Supervised Learning: Classification

Logistic Regression, Decision Trees, Metrics, and Imbalance

Lecture Companion Notes

Data Science with Python – BSc Course

Joerg Osterrieder
March 31, 2026

These notes accompany the classification lectures (L25–L28). They follow a problem-first structure: each section opens with a concrete challenge, builds intuition through visuals and analogies, then formalizes the concept with worked examples. Read before lecture for preparation, revisit after for deeper understanding.

Contents

1 When Numbers Become Decisions – From Regression to Classification 3

2 The S-Curve That Changed Everything – The Sigmoid and Logistic Regression 10

3 Reading the Tea Leaves – Interpreting Coefficients and Odds Ratios 21

4 Twenty Questions – Decision Trees and Splitting Criteria 32

5 The Forest for the Trees – Ensembles, Random Forests, and Overfitting 46
- 5.1 The Wisdom of Crowds . . . 46
- 5.2 Bagging and Random Forests . . . 46
- 5.3 How Many Trees Are Enough? . . . 51
- 5.4 Feature Importance: Gini vs. Permutation . . . 51
- 5.5 Finance Application: Backtesting a Forest . . . 52

6 Beyond the Grade – Confusion Matrix, Precision, Recall, and F1 57
- 6.1 The Grading Analogy . . . 57
- 6.2 The Four Outcomes . . . 57
- 6.3 Worked Example: Fraud Detection Metrics . . . 64

7 The Whole Picture – ROC Curves, AUC, and Threshold Tuning 69
- 7.1 The Speedometer Analogy . . . 69
- 7.2 TPR, FPR, and the ROC Curve . . . 70
- 7.3 Threshold Tuning: Where Theory Meets Business . . . 73
- 7.4 Finance Applications . . . 77

8 Finding Needles in Haystacks – Class Imbalance and Cost-Sensitive Learning 79
- 8.1 Needles, Haystacks, and Three Strategies . . . 79
- 8.2 Resampling Strategies . . . 79
- 8.3 Class Weights: Changing the Loss, Not the Data . . . 82
- 8.4 Stratified Cross-Validation . . . 83
- 8.5 Why PR Curves Matter More Under Imbalance . . . 84
- 8.6 Cost-Sensitive Fraud Detection . . . 86

Solutions to Practice Problems 89

(c) Joerg Osterrieder 2026

1 When Numbers Become Decisions – From Regression to Classification

Opening Problem: The Inconsistent Loan Officer

Maria works as a loan officer at a regional bank. Every day she reviews around 200 applications. For each one she must decide: approve or deny. She has guidelines—income thresholds, debt ratios, credit score cutoffs—but the guidelines leave grey areas. After a long morning she starts approving applications she would have denied on Monday. After a bad lunch she tightens up. Two applicants with nearly identical profiles get opposite outcomes depending on which day they land on her desk.

The bank notices. Bad loans cost money. Rejected good customers walk across the street to the competitor. Management asks: *Can we build a system that makes these decisions consistently, using the same data Maria uses, but without the Monday-vs-Friday drift?*

This is a classification problem. Not "predict a number" (regression), but "pick a category" (approve or deny). The rest of this section explains why regression cannot do this job and what we need instead.

Discovery Question

Suppose you code the target variable as $y = 0$ (deny) and $y = 1$ (approve), then fit a standard linear regression. You get predictions like $\hat{y} = 0.73$ or $\hat{y} = -0.15$ or $\hat{y} = 1.42$. Questions to think about:

(a) What does a prediction of -0.15 mean? Is this person "negative approved"?

(b) What does 1.42 mean? More than fully approved?

(c) If you round to the nearest integer, you get 0 or 1. But is 0.51 really different from 0.49? The model treats them as opposite decisions.

Linear regression was built for continuous targets. Forcing it onto a binary target creates nonsensical outputs. We need a model whose output *lives* in $[0, 1]$ and has a natural interpretation as a probability.

The Thermometer vs. the Traffic Light

Think of a thermometer and a traffic light. A thermometer gives you a *number*: 22.7°C. A traffic light gives you a *category*: red, yellow, or green. Regression is the thermometer. Classification is the traffic light.

The crucial difference is not just the output type. With a thermometer, 22.7°C is "between" 22°C and 23°C in a meaningful sense—you can interpolate. But "yellow" is not "between" red and green in any arithmetic sense. Categories have no natural order (unless they do, which we call ordinal—but even then, the gaps between categories are undefined).

Figure 1 shows the core problem. A straight line fitted to binary data produces predictions outside $[0, 1]$. The logistic curve, by contrast, stays bounded.

Real data rarely separates into clean clusters. Figure 2 shows what happens when classes overlap—some defaulting customers look identical to non-defaulting ones. Classification does not require perfect separation; it learns the *best available* boundary.

Formal Definitions

Classification: Assigning an observation to one of a finite set of categories based on its features. The model learns a mapping $f : \mathbb{R}^p \to \{1, 2, \ldots, K\}$ from training data.

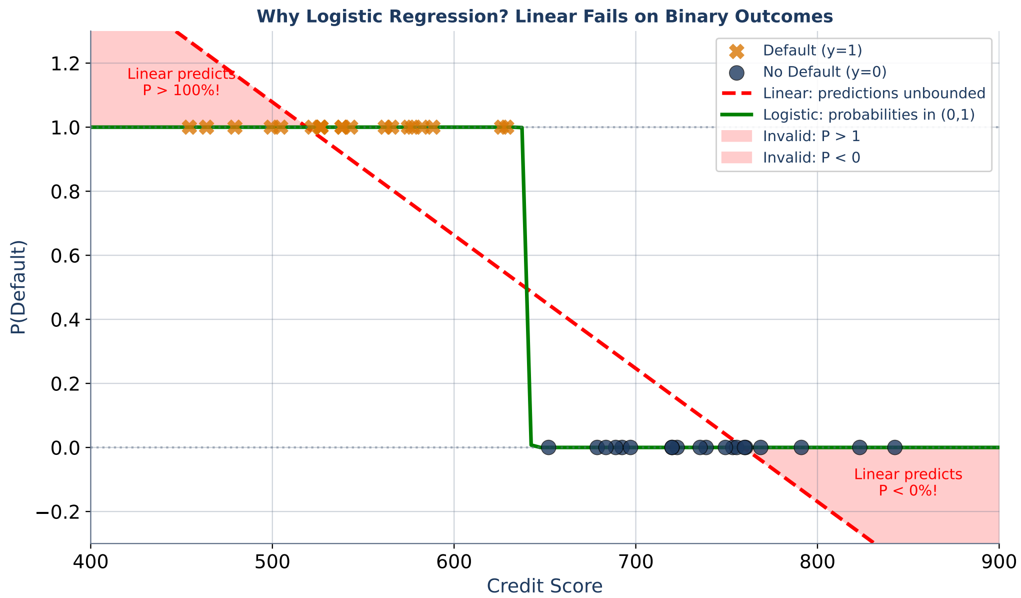

Figure 1: Linear regression (left) vs. logistic regression (right) on binary data. The linear model produces impossible values; the logistic model stays in $(0, 1)$.

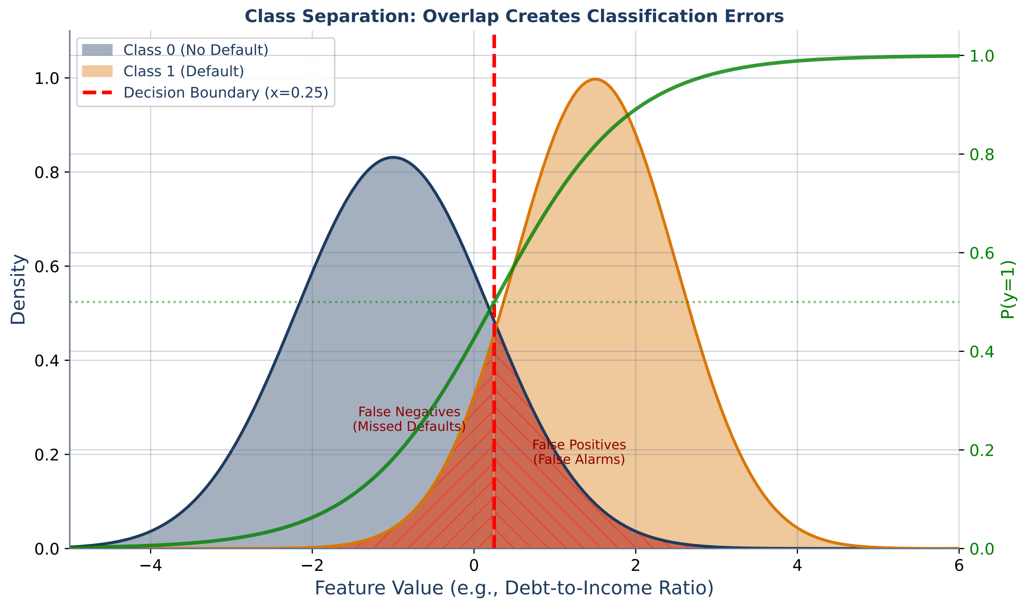

Figure 2: Overlapping classes in feature space. No boundary perfectly separates the two groups. Classification finds the boundary that makes the fewest mistakes.

Binary classification: Classification with exactly two classes (e.g., default/no-default, spam/not-spam, approve/deny). The target is $y \in \{0, 1\}$.

Features (predictors): The input variables $x_1, x_2, \ldots, x_p$ used to make the classification. In the loan example: income, debt-to-income ratio, credit history length.

Decision boundary: The surface in feature space where the model switches its prediction from one class to the other. For linear models, this boundary is a hyperplane.

The standard classification pipeline has three stages: (1) compute a raw score from features, (2) convert the score to a probability, (3) apply a threshold to produce a class label. Figure 3 shows this flow.

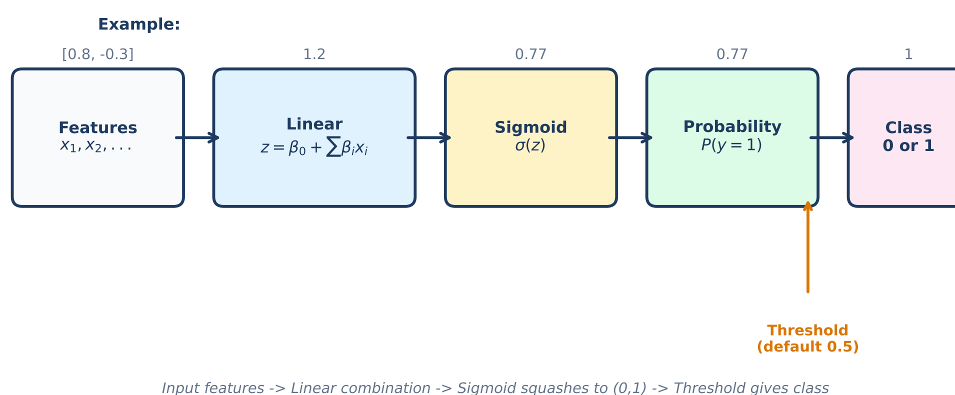

Figure 3: The classification pipeline: features enter a scoring function, get transformed into probabilities, and pass through a decision threshold.

Classification vs. Regression: A Precise Comparison

Property	Regression	Classification
Target type	Continuous ($y \in \mathbb{R}$)	Categorical ($y \in \{0, 1, \ldots, K\}$)
Output	A number (predicted value)	A label (and optionally a probability)
Loss function	MSE, MAE	Log loss, cross-entropy
Example	Predict stock price	Predict default (yes/no)
Evaluation	R^2, RMSE	Accuracy, precision, recall, AUC

Historical Background: Fisher's Irises (1936)

The first systematic use of mathematical classification came from Ronald A. Fisher in 1936. He measured four features of iris flowers—sepal length, sepal width, petal length, petal width—and asked: can we separate three species (*setosa*, *versicolor*, *virginica*) using these measurements?

Fisher invented *linear discriminant analysis* (LDA) to answer this. He found the linear combination of features that maximized the ratio of between-class variance to within-class variance. He had no computer. He did this with a pencil, a ruler, and arithmetic tables. What Fisher did by hand in 1936, you do with `sklearn.LogisticRegression()` in one line of code. The statistical thinking has not changed. The compute budget has.

Common Misconception: "Logistic regression is regression"

Despite the name, logistic regression is a classifier. The word "regression" refers to the fact that the model uses a linear combination of inputs (just like linear regression), but the output passes through a sigmoid function and the target is categorical. If someone asks "Is logistic regression a regression or a classification method?"—the answer is classification.

Common Misconception: "Classification only works with two classes"

Binary classification is the simplest case, but multiclass classification is standard. Techniques like one-vs-rest (OvR) and the softmax function extend binary classifiers to $K > 2$ classes. We cover this in Section 3 (Section 3).

Common Misconception: "You need balanced classes"

Many real-world problems are imbalanced: 1% fraud, 5% default, 0.01% insider trading. Classification still works, but you need to handle the imbalance with appropriate techniques. That is Section 8.

Worked Example: Credit Scoring in Three Steps

A bank builds a credit scoring model with three features:

- x_1: annual income ($10k units)
- x_2: debt-to-income ratio
- x_3: credit history length (years)

Step 1: Compute a score. The model learns weights: $\beta_0 = -3.0$, $\beta_1 = 0.4$, $\beta_2 = -2.5$, $\beta_3 = 0.15$. For a customer with income = 6 ($60k), debt ratio = 0.3, history = 10 years:

$$z = -3.0 + 0.4(6) + (-2.5)(0.3) + 0.15(10) = -3.0 + 2.4 - 0.75 + 1.5 = 0.15$$

Step 2: Convert to probability. Apply the sigmoid: $\sigma(0.15) = 1/(1 + e^{-0.15}) \approx 0.537$. The model estimates a 53.7% probability of repayment.
Step 3: Decide. With a default threshold of 0.5, this customer is approved (barely). A more conservative bank might set the threshold at 0.7, and this customer would be denied.

Problem 1.1 (Easy)

For each of the following, state whether it is a classification or regression problem:

(a) Predicting tomorrow's stock price

(b) Detecting spam emails

(c) Estimating a house's market value

(d) Flagging fraudulent credit card transactions

(e) Forecasting a customer's lifetime spending

Solution: see Appendix.

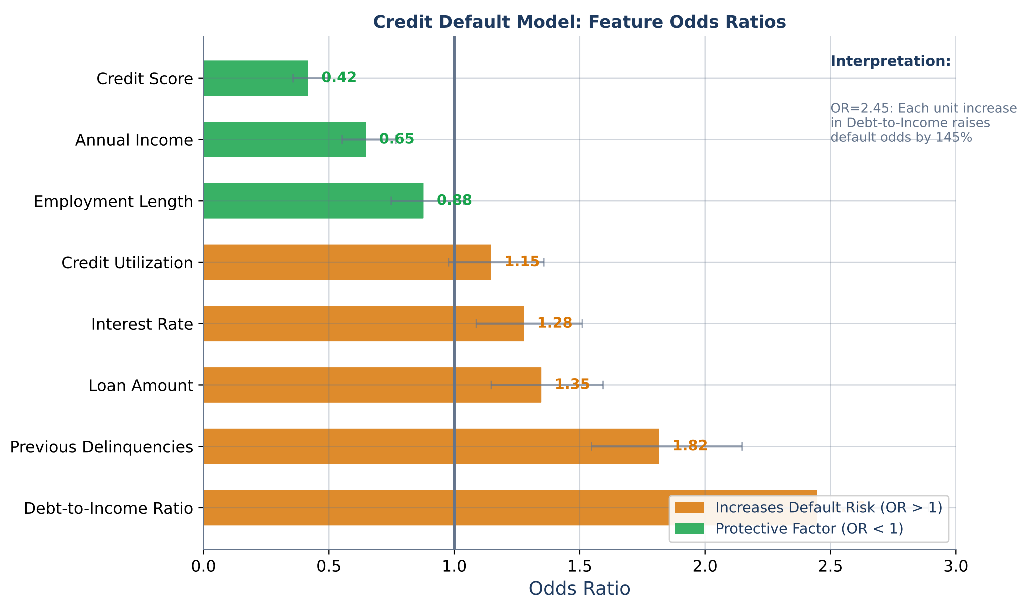

Figure 4: Credit default prediction: features like income and debt ratio feed into a logistic regression that outputs default probability.

Problem 1.2 (Easy)

Sketch what happens when you fit a straight line ($y = \beta_0 + \beta_1 x$) to binary data where $y \in \{0, 1\}$. Draw a scatter plot with about 10 points, fit a line through them by eye, and mark where the line predicts values below 0 or above 1. Why are these predictions problematic?

Solution: see Appendix.

Problem 1.3 (Medium)

A dataset has columns: `annual_income`, `employment_years`, `num_credit_cards`, `loan_amount`, and a target column `defaulted` (0/1).

(a) Identify the features and the target variable.

(b) What type of classification is this (binary or multiclass)?

(c) Which feature do you expect to have the strongest association with default? Why?

Solution: see Appendix.

Problem 1.4 (Medium)

Compare these two scikit-learn calls:

```
1 # Regression
2 LinearRegression().fit(X, y).predict(X_new)
3
4 # Classification
5 LogisticRegression().fit(X, y).predict_proba(X_new)
```

(a) What does `predict()` return for the regression model?

(b) What does `predict_proba()` return for the classification model?

(c) Why does the classifier output probabilities instead of just class labels?

Solution: see Appendix.

Problem 1.5 (Hard)

Design a classification system for detecting insider trading. Consider:

(a) What features would you engineer? (Think: trading volume, timing relative to announcements, network of relationships.)

