T}) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$.

5.4 Where to Partition

Conditional on continuing to grow, how do we determine the best split? For the j-th predictor and a cut position s, we define two half-planes:

$$R_1(j, s) = \{X \mid X_j \leq s\}, \qquad R_2(j, s) = \{X \mid X_j > s\}.$$

We search over all predictors j and all cut points s to minimize the total SSE across the two resulting leaves:

$$\min_{j,\, s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right], \tag{9}$$

where c_1 and c_2 are two constants. For any given split (j, s), the optimal c_1 and c_2 are simply the sample averages of y_i in the respective regions, so the minimization over (j, s) is computationally straightforward.

5.5 When to Stop Growing

The second decision is whether to continue splitting or to stop. A tree that keeps splitting until every leaf contains a single observation will have zero training error—but it will be massively overfit. A tree that stops too early will miss important structure. Too large $\rightarrow$ *overfitting*; too small $\rightarrow$ losing information.

The standard approach is to first grow a large tree and then *prune it back*:

1. **Step 1: Grow a big tree $\mathcal{T}_0$.** Keep splitting until some minimum node size is reached (say, each leaf has only 10 observations).
2. **Step 2: Prune**. Choose the subtree $\mathcal{T} \subset \mathcal{T}_0$ with the lowest cost function $C_\alpha(\mathcal{T})$. Here $\mathcal{T} \subset \mathcal{T}_0$ means any tree that can be obtained by collapsing any number of internal nodes in $\mathcal{T}_0$.

Cost-Complexity Pruning

$$C_\alpha(\mathcal{T}) = \sum_{m=1}^{|\mathcal{T}|} N_m \, Q_m(\mathcal{T}) + \alpha \, |\mathcal{T}|, \tag{10}$$

where $|\mathcal{T}|$ is the number of terminal nodes (leaves) and $\alpha \geq 0$ is a tuning parameter.

This criterion has a familiar structure: it is the total SSE (a measure of bias) plus a penalty for tree size (a measure of complexity). The parameter α determines how hard we penalize tree size. Larger α produces smaller, simpler trees. As with penalized regression, the optimal α can be chosen by cross-validation.

6 Random Forests

6.1 The Problem with a Single Tree

A single decision tree is intuitive and easy to interpret, but it has a significant weakness: high variance. Small changes in the training data can produce very different tree structures. One sample might split first on income, another on education—leading to completely different partitions. This instability is precisely the variance problem from our bias-variance decomposition.

The solution is to *average over many trees*. If individual trees are noisy but unbiased, their average will have the same bias but lower variance. This is the logic behind Random Forests.

6.2 Bagging and De-Correlation

The procedure involves two variance-reduction techniques:

Bagging (Bootstrap Aggregation). We draw B bootstrap samples from the training data (each of size N, drawn with replacement) and grow a tree T_b on each sample. The final prediction is the average:

$$\hat{f}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x).$$

Alternatively, we can use sub-sampling (draw samples of size $n' < N$ without replacement).

De-correlation. Bagging alone is not enough, because trees grown on overlapping bootstrap samples tend to be correlated. If the trees are correlated, averaging them does not reduce variance as much. To break this correlation, Random Forests add a twist: at each split, instead of searching over all p variables, we randomly select m variables (where $m \ll p$) and choose the best split among those m only. This forces

different trees to use different variables, reducing their correlation.

The variance of the forest prediction is approximately:

$$V(\hat{f}) \approx \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2, \tag{11}$$

where ρ is the average pairwise correlation between trees and σ^2 is the variance of a single tree. The first term depends on ρ and cannot be reduced by adding more trees. The second term shrinks as B increases. To reduce $V(\hat{f})$, we can either increase B (more trees) or decrease ρ (smaller m, more de-correlation).

Random Forests

Random Forests = Tree Method + Sampling Average (Many De-correlated Trees).

The algorithm:

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a tree T_b on $\mathbf{Z}^*$: at each split, randomly select m variables from the p available, pick the best variable/split-point among those m, and split. Continue until the minimum node size $n_{\min}$ is reached.
2. Output the ensemble $\{T_b\}_{b=1}^B$.

