Data Science for Mathematicians
Lesson 4: Model Evaluation and Statistical Inference

February 9, 2026

Abstract

This lecture transitions from the mechanics of model fitting to the critical discipline of model evaluation. We establish a formal mathematical framework for understanding and quantifying a model's ability to generalize to unseen data. The cornerstone of this framework is the Bias-Variance Decomposition, which we will derive from first principles to decompose the expected prediction error into its constituent parts: bias, variance, and irreducible noise. Building upon this theoretical foundation, we develop a practical toolkit of evaluation metrics for both regression (Mean Squared Error, R^2, Adjusted R^2) and classification (Accuracy, Precision, Recall, F_1-Score), with a rigorous analysis of their properties and limitations. Finally, we introduce Cross-Validation as the canonical, robust procedure for estimating generalization error, facilitating principled model selection and hyperparameter tuning. This lecture provides the necessary statistical machinery to assess the quality of the models constructed in previous weeks and motivates the need for the numerical optimization techniques to be studied subsequently.

1 Beyond Model Fitting

In previous lessons, we derived the OLS estimator for linear regression and built our first probabilistic classifier with Naive Bayes. Both focused on *fitting* models to data.

We now arrive at a question of paramount importance, one that separates the practice of data science from mere curve-fitting: having fit a model, is it any good? This question is far more subtle than it appears. A low error on the training data, while desirable, is not a sufficient, nor even always a necessary, condition for a model to be considered effective. This lecture is dedicated to building a rigorous mathematical framework to answer this question.

1.1 From Training Error to Generalization

The OLS solution $\hat{\beta}$ was derived by solving an optimization problem:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \|y - X\beta\|_2^2.$$

The quantity being minimized, $\|y - X\beta\|_2^2$, is the sum of squared residuals on the training data. Let us call the minimized value of this objective, scaled by the number of data points, the *training error*. Similarly, when we used maximum likelihood estimation, we found parameters that maximized the probability of observing the training data.

The central intellectual shift we must now make is from a deterministic, optimization-focused perspective to a statistical, inferential one. The dataset we possess, $D = \{(x_i, y_i)\}_{i=1}^n$, is almost never the entire population of interest. It is a *sample*. If we were to draw a different sample, D', from the same underlying data-generating process, we would obtain a different set of observations (X', y') and consequently, a different OLS estimate $\hat{\beta}' = ((X')^T X')^{-1}(X')^T y'$. This

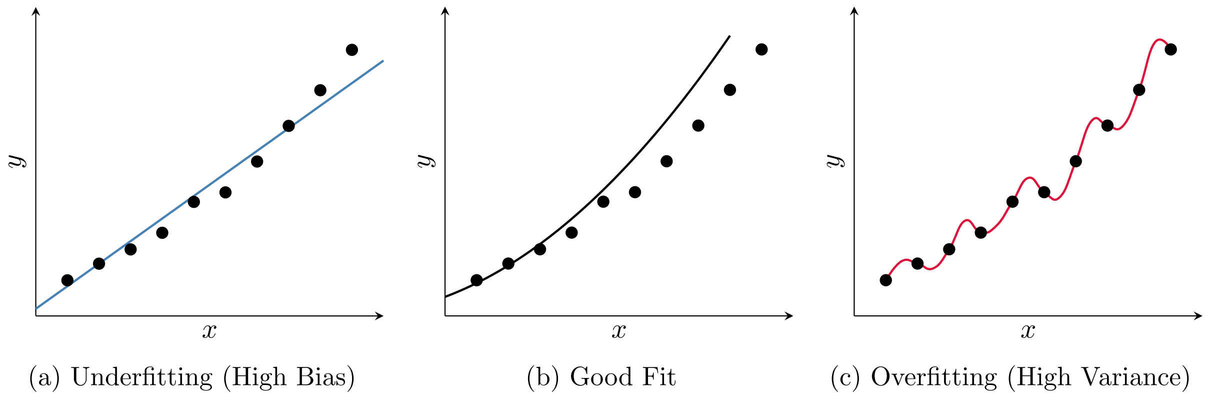

Figure 1: Illustration of model complexity. (a) An underfit linear model fails to capture the quadratic nature of the data. (b) A well-fit quadratic model captures the underlying trend. (c) An overfit high-degree polynomial wiggles to pass through every training point, fitting the noise.

reveals a crucial fact: the estimator $\hat{\beta}$ is not a fixed vector, but a *random variable*. Its value is a function of the random sample drawn from the population.

Therefore, our goal cannot be simply to minimize the error on our particular, idiosyncratic sample. That would be akin to claiming a universal truth based on a single, limited experiment. Our true objective is to find a model that performs well on average on *new, unseen data* drawn from the same underlying process. The performance on unseen data is known as the *generalization error*. The entire discipline of model evaluation is fundamentally about estimating this quantity.

1.2 Memorization vs. Generalization

The distinction between training error and generalization error is most starkly illustrated by the phenomenon of *overfitting*. An overfit model is one that corresponds too closely to the particularities of the training data, including its random noise, rather than the true underlying relationship.

Consider the analogy of a student preparing for an examination. One student diligently works to understand the fundamental principles and concepts of the subject matter. Another student simply memorizes the exact answers to every question in the practice textbook. On an exam composed of questions taken verbatim from the textbook, the second student may outperform the first. However, on an exam with new questions that test the same underlying principles, the first student will excel while the second will fail catastrophically.

The first student has learned a model that *generalizes*. The second has constructed a model that has merely *memorized* the training data. In machine learning, a model with excessive complexity—for instance, a high-degree polynomial for a simple relationship—can *memorize* the training data by contorting itself to pass through every data point. This results in a training error of or near zero, but the model will perform poorly on new data because it has fit the noise, not the signal. This pathology, overfitting, is a primary concern in all of statistical modeling.

Our goal is not memorization, but generalization. We seek a model that captures the stable, repeatable patterns in the data-generating process, while ignoring the stochastic noise inherent in any single sample.

1.3 A Mathematical Framework for Evaluation

To navigate the perils of overfitting and to formalize the concept of generalization, this lecture will proceed in three parts:

1. **A Theoretical Framework for Error:** We will begin by formally deriving the **Bias-Variance Decomposition**. This foundational result in statistical learning theory provides a mathematical language for understanding generalization error, decomposing it into three distinct components. This will give us a theoretical lens through which to analyze the behavior of our models.

2. **A Practical Toolkit of Metrics:** We will then translate this theory into practice by defining and critically analyzing a suite of evaluation metrics. We will cover metrics for regression tasks, where the goal is to predict a continuous value, and for classification tasks, where the goal is to predict a discrete label.

3. **A Robust Procedure for Estimation:** Finally, we will address the problem of how to estimate these metrics reliably. We will show that evaluating a model on its training data is a flawed approach and introduce **Cross-Validation** as the gold-standard statistical procedure for obtaining a stable and unbiased estimate of a model's true generalization error. This procedure is the primary tool for model selection and hyperparameter tuning.

2 The Bias-Variance Tradeoff

We now develop the central theoretical tool for understanding generalization error. The Bias-Variance Decomposition allows us to dissect the expected error of a model into three fundamental components, revealing the sources of error and the inherent tradeoffs in model complexity.

2.1 The Data Generating Process

To proceed with mathematical rigor, we must first formalize our assumptions about where the data comes from.

We assume that our observed data arises from a true, but unknown, function $f : \mathbb{R}^p \to \mathbb{R}$. The relationship between an input vector $x \in \mathbb{R}^p$ and its corresponding scalar output $y \in \mathbb{R}$ is given by the model:

$$y = f(x) + \epsilon$$

where ϵ is a random noise variable.

The function f represents the systematic, deterministic part of the relationship. It is the *signal* that we wish to learn. The term ϵ represents the stochastic, or *noisy*, part of the process. It captures measurement error, unmodeled variables, and inherent randomness in the system.

We make the following standard assumptions about the noise term:

1. The expected value of the noise is zero:

$$\mathbb{E}[\epsilon] = 0.$$

   This implies that the noise is unbiased; it does not systematically push the observed value y above or below the true value $f(x)$.

2. The variance of the noise is constant and finite:

$$\mathrm{Var}(\epsilon) = \mathbb{E}[\epsilon^2] - (\mathbb{E}[\epsilon])^2 = \mathbb{E}[\epsilon^2] = \sigma^2.$$

   The value σ^2 is a measure of the inherent variability in the data-generating process.

Our goal is to use a training dataset, $D = \{(x_i, y_i)\}_{i=1}^n$, which is a random sample from this process, to learn an estimator, denoted $\hat{f}(x)$. This estimator is our model's prediction for a given input x. Crucially, since $\hat{f}$ is constructed from the random sample D, the function $\hat{f}$ is itself a random quantity. If we were to draw a different training set D', we would obtain a different function $\hat{f}'$. We can make this dependence explicit by writing $\hat{f}(x; D)$.