(b) What makes this harder than credit default prediction?

(c) What is the likely class balance, and why does that matter?

Solution: see Appendix.

Connecting Forward

We now know *what* classification is and *why* linear regression fails at it. The missing piece is the mathematical bridge—a function that takes any real-valued score and maps it to a probability. That function is the sigmoid, and it is the star of Section 2.

Key Takeaway: Classification turns features into decisions by learning the boundary between categories from data.

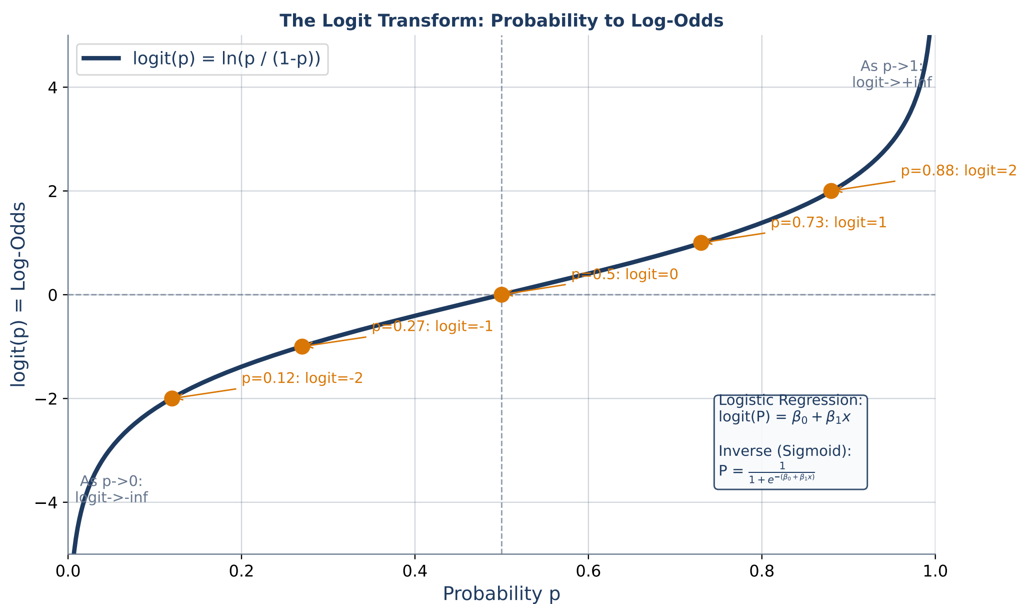

Figure 5: The logit transform: the mathematical bridge from linear scores to probabilities.

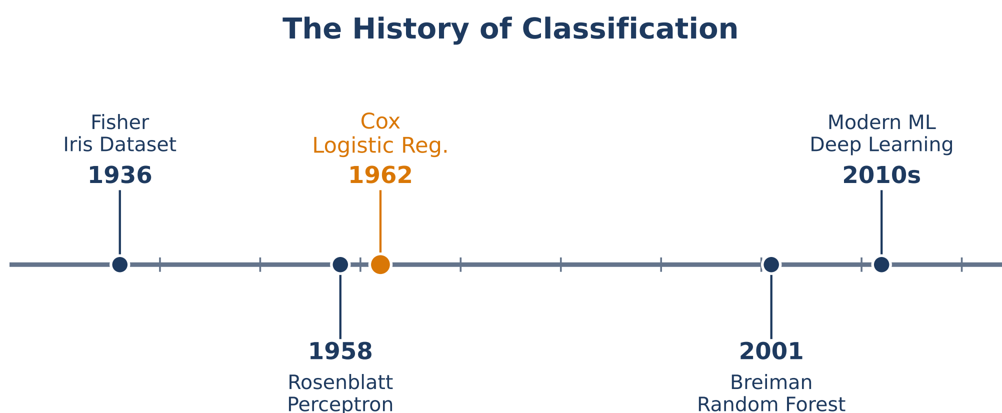

Figure 6: Timeline of classification methods—from Fisher's LDA (1936) to modern ensemble methods.

2 The S-Curve That Changed Everything – The Sigmoid and Logistic Regression

Opening Problem: The Bank Wants Probabilities, Not Just Labels

The bank from Section 1 does not just want "approve" or "deny." Their pricing department needs granularity:

- A customer with 95% repayment probability gets a premium low-interest rate.
- A customer at 60% gets approved but at a higher rate to compensate for risk.
- A customer at 30% gets denied.

Linear regression gives unbounded scores: $z = -2.7$ or $z = 4.3$. These are not probabilities. You cannot charge an interest rate based on a score of -2.7—what does that even mean? We need a function that takes any real number z and squeezes it into the interval $(0, 1)$, smoothly, with a natural interpretation as probability. And ideally, the function should be simple enough to differentiate and optimize.

Discovery Question

What mathematical function maps any real number to the interval $(0, 1)$, is smooth and differentiable everywhere, is symmetric around its midpoint, and has the property that its derivative can be expressed in terms of itself?
A first attempt: just clip the linear output at 0 and 1. But this creates a flat region where the gradient is zero—the model cannot learn there.
A second attempt: the cumulative distribution function (CDF) of the standard normal (this is the "probit" model). It works, but the CDF involves an integral with no closed form.
A third attempt: $\sigma(z) = 1/(1 + e^{-z})$. Simple. Closed form. Derivative is $\sigma(z)(1 - \sigma(z))$. This is the sigmoid, and it won.

The Dimmer Switch

A light switch is binary: on or off. A dimmer switch is continuous: you can set it anywhere from fully off to fully on, with a smooth transition in between. The sigmoid is the mathematical dimmer switch. Near $z = -5$, it is "almost off" (close to 0). Near $z = 5$, it is "almost on" (close to 1). At $z = 0$, it is exactly at 50%.

Figure 7 shows the shape. Notice the S-curve: steep in the middle, flat at the extremes. This means the model is most sensitive to changes near the decision boundary and least sensitive to changes far from it—exactly the behavior you want.

The Sigmoid Function

Sigmoid function: The function $\sigma(z) = \frac{1}{1+e^{-z}}$, mapping any real value z to the open interval $(0, 1)$. It is the canonical link function for binary classification.

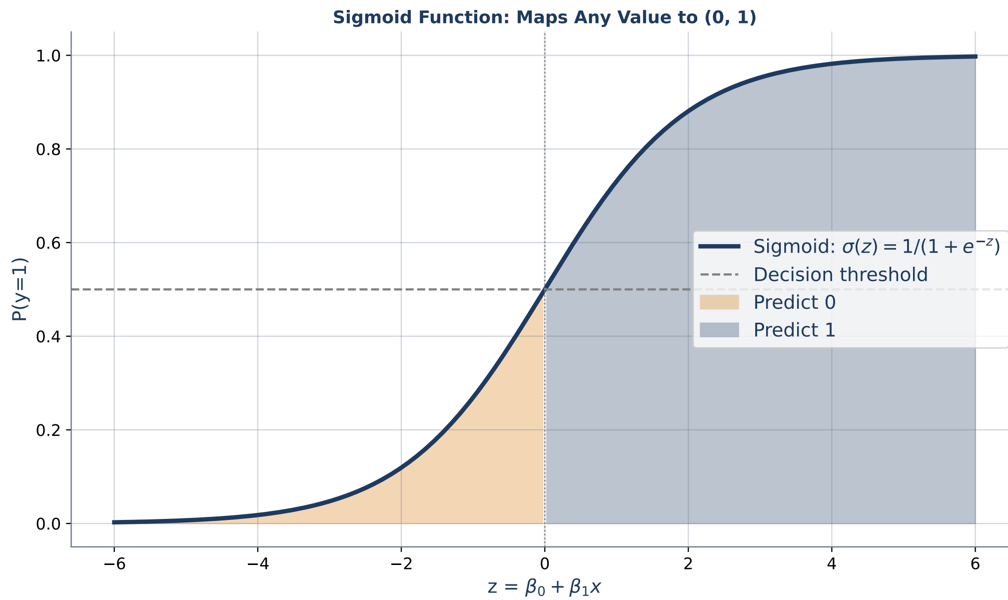

Figure 7: The sigmoid function $\sigma(z) = 1/(1 + e^{-z})$. The S-curve maps any real value to $(0, 1)$.

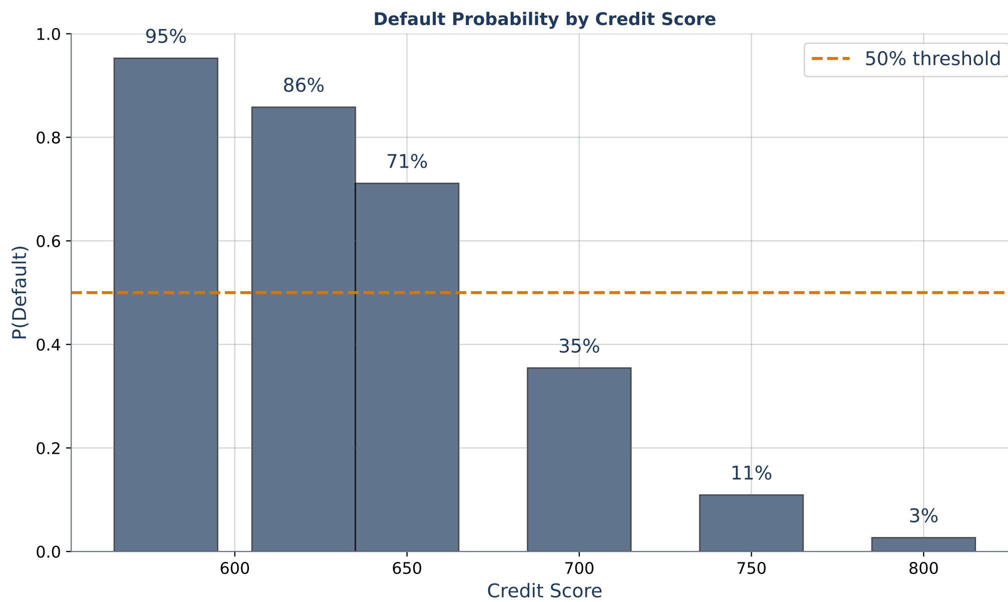

Figure 8: The sigmoid output interpreted as probability. Each input z maps to a probability between 0 and 1.

Key Formula: The Sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

where:

- $z = \beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_p x_p$ is the linear combination of features (the "log-odds")
- $\sigma(z) \in (0,1)$ is interpreted as $P(y=1 \mid \mathbf{x})$
- At $z=0$: $\sigma(0)=0.5$ (maximum uncertainty)
- As $z \to +\infty$: $\sigma(z) \to 1$
- As $z \to -\infty$: $\sigma(z) \to 0$

Key property: $\sigma'(z) = \sigma(z)\big(1-\sigma(z)\big)$. The derivative depends only on the function value itself. This makes gradient computation fast.

Why Not MSE? The Log Loss

In linear regression we minimize the mean squared error. For classification, MSE is a poor choice: it treats a confident wrong prediction (say, predicting 0.99 when the true label is 0) as only slightly worse than a mildly wrong one (predicting 0.6). We need a loss function that *punishes confident mistakes severely.*

Log loss (binary cross-entropy): The loss function for logistic regression: $\mathcal{L} = -\big[y\log p+(1-y)\log(1-p)\big]$, where $p=\sigma(z)$ is the predicted probability and $y \in \{0,1\}$ is the true label.

Key Formula: Log Loss (Binary Cross-Entropy)

For a single sample:

$$\mathcal{L}(y,p) = -\big[y\log p + (1-y)\log(1-p)\big]$$

where:

- $y \in \{0,1\}$ is the true label
- $p = \sigma(\mathbf{x}^\top\boldsymbol{\beta}) \in (0,1)$ is the predicted probability
- If $y=1$ and $p\approx 1$: loss ≈ 0 (correct and confident—good)
- If $y=1$ and $p\approx 0$: loss $\to +\infty$ (confident and wrong—catastrophic)

For the full dataset of n samples:

$$J(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}\big[y_i\log p_i + (1-y_i)\log(1-p_i)\big]$$

This is minimized using gradient descent (or a quasi-Newton method like L-BFGS).

Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE): Finding the parameters $\boldsymbol{\beta}$ that maximize the probability (likelihood) of observing the training data. Minimizing log loss is equivalent to maximizing the log-likelihood.

The connection is direct. If we treat each sample as an independent Bernoulli trial with success

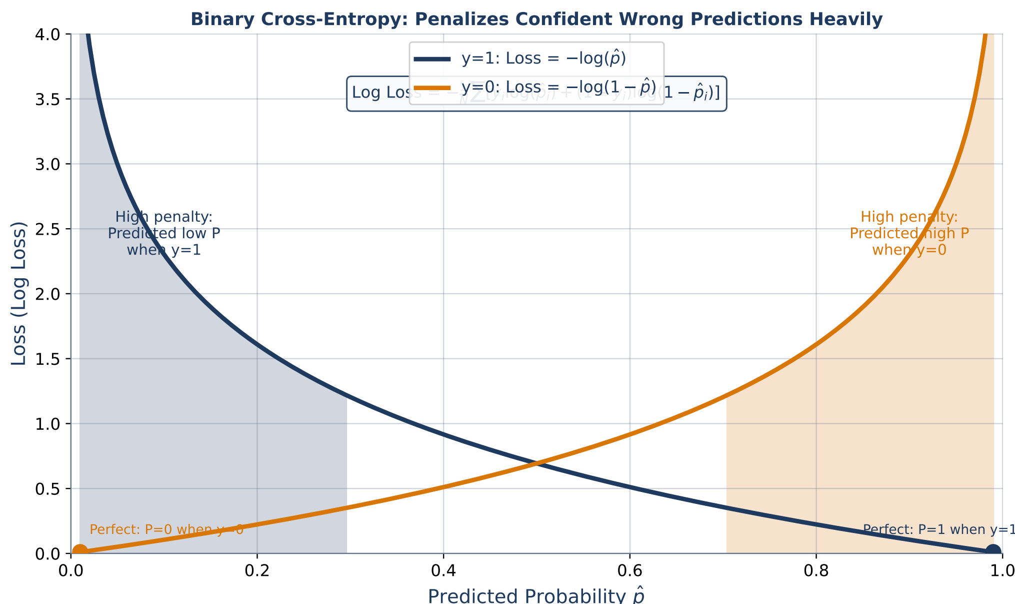

Figure 9: Log loss as a function of predicted probability. Confident wrong predictions incur enormous loss.

probability $p_i = \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})$, then the likelihood of the dataset is:

$$\mathcal{L}(\boldsymbol{\beta}) = \prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i}$$

Taking the log and negating gives exactly the log loss. So minimizing log loss *is* maximum likelihood.

Historical Background: Berkson and the Logit (1944)

Joseph Berkson, a statistician at the Mayo Clinic, coined the term “logit” in 1944 as the log-odds transformation: $\text{logit}(p) = \log \frac{p}{1-p}$. He proposed the logistic model as an alternative to the *probit* model (based on the normal CDF) for analyzing dose-response relationships in medicine.

Berkson and Fisher debated fiercely over estimation: Fisher championed maximum likelihood, Berkson preferred minimum chi-square. Fisher won the argument—MLE became the standard.

The word “logistic” for the S-curve predates Berkson by over a century. Pierre-François Verhulst introduced the *logistic growth curve* in 1838 to model population dynamics under resource constraints. The mathematical function is the same; the application is entirely different. Verhulst studied rabbits. We study loan defaults.

Common Misconception: “The sigmoid output IS the prediction”

The sigmoid gives you a probability, not a class label. To get a label you must choose a threshold (commonly 0.5) and classify: if $p \geq 0.5$, predict class 1; otherwise predict class 0. The threshold is a separate decision. Section 7 covers how to tune it.

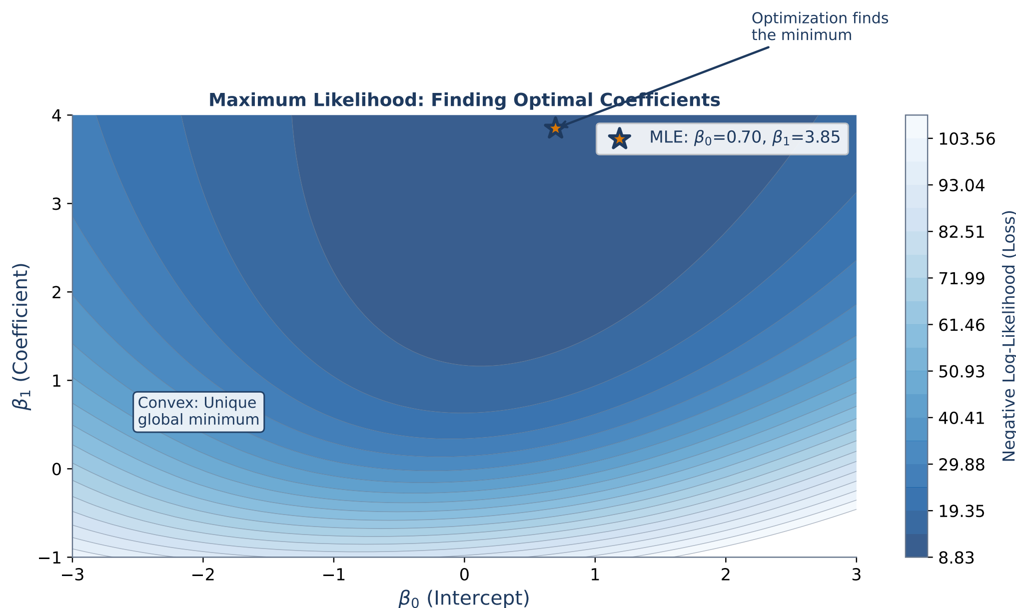

Figure 10: The log-likelihood surface over parameter space. MLE finds the peak—the parameters that make the observed data most probable.