Prediction: $\hat{f}_{\mathrm{RF}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

For classification: let $\hat{C}_b(x)$ be the class predicted by tree b. The forest prediction is $\hat{C}_{\mathrm{RF}}^B(x) = \text{majority vote}\{\hat{C}_b(x)\}_{b=1}^B$.

6.3 Connection to Kernel Methods

Random Forests are conceptually similar to the kernel and nearest-neighbor methods we studied in Chapter 2. All of these methods make predictions by taking weighted averages of "nearby" observations. The difference lies in how "nearby" is defined:

- **Nearest neighbor and kernel methods**: the notion of proximity is fixed by a distance metric (e.g., Euclidean distance) or a bandwidth h. It does not adapt to the data.
- **Random Forests**: the notion of proximity is *adaptive*. The tree structure learns which directions in the covariate space are most informative for prediction. Two

observations are "nearby" if they frequently end up in the same leaf—not if their Euclidean distance is small.

An important application of Random Forests in economics is **Causal Forests**, which we discuss next.

7 Causal Forests

7.1 Motivation: Heterogeneous Treatment Effects

In Chapter 1, we focused on estimating summary measures of the treatment effect: the Average Treatment Effect (ATE). This tells us the average impact of a policy across the entire population. But in many applications, the most interesting question is: *for whom does the treatment work best?*

A drug may cure some patients but have no effect on others. A job training program may benefit workers with low baseline skills but do little for highly skilled workers. These are *heterogeneous treatment effects* (HTEs), and they are crucial for policy design: if we know who benefits most, we can target the treatment more efficiently.

The traditional approach to detecting HTEs is subgroup analysis: split the sample by gender, age, education, etc., and estimate the treatment effect within each subgroup. The problem is that this invites *cherry picking*—running many subgroup analyses until a statistically significant difference appears by chance. This multiple testing problem is severe when the number of potential subgroups is large.

Wager and Athey (2018) propose **Causal Forests** as a principled, data-driven solution. Causal Forests are an extension of Random Forests designed specifically to estimate heterogeneous treatment effects without the researcher pre-specifying which subgroups to examine.

7.2 Setup and Assumptions

We observe data (X_i, Y_i, W_i) for $i = 1, \ldots, n$, where X_i is a vector of covariates, Y_i is the outcome, and $W_i \in \{0, 1\}$ is the treatment indicator. The *Conditional Average Treatment Effect* (CATE) at a covariate value x is:

$$\tau(x) = E\Big[Y_i^{(1)} - Y_i^{(0)} \mid X_i = x\Big]. \tag{12}$$

This is the treatment effect for the subpopulation with characteristics x. It varies with x—that is what "heterogeneous" means.

Causal Forests Require Unconfoundedness

Causal Forests assume **unconfoundedness** (selection on observables):

$$\{Y_i^{(0)}, Y_i^{(1)}\} \perp\!\!\!\perp W_i \mid X_i.$$

Treatment assignment is as good as random once we condition on X_i. Therefore, *causal forests are not a method to deal with endogeneity.* If there is unobserved confounding (omitted variables that affect both treatment and outcome), causal forests will not solve the problem. They are a tool for uncovering treatment effect heterogeneity under the maintained assumption that selection on observables holds.

7.3 Estimation: Comparing Treated and Control Within Leaves

The estimation logic is natural. Each tree partitions the covariate space into leaves. Within a leaf $L(x)$ containing the point x, the treatment effect is estimated by comparing the average outcome of treated units to the average outcome of control units:

$$\hat{\tau}(x) = \frac{1}{|\{i : W_i = 1, X_i \in L\}|} \sum_{\substack{i: W_i=1 \\ X_i \in L}} Y_i - \frac{1}{|\{i : W_i = 0, X_i \in L\}|} \sum_{\substack{i: W_i=0 \\ X_i \in L}} Y_i. \tag{13}$$

This is just the difference in means within the leaf—exactly the estimator we would use in a randomized experiment restricted to the subpopulation in the leaf.

The forest aggregates over many such trees by averaging the leaf-level estimates. The splitting criterion is adapted: instead of minimizing prediction error for Y, the algorithm implements the Random Forests using the criterion of *maximizing the variance of the treatment effect $\hat{\tau}(X_i)$ across groups (leaves).* This encourages the tree to find splits where the treatment effect genuinely differs, rather than splits that merely predict the level of Y.