2.2 Decomposing the Expected Prediction Error

Let us consider a new, unseen data point (x_0, y_0), where $y_0 = f(x_0) + \epsilon_0$. We are interested in the expected squared prediction error of our estimator $\hat{f}$ at this point. This is the quantity we aim to minimize to achieve good generalization.

Definition 2.1 (Expected Prediction Error)**.** The expected squared prediction error of an estimator $\hat{f}$ at a point x_0 is given by:

$$\mathbb{E}[(y_0 - \hat{f}(x_0))^2].$$

The expectation is taken over all sources of randomness: the random choice of the training dataset D and the random noise ϵ_0 in the new data point y_0.

We now arrive at the main result of this section.

Theorem 2.2. *For a data generating process $y = f(x) + \epsilon$ with $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Var}(\epsilon) = \sigma^2$, the expected squared prediction error of an estimator $\hat{f}(x)$ at a point x_0 can be decomposed as:*

$$\mathbb{E}[(y_0 - \hat{f}(x_0))^2] = \underbrace{\left(\mathbb{E}[\hat{f}(x_0)] - f(x_0)\right)^2}_{\textit{Squared Bias}} + \underbrace{\mathbb{E}\left[\left(\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)]\right)^2\right]}_{\textit{Variance}} + \underbrace{\sigma^2}_{\textit{Irreducible Error}}$$

where the expectations involving $\hat{f}$ are taken over the distribution of training datasets D.

Proof. To simplify the notation, let us write $\hat{f}$ for $\hat{f}(x_0; D)$ and f for $f(x_0)$. Our goal is to decompose the quantity $\mathbb{E}[(y_0 - \hat{f})^2]$. The key algebraic step is to add and subtract the expected value of our estimator, $\mathbb{E}[\hat{f}]$, inside the squared term.

$$\begin{aligned}
\mathbb{E}[(y_0 - \hat{f})^2] &= \mathbb{E}\left[\left((y_0 - \mathbb{E}[\hat{f}]) + (\mathbb{E}[\hat{f}] - \hat{f})\right)^2\right] \\
&= \mathbb{E}\left[(y_0 - \mathbb{E}[\hat{f}])^2 + (\mathbb{E}[\hat{f}] - \hat{f})^2 + 2(y_0 - \mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}] - \hat{f})\right] \\
&= \mathbb{E}[(y_0 - \mathbb{E}[\hat{f}])^2] + \mathbb{E}[(\mathbb{E}[\hat{f}] - \hat{f})^2] + 2\mathbb{E}[(y_0 - \mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}] - \hat{f})].
\end{aligned}$$

We will now analyze each of these three terms separately.

Term 2: The Variance. The second term is $\mathbb{E}[(\hat{f} - \mathbb{E}[\hat{f}])^2]$. This is precisely the definition of the variance of the random variable $\hat{f}$, i.e.,

$$\mathbb{E}[(\hat{f} - \mathbb{E}[\hat{f}])^2] = \mathrm{Var}(\hat{f}).$$

Term 3: The Cross-Term. The third term is $2\mathbb{E}[(y_0 - \mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}] - \hat{f})]$. We analyze the expectation. The term $\hat{f}$ depends on the training data D, while y_0 depends on the new noise ϵ_0. These are independent. $\mathbb{E}[\hat{f}]$ is a constant, $\mathbb{E}_D[\hat{f}]$. Therefore

$$\begin{aligned}
\mathbb{E}[(y_0 - \mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}] - \hat{f})] &= \mathbb{E}_{y_0,D}\left[(y_0 - \mathbb{E}[\hat{f}])(\mathbb{E}[\hat{f}] - \hat{f})\right] \\
&= \mathbb{E}_{y_0}\left[y_0 - \mathbb{E}[\hat{f}]\right] \cdot \mathbb{E}_D\left[\mathbb{E}[\hat{f}] - \hat{f}\right].
\end{aligned}$$

Let's evaluate the second factor:

$$\mathbb{E}_D\left[\mathbb{E}[\hat{f}] - \hat{f}\right] = \mathbb{E}_D\left[\mathbb{E}[\hat{f}]\right] - \mathbb{E}_D\left[\hat{f}\right].$$

By definition, $\mathbb{E}[\hat{f}]$ is $\mathbb{E}_D[\hat{f}]$, which is a constant. The expectation of a constant is the constant itself. Thus

$$\mathbb{E}_D\left[\mathbb{E}[\hat{f}]\right] - \mathbb{E}_D\left[\hat{f}\right] = \mathbb{E}[\hat{f}] - \mathbb{E}[\hat{f}] = 0.$$

Therefore, the entire third term is zero.

Term 1: The Bias and Irreducible Error. We are left with the first term, $\mathbb{E}[(y_0 - \mathbb{E}[\hat{f}])^2]$. Here, the only random quantity is $y_0 = f + \epsilon_0$, as $\mathbb{E}[\hat{f}]$ is a fixed value (the average prediction of our model over all possible datasets).

$$\begin{aligned}
\mathbb{E}[(y_0 - \mathbb{E}[\hat{f}])^2] &= \mathbb{E}[(f + \epsilon_0 - \mathbb{E}[\hat{f}])^2] \\
&= \mathbb{E}\left[\left((f - \mathbb{E}[\hat{f}]) + \epsilon_0\right)^2\right] \\
&= \mathbb{E}\left[(f - \mathbb{E}[\hat{f}])^2 + 2\epsilon_0(f - \mathbb{E}[\hat{f}]) + \epsilon_0^2\right] \\
&= \mathbb{E}[(f - \mathbb{E}[\hat{f}])^2] + 2(f - \mathbb{E}[\hat{f}])\mathbb{E}[\epsilon_0] + \mathbb{E}[\epsilon_0^2].
\end{aligned}$$

From our initial assumptions, we know $\mathbb{E}[\epsilon_0] = 0$ and $\mathbb{E}[\epsilon_0^2] = \mathrm{Var}(\epsilon_0) = \sigma^2$. The term $(f - \mathbb{E}[\hat{f}])^2$ is a constant with respect to the expectation over ϵ_0. So, this simplifies as follows.

$$(f - \mathbb{E}[\hat{f}])^2 + \sigma^2.$$

The term $(f - \mathbb{E}[\hat{f}])^2$ is the squared difference between the true function value and the average prediction of our model. This is the definition of the squared bias.

Putting it all together: Combining our results for the three terms, we have shown that the expected squared prediction error is the sum of the results from Term 1 and Term 2 (since Term 3 is zero):

$$\mathbb{E}[(y_0 - \hat{f})^2] = \left(f(x_0) - \mathbb{E}[\hat{f}(x_0)]\right)^2 + \mathrm{Var}(\hat{f}(x_0)) + \sigma^2$$

This completes the proof. □

2.3 Analysis of Components

Theorem 2.2 is profound because it tells us that a model's expected error comes from three distinct sources.

Definition 2.3 (Irreducible Error)**.** The irreducible error of the data-generating process $y = f(x) + \epsilon$ is defined as:

$$\sigma^2 = \mathrm{Var}(\epsilon) = \mathbb{E}[\epsilon^2].$$

It is the variance of the noise term ϵ and represents a uniform lower bound on the expected prediction error for any estimator $\hat{f}$.

The irreducible error is inherent to the data-generating process itself. No model, no matter how sophisticated, can eliminate it. It arises from measurement error, unmodeled variables, and intrinsic randomness. This is the price we pay for modeling a stochastic, rather than a purely deterministic, world.

Definition 2.4 (Bias)**.** The **bias** of an estimator $\hat{f}$ at a point x_0 is defined as:

$$\text{Bias}(\hat{f}(x_0)) = \mathbb{E}_D[\hat{f}(x_0)] - f(x_0),$$

where the expectation is taken over the distribution of training datasets D. The squared bias is therefore:

$$\text{Bias}^2(\hat{f}(x_0)) = \left(\mathbb{E}_D[\hat{f}(x_0)] - f(x_0)\right)^2.$$

The squared bias measures the systematic error introduced by the model's simplifying assumptions. It quantifies the discrepancy between the average prediction of the model—computed over all possible training datasets—and the true value of the underlying function. A model with high bias is systematically incorrect; its functional form is not flexible enough to capture the true relationship in the data. This leads to *underfitting*. For example, using a linear model to fit a cubic relationship will result in high bias.

Definition 2.5 (Variance)**.** The variance of an estimator $\hat{f}$ at a point x_0 is defined as:

$$\text{Var}(\hat{f}(x_0)) = \mathbb{E}_D\left[\left(\hat{f}(x_0) - \mathbb{E}_D[\hat{f}(x_0)]\right)^2\right],$$

where the expectation is taken over the distribution of training datasets D.