Common Misconception: "Higher sigmoid output means more confidence"

Not necessarily. A poorly calibrated model might output 0.92 for almost every positive case, regardless of the actual evidence. Calibration—whether a predicted 80% actually corresponds to 80% of true positives—is a separate property from accuracy.

Common Misconception: "Logistic regression minimizes MSE"

No. It minimizes log loss (cross-entropy), which is derived from maximum likelihood estimation. MSE applied to classification produces non-convex optimization with bad local minima. Log loss is convex, which guarantees a unique global minimum.

Common Misconception: "The decision boundary is always at $P = 0.5$"

Only by default. In medical diagnosis you might set the threshold at 0.3 to catch more true positives (at the cost of more false positives). In spam filtering you might set it at 0.8 to avoid blocking legitimate emails. The threshold is a business decision, not a mathematical requirement.

Decision Boundaries: From 1D to 2D

In one dimension, the decision boundary is a single point on the number line—the value of x where $\sigma(z) = 0.5$, i.e., where $z = 0$.

In two dimensions, the decision boundary is a line. On one side, $P(y = 1) > 0.5$; on the other side, $P(y = 1) < 0.5$. The line's equation comes from setting $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$.

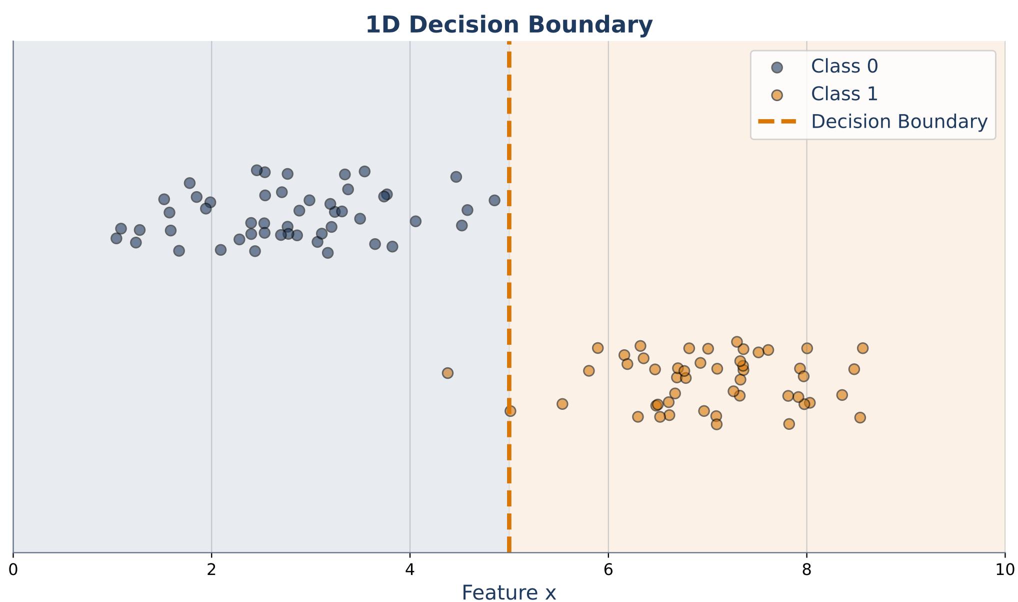

Figure 11: One-dimensional decision boundary. The sigmoid crosses 0.5 at the threshold point.

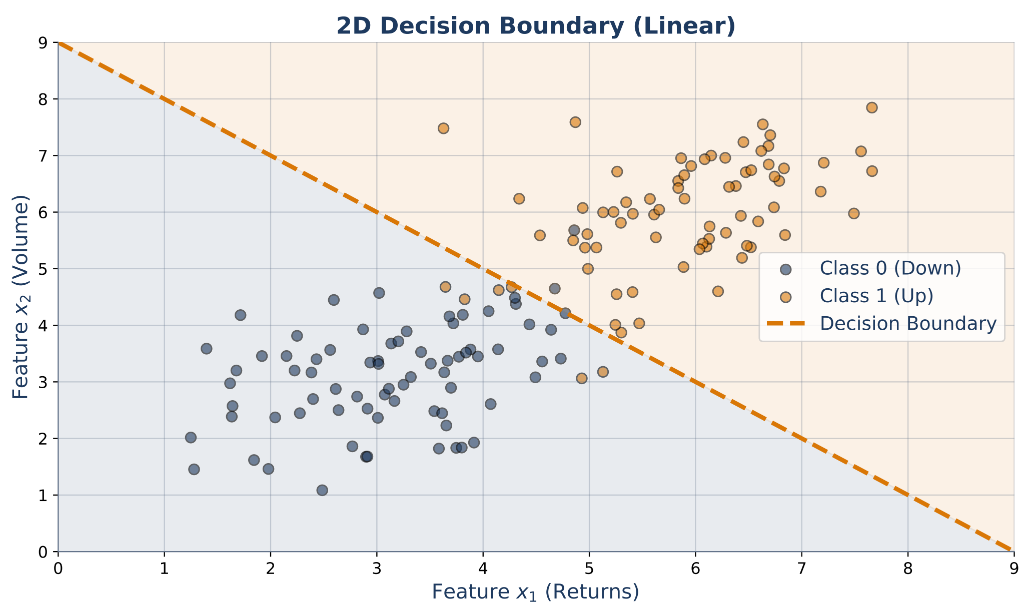

Figure 12: Two-dimensional decision boundary. The line separates the two classes; the color gradient shows how probability changes with distance from the boundary.

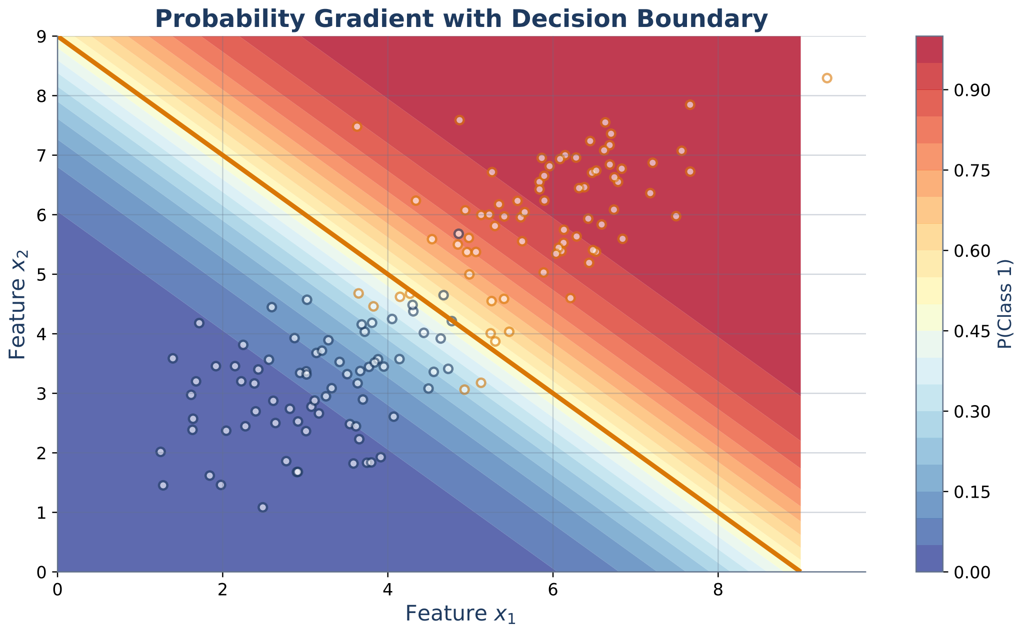

Figure 13: The probability gradient across feature space. Dark regions correspond to high confidence in one class, light regions to the other.

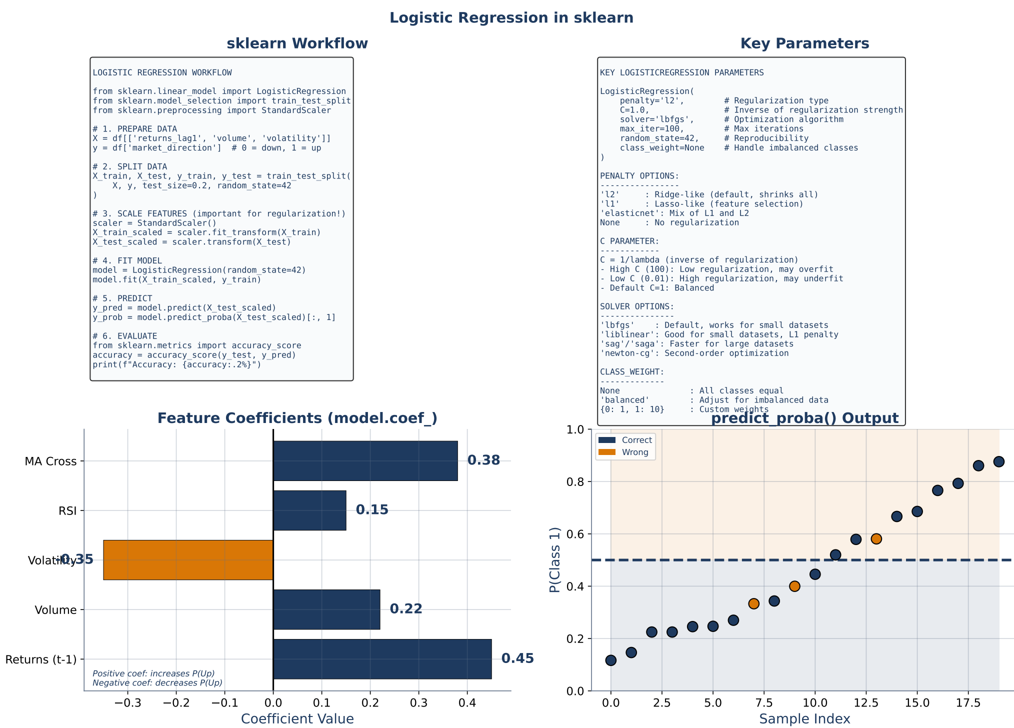

Figure 14: Logistic regression fitted with scikit-learn: the decision boundary and class regions.

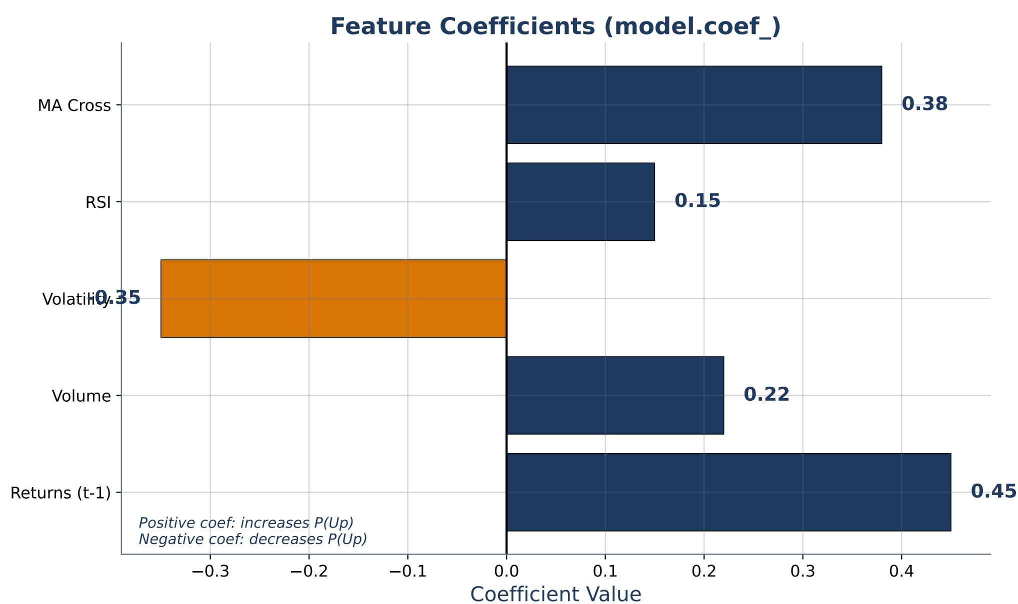

Figure 15: Learned feature coefficients. Positive coefficients push toward class 1; negative coefficients push toward class 0.

Scikit-Learn in Action

Worked Example: Computing the Sigmoid by Hand

Compute $\sigma(z)$ for several values of z:

z	e^{-z}	$1+e^{-z}$	$\sigma(z)=1/(1+e^{-z})$
-3	20.09	21.09	0.047
-1	2.718	3.718	0.269
0	1.000	2.000	0.500
1	0.368	1.368	0.731
2	0.135	1.135	0.881
5	0.0067	1.0067	0.993

Observations: $\sigma(0) = 0.5$ always. $\sigma(-z) = 1-\sigma(z)$ (symmetry). For $|z| > 5$, the output is practically 0 or 1. The function is steepest around $z = 0$.

Python Code: Logistic Regression with scikit-learn

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import numpy as np

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Fit logistic regression
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Get probabilities (not just labels)
probabilities = model.predict_proba(X_test)[:, 1]  # P(y=1)
labels = model.predict(X_test)                       # 0 or 1

# Inspect learned parameters
print(f"Intercept: {model.intercept_[0]:.3f}")
print(f"Coefficients: {model.coef_[0]}")
```

Problem 2.1 (Easy)

Compute $\sigma(z)$ for $z = -2, 0, 2, 5$. Show the full arithmetic: write out e^{-z}, then $1 + e^{-z}$, then the final value. Verify that $\sigma(-2) + \sigma(2) = 1$.
Solution: see Appendix.

Problem 2.2 (Medium)

A credit scoring model has coefficients: $\beta_0 = -1.5$, $\beta_{\text{income}} = 0.3$, $\beta_{\text{debt}} = -2.1$.
A customer has income = 5 (in $10k units) and debt ratio = 0.4.

(a) Compute $z = \beta_0 + \beta_{\text{income}} \cdot 5 + \beta_{\text{debt}} \cdot 0.4$.

(b) Compute $P(\text{default}) = \sigma(z)$.

(c) Would this customer be approved at a threshold of 0.5?

Solution: see Appendix.

Problem 2.3 (Medium)

Look at Figure 12. The decision boundary is a straight line.

(a) Explain why the boundary is a straight line (hint: what equation defines it?).

(b) What determines the slope of this line?

(c) What determines where the line crosses the axes?

Solution: see Appendix.

Problem 2.4 (Medium)

Compute the log loss for two scenarios:

(a) True label $y = 1$, predicted probability $p = 0.9$

(b) True label $y = 1$, predicted probability $p = 0.01$

Compute $\mathcal{L} = -[y \log p + (1-y)\log(1-p)]$ for each case. How many times larger is the loss in case (b)?
Solution: see Appendix.

Problem 2.5 (Hard)

Derive the gradient of the log loss with respect to z for a single sample. Start from:

$$\mathcal{L} = -\left[y \log \sigma(z) + (1-y)\log(1-\sigma(z))\right]$$

Use the fact that $\sigma'(z) = \sigma(z)(1-\sigma(z))$.
Show that $\frac{\partial \mathcal{L}}{\partial z} = \sigma(z) - y$.
Why is this result remarkably simple? What does it mean: the gradient is just the error (predicted minus actual)?
Solution: see Appendix.

Connecting Forward

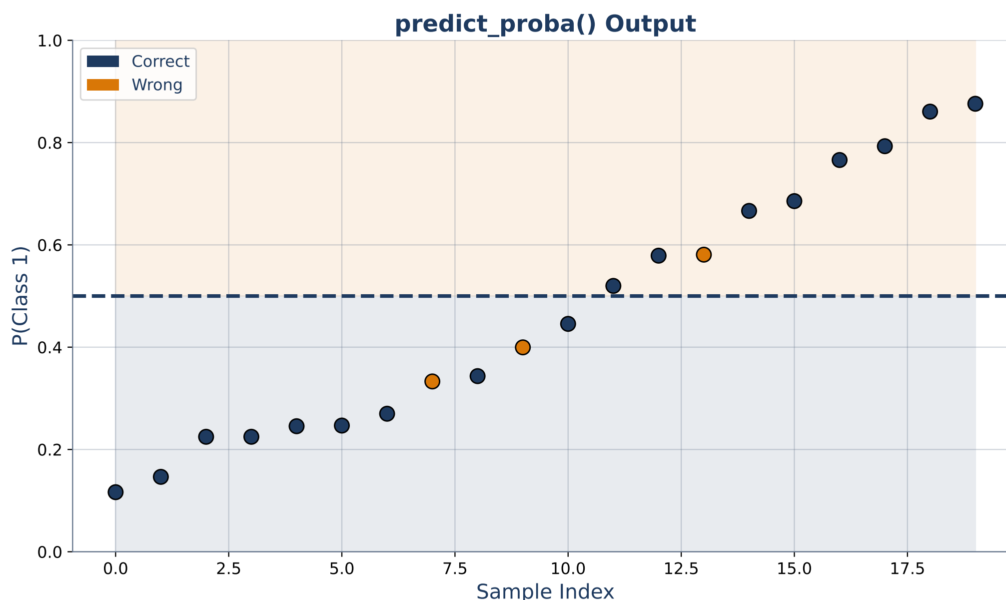

Figure 16: The `predict_proba` output: each sample gets a probability, which we can use for nuanced decisions beyond simple approve/deny.

We now have a model that outputs probabilities. We know the sigmoid, the loss function, and how to fit the model. But something is missing. If the coefficient on income is 0.7—does that mean income increases the probability of repayment by 0.7? By 70 percentage points? By some other amount?

The answer is none of the above. Coefficients in logistic regression live in log-odds space, not probability space. To interpret them, you need the odds ratio. That is Section 3.

Key Takeaway: The sigmoid function is the bridge between a linear score and a probability—it enables principled classification.

3 Reading the Tea Leaves – Interpreting Coefficients and Odds Ratios

Opening Problem: The Regulator Wants an Explanation

A banking regulator reviews your credit scoring model and asks: "Your model denied this customer's loan application. Why?"
The data scientist answers: "The model predicted a 23% probability of repayment."
The regulator presses: "But *why*? Which factors drove the denial? By how much? If the customer paid off one credit card, would the decision change?"
This is not a hypothetical. The EU's AI Act and the US Equal Credit Opportunity Act require that lenders explain automated decisions. A black-box probability is not enough. You need to point to specific features and quantify their contribution.
Logistic regression can answer this—if you know how to read its coefficients. The coefficients are not in probability space. They are in *log-odds space*. Converting them to a human-readable form requires one more concept: the odds ratio.

Discovery Question

If the coefficient on income is $\beta = 0.7$, does that mean:

(a) A one-unit increase in income raises the probability of approval by 0.7?

(b) A one-unit increase in income raises the probability by 70 percentage points?

(c) Something else entirely?

Try this: if $P(\text{approval}) = 0.5$, and we increase income by one unit, the new log-odds is $\text{logit}(0.5) + 0.7 = 0 + 0.7 = 0.7$. Then $P = \sigma(0.7) = 0.668$. The probability went up by 0.168, not 0.7.
Now try starting at $P = 0.9$. The log-odds is $\text{logit}(0.9) = 2.197$. Adding 0.7 gives 2.897, so $P = \sigma(2.897) = 0.948$. The probability went up by only 0.048.
Same coefficient. Different starting probability. Different change in probability. The coefficient is constant in *log-odds space*, not in probability space. That is why we need odds ratios.

Odds Ratios as Betting Multipliers

Think about gambling odds. If a horse has 3:1 odds of winning, that means for every 1 time it loses, it wins 3 times. The odds are $3/1 = 3.0$.

Now suppose you learn that the horse has a new jockey. The odds ratio for this jockey is 2.0. That means the odds *double*: from 3:1 to 6:1. The horse is now twice as likely (in odds terms) to win.

This is exactly how logistic regression coefficients work. Each coefficient β_j corresponds to an odds ratio of e^{β_j}. A one-unit increase in feature x_j *multiplies* the odds by e^{β_j}, regardless of the starting probability.

From Coefficients to Odds Ratios

Odds: The ratio $p/(1-p)$, where p is the probability of the event. If $p = 0.75$, the odds are $0.75/0.25 = 3$, meaning the event is three times more likely to happen than not. Odds range from 0 to $+\infty$.

Log-odds (logit): The natural logarithm of the odds: $\text{logit}(p) = \log \frac{p}{1-p}$. This is the quantity that logistic regression models as a linear function of features: $\text{logit}(p) = \mathbf{x}^\top \boldsymbol{\beta}$.

Odds Ratios: Interpreting Logistic Regression Coefficients

Odds Ratio Concept