7.4 Honesty and Asymptotic Properties

A subtle but important issue arises: if the same data are used both to decide where to split and to estimate τ within each leaf, the estimates can be biased. The splits are

chosen to maximize differences in treatment effects, so the within-leaf estimates will tend to overstate heterogeneity—a form of overfitting.

Wager and Athey (2018) solve this problem with **honest estimation**. A tree is *honest* if, for each training observation i, that observation is used either to decide the splits *or* to estimate τ, but not both.

The practical implementation uses **double-sample trees**:

1. Randomly divide the training sample into two halves, $\mathcal{I}$ and $\mathcal{J}$.
2. Use $\mathcal{I}$ to grow the tree (decide the split structure).
3. Use $\mathcal{J}$ to estimate τ in each leaf.

This separation ensures that the estimation step is not contaminated by the adaptive splitting decisions. Wager and Athey (2018) show that honest causal forests are *consistent* and *asymptotically normal*, which means we can construct valid confidence intervals for the CATE $\tau(x)$—a remarkable result for a machine learning method.

7.5 Application: Social Media and Polarization

Levy (2021) studies the effect of social media on news consumption and political polarization using a field experiment. The main experiment randomly assigns Facebook users to be shown counter-attitudinal news content. Causal forests are used (see Online Appendix C.5 of the paper) to explore heterogeneous treatment effects—that is, to identify which subgroups of participants are most affected by exposure to content from the other side. This is a natural application: the experimenter cannot pre-specify all relevant subgroups, and causal forests let the data reveal which characteristics matter for treatment effect heterogeneity.

8 Neural Networks

8.1 Overview

The third family of machine learning methods we introduce is **neural networks**. Neural networks have attracted enormous public attention in recent years, powering applications ranging from ChatGPT to AlphaGo. The underlying mathematics, however, is remarkably straightforward.

8.2 Architecture of a Single-Hidden-Layer Network

Consider a prediction problem where we observe inputs $X = (X_1, \ldots, X_p)$ and want to predict an output Y. A single-hidden-layer feed-forward neural network introduces an intermediate set of *hidden units* $Z = (Z_1, \ldots, Z_M)$ that transform the inputs before they are used to predict Y. The model has three layers:

- **Input layer**: the covariates X.
- **Hidden layer**: the derived features $Z_1, \ldots, Z_M$.
- **Output layer**: the predicted values $Y_1, \ldots, Y_K$.

Step 1: From inputs to hidden units. Each hidden unit $m = 1, \ldots, M$ computes a linear combination of the inputs and then passes it through a nonlinear *activation function* σ:

$$Z_m = \sigma\big(\alpha_{0m} + \alpha_m^T X\big), \qquad m = 1, \ldots, M. \tag{14}$$

The activation function $\sigma(\cdot)$ is typically the logistic function $\sigma(v) = 1/(1+e^{-v})$, a step function, or the popular ReLU function $\sigma(v) = \max(0, v)$. This nonlinearity is critical: it is what distinguishes a neural network from a linear model. If σ is a linear function, the composition of two linear transformations would be another linear transformation, and the hidden layer would be redundant.

Step 2: From hidden units to outputs. Each output unit $k = 1, \ldots, K$ computes:

$$T_k = \beta_{0k} + \beta_k^T Z, \qquad Y_k = f_k(X) = g_k(T), \quad k = 1, \ldots, K, \tag{15}$$

where $g(\cdot)$ is another nonlinear function (e.g., Logit model).

Figure 4 illustrates the architecture of a single-hidden-layer feed-forward neural network.

8.3 Why "Neural" Networks?

The name reflects the original motivation: modeling the human brain. Each unit represents a *neuron*, and each weighted connection represents a *synapse*. When a step function is used as the activation function, a neuron "fires" (outputs 1) when the total input signal $\alpha_{0m} + \alpha_m^T X$ exceeds a threshold—mimicking the way biological neurons fire when stimulated beyond a certain level.