The variance measures the sensitivity of the estimator to the particular training dataset used. It quantifies the extent to which the model's prediction at x_0 fluctuates across different draws of the training data. A model with high variance is unstable: small changes in the training sample lead to large changes in the fitted function. This occurs when a model is too complex, allowing it to fit the random noise in the training sample, and leads to *overfitting*. For example, a 20th-degree polynomial will likely have very high variance, as its shape will change dramatically depending on the exact training points it is given.

2.4 Model Complexity and the Tradeoff Curve

The key insight from the decomposition is that bias and variance are often in opposition. The mechanism we can control to navigate this opposition is *model complexity*.

- **Simple Models (Low Complexity):** Models with few parameters or strong assumptions (e.g., linear regression) tend to have **high bias** and **low variance**. They are systematically wrong if the true function is complex, but their predictions are stable and do not change much from one dataset to the next.
- **Complex Models (High Complexity):** Models with many parameters or weak assumptions (e.g., high-degree polynomials, deep neural networks) tend to have **low bias** and **high variance**. They are flexible enough to approximate the true function closely, but they are also so flexible that they fit the noise in the training data, making them unstable.

This relationship is known as the **bias-variance tradeoff**. As we increase a model's complexity, its bias decreases, but its variance increases. The total expected error, being the sum of squared bias and variance (plus the irreducible error), will typically be a U-shaped curve. Our goal in model selection is to find the *sweet spot* at the bottom of this curve, where the complexity is just right to minimize the total error.

This tradeoff provides a powerful theoretical justification for why we might prefer a simpler, slightly *wrong* (biased) model over a highly complex one. For example, regularization techniques like Ridge and LASSO regression, which you will encounter later, intentionally introduce bias into the OLS estimator by shrinking its coefficients. They do this because the corresponding reduction in variance is often much larger than the increase in squared bias, leading to a lower overall expected prediction error. The goal is not to eliminate bias at all costs, but to find the optimal balance that minimizes the total reducible error, $\text{Bias}^2 + \text{Var}$.

3 Metrics for Regression Models

The Bias-Variance Decomposition provides a theoretical understanding of error. We now turn to the practical task of estimating this error from a finite sample. This section develops a toolkit of metrics for evaluating regression models, where the goal is to predict a continuous outcome.

3.1 Mean Squared Error and Root Mean Squared Error

The most direct empirical analogue to the theoretical expected prediction error is the Mean Squared Error.

Definition 3.1. Given a set of n observations (y_i) and a model's predictions $(\hat{y}_i)$, the **Mean Squared Error (MSE)** is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Remark 3.2. Note that the MSE is precisely the objective function that we minimized to derive the OLS estimator. Therefore, by construction, OLS finds the linear model with the lowest possible MSE on the training data. Our task now is to estimate the MSE on *unseen* data.

While MSE is a cornerstone metric, its units are the square of the units of the target variable (e.g., dollars squared), which can make interpretation difficult. A simple modification resolves this.

Definition 3.3. The **Root Mean Squared Error (RMSE)** is the square root of the MSE:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

The utility of RMSE lies in its interpretability. It has the same units as the target variable y. An RMSE of \$5,000 is more directly understandable than an MSE of 25,000,000 $\$^2$. It can be loosely interpreted as the typical or average magnitude of the prediction error.

3.2 The Coefficient of Determination

MSE and RMSE provide an absolute measure of a model's error. Often, we want a relative measure: how much better is our model than a very simple baseline? The coefficient of determination, or R^2, provides such a measure.

To define R^2, we first establish a baseline model. The simplest possible prediction model is one that ignores all predictors X and simply predicts the sample mean of the outcome, $\bar{y}$, for every observation.

Definition 3.4. The **Total Sum of Squares** (SS_{tot}) is the sum of squared differences between the observed values and their mean. It is proportional to the sample variance of y and represents the total variation in the data that we seek to explain.

$$\text{SS}_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

SS_{tot} can be interpreted as the error of our baseline *mean-only* model.

Next, we define the error made by our actual model.

Definition 3.5. The **Residual Sum of Squares** ($\mathrm{SS}_{\mathrm{res}}$) is the sum of squared differences between the observed values and the model's predictions. It represents the variation left unexplained by the model.

$$\mathrm{SS}_{\mathrm{res}} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

With these components, we can now define R^2.

Definition 3.6. The **Coefficient of Determination** (R^2) is defined as:

$$\mathrm{R}^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}}.$$

The most common interpretation of R^2 is as the *proportion of variance in the dependent variable that is explained by the independent variables* in the model.

- An R^2 of 1.0 indicates that the model perfectly explains the data ($\mathrm{SS}_{\mathrm{res}} = 0$).
- An R^2 of 0.0 indicates that the model is no better than the baseline model that simply predicts the mean ($\mathrm{SS}_{\mathrm{res}} = \mathrm{SS}_{\mathrm{tot}}$).
- A negative R^2 is possible if the model is actively worse than the baseline mean model ($\mathrm{SS}_{\mathrm{res}} > \mathrm{SS}_{\mathrm{tot}}$).

3.3 Critical Limitations of R^2

Despite its widespread use, R^2 suffers from several severe limitations that make it a potentially misleading metric if used uncritically. For a student of mathematics, it is essential to understand these flaws not just as trivia, but as deep indicators of the complexities of statistical modeling.

The most critical flaw is its behavior with respect to model complexity.

Lemma 3.7. *The value of* R^2 *is non-decreasing when a new predictor is added to a linear regression model.*

Proof. Let Model 1 have predictors X_1 and Model 2 have predictors X_1 and X_2. The OLS fit for Model 2 is found by minimizing $\mathrm{SS}_{\mathrm{res}}$ over a larger set of parameters than Model 1. The parameter space for Model 1 is a subspace of the parameter space for Model 2 (where the coefficients for X_2 are constrained to be zero). Since OLS minimizes $\mathrm{SS}_{\mathrm{res}}$, the minimum found over the larger space must be less than or equal to the minimum found over the constrained subspace. Thus, $\mathrm{SS}_{\mathrm{res}}(\text{Model 2}) \leq \mathrm{SS}_{\mathrm{res}}(\text{Model 1})$. Since $\mathrm{SS}_{\mathrm{tot}}$ is constant, it follows that $\mathrm{R}^2(\text{Model 2}) \geq \mathrm{R}^2(\text{Model 1})$. □

This property means that R^2 actively encourages overfitting. One can always increase the R^2 of a model by adding more variables, even if those variables are pure random noise. This makes R^2 a poor tool for comparing models with different numbers of predictors.

Furthermore, a high R^2 is not a synonym for a *good model*:

- ***R^2* does not measure goodness of fit:** A model can be fundamentally misspecified (e.g., fitting a linear model to data with a clear non-linear pattern) and still achieve a high R^2. It does not validate the model's underlying assumptions.
- **R^2 does not measure predictive accuracy:** As a relative measure, it can be misleading. Two models can have the same predictive error (MSE), but vastly different R^2 values if the inherent variance ($\mathrm{SS}_{\mathrm{tot}}$) of the datasets differs. MSE is the direct measure of predictive error.

- R^2 **is not comparable across transformations:** One cannot compare the R^2 of a model predicting y with the R^2 of a model predicting $\log(y)$. The $\mathrm{SS}_{\mathrm{tot}}$ values are on different scales.

The uncritical pursuit of a high R^2 score is a common anti-pattern in data analysis, often leading to overly complex, overfit models that generalize poorly. A model's quality is a multi-faceted concept that cannot be collapsed into a single number. A thorough evaluation must also include residual analysis, checks of model assumptions, and consideration of the Bias-Variance tradeoff.

3.4 A Penalized Alternative

To address the most glaring flaw of R^2—its monotonic increase with the number of predictors—the Adjusted R^2 was developed. It modifies the R^2 formula to include a penalty for model complexity.

Definition 3.8 (Adjusted R^2)**.** The **Adjusted** R^2 is defined as:

$$\mathrm{R}^2_{\mathrm{adj}} = 1 - \frac{(1-\mathrm{R}^2)(n-1)}{n-p-1}$$

where n is the number of samples and p is the number of predictors in the model.

The terms $(n-p-1)$ and $(n-1)$ are the degrees of freedom for the residuals and the total, respectively. The intuition is that as we add a predictor, p increases. This increases the penalty term $(n-1)/(n-p-1)$, which in turn increases the quantity being subtracted from 1, thus pushing $\mathrm{R}^2_{\mathrm{adj}}$ down. The adjusted R-squared will only increase if the addition of the new variable reduces $\mathrm{SS}_{\mathrm{res}}$ by an amount sufficient to overcome this penalty. This makes $\mathrm{R}^2_{\mathrm{adj}}$ a more suitable metric for comparing models with different numbers of predictors, as it rewards parsimony.