```
ODDS AND ODDS RATIOS

PROBABILITY vs ODDS:
--------------------
Probability P = 0.75 (75% chance)
Odds = P / (1-P) = 0.75 / 0.25 = 3

"3 to 1 odds" means 3 times more likely
to happen than not happen.

ODDS RATIO (OR):
----------------
Compares odds between two groups

OR = (Odds with feature) / (Odds without feature)

OR = 2.0 means:
"The odds of success DOUBLE when the
 feature increases by 1 unit"

FROM LOGISTIC COEFFICIENTS:
---------------------------
log(odds) = beta_0 + beta_1*x_1 + ...

Therefore:
    Odds Ratio = exp(beta)

Example: beta = 0.693
    OR = exp(0.693) = 2.0
```

Odds Ratios for Market Direction

Earnings Beat 3.10x
High Volatility 0.65x
Above MA 1.45x
Positive Sentiment 2.30x
High Volume 1.85x
OR = 1 (no effect)
0.0 0.5 1.0 1.5 2.0 2.5 3.0
Odds Ratio

Coefficient to Odds Ratio: OR = exp(beta)

Odds Ratio
beta=-1 OR=0.37
beta=-0.5 OR=0.61
beta=0 OR=1.00
beta=0.5 OR=1.65
beta=1 OR=2.72
beta=1.5 OR=4.48
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0
Coefficient (beta)

Python Implementation

```
Computing Odds Ratios in Python

import numpy as np
from sklearn.linear_model import LogisticRegression
# Fit model
model = LogisticRegression()
model.fit(X_train, y_train)

# Get coefficients
coefficients = model.coef_[0]
feature_names = X_train.columns

# Calculate odds ratios
odds_ratios = np.exp(coefficients)

# Display results
print("Feature              Coef     OR      Interpretation")
print("-" * 60)
for name, coef, or_ in zip(feature_names, coefficients, odds_ratios):
    if or_ > 1:
        interp = f"+{(or_-1)*100:.0f}% odds"
    else:
        interp = f"-{(1-or_)*100:.0f}% odds"
    print(f"{name:20} {coef:+.3f}  {or_:.2f}x   {interp}")