There can be multiple hidden layers, giving rise to *deep* neural networks. The depth

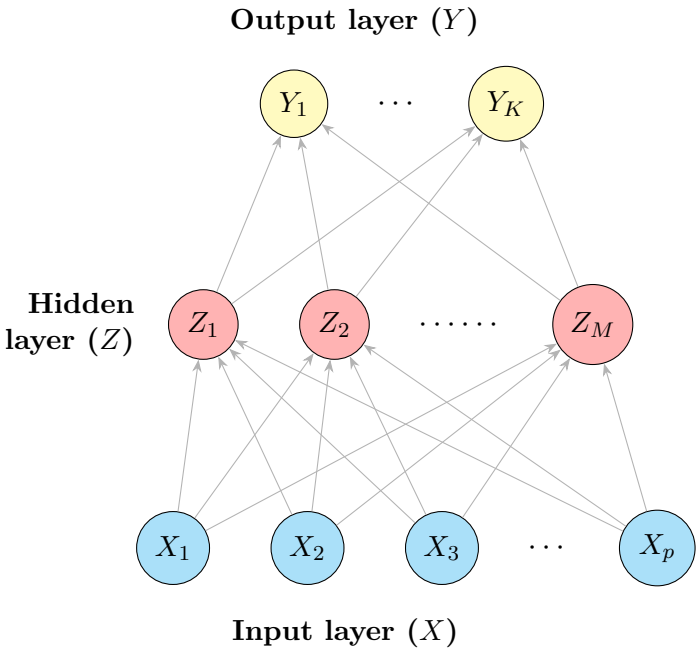


Figure 4: Schematic of a single-hidden-layer, feed-forward neural network. Input features X are transformed into hidden units Z via nonlinear activation functions, then combined linearly to produce outputs Y.

of the network is the number of hidden layers, and it is this depth that gives "deep learning" its name. Each additional layer allows the network to learn increasingly abstract representations of the data.

8.4 Estimation and Regularization

The parameters $\theta = (\alpha, \beta)$ are estimated by minimizing a loss function—nonlinear least squares for regression.

With many hidden units and many layers, neural networks have an enormous number of parameters. A single-hidden-layer network with 100 inputs and 50 hidden units already has $100 \times 50 + 50 = 5{,}050$ parameters in the first layer alone. To prevent overfitting, we **regularize** by adding a penalty term, exactly as in Ridge regression:

Neural Network Regularization

$$\min_{\theta} \; R(\theta) + \lambda J(\theta), \qquad J(\theta) = \sum_{k,m} \beta_{km}^2 + \sum_{m,l} \alpha_{ml}^2, \tag{16}$$

where $R(\theta)$ is the loss function (e.g., SSR) and λ is the tuning parameter controlling the strength of regularization.

The penalty $J(\theta)$ is the sum of squared weights throughout the network—the neural network analogue of the Ridge penalty. It prevents any single connection from becoming too large, which would allow the network to memorize noise. As always, λ is chosen by cross-validation.

9 Conclusion and Caveats

Let us step back and take stock of what we have learned.

The bias-variance tradeoff is the organizing principle of this chapter. Model complexity is double-edged: too simple and we miss the signal (high bias); too complex and we memorize the noise (high variance). Every model selection method is, at its core, a strategy for navigating this tradeoff.

Goodness-of-fit criteria—adjusted R^2, AIC, BIC, and cross-validation—provide the standards by which we evaluate models. Among these, cross-validation is particularly valuable because it directly measures out-of-sample prediction quality.

Machine learning algorithms give us automatic procedures for searching the model space. Penalized regression (LASSO, Ridge, Elastic Net) stays within the linear model and controls complexity by shrinking coefficients. Tree-based methods (CART, Random Forests) partition the covariate space and average within regions. Neural networks compose nonlinear transformations to learn flexible representations.

Causal Forests are an important application for economists: they adapt Random Forests to estimate heterogeneous treatment effects under unconfoundedness, providing a principled alternative to ad hoc subgroup analysis.

Machine Learning Is Not a Substitute for Economic Reasoning

Remember: these are *statistical tools* for prediction and model selection. The most important ingredient in empirical economics is still your **economic intuition**. Machine learning can tell you which variables predict Y well, but it cannot tell you which variables have a *causal* effect on Y. Never let a data-driven algorithm

override substantive economic reasoning. For example, you should never exclude *education* from a wage equation just because AIC or BIC tells you to do so—economic theory demands its inclusion.