4 Metrics for Classification Models

We now shift our focus to classification problems, where the goal is to predict a discrete categorical label. The evaluation of classifiers presents a different set of challenges, particularly when dealing with datasets where the classes are not equally represented.

4.1 The Pitfall of Accuracy in Imbalanced Scenarios

The most intuitive metric for a classifier is accuracy: the proportion of predictions that were correct.

Definition 4.1 (Accuracy)**.** Given a set of predictions, **Accuracy** is defined as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

While simple, accuracy can be dangerously misleading, especially in the presence of class imbalance.

Example 4.2. Consider a dataset of 1,000,000 credit card transactions, of which 100 are fraudulent (0.01%) and 999,900 are legitimate (99.99%). Let's evaluate a trivial *model* that always predicts *not fraudulent*.

- Correct predictions: 999,900 (the legitimate transactions).
- Incorrect predictions: 100 (the fraudulent transactions it missed).

Table 1: The Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- Accuracy $= \dfrac{999,900}{1,000,000} = 99.99\%$.

This model achieves 99.99% accuracy, yet it is completely useless because it fails to identify a single case of fraud, which is the entire purpose of the model.[18] This example demonstrates that for imbalanced problems, accuracy is not a meaningful metric.

4.2 A Comprehensive View of Performance

To move beyond the limitations of accuracy, we must disaggregate a model's performance. The fundamental tool for this is the confusion matrix, which provides a complete summary of the classification results. For a binary classification problem with classes *Positive* and *Negative*:

Definition 4.3 (Confusion Matrix)**.** Let $\{(x_i, y_i)\}_{i=1}^{n}$ be a set of n observations with true binary labels $y_i \in \{0,1\}$, and let $\hat{y}_i \in \{0,1\}$ denote the corresponding predicted labels, where 1 represents the *positive* class and 0 represents the *negative* class. The **confusion matrix** is defined by the following four counts:

$$\begin{aligned}
\mathrm{TP} &= \left|\{i : y_i = 1 \text{ and } \hat{y}_i = 1\}\right|, && \text{(True Positives)}\\
\mathrm{TN} &= \left|\{i : y_i = 0 \text{ and } \hat{y}_i = 0\}\right|, && \text{(True Negatives)}\\
\mathrm{FP} &= \left|\{i : y_i = 0 \text{ and } \hat{y}_i = 1\}\right|, && \text{(False Positives / Type I Error)}\\
\mathrm{FN} &= \left|\{i : y_i = 1 \text{ and } \hat{y}_i = 0\}\right|. && \text{(False Negatives / Type II Error)}
\end{aligned}$$

These four quantities partition the n observations exhaustively, i.e., $\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN} = n$.

These four quantities can be arranged in a 2×2 matrix that forms the basis for nearly all other classification metrics.

Using this notation, accuracy can be expressed as

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}.$$

The power of the confusion matrix is that it separates the two types of errors (FP and FN), which often have vastly different real-world consequences.

Example 4.4. Suppose a medical test is applied to $n = 200$ patients, of whom 50 truly have a disease ($y_i = 1$) and 150 are healthy ($y_i = 0$). The model predicts positive for 60 patients. Of these 60 positive predictions, 40 are correct and 20 are incorrect. The confusion matrix is then:

$$\mathrm{TP} = 40, \quad \mathrm{FP} = 20, \quad \mathrm{FN} = 10, \quad \mathrm{TN} = 130.$$

We can verify the partition property: $40 + 130 + 20 + 10 = 200 = n$. From these counts we can immediately read off that the model correctly identified 40 out of 50 sick patients but also raised 20 false alarms among the 150 healthy patients.

4.3 Quantifying Error Types

From the confusion matrix, we can define two crucial metrics that focus on the performance on the positive class: Precision and Recall.

Definition 4.5 (Precision)**.** Precision, also known as positive predictive value (PPV), is the proportion of positive predictions that were actually correct.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Precision answers the question: *"Of all the instances we predicted to be positive, what fraction was correct?"*. High precision is critical when the cost of a false positive is high.

Example 4.6. Continuing the medical test scenario with TP = 40, FP = 20, FN = 10, and TN = 130, we compute:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{40}{40 + 20} = \frac{40}{60} = \frac{2}{3} \approx 0.667.$$

This means that of the 60 patients the model flagged as having the disease, only two-thirds actually had it. The remaining one-third were false alarms. In a context where a positive prediction triggers an invasive or costly follow-up procedure, such a precision rate may be unacceptable.

Example 4.7. In an email spam filter, a false positive occurs when a legitimate email is incorrectly marked as spam. This can be very costly, as the user might miss an important message. Therefore, a spam filter should be optimized for high precision.

Definition 4.8 (Recall)**.** Recall, also known as sensitivity or true positive rate (TPR), is the proportion of actual positive cases that were correctly identified by the model.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Recall answers the question: *"Of all the actual positive instances, what fraction did our model successfully identify?"*. High recall is critical when the cost of a false negative is high.

Example 4.9. Continuing the medical test scenario with TP = 40, FP = 20, FN = 10, and TN = 130, we compute:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{40}{40 + 10} = \frac{40}{50} = 0.80.$$

The model correctly identifies 80% of the 50 patients who truly have the disease, but misses the remaining 10. In a screening context where undetected illness poses a serious risk, a recall of 0.80 means that 1 in 5 sick patients goes undiagnosed.

Example 4.10. In a medical screening test for a serious disease, a false negative occurs when a sick patient is incorrectly told they are healthy. The consequences of this error are catastrophic. Therefore, such a diagnostic tool must be optimized for high recall.

4.4 The Precision-Recall Tradeoff

It is rare for a model to excel at both precision and recall simultaneously. In fact, there is an inherent tradeoff between them. Most classification models, such as logistic regression, do not output a binary label directly. Instead, they output a probability or a score, $p(y = 1|x)$. A decision threshold, τ, is then used to make a classification: if $p(y = 1|x) > \tau$, predict *Positive*; otherwise, predict *Negative*. The standard choice is $\tau = 0.5$, but this threshold is a tunable parameter that directly controls the balance between precision and recall.

- **Increasing the threshold (e.g., $\tau = 0.9$):** The model becomes more *conservative.* It will only predict *Positive* when it is extremely confident. This will reduce the number of False Positives, thus **increasing precision**. However, it will also cause the model to miss more borderline positive cases, increasing the number of False Negatives and thus **decreasing recall**.
- **Decreasing the threshold (e.g., $\tau = 0.1$):** The model becomes more *liberal.* It will predict *Positive* even with very little evidence. This will capture more of the true positive cases, reducing False Negatives and thus **increasing recall**. However, this comes at the cost of incorrectly classifying more negative instances as positive, increasing False Positives and thus **decreasing precision**.

This tradeoff implies that there is no single *best* model, but rather a spectrum of models depending on the chosen threshold. The choice of the optimal threshold is not a purely statistical decision; it is a business or domain decision that depends on the relative costs of Type I and Type II errors. A data scientist's role is often to present the full *precision-recall curve*—a plot of precision versus recall for all possible thresholds—to stakeholders, who can then choose an operating point that aligns with their objectives.

4.5 A Unified Metric

While the precision-recall curve provides a comprehensive picture, it is often convenient to have a single-number summary of a model's performance that balances both metrics. A simple arithmetic mean of precision and recall is not ideal, as it would allow a model to score well by having perfect recall and terrible precision, or vice-versa. We need a metric that requires both to be high. The harmonic mean provides this property.

Lemma 4.11 (Property of the Harmonic Mean)**.** *For two positive numbers a and b, their harmonic mean $H(a,b) = \dfrac{2ab}{a+b}$ is always closer to the minimum of $\{a,b\}$ than their arithmetic mean $A(a,b) = \dfrac{a+b}{2}$. The harmonic mean is low if either a or b is low.*

This property makes the harmonic mean an excellent candidate for combining precision and recall.

Definition 4.12. The F_1-Score is the harmonic mean of Precision and Recall:

$$\mathrm{F}_1 = \frac{2 \cdot \mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}.$$

Example 4.13. Continuing the medical test scenario with $\mathrm{TP} = 40$, $\mathrm{FP} = 20$, $\mathrm{FN} = 10$, and $\mathrm{TN} = 130$, we first compute:

$$\mathrm{Precision} = \frac{40}{40+20} = \frac{2}{3}, \qquad \mathrm{Recall} = \frac{40}{40+10} = \frac{4}{5}.$$

The F_1-Score is then:

$$\mathrm{F}_1 = \frac{2 \cdot \frac{2}{3} \cdot \frac{4}{5}}{\frac{2}{3} + \frac{4}{5}} = \frac{\frac{16}{15}}{\frac{22}{15}} = \frac{16}{22} = \frac{8}{11} \approx 0.727.$$

Notice that the F_1-score of 0.727 lies below the arithmetic mean of precision and recall, $\frac{1}{2}\left(\frac{2}{3} + \frac{4}{5}\right) \approx 0.733$, and is pulled closer to the lower of the two values (precision). This illustrates the harmonic mean's property of penalizing imbalance between the two metrics.