# Example output:
# Feature              Coef     OR      Interpretation
# ------------------------------------------------------------
# volume               +0.616  1.85x   +85% odds
# sentiment            +0.833  2.30x   +130% odds
# above_ma             +0.372  1.45x   +45% odds
# volatility           -0.431  0.65x   -35% odds
# earnings_beat        +1.131  3.10x   +210% odds
```

Figure 17: The odds ratio concept: coefficients in logistic regression are multiplicative factors on the odds.

Odds ratio (OR): The factor e^{β_j} by which the odds change for a one-unit increase in feature x_j. An OR > 1 means the odds increase; an OR < 1 means the odds decrease; an OR $= 1$ means the feature has no effect.

Key Formula: The Log-Odds Relationship

The logistic regression model says:

$$\log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

Increasing x_j by one unit adds β_j to the log-odds. Exponentiating both sides:

$$\frac{p_{\text{new}}}{1-p_{\text{new}}} = e^{\beta_j} \cdot \frac{p_{\text{old}}}{1-p_{\text{old}}}$$

So the odds ratio for feature j is $\text{OR}_j = e^{\beta_j}$.
Interpretation guide:

- $\beta_j > 0 \implies \text{OR}_j > 1$: feature increases the odds of class 1
- $\beta_j < 0 \implies \text{OR}_j < 1$: feature decreases the odds of class 1
- $\beta_j = 0 \implies \text{OR}_j = 1$: feature has no effect
- $|\beta_j|$ large $\implies$ strong effect (but check the feature's scale!)

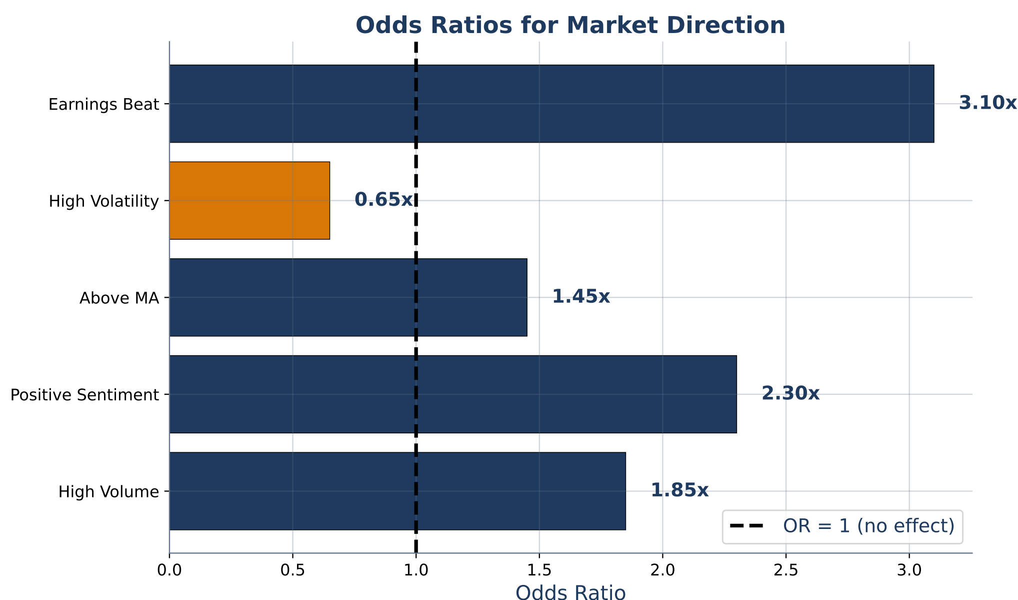

Figure 18: Odds ratios for a credit scoring model. Values above 1 increase default risk; values below 1 decrease it.

Regularization: Keeping Coefficients Honest

When features are correlated or when the model has many features relative to the number of samples, coefficients can blow up to extreme values. Regularization adds a penalty to the loss function that keeps coefficients small.

(c) Joerg Osterrieder 2026

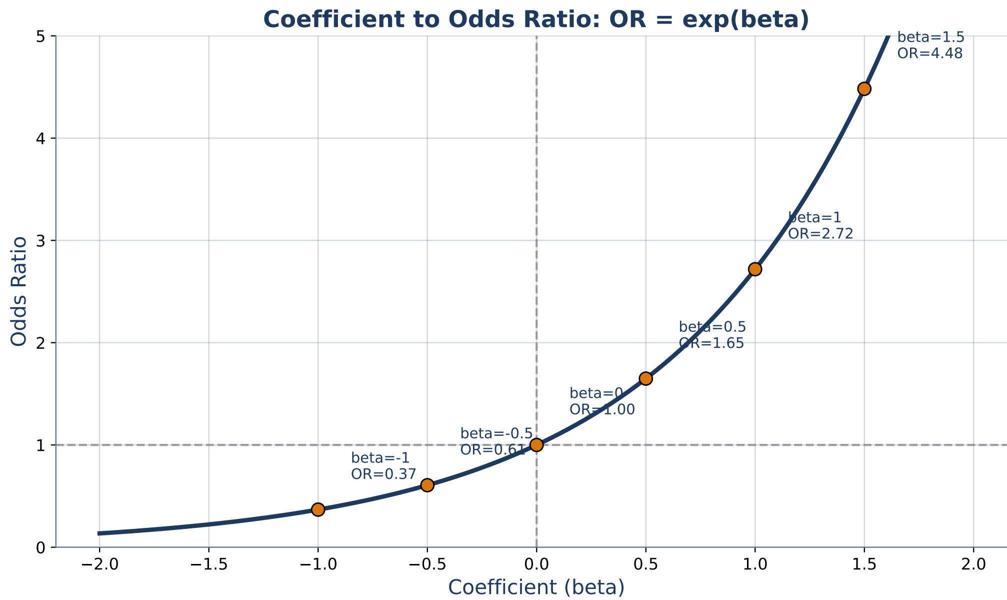

Figure 19: From raw coefficients to odds ratios: the exponential transformation.

Key Formula: Regularized Log Loss

$$J(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log p_i + (1-y_i)\log(1-p_i)\right] + \frac{1}{C}\cdot R(\boldsymbol{\beta})$$

where C is the regularization strength (confusingly, higher C means *less* regularization in scikit-learn) and R is either:

- **L2 (Ridge):** $R = \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 = \frac{1}{2}\sum_j \beta_j^2$ — shrinks all coefficients toward zero
- **L1 (Lasso):** $R = \|\boldsymbol{\beta}\|_1 = \sum_j |\beta_j|$ — drives some coefficients to exactly zero (feature selection)

Historical Background: David Cox and the General Logistic Model (1958)

David Cox published the paper that made logistic regression a general-purpose statistical tool for binary outcomes. Before Cox, researchers primarily used probit analysis (based on the normal CDF). Cox showed that logistic regression was mathematically more convenient—the log-odds are linear in the features—and more interpretable: coefficients map directly to odds ratios.

Cox's framework became the standard in medicine (clinical trials), social science (survey analysis), and eventually finance (credit scoring, default prediction). The simplicity of the odds ratio interpretation is the main reason logistic regression remains the first-choice model for binary classification when interpretability matters, even in an era of deep learning.

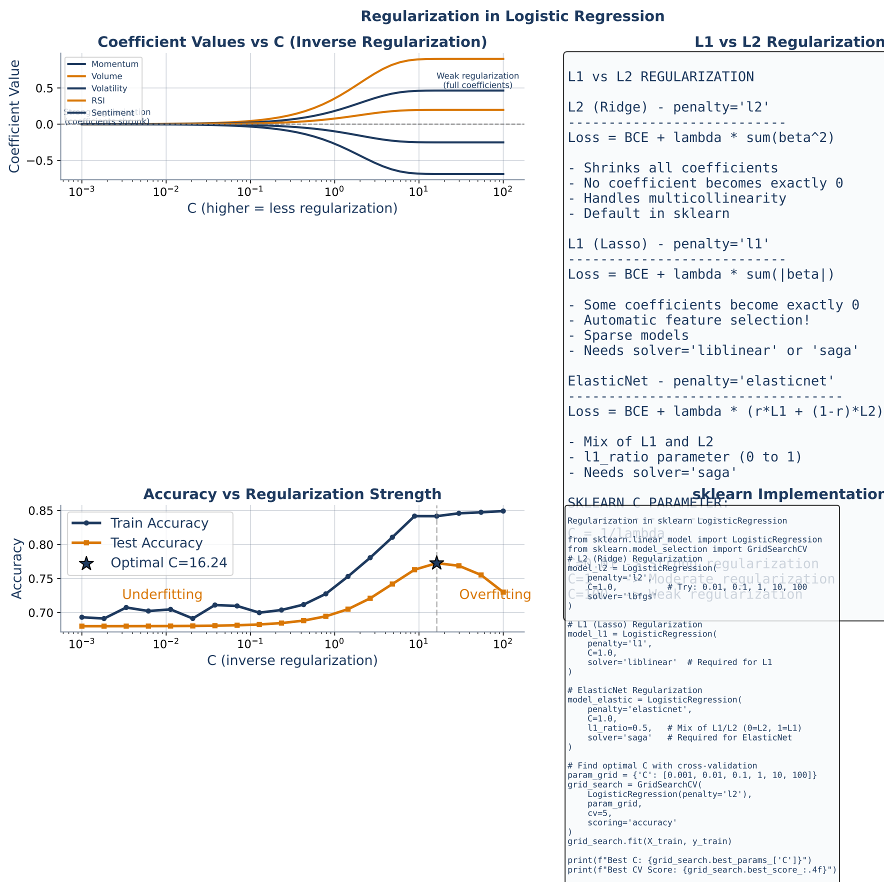

Figure 20: Effect of regularization on the decision boundary. Stronger regularization (smaller C) produces smoother boundaries.

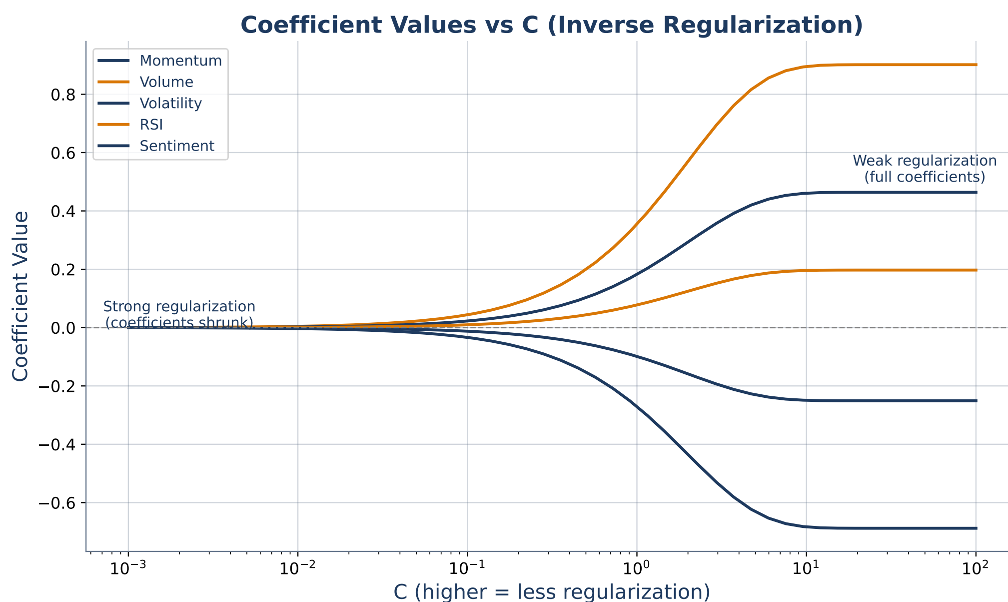

Figure 21: The C parameter controls the trade-off: small C = heavy regularization (simpler model); large C = light regularization (flexible model).

Common Misconception: "A large coefficient means the feature is important"

A coefficient of 100 on a feature measured in millions (e.g., revenue in dollars) has a *tiny* practical effect. A coefficient of 0.5 on a feature measured in standard deviations has a *large* effect. Always consider the feature's scale. To compare importance across features, standardize them first (subtract mean, divide by standard deviation) and then compare coefficients.

Common Misconception: "An odds ratio of 2.0 means twice the probability"

No. It means twice the *odds*. If $p = 0.1$, the odds are $0.1/0.9 \approx 0.111$. Doubling the odds gives 0.222, so the new probability is $0.222/(1 + 0.222) \approx 0.182$. The probability went from 0.1 to 0.18—not from 0.1 to 0.2. The relationship between odds ratios and probability changes is nonlinear.

Common Misconception: "Regularization always improves the model"

Strong regularization (small C) can *underfit* by shrinking informative coefficients toward zero. If the true relationship is strong and the data is plentiful, regularization can hurt. The optimal C should be selected via cross-validation, not assumed.

Beyond Binary: Multiclass Classification

What if the target has more than two categories? For example, classifying a stock into one of three sectors: Technology, Finance, Healthcare.

Softmax function: The generalization of the sigmoid to K classes. For class k: $p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$. All probabilities sum to 1. When $K = 2$, softmax reduces to the sigmoid.

Key Formula: Softmax

$$P(y = k \mid \mathbf{x}) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \qquad z_k = \mathbf{x}^\top \boldsymbol{\beta}_k$$

Each class k has its own coefficient vector $\boldsymbol{\beta}_k$. The softmax normalizes the raw scores so they form a proper probability distribution.

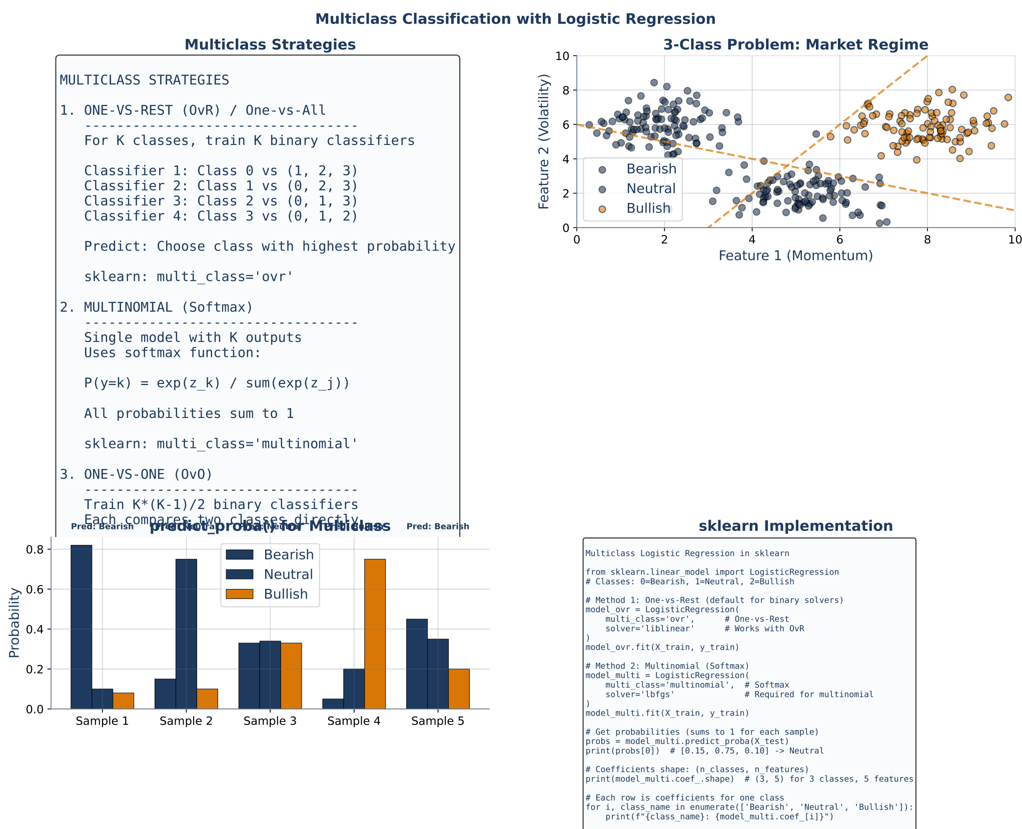

Figure 22: Multiclass logistic regression: the decision boundary consists of multiple lines separating three or more classes.

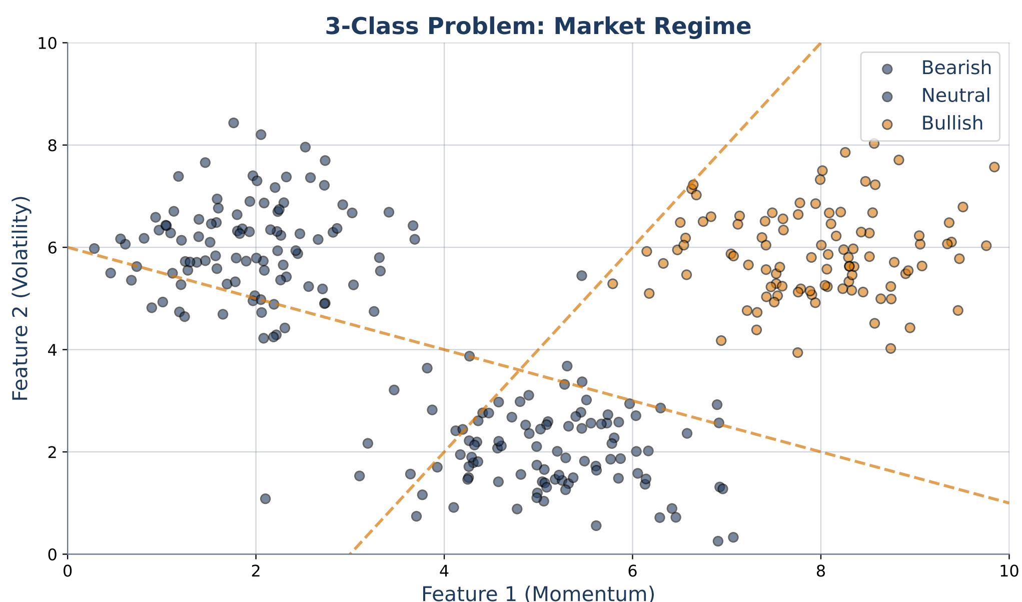

Figure 23: Three-class scatter plot with decision regions. Each region corresponds to one predicted class.

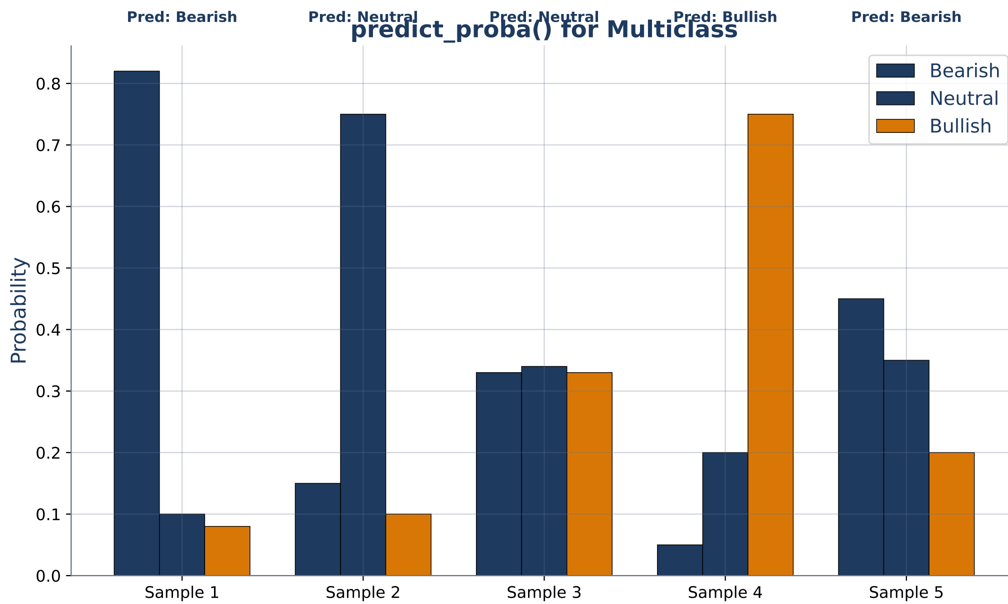

Figure 24: Softmax probabilities for each sample. The three probabilities always sum to 1.

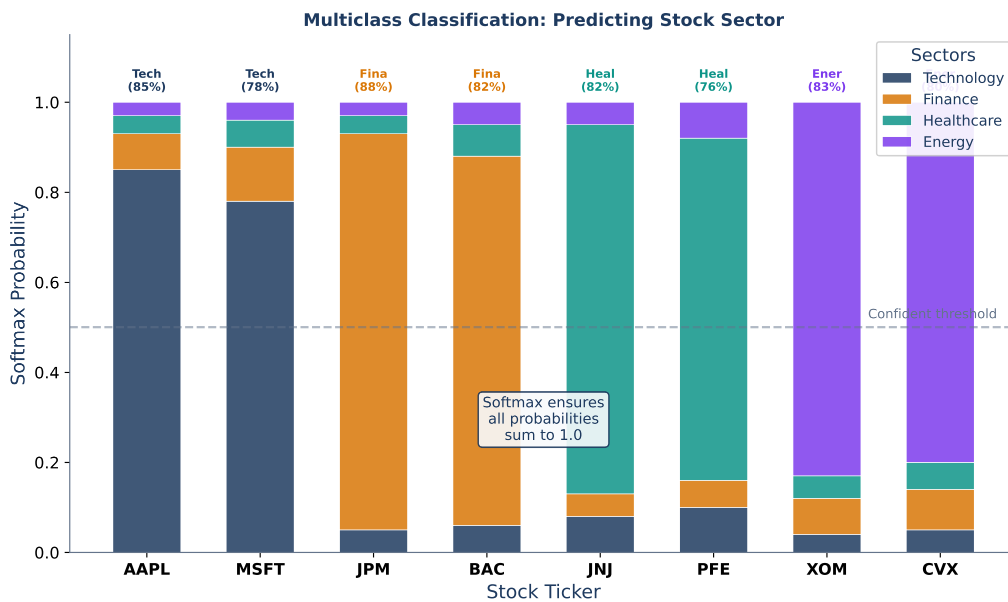

Figure 25: Finance application: classifying stocks into sectors using financial features.

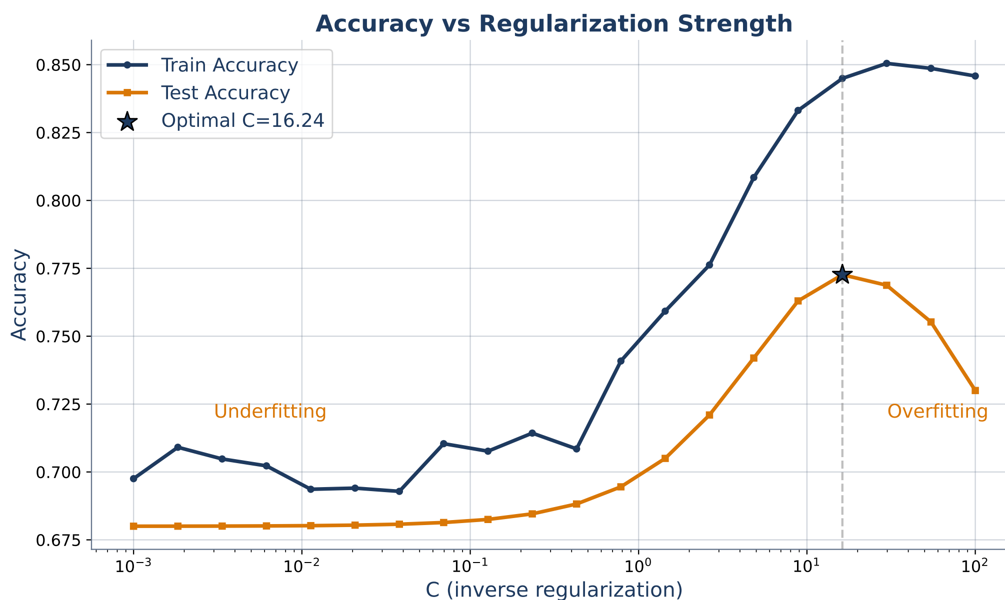

Figure 26: Training vs. test accuracy as regularization strength varies. The gap between them indicates overfitting.

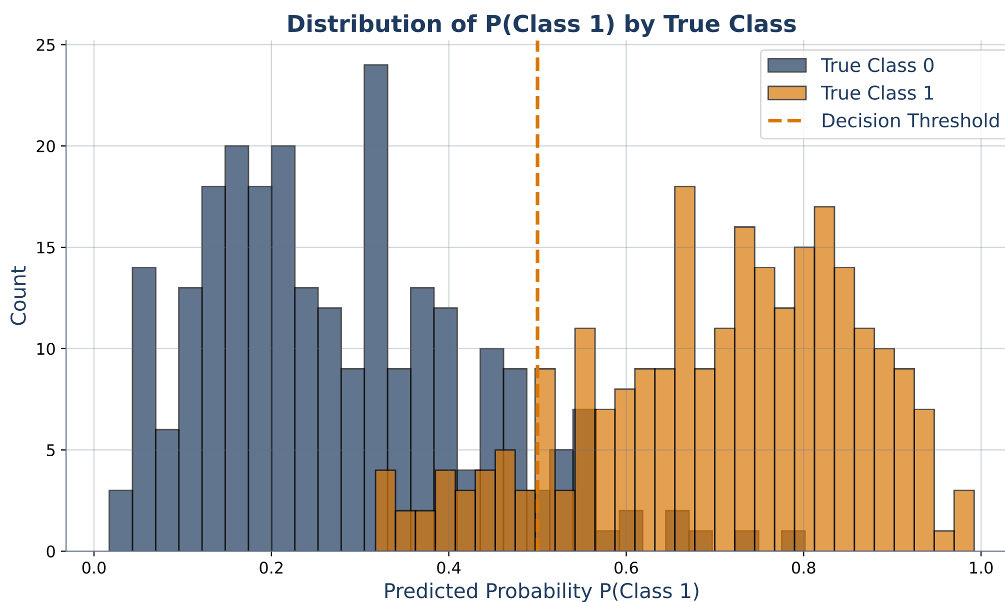

Figure 27: Distribution of predicted probabilities for the two classes. Well-separated distributions indicate a confident model.

Worked Example: Interpreting a Credit Scoring Model

A bank's logistic regression model for loan default has the following coefficients:

Feature	Coeff. (β)	OR (e^β)	Interpretation
Intercept	-2.30	—	Baseline log-odds
Income ($10k)	-0.18	0.835	Each $10k reduces odds by 16.5%
Debt-to-income	1.95	7.03	Each 1.0 increase multiplies odds by 7