One final observation. In this chapter, we focused on model selection conditional on the *unconfoundedness* assumption. Everything we discussed—penalized regression, trees, forests, neural networks—is about prediction: how to best approximate $E[Y \mid X]$. We did not use any prior knowledge about causal structure.

But in economics, we often *do* have such knowledge. We know that education causally affects wages. We know that prices causally affect demand. Can we incorporate this knowledge into the model selection process? The answer is yes, and the tool for doing so is the **causal graph** (Directed Acyclic Graph, or DAG). This will be the topic of Chapter 4.

A Homework Problems

1. **Bias-variance decomposition.** Consider the data-generating process $Y = f(X) + \varepsilon$ with $E[\varepsilon] = 0$ and $\text{Var}(\varepsilon) = \sigma_\varepsilon^2$. Let $\hat{f}(x)$ be an estimator trained on a random sample. Derive the bias-variance decomposition:

$$E\Big[(Y - \hat{f}(x_0))^2\Big] = \sigma_\varepsilon^2 + \Big[\text{Bias}(\hat{f}(x_0))\Big]^2 + \text{Var}(\hat{f}(x_0)).$$

   Clearly state where you use the assumption that ε is independent of $\hat{f}$.

2. **LASSO vs. Ridge.** Consider the penalized regression problem:

$$\min_\beta \sum_{i=1}^{n} (y_i - x_i'\beta)^2 + \lambda \left(\|\beta\|_p\right)^p.$$

   (a) Explain intuitively why the ℓ_1 penalty ($p = 1$) can produce exact zeros in $\hat{\beta}$ while the ℓ_2 penalty ($p = 2$) cannot.

   (b) In a wage regression with 50 potential covariates, would you prefer LASSO or Ridge? Justify your choice in terms of interpretability and the economic context.

3. **Cross-validation for polynomial order selection.** Suppose you have $n = 100$

observations from the DGP $Y = 1 + 2X - 0.5X^2 + \varepsilon$, $\varepsilon \sim N(0, 4)$. Describe how you would use 5-fold cross-validation to choose between a linear, quadratic, and cubic polynomial model. Which model do you expect to be selected, and why?

Practice Example: LASSO and Ridge Regression in Stata

This practice example illustrates penalized regression methods in Stata. We simulate a dataset with a known sparse data-generating process—only a few covariates truly matter—and compare the performance of OLS, LASSO, and Ridge regression.

Setup

The data-generating process is:

$$Y_i = 2X_{1i} + 3X_{2i} - 1.5X_{3i} + \varepsilon_i, \qquad \varepsilon_i \sim N(0,1), \qquad X_{ji} \sim N(0,1) \text{ for } j = 1, \ldots, 20, \qquad n = 200.$$

Only three of the twenty covariates have nonzero coefficients. The remaining seventeen are pure noise variables with $\beta_j = 0$. This "sparse" structure is exactly the kind of problem where LASSO should outperform OLS.

Learning Objectives

1. OLS uses all 20 regressors and assigns nonzero (noisy) coefficients to the irrelevant variables.
2. LASSO identifies the three relevant variables by shrinking the irrelevant coefficients to zero.
3. Ridge shrinks all coefficients toward zero but does not set any exactly to zero.
4. Cross-validation selects the tuning parameter λ that minimizes out-of-sample prediction error.
5. In-sample RMSE comparison: OLS, LASSO, Ridge, and the irreducible noise $\sigma = 1$.

Replication Code

The Stata do-file `src/chapter 3/ch3_ml.do` reproduces all results. The simulated dataset is saved to `data/chapter 3/ch3_ml.dta`.

References

Athey, S. and Imbens, G. W. (2019). Machine learning methods that economists should know about. *Annual Review of Economics*, 11:685–725.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition.

Levy, R. (2021). Social media, news consumption, and polarization: Evidence from a field experiment. *American Economic Review*, 111(3):831–870.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288.

Wager, S. and Athey, S. (2018). Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523):1228–1242.