The F_1-score provides a balanced measure of performance. It will only be high if both precision and recall are high. If one of the metrics is very low, the F_1-score will also be very low, severely penalizing the model. This makes it a more robust summary metric than accuracy for imbalanced classification problems.

4.6 A Generalized Unified Metric

The F_1-Score assigns equal importance to precision and recall. However, as we have seen, many real-world applications demand that one be prioritized over the other. The F_β-Score generalizes the F_1-Score by introducing a parameter $\beta > 0$ that controls this balance.

Definition 4.14 (F_β-Score)**.** For a real-valued parameter $\beta > 0$, the F_β-Score is defined as the weighted harmonic mean of Precision and Recall:

$$F_\beta = \frac{(1+\beta^2)\cdot \text{Precision}\cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}.$$

The parameter β determines the relative weight of recall versus precision:

- $\beta = 1$: Precision and recall are weighted equally, recovering the F_1-Score.
- $\beta > 1$: Recall is weighted more heavily than precision. The score penalizes false negatives more severely.
- $0 < \beta < 1$: Precision is weighted more heavily than recall. The score penalizes false positives more severely.

Intuitively, β represents the factor by which recall is considered more important than precision. An F_2-Score, for instance, weights recall twice as much as precision.

Example 4.15. Continuing the medical test scenario with Precision $= \frac{2}{3}$ and Recall $= \frac{4}{5}$, we compute the F_2-Score, which emphasizes recall—appropriate for a disease screening context where missing a sick patient (false negative) is far more costly than a false alarm.

$$F_2 = \frac{(1+2^2)\cdot\frac{2}{3}\cdot\frac{4}{5}}{2^2\cdot\frac{2}{3}+\frac{4}{5}} = \frac{5\cdot\frac{8}{15}}{\frac{8}{3}+\frac{4}{5}} = \frac{\frac{8}{3}}{\frac{52}{15}} = \frac{8}{3}\cdot\frac{15}{52} = \frac{120}{156} = \frac{10}{13} \approx 0.769.$$

Compare this with the $F_{0.5}$-Score, which emphasizes precision—appropriate for a spam filter where falsely marking a legitimate email as spam is the primary concern.

$$F_{0.5} = \frac{(1+0.5^2)\cdot\frac{2}{3}\cdot\frac{4}{5}}{0.5^2\cdot\frac{2}{3}+\frac{4}{5}} = \frac{1.25\cdot\frac{8}{15}}{\frac{1}{6}+\frac{4}{5}} = \frac{\frac{2}{3}}{\frac{29}{30}} = \frac{2}{3}\cdot\frac{30}{29} = \frac{20}{29} \approx 0.690.$$

Since recall (0.80) exceeds precision (0.667) for this model, the F_2-Score (0.769) is higher than the F_1-Score (0.727), which in turn is higher than the $F_{0.5}$-Score (0.690). This confirms that the F_β-Score shifts the emphasis as intended: higher β rewards the model for its stronger recall, while lower β penalizes it for its weaker precision.

4.7 The ROC Curve and AUC

The metrics developed so far—Precision, Recall, the F_β-Score—all depend on a fixed decision threshold τ. In Section 4.4 we saw how varying τ induces a tradeoff between Precision and Recall. We now examine a closely related but distinct tradeoff that varies τ along a different pair of axes: the True Positive Rate (TPR) and the False Positive Rate (FPR).

Recall that the True Positive Rate is simply another name for Recall:

$$\text{TPR} = \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}.$$

We now introduce the complementary quantity that measures errors among the actual negatives.

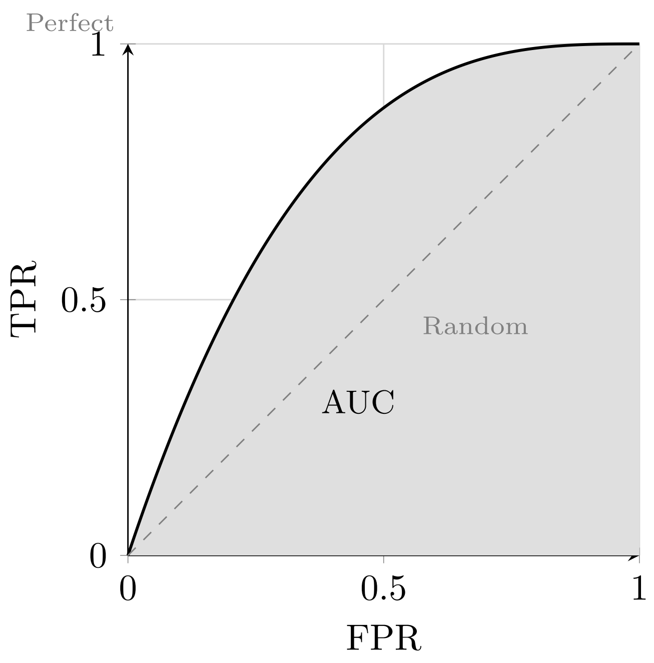

Figure 2: An ROC curve with the shaded region representing the AUC. The dashed diagonal corresponds to a random classifier.

Definition 4.16 (False Positive Rate)**.** The False Positive Rate (FPR) is the proportion of actual negatives that are incorrectly classified as positive:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}.$$

Note that $1 - \mathrm{FPR} = \frac{\mathrm{TN}}{\mathrm{FP+TN}}$ is called the *Specificity* (or True Negative Rate), so $\mathrm{FPR} = 1 - \mathrm{Specificity}$.

Definition 4.17 (ROC Curve)**.** The Receiver Operating Characteristic (ROC) curve is the parametric curve obtained by plotting TPR against FPR as the decision threshold τ varies over $[0, 1]$:

$$\mathrm{ROC} = \big\{\big(\mathrm{FPR}(\tau),\ \mathrm{TPR}(\tau)\big) : \tau \in [0, 1]\big\}.$$

Consider the behavior at the extreme thresholds. When $\tau = 1$ the classifier predicts every instance as negative, so $\mathrm{TP} = 0$, $\mathrm{FP} = 0$, giving the point $(0, 0)$. When $\tau = 0$ the classifier predicts every instance as positive, so $\mathrm{FN} = 0$, $\mathrm{TN} = 0$, giving the point $(1, 1)$. A classifier that assigns scores independently of the true class produces a ROC curve along the diagonal $\mathrm{TPR} = \mathrm{FPR}$; this corresponds to random guessing. A curve that rises steeply toward the top-left corner $(0, 1)$ indicates strong discriminative ability: the classifier achieves a high true positive rate while maintaining a low false positive rate.

Definition 4.18 (Area Under the ROC Curve (AUC))**.** The AUC is the area under the ROC curve:

$$\mathrm{AUC} = \int_0^1 \mathrm{TPR}\big(\mathrm{FPR}^{-1}(t)\big)\, dt.$$

Figure 2 illustrates an ROC curve together with its AUC, shown as the shaded region beneath the curve. The AUC has an elegant probabilistic interpretation: it equals the probability that the classifier assigns a higher score to a randomly chosen positive instance than to a randomly chosen negative instance. That is, if X^+ is the score of a random positive and X^- the score of a random negative,

$$\mathrm{AUC} = \mathbb{P}\big(X^+ > X^-\big).$$

Key reference values for interpretation:

- AUC = 1.0: Perfect classifier—positive instances always receive higher scores than negative instances.
- AUC = 0.5: Random classifier—no discriminative power; the ROC curve lies on the diagonal.
- AUC < 0.5: Worse than random—the classifier systematically ranks negatives above positives (inverting its predictions would improve performance).