Credit score (100s)	-0.45	0.638	Each 100 pts reduces odds by 36%
Years employed	-0.12	0.887	Each year reduces odds by 11.3%
Late payments	0.73	2.075	Each late payment doubles odds

Strongest predictor: Debt-to-income ratio (OR = 7.03). A one-unit increase in debt-to-income (e.g., from 0.3 to 1.3) multiplies the odds of default by 7.
Weakest predictor: Years employed (OR = 0.887). Each additional year reduces default odds by about 11%.
Negative coefficients (income, credit score, years employed) reduce default odds—they are protective factors. **Positive coefficients** (debt-to-income, late payments) increase default risk.

Problem 3.1 (Easy)

Convert $\beta = 0.5$ to an odds ratio. Complete the sentence: "For each one-unit increase in X, the odds of default are multiplied by ________."
Solution: see Appendix.

Problem 3.2 (Medium)

A credit scoring model produces these odds ratios: income (OR = 0.85), debt-to-income (OR = 3.2), years employed (OR = 0.70).

(a) Which feature most strongly predicts default?

(b) How do you interpret OR = 0.85 for income?

(c) If a customer's debt-to-income increases from 0.3 to 0.5 (a 0.2-unit change), by what factor do the odds of default change? (Hint: $\text{OR}^{0.2}$.)

Solution: see Appendix.

Problem 3.3 (Medium)

Explain how changing the C parameter from 0.01 to 100 affects:

(a) The magnitude of the coefficients

(b) The complexity of the decision boundary

(c) The risk of overfitting

Use the terms "regularization strength" and "bias-variance tradeoff."
Solution: see Appendix.

Problem 3.4 (Medium)

A multiclass model outputs softmax probabilities: $[0.15, 0.72, 0.13]$ for classes [Technology, Finance, Healthcare].

(a) What is the predicted class?

(b) What is the model's confidence (as measured by the winning probability)?

(c) The probabilities sum to 1.00. Is this always the case with softmax? Why?

(d) If the probabilities were $[0.34, 0.35, 0.31]$, would you trust this prediction?

Solution: see Appendix.

Problem 3.5 (Hard)

Compare L1 (Lasso) and L2 (Ridge) regularization for logistic regression.

(a) What does L1 do to small coefficients that L2 does not?

(b) Under what conditions does L1 produce a sparse model (many coefficients exactly zero)?

(c) A financial regulator requires that you explain your model using at most 5 features. Which regularization do you use? Why?

(d) In scikit-learn, how do you switch between L1 and L2? (Hint: the `penalty` parameter.)

Solution: see Appendix.

Connecting Forward

Logistic regression draws a straight line through feature space. Powerful, interpretable, and fast. But what if the real boundary between classes is not a straight line? What if customers with both very low and very high income default (the former cannot pay, the latter take excessive risks), while middle-income customers are safe? A straight line cannot capture this U-shaped relationship.

A decision tree draws something very different: axis-aligned rectangles. It asks a series of yes/no questions, splitting the data into increasingly pure groups. The resulting boundaries are jagged but can capture complex, nonlinear patterns. That is Section 4.

Key Takeaway: Coefficients are log-odds: exponentiate to get odds ratios, the universal language for explaining logistic regression predictions.

4 Twenty Questions – Decision Trees and Splitting Criteria

Opening Problem: The Loan Officer's Mental Flowchart

A credit officer at a small bank does not use a formula. He uses a mental flowchart:

1. Is the applicant's income above €50 000? If no, go to step 4.
2. Is their employment stable (more than 3 years)? If yes, approve.
3. Check credit score. If above 700, approve. Otherwise, deny.
4. Is the loan amount below €10 000? If yes, check credit history length...

This *is* a decision tree. The officer just does not call it that. He has learned it from years of experience—which applications ended well, which ended badly.
The question: can a machine learn a *better* tree from thousands of past decisions? One that is consistent, quantified, and free from the officer's biases and blind spots? And how does the machine decide which question to ask first?

Discovery Question

You have 100 loan applications: 50 were approved and repaid, 50 defaulted. You are allowed to ask **one yes/no question** about the data to split the group.
You could ask: "Is income above $50k?" Suppose this produces two groups:

- Left group (income $\leq$ $50k): 40 defaults, 10 repaid
- Right group (income $>$ $50k): 10 defaults, 40 repaid

Or you could ask: "Is the applicant's name longer than 5 letters?" This gives:

- Left group: 25 defaults, 27 repaid
- Right group: 25 defaults, 23 repaid

The income question creates two groups that are mostly one class each. The name question creates two groups that are still mixed. The income split is "better." But how do we measure "better" mathematically? That is what Gini impurity and information gain do.

The 20 Questions Game

You know the parlor game "20 Questions": one person thinks of something, the other asks up to 20 yes/no questions. The key strategy is to ask questions that *eliminate half the remaining possibilities*. "Is it alive?" is a great first question. "Is it a Labrador retriever?" is a terrible one (too specific too early).

A decision tree does the same thing, but with data. Each question (called a *split*) divides the dataset into two groups. The best question is the one that creates the purest groups—where each group is dominated by one class.

Key Definitions

Decision tree: A model that recursively partitions the feature space using binary yes/no splits on individual features. Internal nodes contain splitting conditions; leaf nodes contain class predictions. The model can be read as a set of if-then-else rules.

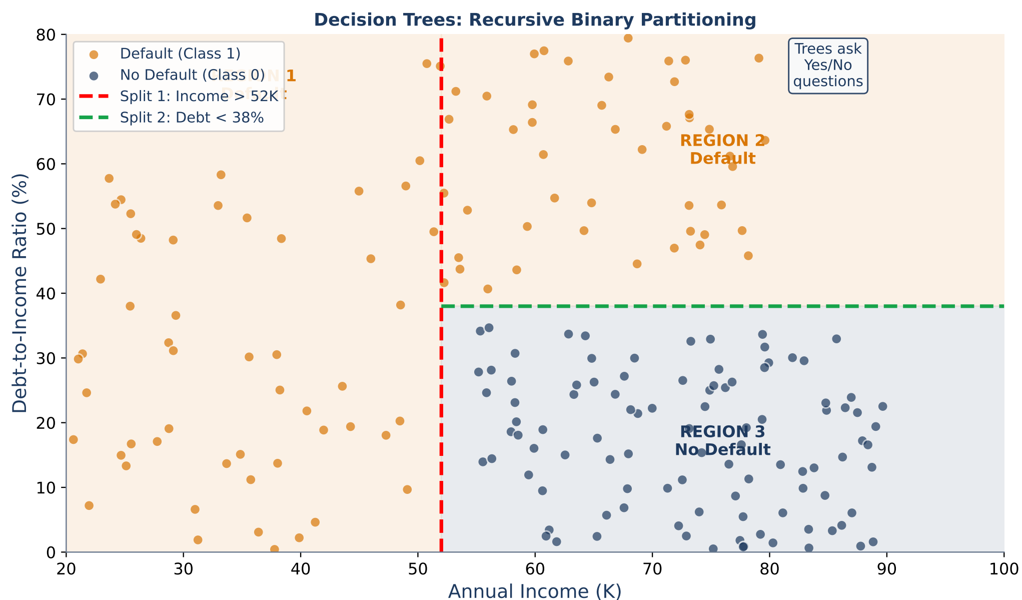

Figure 28: Decision tree intuition: each internal node asks a question; each leaf gives a prediction.

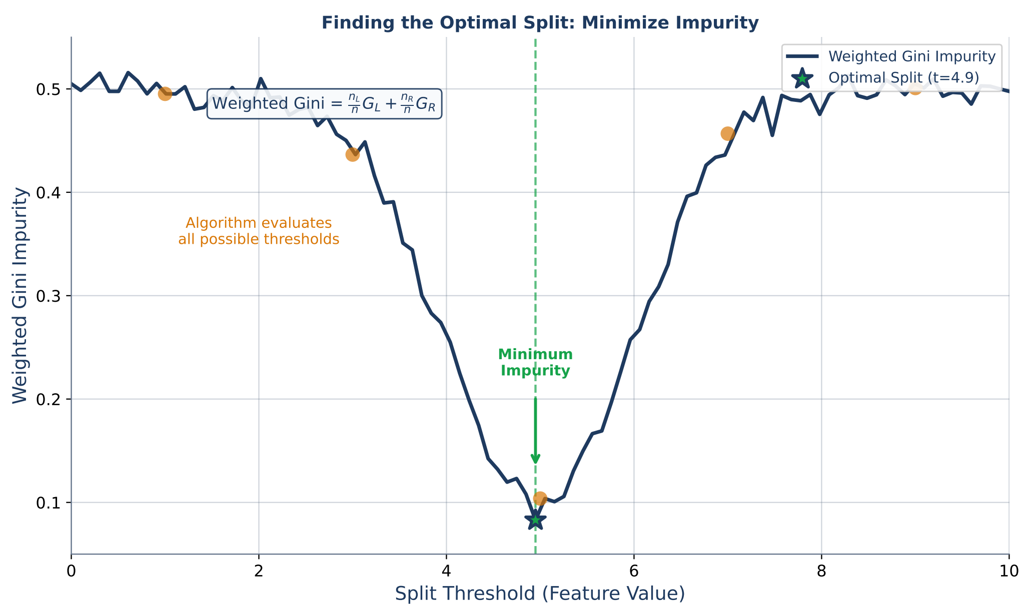

Figure 29: The splitting process: the algorithm evaluates every possible split on every feature and picks the one that maximizes purity.

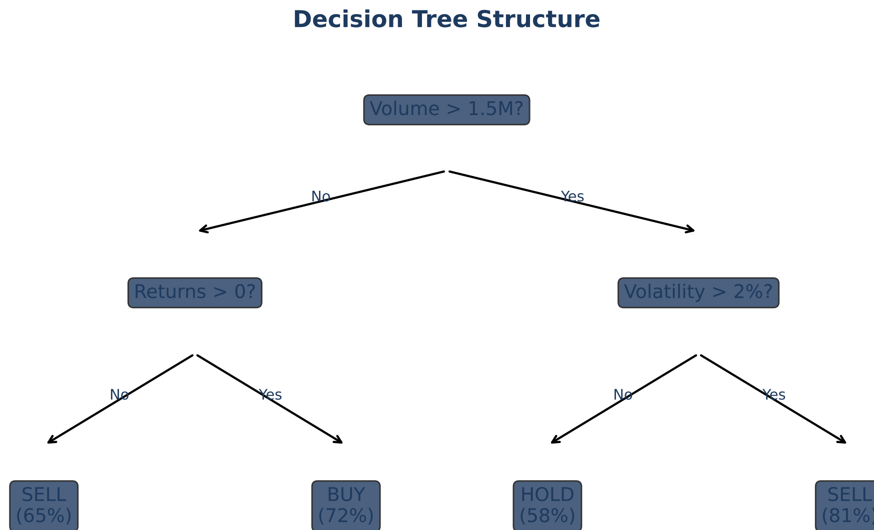

Figure 30: A complete decision tree for a classification problem. Follow the branches from root to leaf to get a prediction.

Node: A point in the tree. The *root* node contains all training data. *Internal* nodes split data according to a condition. *Leaf* nodes produce a final prediction (the majority class in that leaf).

Depth: The number of edges from the root to the deepest leaf. A depth-0 tree has only a root (no splits). A depth-3 tree has at most $2^3 = 8$ leaves.

Measuring Purity: Gini and Entropy

A node is "pure" if it contains only one class. A node is "impure" if classes are mixed. We need a number that quantifies impurity. Two measures dominate:

Gini impurity: $G(p) = 2p(1-p)$ for binary classification, where p is the proportion of the positive class. $G = 0$ means the node is pure (all one class). $G = 0.5$ means maximum impurity (50/50 split). More generally for K classes: $G = 1 - \sum_{k=1}^{K} p_k^2$.

Entropy: $H(p) = -p\log_2 p - (1-p)\log_2(1-p)$ for binary classification. $H = 0$ means pure. $H = 1$ means maximum uncertainty. More generally: $H = -\sum_{k=1}^{K} p_k \log_2 p_k$.

Key Formula: Gini Impurity

For a node with class proportions $p_1, p_2, \ldots, p_K$:

$$G = 1 - \sum_{k=1}^{K} p_k^2$$

Binary case ($K = 2$): Let p be the fraction of class 1. Then $G = 2p(1-p)$.

- Pure node ($p = 0$ or $p = 1$): $G = 0$
- Maximum impurity ($p = 0.5$): $G = 0.5$
- Intuition: G measures the probability that a randomly chosen sample would be misclassified if labeled randomly according to the node's class distribution.

Key Formula: Entropy and Information Gain

Entropy of a node:

$$H = -\sum_{k=1}^{K} p_k \log_2 p_k$$

Information gain from a split:

$$IG = H(\text{parent}) - \sum_{\text{child } c} \frac{n_c}{n_{\text{parent}}} \cdot H(\text{child}_c)$$

where n_c is the number of samples in child c. The split that maximizes IG is chosen. This is exactly the "best question" in the 20 Questions analogy: the one that reduces uncertainty the most.

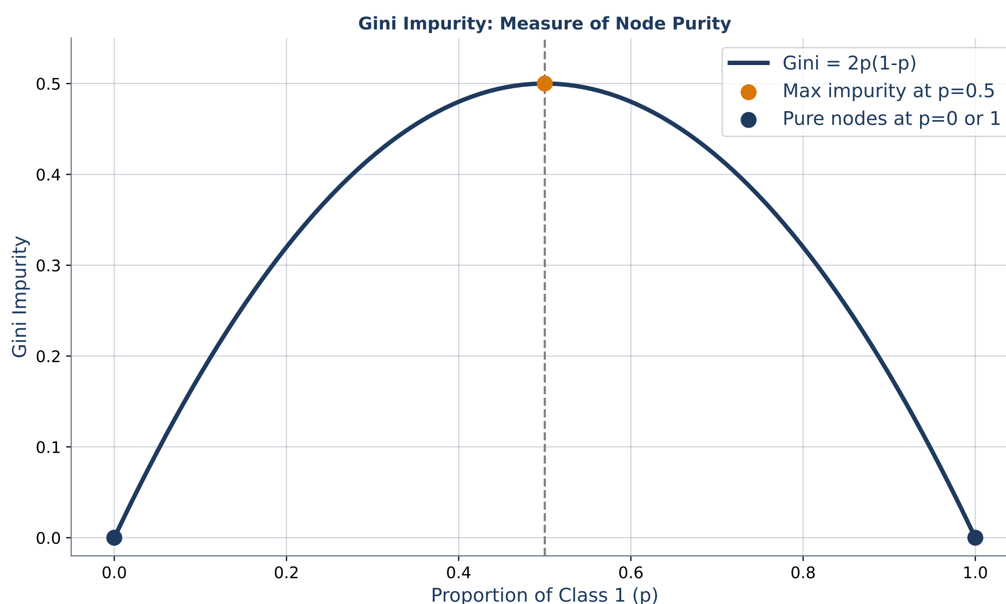

Figure 31: Gini impurity as a function of p. Maximum at $p = 0.5$; zero at $p = 0$ and $p = 1$.

Historical Background: Quinlan, Breiman, and the Birth of Tree Algorithms

J. Ross Quinlan published the ID3 algorithm in 1986—the first practical decision tree algorithm that used information gain to select splits. He improved it to C4.5 in 1993, adding support for continuous features, missing values, and pruning.

Meanwhile, Leo Breiman and colleagues published the CART (Classification and Regression Trees) algorithm in 1984, using Gini impurity instead of entropy. CART introduced cost-complexity pruning and handled both classification and regression.

The two lineages—Quinlan's entropy-based approach and Breiman's Gini-based approach—converge in modern implementations. Scikit-learn's `DecisionTreeClassifier` supports both criteria via the `criterion` parameter. Breiman went on to invent the random forest (1996/2001), which builds on CART. We cover that in Section 5.

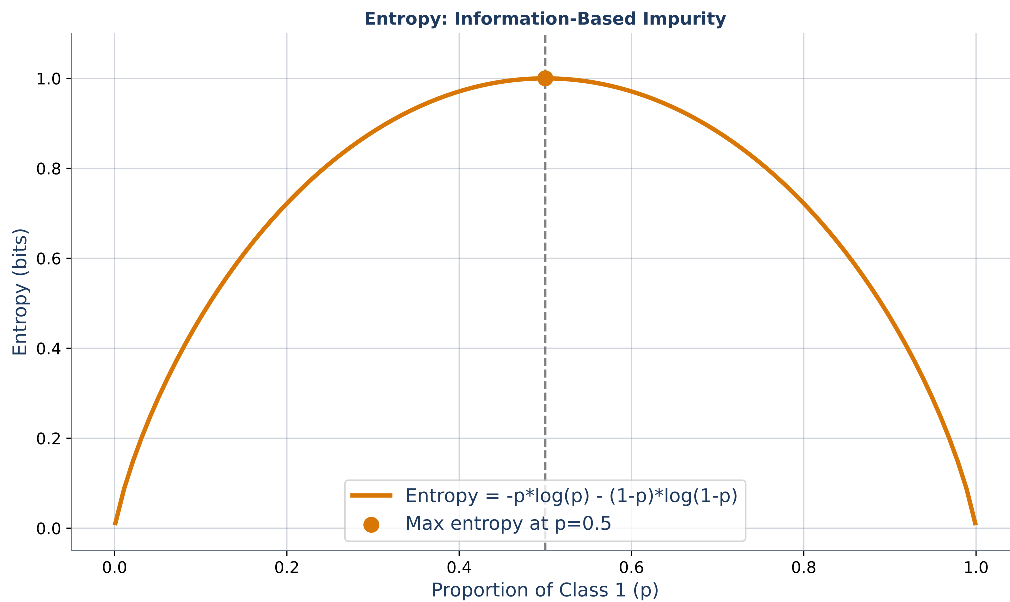

Figure 32: Entropy as a function of p. Same shape as Gini but with a peak of 1.0 instead of 0.5.

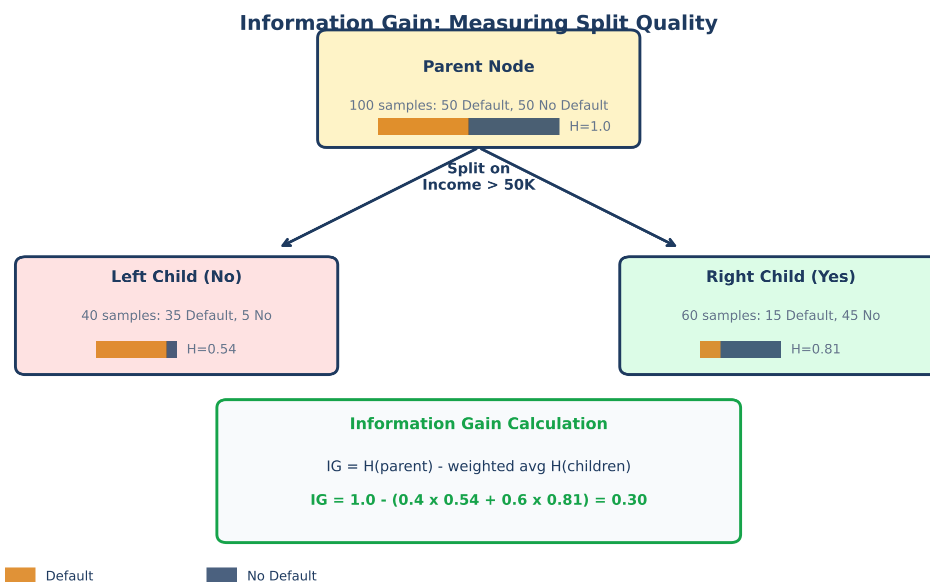

Figure 33: Information gain: the reduction in entropy achieved by a split. Higher is better.

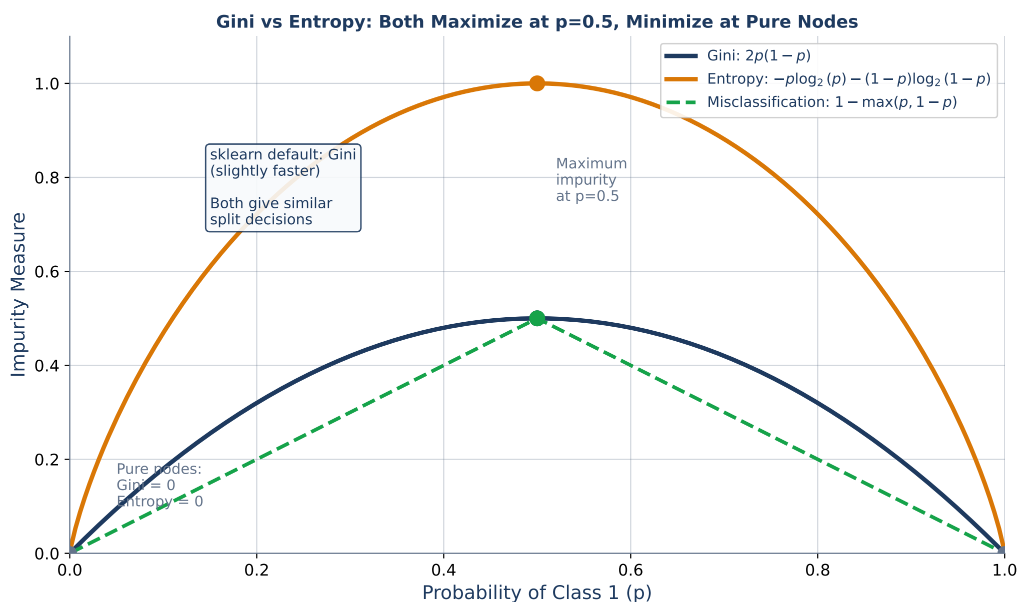

Figure 34: Gini vs. entropy: both peak at $p = 0.5$ and agree on most splits. In practice, the choice rarely matters.

Common Misconception: "Deeper trees are always better"

A deep tree memorizes the training data, including its noise. A depth-20 tree on 100 training samples will have nearly one leaf per sample—perfect training accuracy but terrible generalization. This is classic overfitting. The bias-variance tradeoff applies: shallow trees have high bias (underfitting); deep trees have high variance (overfitting). The sweet spot is in between.

Common Misconception: "Gini and entropy always give different splits"

They agree about 95% of the time. Both are concave functions of class proportions that peak at the uniform distribution. The few cases where they disagree are typically marginal—where multiple candidate splits are nearly equivalent. Do not waste time agonizing over this choice.

Common Misconception: "Decision trees can learn any function"

Technically, a sufficiently deep tree can approximate any decision boundary on the training set. But "approximate on the training set" is not "learn." A tree that perfectly fits noisy training data has *memorized*, not learned. On new data it will fail. Trees need regularization (max depth, min samples per leaf) to generalize.

Common Misconception: "Trees handle continuous features naturally"

Trees discretize continuous features into binary splits: "Is income > $50k?" This is powerful (it handles nonlinearity) but crude (it ignores the ordering within each bin). The split threshold is chosen greedily—the tree considers every unique value of every feature and picks the best one. With 1000 samples and 10 features, that is potentially 10,000 candidate splits per node.

What Decision Boundaries Look Like

Logistic regression produces straight-line boundaries. Decision trees produce axis-aligned rectangles. Figure 35 shows the difference.

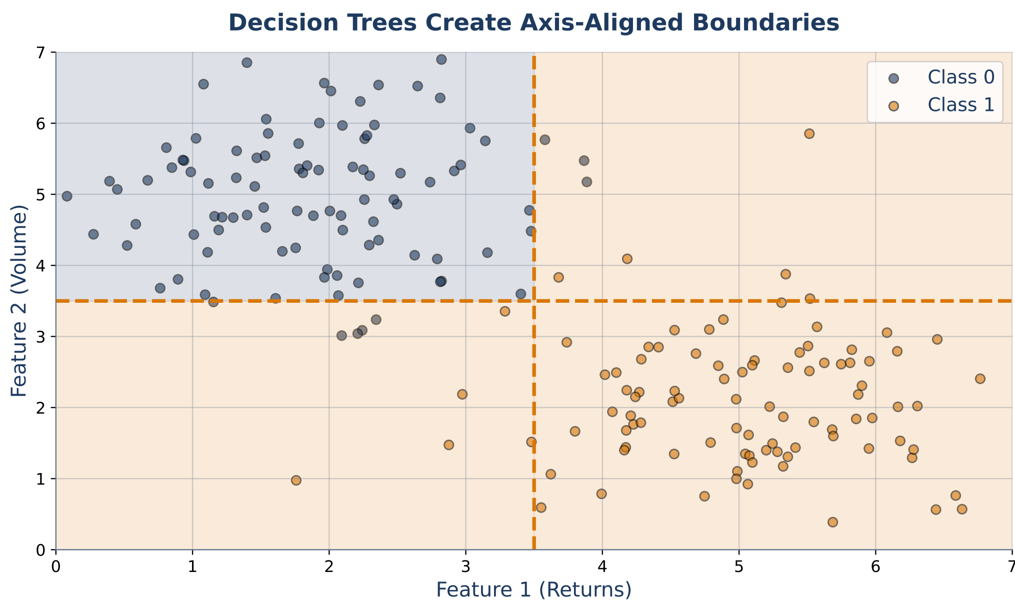

Figure 35: Decision tree boundaries are axis-aligned: each split is parallel to one axis. This creates rectangular regions.

Trees can also solve problems that linear models cannot. The XOR problem (Figure 36) has two classes arranged in a checkerboard pattern. No single straight line can separate them. A depth-2 tree handles it easily with two splits.

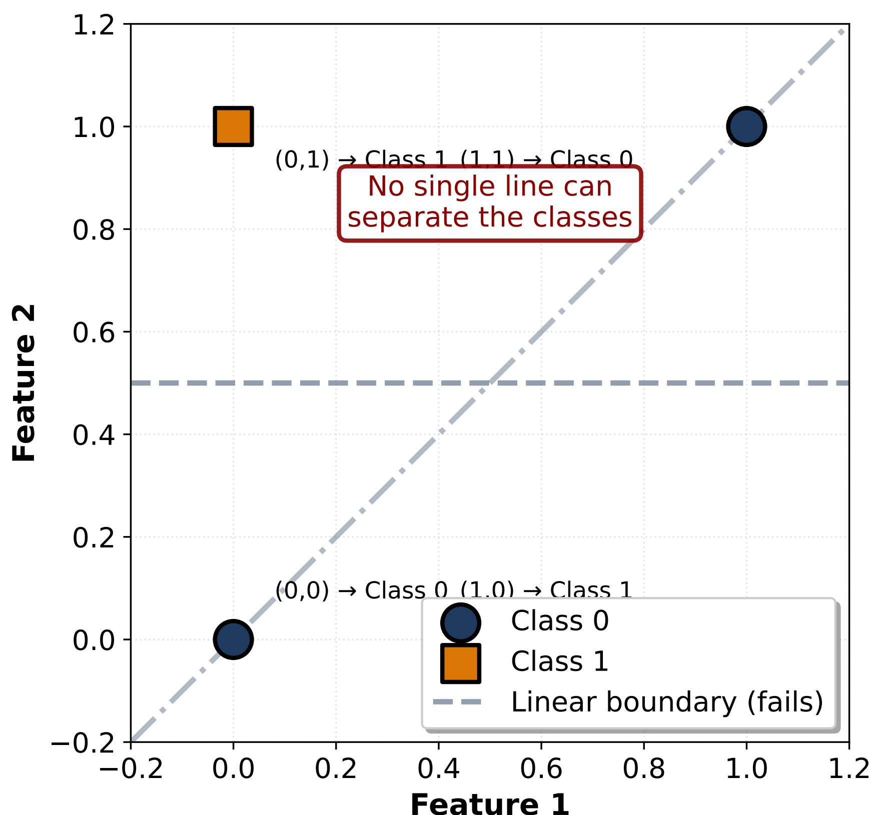

Figure 36: The XOR problem: no linear boundary works, but a decision tree solves it with two splits.

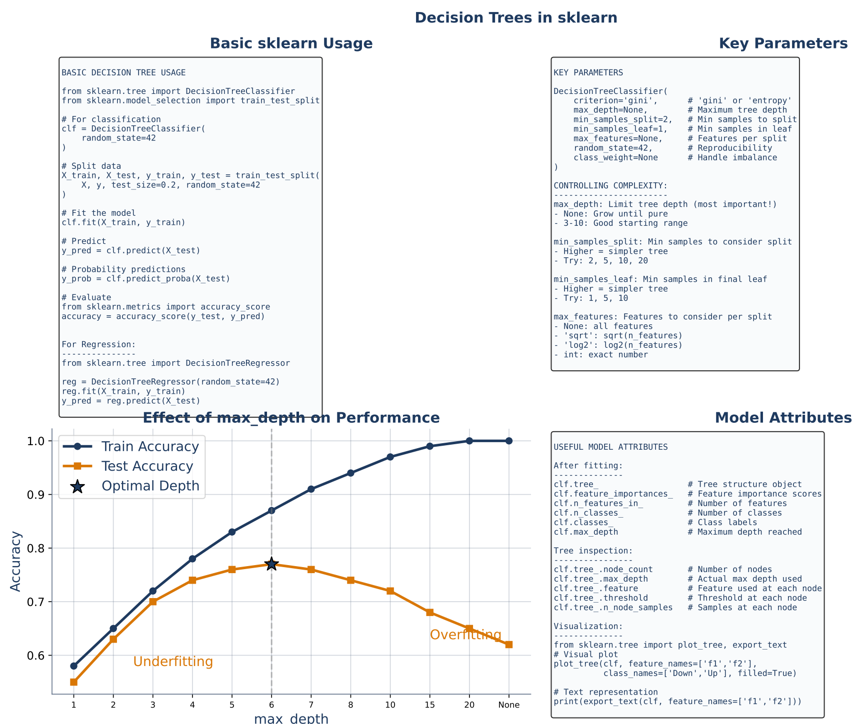

Figure 37: A decision tree trained with scikit-learn on a two-feature dataset.

(c) Joerg Osterrieder 2026

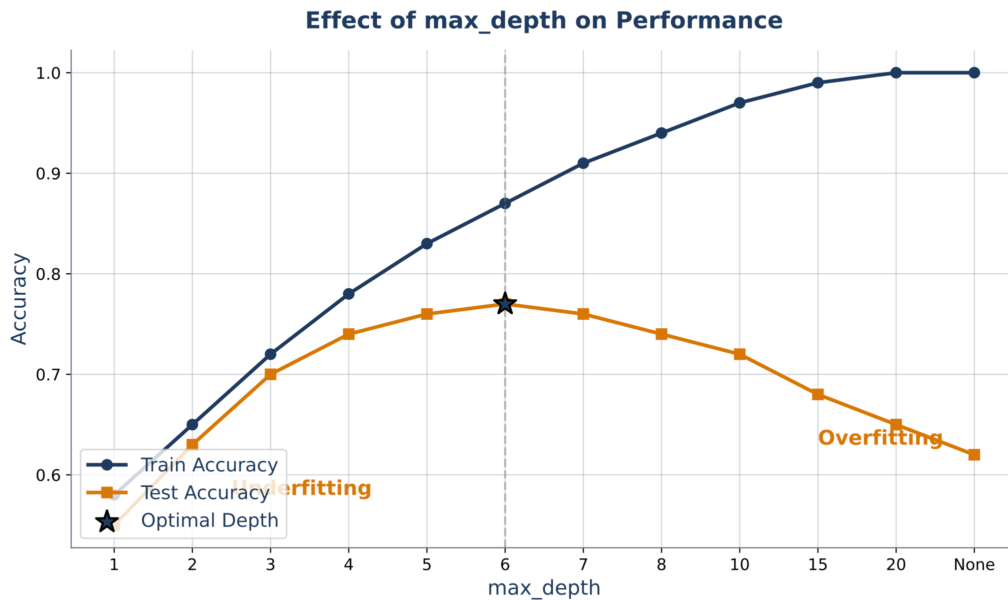

Figure 38: Effect of `max_depth` on the decision boundary. Shallow trees underfit; deep trees overfit.

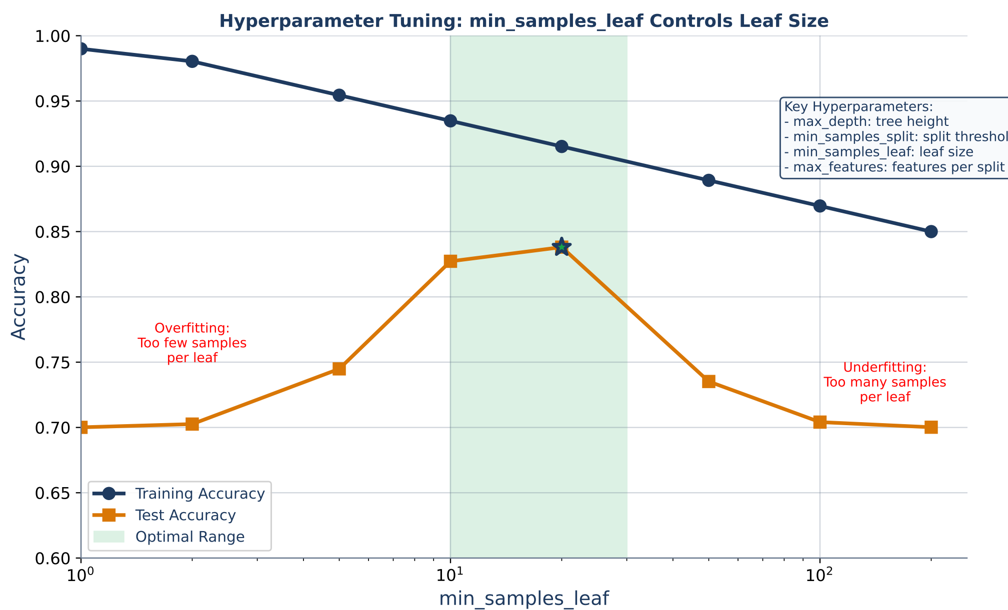

Figure 39: Hyperparameter effects: `max_depth`, `min_samples_split`, and `min_samples_leaf` all control tree complexity.

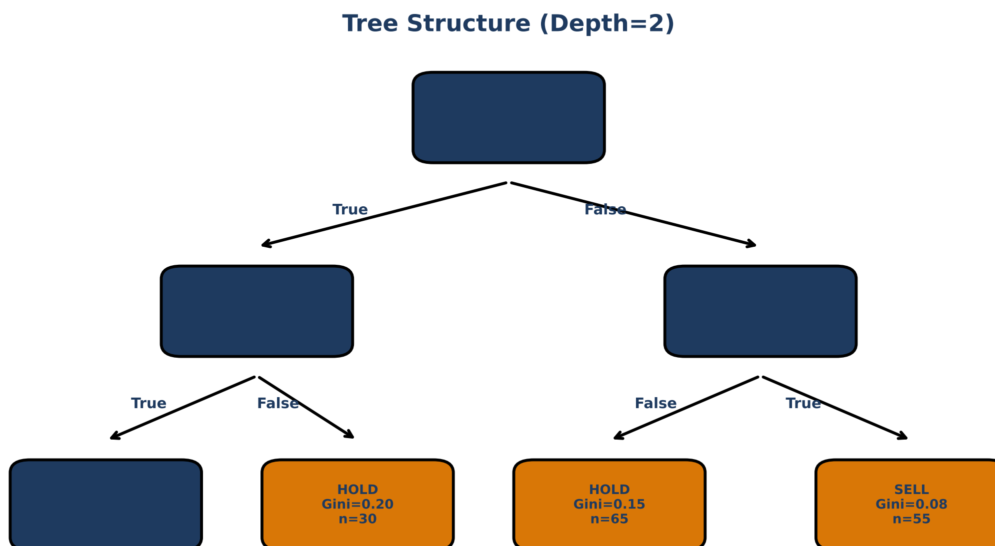

Figure 40: Tree structure visualization: each node shows the split condition, Gini impurity, sample count, and class distribution.

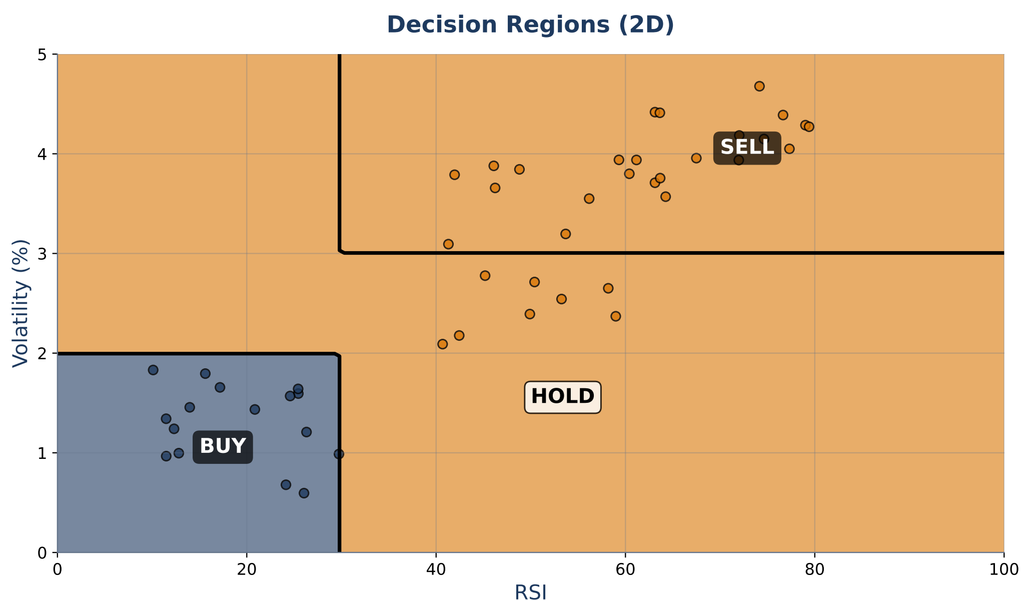

Figure 41: The decision surface produced by the tree. Each rectangular region corresponds to one leaf node.

The Danger of Depth

Visualizing Tree Structure

Worked Example: Building a Tree by Hand

Consider 6 loan applications:

ID	Income ($k)	Credit Score	Default?
1	35	620	Yes
2	80	750	No
3	45	580	Yes
4	70	700	No
5	30	640	Yes
6	90	720	No

Step 1: Compute parent Gini. 3 defaults out of 6: $p = 3/6 = 0.5$. $G = 2(0.5)(0.5) = 0.5$.
Step 2: Try splitting on Income $\leq$ 50.

- Left (≤ 50): IDs 1, 3, 5 $\rightarrow$ 3 defaults, 0 non-defaults. $G_L = 2(1)(0) = 0$.
- Right (> 50): IDs 2, 4, 6 $\rightarrow$ 0 defaults, 3 non-defaults. $G_R = 2(0)(1) = 0$.
- Weighted Gini: $(3/6)(0) + (3/6)(0) = 0$.
- Information gain: $0.5 - 0 = 0.5$. Perfect split!

Step 3: Try splitting on Credit Score $\leq$ 650.

- Left (≤ 650): IDs 1, 3, 5 $\rightarrow$ 3 defaults, 0 non-defaults. $G_L = 0$.
- Right (> 650): IDs 2, 4, 6 $\rightarrow$ 0 defaults, 3 non-defaults. $G_R = 0$.
- Information gain: $0.5 - 0 = 0.5$. Also perfect!

Both splits achieve maximum gain. The algorithm picks one (e.g., the first encountered). This toy example separates cleanly—real data rarely does.

Python Code: Decision Tree with scikit-learn