Example 4.19. To illustrate how the ROC curve is constructed, consider a small dataset of $n = 10$ patients. A diagnostic model outputs a score $s(x_i) \in [0, 1]$ for each patient, and we know the true label $y_i \in \{0, 1\}$ (healthy or sick). The patients, sorted by decreasing score, are:

Patient	Score $s(x_i)$	True label y_i
1	0.95	1
2	0.90	1
3	0.82	0
4	0.78	1
5	0.65	1
6	0.55	0
7	0.40	1
8	0.35	0
9	0.20	0
10	0.10	0

There are $P = 5$ actual positives and $N = 5$ actual negatives. The ROC curve is built by sweeping the threshold τ through each distinct score value. At each threshold, every patient with $s(x_i) \geq \tau$ is classified as positive. We record the resulting TP, FP, and compute TPR = TP$/P$ and FPR = FP$/N$:

Threshold τ	TP	FP	TPR	FPR
> 0.95 (predict all negative)	0	0	$0/5 = 0.0$	$0/5 = 0.0$
0.95	1	0	$1/5 = 0.2$	$0/5 = 0.0$
0.90	2	0	$2/5 = 0.4$	$0/5 = 0.0$
0.82	2	1	$2/5 = 0.4$	$1/5 = 0.2$
0.78	3	1	$3/5 = 0.6$	$1/5 = 0.2$
0.65	4	1	$4/5 = 0.8$	$1/5 = 0.2$
0.55	4	2	$4/5 = 0.8$	$2/5 = 0.4$
0.40	5	2	$5/5 = 1.0$	$2/5 = 0.4$
0.35	5	3	$5/5 = 1.0$	$3/5 = 0.6$
0.20	5	4	$5/5 = 1.0$	$4/5 = 0.8$
0.10	5	5	$5/5 = 1.0$	$5/5 = 1.0$

The ROC curve is the plot of these (FPR, TPR) pairs, connected by line segments. The AUC can be computed exactly by summing the areas of the trapezoids (or rectangles) under each segment. Each time the FPR increases from FPR_k to FPR_{k+1}, the added area is $\frac{1}{2}(\text{TPR}_k + \text{TPR}_{k+1}) \cdot (\text{FPR}_{k+1} - \text{FPR}_k)$. From the table, FPR changes at four transitions:

$$\begin{aligned}
\Delta_1 &: (0, 0.4) \to (0.2, 0.4), && \text{area} = \tfrac{1}{2}(0.4 + 0.4)(0.2 - 0) = 0.08,\\
\Delta_2 &: (0.2, 0.8) \to (0.4, 0.8), && \text{area} = \tfrac{1}{2}(0.8 + 0.8)(0.4 - 0.2) = 0.16,\\
\Delta_3 &: (0.4, 1.0) \to (0.6, 1.0), && \text{area} = \tfrac{1}{2}(1.0 + 1.0)(0.6 - 0.4) = 0.20,\\
\Delta_4 &: (0.8, 1.0) \to (1.0, 1.0), && \text{area} = \tfrac{1}{2}(1.0 + 1.0)(1.0 - 0.8) = 0.20.
\end{aligned}$$

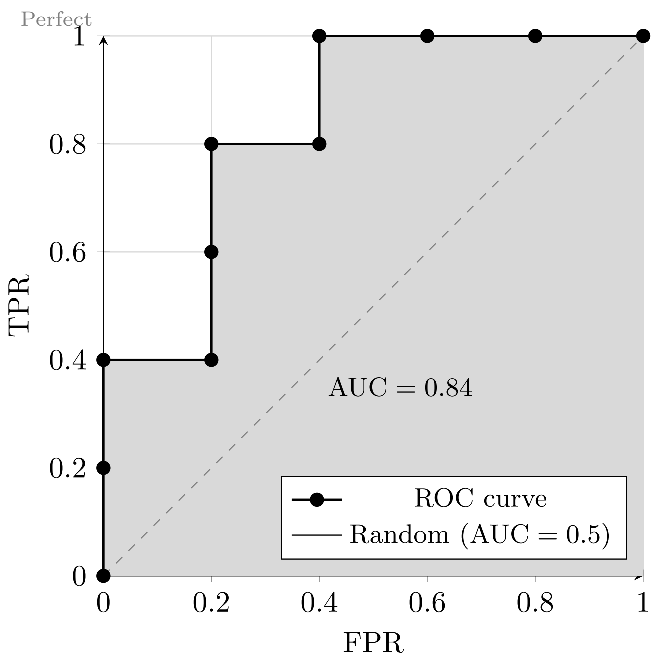

Figure 3: An illustrative ROC curve (solid) with the shaded region representing the AUC. The dashed diagonal corresponds to a random classifier (AUC = 0.5). The marked points are the operating points from the medical test example.

Summing these gives AUC $= 0.08+0.16+0.20+0.20 = 0.64+0.20 = 0.84+\cdots$. Let us be more careful—the vertical jumps (where FPR stays constant) contribute no area, and the horizontal steps occur between the following consecutive points where FPR changes:

$$\begin{aligned}
(0, 0.4) \to (0.2, 0.4) :& \quad \text{area} = 0.4 \times 0.2 = 0.08,\\
(0.2, 0.8) \to (0.4, 0.8) :& \quad \text{area} = 0.8 \times 0.2 = 0.16,\\
(0.4, 1.0) \to (0.6, 1.0) :& \quad \text{area} = 1.0 \times 0.2 = 0.20,\\
(0.6, 1.0) \to (0.8, 1.0) :& \quad \text{area} = 1.0 \times 0.2 = 0.20,\\
(0.8, 1.0) \to (1.0, 1.0) :& \quad \text{area} = 1.0 \times 0.2 = 0.20.
\end{aligned}$$

Therefore AUC $= 0.08 + 0.16 + 0.20 + 0.20 + 0.20 = 0.84$.

This value is well above 0.5, confirming that the model has strong discriminative ability. The ROC curve and the shaded AUC region for this type of classifier are illustrated in Figure 3.

The ROC curve and the Precision-Recall curve offer complementary perspectives on classifier performance, but they can diverge significantly under class imbalance. Because the FPR denominates over FP + TN, a large number of true negatives (as is typical when the positive class is rare) can make the FPR appear small even when the absolute number of false positives is large. In such settings the ROC curve—and hence the AUC—may present an overly optimistic picture of performance. Precision-Recall curves, which ignore TN entirely, are generally preferred when the positive class is rare and the cost of false positives relative to true positives is the primary concern.

5 Estimating Generalization Error

We have defined a theoretical framework for error (Bias-Variance) and a toolkit of practical metrics (MSE, F_1-score, etc.). The final piece of the puzzle is a robust procedure for estimating these metrics. How do we get a reliable estimate of a model's performance on unseen data?

5.1 The Flaw of Evaluating on Training Data

As established in our discussion of overfitting, calculating metrics on the same data that was used to train the model is a fundamentally flawed approach. The resulting performance estimate, known as the *training error*, will be systematically and often dramatically optimistic. A complex model can achieve a near-perfect score on the training data, only to fail completely on new data. Training error is a poor proxy for generalization error.

5.2 The Simple Validation Set

A straightforward improvement is to partition the data before modeling begins. We can split our dataset D into two disjoint subsets: a training set D_{train} (e.g., 80% of the data) and a validation or hold-out set D_{val} (e.g., 20% of the data).

1. Train the model $\hat{f}$ exclusively on D_{train}.
2. Evaluate the model's performance on the unseen D_{val}.

This provides a much more honest estimate of generalization error. However, this simple approach has two significant drawbacks:

1. **High Variance of the Estimate:** The performance metric calculated on D_{val} can be highly sensitive to the specific random split of the data. A different split could result in a very different performance estimate, making the evaluation unreliable.
2. **Data Inefficiency:** We have held back a portion of our data from the training process. Since statistical models generally perform better with more data, the model trained on D_{train} is likely to be systematically worse than a model trained on the full dataset D. This means we are evaluating a suboptimal model.

Example 5.1. Consider a dataset of $n = 10$ observations with a single feature x and response y:

x_i	1	2	3	4	5	6	7	8	9	10
y_i	3	5	7	8	11	12	15	16	19	24

We fit a simple linear model $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ using OLS and evaluate with MSE on a held-out 80/20 split. We examine two different random splits of the same data.

Split A. Validation set: $D_{\text{val}}^{(A)} = \{(5, 11),\ (6, 12)\}$, training set: the remaining 8 points. OLS on $D_{\text{train}}^{(A)}$ yields $\hat{y} \approx 0.29 + 2.15x$. Predicting on the validation points:

$$\hat{y}(5) = 0.29 + 2.15(5) = 11.04, \qquad \hat{y}(6) = 0.29 + 2.15(6) = 13.19.$$

The validation MSE is:

$$\text{MSE}^{(A)} = \frac{1}{2}\big[(11 - 11.04)^2 + (12 - 13.19)^2\big] = \frac{0.002 + 1.416}{2} \approx 0.71.$$

Split B. Validation set: $D_{\text{val}}^{(B)} = \{(9, 19),\ (10, 24)\}$, training set: the remaining 8 points. OLS on $D_{\text{train}}^{(B)}$ yields $\hat{y} \approx 1.11 + 1.89x$. Predicting on the validation points:

$$\hat{y}(9) = 1.11 + 1.89(9) = 18.12, \qquad \hat{y}(10) = 1.11 + 1.89(10) = 20.01.$$

The validation MSE is:

$$\text{MSE}^{(B)} = \frac{1}{2}\big[(19 - 18.12)^2 + (24 - 20.01)^2\big] = \frac{0.774 + 15.920}{2} \approx 8.35.$$

Drawback 1: High Variance. The two splits produce validation MSE estimates of 0.71 and 8.35—a difference of more than a factor of ten—even though they evaluate the same underlying model class on the same dataset. The estimate is dominated by which two points happen to land in the validation set. With only $n_{\text{val}} = 2$ observations, a single unusual point (here $y_{10} = 24$, which lies above the linear trend) can inflate the MSE dramatically. This makes any single hold-out estimate unreliable.

Drawback 2: Data Inefficiency. In Split B, the training set contains only the points $x = 1, \ldots, 8$. The model never observes the steep increase at $x = 9, 10$ during training, so it underestimates the slope ($\hat{\beta}_1 = 1.89$ versus 2.15 when training includes those points). Withholding 20% of a small dataset materially degrades the fitted model, meaning we are evaluating a model that is systematically worse than the one we would deploy in practice.

These limitations motivate a more systematic procedure that uses every data point for both training and validation.

5.3 A Robust Estimation Protocol

k-Fold Cross-Validation (CV) is a resampling procedure designed to mitigate both drawbacks of the simple validation set approach. It provides a more stable and less biased estimate of generalization error by using all data for both training and validation, systematically rotated over several iterations.

Definition 5.2 (k-Fold Cross-Validation Algorithm)**.** The k-Fold CV procedure is as follows:

1. **Partition:** Randomly partition the full dataset D of size n into k disjoint subsets (or *folds*) of approximately equal size, $D_1, D_2, \ldots, D_k$. Common choices are $k = 5$ or $k = 10$.
2. **Iterate:** For each fold $i = 1, \ldots, k$:
   (a) Define the validation set for this iteration as $D_{\text{val}}^{(i)} = D_i$.
   (b) Define the training set as all other folds: $D_{\text{train}}^{(i)} = D \setminus D_i$.
   (c) Train a new model, $\hat{f}_i$, on the training set $D_{\text{train}}^{(i)}$.
   (d) Calculate the performance metric of interest, M_i, by evaluating $\hat{f}_i$ on the validation set $D_{\text{val}}^{(i)}$.
3. **Average:** The final cross-validation score is the average of the metrics computed in each iteration:

$$\text{CV}_k = \frac{1}{k} \sum_{i=1}^{k} M_i$$

Example 5.3. We apply 5-fold cross-validation to the same dataset from the previous example ($n = 10$, $y_i \in \{3, 5, 7, 8, 11, 12, 15, 16, 19, 24\}$), again fitting a linear model $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ by OLS.

Step 1: Partition. We partition the dataset into $k = 5$ folds, each of size 2:

$$D_1 = \{(1,3),\ (2,5)\}, \quad D_2 = \{(3,7),\ (4,8)\}, \quad D_3 = \{(5,11),\ (6,12)\},$$
$$D_4 = \{(7,15),\ (8,16)\}, \quad D_5 = \{(9,19),\ (10,24)\}.$$

Step 2: Iterate. For each fold i, we hold out D_i as the validation set and train OLS on the remaining 8 points. We illustrate the computation for Fold 1 in detail.

Fold 1. Training set $D^{(1)}_{\text{train}} = D \setminus D_1 = \{(3,7),(4,8),\dots,(10,24)\}$. OLS yields $\hat{y} \approx -0.86 + 2.29x$. Predicting on D_1:

$$\hat{y}(1) \approx 1.43, \qquad \hat{y}(2) \approx 3.71.$$

The fold MSE is:

$$M_1 = \frac{1}{2}\big[(3-1.43)^2 + (5-3.71)^2\big] = \frac{2.47+1.66}{2} \approx 2.06.$$

Applying the same procedure to the remaining folds:

Fold i	Validation set D_i	OLS on $D \setminus D_i$	M_i
1	$\{(1,3),\ (2,5)\}$	$\hat{y} \approx -0.86 + 2.29x$	2.06
2	$\{(3,7),\ (4,8)\}$	$\hat{y} \approx 0.29 + 2.14x$	0.40
3	$\{(5,11),\ (6,12)\}$	$\hat{y} \approx 0.29 + 2.15x$	0.71
4	$\{(7,15),\ (8,16)\}$	$\hat{y} \approx 0.08 + 2.21x$	1.68
5	$\{(9,19),\ (10,24)\}$	$\hat{y} \approx 1.11 + 1.89x$	8.35

Step 3: Average.

$$\text{CV}_5 = \frac{1}{5}\sum_{i=1}^{5} M_i = \frac{2.06+0.40+0.71+1.68+8.35}{5} = \frac{13.20}{5} = 2.64.$$

Compare this with the single-split estimates from the previous example: Split A gave MSE $= 0.71$ and Split B gave MSE $= 8.35$. The cross-validation estimate of 2.64 is far more representative than either extreme. Moreover, every data point contributed to both training and validation exactly once, resolving the data inefficiency problem. Note that Fold 5 still yields a high $M_5 = 8.35$ due to the outlier at $y_{10} = 24$, but its influence is moderated by averaging with the other four folds.

This procedure produces a single, more robust estimate of the model's performance. It is less variable because it is an average over k different splits, and it is more efficient with data because every data point is used in a validation set exactly once.

5.4 Bias and Variance of the Cross-Validation Estimator

It is important to recognize a subtle but profound point: the cross-validation score, CV_k, is itself an *estimator* of the true, unobservable generalization error. As with any estimator, it has its own bias and variance. The choice of k involves a tradeoff between the bias and variance of this error estimate.

- **Bias of the CV Estimate:** In each fold of k-fold CV, we train our model on a dataset of size $\frac{k-1}{k} \cdot n$, which is smaller than the full dataset of size n. Since models trained on less data tend to perform worse, the error metric M_i from each fold is likely a slightly pessimistic (biased upwards) estimate of the error of a model trained on the full dataset. This bias is largest for small k (e.g., $k = 2$, where we train on only 50% of the data) and smallest for large k. As $k \to n$, the training set size approaches $n-1$, and the bias of the CV estimate approaches zero.
- **Variance of the CV Estimate:** The variance of the CV_k estimate depends on the correlation between the metrics M_i from each fold. For large k, the training sets $D^{(i)}_{\text{train}}$ are highly overlapping. For example, in 10-fold CV, any two training sets share 8/9 of their

data. This means the models $\hat{f}_i$ are highly correlated with each other, and so are their errors M_i. Averaging highly correlated quantities does not reduce variance as effectively as averaging independent quantities. Therefore, for large k, the CV estimate can have high variance. For small k, the training sets are less correlated, leading to a lower variance estimate.

This reveals a meta-level bias-variance tradeoff in our choice of evaluation protocol. There is no single *best* value of k. The common choices of $k = 5$ or $k = 10$ are empirically found to provide a good balance between the bias and variance of the generalization error estimate for many problems.

5.5 Special Cases and Variants

Definition 5.4 (Leave-One-Out Cross-Validation (LOOCV))**.** Let $D = \{(x_i, y_i)\}_{i=1}^n$ be a dataset of size n. leave-one-out cross-validation is the special case of k-fold cross-validation with $k = n$. For each $i = 1, \ldots, n$:

1. Define the validation set $D_{\text{val}}^{(i)} = \{(x_i, y_i)\}$ (a single observation).
2. Define the training set $D_{\text{train}}^{(i)} = D \setminus \{(x_i, y_i)\}$ (the remaining $n - 1$ observations).
3. Train a model $\hat{f}_i$ on $D_{\text{train}}^{(i)}$ and compute the prediction error $e_i = L(y_i, \hat{f}_i(x_i))$ for a chosen loss function L.

The LOOCV estimate of the generalization error is:

$$\text{CV}_n = \frac{1}{n} \sum_{i=1}^{n} e_i.$$

Since each training set has size $n - 1$, the LOOCV estimate has approximately zero bias. However, the n training sets overlap in $n - 2$ points, making the individual errors e_i highly correlated, which can result in high variance of the estimate. The procedure requires training n separate models, making it computationally expensive for large datasets.

Example 5.5. Consider our running dataset with $n = 10$ observations and a linear model $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$. LOOCV requires fitting 10 separate OLS models. For illustration, we compute the first two iterations using squared error loss $e_i = (y_i - \hat{y}_i)^2$:

- *Iteration 1*: Hold out $(1, 3)$. Train OLS on the 9 remaining points to obtain $\hat{y} \approx 0.17 + 2.13x$. Predict $\hat{y}(1) = 2.30$. Error: $e_1 = (3 - 2.30)^2 = 0.49$.
- *Iteration 2*: Hold out $(2, 5)$. Train OLS on the 9 remaining points to obtain $\hat{y} \approx 0.10 + 2.15x$. Predict $\hat{y}(2) = 4.40$. Error: $e_2 = (5 - 4.40)^2 = 0.36$.

The procedure continues for all 10 points, producing errors $e_1, e_2, \ldots, e_{10}$. The final LOOCV estimate is

$$\text{CV}_{10} = \frac{1}{10} \sum_{i=1}^{10} e_i.$$

Note that each training set contains 9 of the 10 points, so each fitted model closely approximates the model trained on the full dataset. This minimizes bias but, as discussed, does not guarantee low variance.

Table 2: Comparison of Cross-Validation Methods

	Validation Set	**5/10-Fold CV**	**LOOCV**	**Stratified k-Fold**
Training Size	$0.8n$	$\frac{k-1}{k} \cdot n$	$n-1$	$\frac{k-1}{k} \cdot n$
Bias of Estimate	High	Moderate	Low	Moderate
Variance of Estimate	High	Moderate	High	Moderate
Comp. Cost	Low	Moderate	High	Moderate
Best Use Case	Very large datasets	General purpose	Very small datasets	Imbalanced classification

Definition 5.6 (Stratified k-Fold Cross-Validation)**.** Let $D = \{(x_i, y_i)\}_{i=1}^n$ be a dataset for a classification problem with C distinct classes, where class c has n_c instances, so that $\sum_{c=1}^C n_c = n$. Stratified k-fold cross-validation is a variant of k-fold CV in which the partition $D_1, D_2, \ldots, D_k$ is constructed such that each fold D_j preserves the class proportions of D. That is, for each fold $j = 1, \ldots, k$ and each class $c = 1, \ldots, C$:

$$\frac{|\{i : (x_i, y_i) \in D_j \text{ and } y_i = c\}|}{|D_j|} \approx \frac{n_c}{n}.$$

The remainder of the procedure—training on $D \setminus D_j$, evaluating on D_j, and averaging—is identical to standard k-fold CV.

Example 5.7. Suppose we have a binary classification dataset with $n = 100$ observations, of which 80 belong to class 0 and 20 belong to class 1 (an 80/20 split). We wish to perform 5-fold CV.

With *unstratified* random splitting, some folds could receive as few as 1 or even 0 instances of the minority class 1 by chance. For instance, a fold of 20 observations might contain 19 from class 0 and only 1 from class 1, making the validation estimate for class 1 unreliable.

With *stratified* 5-fold CV, each fold is constructed to contain exactly $80/5 = 16$ instances of class 0 and $20/5 = 4$ instances of class 1, preserving the 80/20 proportion in every fold. This ensures that each fold's training set also maintains the original class distribution ($64/16 = 80\%$ class 0 and $16/4 = 80\%$ class 1 in the training portion), yielding more stable and representative performance estimates—particularly for metrics like Recall and Precision that depend on the minority class.

Table 2 summarizes the key properties of the cross-validation methods discussed in this section. Each method represents a different point in the tradeoff between the bias and variance of the generalization error estimate, computational cost, and suitability to the problem at hand. In practice, 5- or 10-fold CV offers a reliable default for most applications, while stratified k-fold should be preferred whenever class imbalance is present and LOOCV reserved for settings where the sample size is too small to afford holding out an entire fold.

5.6 The Role of CV in Model Selection and Hyperparameter Tuning

The primary application of cross-validation is to guide *model selection*. Suppose we are considering several different models for a task (e.g., OLS vs. a 2nd-degree polynomial vs. a 3rd-degree polynomial). We can compute the k-fold CV score for each model. The model with the best (e.g., lowest MSE or highest F_1-score) cross-validated performance is selected as the one most likely to generalize well to new data.

Example 5.8 (Model Selection via Cross-Validation)**.** Returning to our running dataset with $n = 10$ observations, suppose we wish to choose among three polynomial regression models of increasing complexity:

- Model A: Linear, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ (degree 1, $p = 1$).
- Model B: Quadratic, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2$ (degree 2, $p = 2$).
- Model C: Cubic, $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \hat{\beta}_3 x^3$ (degree 3, $p = 3$).

We perform 5-fold CV with MSE as the evaluation metric on each model separately, using the same fold partition in all three cases to ensure a fair comparison. Suppose the results are:

	M_1	M_2	M_3	M_4	M_5	CV_5
Model A (linear)	2.06	0.40	0.71	1.68	8.35	2.64
Model B (quadratic)	0.52	0.18	0.33	0.41	1.06	0.50
Model C (cubic)	0.61	0.25	0.40	0.55	1.89	0.74

Model B achieves the lowest cross-validated MSE of 0.50, and is therefore selected. The linear model (A) underfits, producing the highest error. The cubic model (C) is more flexible than the quadratic but does not improve the CV score—its additional parameter fits noise in each training fold, slightly increasing variance without meaningfully reducing bias. This outcome illustrates the bias-variance tradeoff in action: the quadratic model strikes the best balance for this dataset.

Note that the training MSE tells a different story. On the full dataset, the cubic model would achieve the lowest training MSE (and a higher R^2), since it has more parameters. Cross-validation correctly identifies this as overfitting by evaluating on held-out data.

This same procedure is used for *hyperparameter tuning.* Many models have parameters that are not learned from the data but must be set beforehand (e.g., the degree p of a polynomial, or the regularization strength λ we will see later). We can use cross-validation to test a range of values for a hyperparameter and select the value that yields the best CV score.

Example 5.9 (Hyperparameter Tuning via Cross-Validation)**.** Consider a polynomial regression model $\hat{y} = \sum_{j=0}^{d} \hat{\beta}_j x^j$ where the polynomial degree d is a hyperparameter. We wish to select the optimal d from the candidate set $\{1, 2, 3, 4, 5\}$.

For each candidate degree d, we perform 5-fold CV and record the average MSE:

Degree d	CV_5 (MSE)
1	2.64
2	0.50
3	0.74
4	1.35
5	3.92

The CV error initially decreases as d increases from 1 to 2, reflecting a reduction in bias as the model gains the flexibility to capture the curvature in the data. Beyond $d = 2$, the CV error rises monotonically: each additional degree of freedom allows the model to fit noise in the training folds, increasing variance faster than it reduces bias. The optimal hyperparameter is $d^* = 2$.

After selecting $d^* = 2$, the final model is retrained on the *entire* dataset D using $d = 2$. This is a crucial and often overlooked step: cross-validation is used to *select* the hyperparameter, but the deployed model should leverage all available data.

6 Conclusion

This lecture has established a rigorous mathematical framework for answering the question that separates principled data science from mere curve-fitting: given a fitted model, how well will it perform on data it has never seen?

We began with the Bias-Variance Decomposition (Theorem 2.2), deriving from first principles the result

$$\mathbb{E}[(y_0 - \hat{f}(x_0))^2] = \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\hat{f}(x_0)) + \sigma^2.$$

This decomposition revealed that prediction error arises from three fundamentally distinct sources—the systematic error of the model's assumptions (bias), the instability of the model across different training samples (variance), and the irreducible noise inherent to the data-generating process (σ^2). Crucially, bias and variance are in opposition: increasing model complexity reduces one at the expense of the other. The art of model building lies in navigating this tradeoff to minimize the total error.

We then translated this theoretical understanding into a practical toolkit of evaluation metrics. For regression, we defined MSE and RMSE as direct measures of predictive error, and the coefficient of determination R^2 as a relative measure of explained variance—while carefully exposing its critical limitations, particularly its monotonic inflation with added predictors. The Adjusted R^2 was introduced as a penalized alternative that rewards parsimony. For classification, we demonstrated that accuracy is a dangerously misleading metric under class imbalance and constructed a richer framework from the confusion matrix: Precision quantifies the reliability of positive predictions, Recall quantifies the completeness of positive detection, and the F_β-Score unifies the two via the weighted harmonic mean. The ROC curve and its summary statistic, the AUC, provide a threshold-independent assessment of a classifier's discriminative ability.

Finally, we addressed the fundamental question of *how* to estimate these metrics reliably. We showed that evaluating on training data is systematically optimistic and that a single validation split suffers from high variance and data inefficiency. k-Fold Cross-Validation resolves both problems by rotating every observation through the roles of training and validation, producing a stable, nearly unbiased estimate of generalization error. We analyzed the bias-variance tradeoff of the CV estimator itself, examined LOOCV and stratified k-fold as important variants, and demonstrated how cross-validation serves as the primary tool for both model selection and hyperparameter tuning.

Together, these three pillars—a theoretical decomposition of error, a suite of metrics to measure it, and a robust resampling procedure to estimate it—provide the complete statistical machinery needed to evaluate the models constructed in previous lectures. They also set the stage for the coming lectures on numerical optimization and regularization, where the goal will be to actively control the bias-variance tradeoff by constraining model complexity during the fitting process itself.