```python
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split

# Train a decision tree
tree = DecisionTreeClassifier(
    max_depth=4,           # Limit depth to prevent overfitting
    min_samples_leaf=5,    # At least 5 samples per leaf
    random_state=42
)
tree.fit(X_train, y_train)

# Evaluate
print(f"Training accuracy: {tree.score(X_train, y_train):.3f}")
print(f"Test accuracy:     {tree.score(X_test, y_test):.3f}")

# Print the tree rules
print(export_text(tree, feature_names=feature_names))
```

Problem 4.1 (Easy)

Using the 6-row dataset from the worked example, draw a depth-1 tree that splits on Income ≤ 50. Label each leaf with "Default" or "No Default" based on the majority class. How many samples are correctly classified?
Solution: see Appendix.

Problem 4.2 (Easy)

Compute the Gini impurity for a node containing 30 positive samples and 70 negative samples.

(a) What is p (the proportion of positives)?

(b) Compute $G = 2p(1 - p)$.

(c) Is this node "pure"? How does this Gini compare to the maximum possible value?

Solution: see Appendix.

(c) Joerg Osterrieder 2026

Problem 4.3 (Medium)

A parent node has 100 samples: 60 positive, 40 negative. A candidate split produces:

- Left child: 40 samples (35 positive, 5 negative)
- Right child: 60 samples (25 positive, 35 negative)

(a) Compute G_{parent}.

(b) Compute G_{left} and G_{right}.

(c) Compute the weighted average Gini after the split: $G_{\text{split}} = \frac{40}{100}G_L + \frac{60}{100}G_R$.

(d) What is the Gini gain (reduction in impurity)?

(e) Is this a "good" split? Why or why not?

Solution: see Appendix.

Problem 4.4 (Medium)

You train two trees on the same dataset:

- Tree A: `max_depth=2`. Training accuracy: 72%. Test accuracy: 70%.
- Tree B: `max_depth=20`. Training accuracy: 99%. Test accuracy: 55%.

(a) Which tree is overfitting? How can you tell?

(b) Which tree would you deploy in production?

(c) What hyperparameters could you tune to find a better tree between A and B?

Solution: see Appendix.

Problem 4.5 (Hard)

Decision tree boundaries are always axis-aligned (parallel to the coordinate axes).

(a) Explain why this is the case. (Hint: each split tests a single feature against a threshold.)

(b) Draw a 2D dataset where the true boundary is a diagonal line ($x_2 = x_1$). Show that a decision tree needs many splits to approximate it, creating a "staircase" pattern.

(c) Which classifier from Section 2 would handle a diagonal boundary with a single line? Why?

(d) Can you think of a way to let a tree capture diagonal boundaries? (Hint: feature engineering.)

Solution: see Appendix.

(c) Joerg Osterrieder 2026

Connecting Forward

A single decision tree is fast, interpretable, and requires no feature scaling. But it is fragile. Change a few training points and the tree can look completely different. The splits are greedy—each node picks the locally best question without considering what comes next.

What if we grew hundreds of trees, each on a slightly different random sample of the data, and let them vote? The individual trees are still noisy, but their *average* is stable. That is the random forest, and it is one of the most successful machine learning algorithms ever invented. It is the topic of Section 5.

Key Takeaway: A decision tree is an automated flowchart that finds the most informative questions to ask—but a single tree is rarely enough.

(c) Joerg Osterrieder 2026

5 The Forest for the Trees – Ensembles, Random Forests, and Overfitting

Opening Problem

The credit model from Section 4 worked well in Q1 2024. Recall the tree that split first on debt-to-income ratio, then on employment length. The bank was satisfied.
Then Q2 arrived. Five hundred new applications came in. The team retrained the tree on the combined dataset. The result: the tree changed *completely*. Different features at the root. Different split points. Different predictions for the same applicant profiles that had been correctly classified last quarter.
The risk manager called an emergency meeting: "We trusted this model. We explained it to regulators. Now one quarter of new data has rewritten everything. How can we rely on something this fragile?"
This is the *variance problem* of single decision trees. Small changes in training data produce wildly different models. The tree from Section 4 was not wrong—it was unstable.

Discovery Question

Why does asking 100 weak experts and averaging their opinions beat asking one strong expert? Intuitively, the strong expert should know more—so why does the crowd win?

5.1 The Wisdom of Crowds

There is a famous experiment. Put a jar of jellybeans on a table. Ask 100 people to guess the count. Each individual guess is wildly off—some guess 200, others 900, the true count is 547. But the *average* of all 100 guesses? Remarkably close to 547. Almost always within 5%.

This is not magic. Each person brings different biases. Some overcount. Some undercount. When you average, the biases cancel. What remains is signal.

Decision trees have the same problem as individual jellybean guessers: high variance. Depending on which training samples they see, they produce very different models. But if you grow many trees—each seeing a slightly different version of the data—and average their predictions, the variance cancels. The signal remains.

Before we formalize this, look at what variance does to a single tree.

Three specific examples make the problem concrete. A shallow tree (Figure 43) misses obvious patterns—it *underfits*. A well-tuned tree (Figure 44) captures real structure without memorizing noise. A deep tree (Figure 45) draws absurdly complex boundaries that fit every training point, including the noisy ones.

The gap between the good tree and the deep tree is exactly the variance problem. Different training samples push a single tree toward different jagged boundaries. An ensemble smooths this out.

5.2 Bagging and Random Forests

Bagging: Bootstrap Aggregating—train multiple models on random subsets of the data (drawn with replacement), then average their predictions. Proposed by Leo Breiman in 1996.

Random forest: An ensemble of decision trees, each trained on a bootstrap sample with a random subset of features considered at each split. The double randomization (random samples *and* random features) ensures the trees are diverse.

Out-of-bag (OOB) error: Error estimated on samples not included in a tree's bootstrap sample. Each bootstrap draw leaves out roughly 37% of the data. These leftover samples provide a free validation set—no separate holdout needed.

(c) Joerg Osterrieder 2026

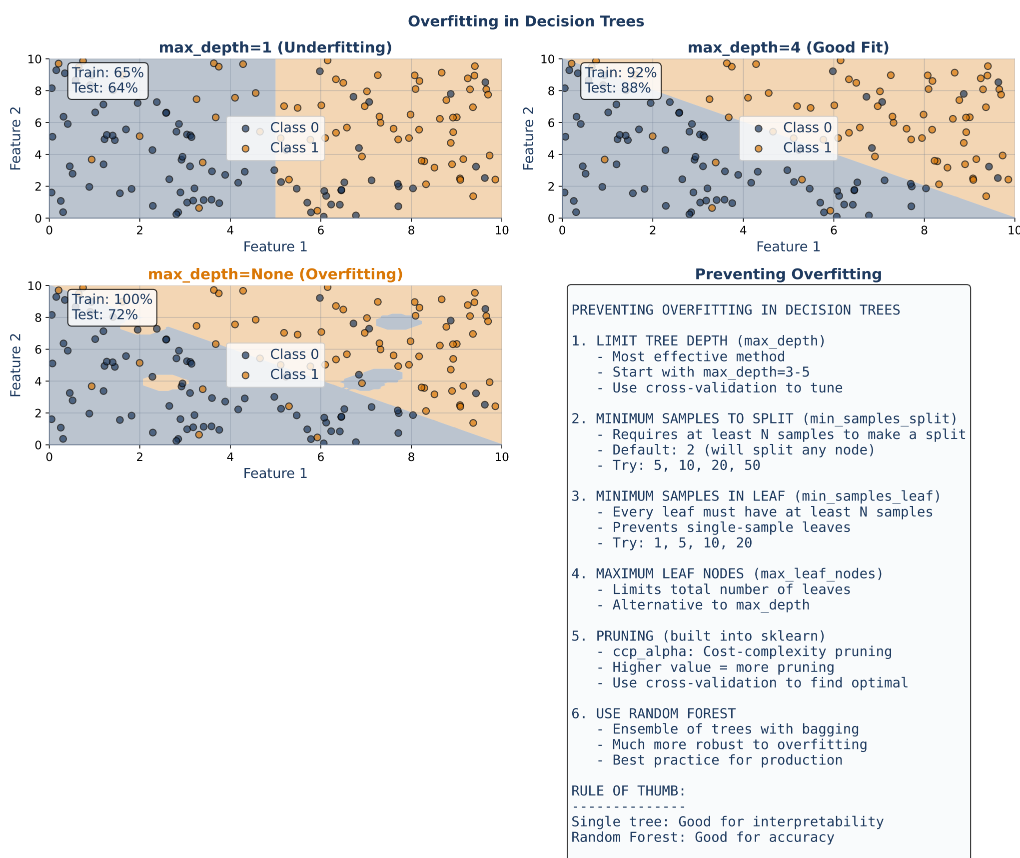

Figure 42: The overfitting spectrum: a single decision tree ranges from underfitting (too simple) to overfitting (too complex). The sweet spot is narrow and data-dependent.

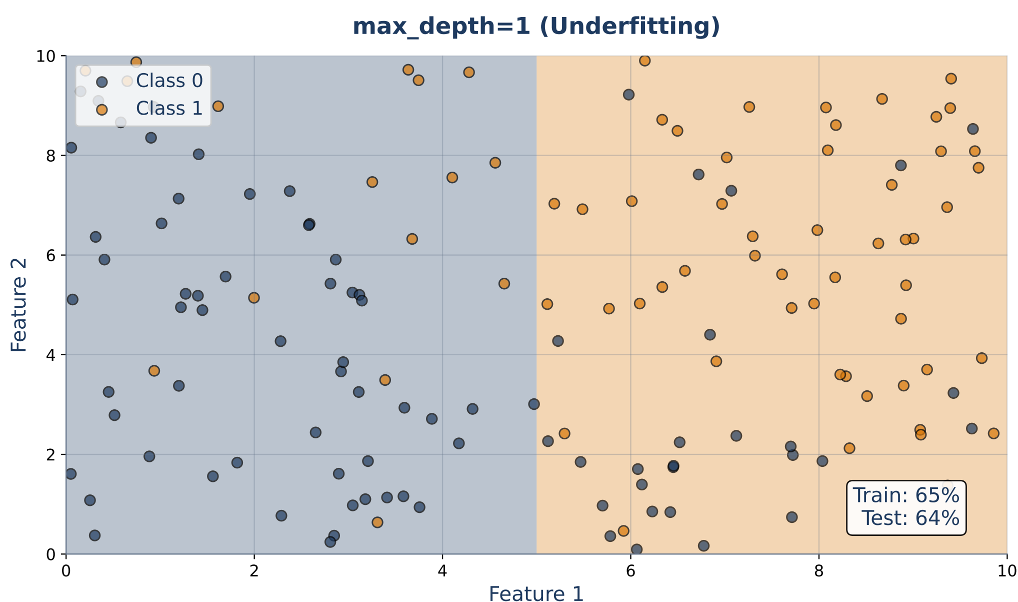

Figure 43: Underfitting: a shallow tree with max_depth=1 captures only the coarsest pattern and misses everything else.

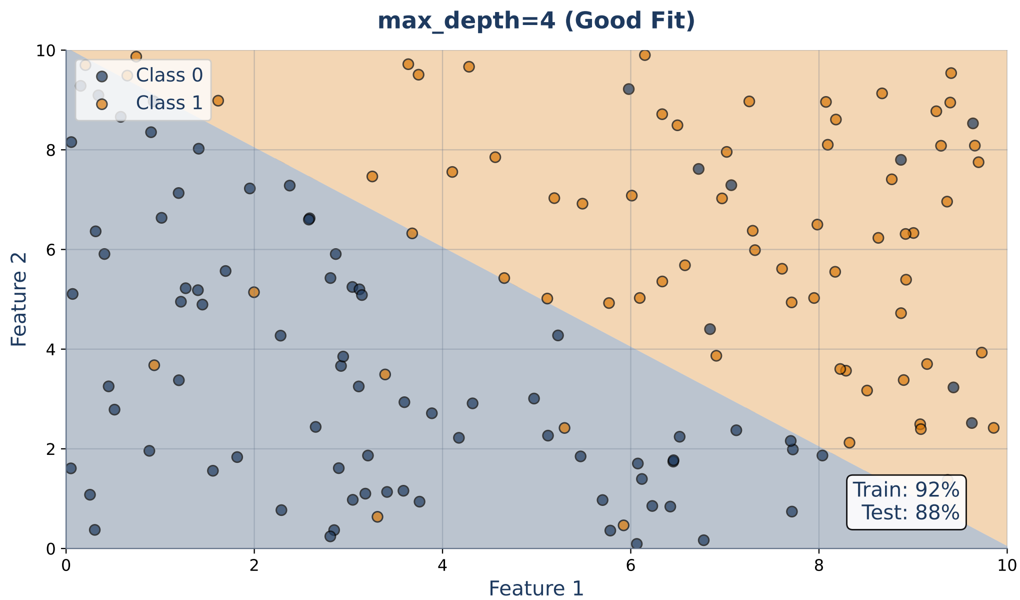

Figure 44: Good fit: a tree with moderate depth captures the real decision boundary without chasing noise.

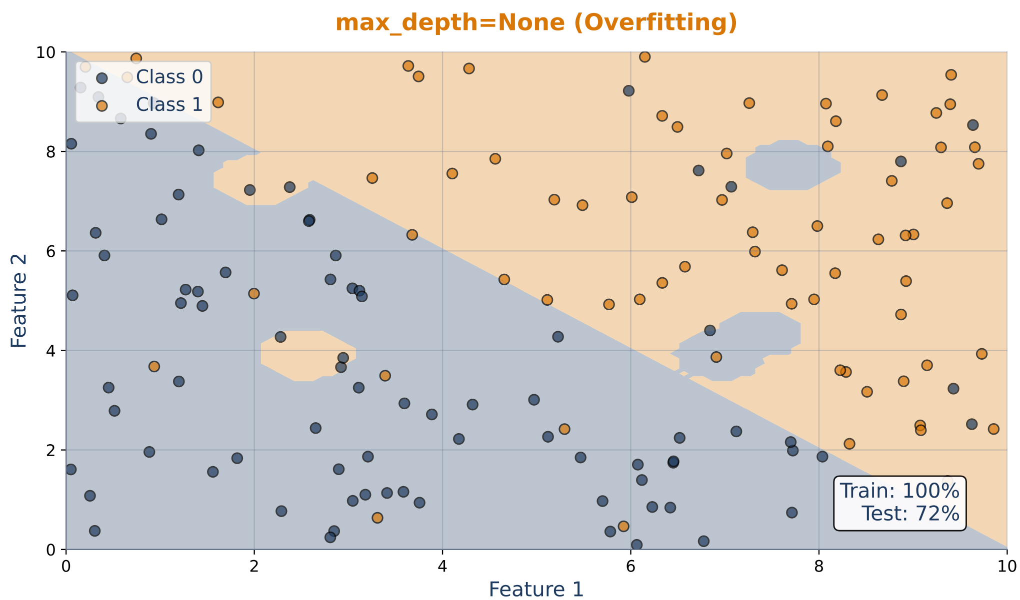

Figure 45: Overfitting: a deep tree memorizes training noise. The boundary is jagged, unstable, and will not generalize.

Bagging Prediction

For a classification task with B trees, the bagged prediction for input $\mathbf{x}$ is the majority vote:

$$\hat{y}_{\text{bag}}(\mathbf{x}) = \text{mode}\{\hat{y}_1(\mathbf{x}),\ \hat{y}_2(\mathbf{x}),\ \ldots,\ \hat{y}_B(\mathbf{x})\}$$

For probability estimates, average the individual tree probabilities:

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

At each split, a random forest considers only m features out of p total. The default for classification:

$$m = \lfloor\sqrt{p}\rfloor$$

This forces trees to use different features at their roots, making them less correlated. Less correlation means more variance reduction when averaging.

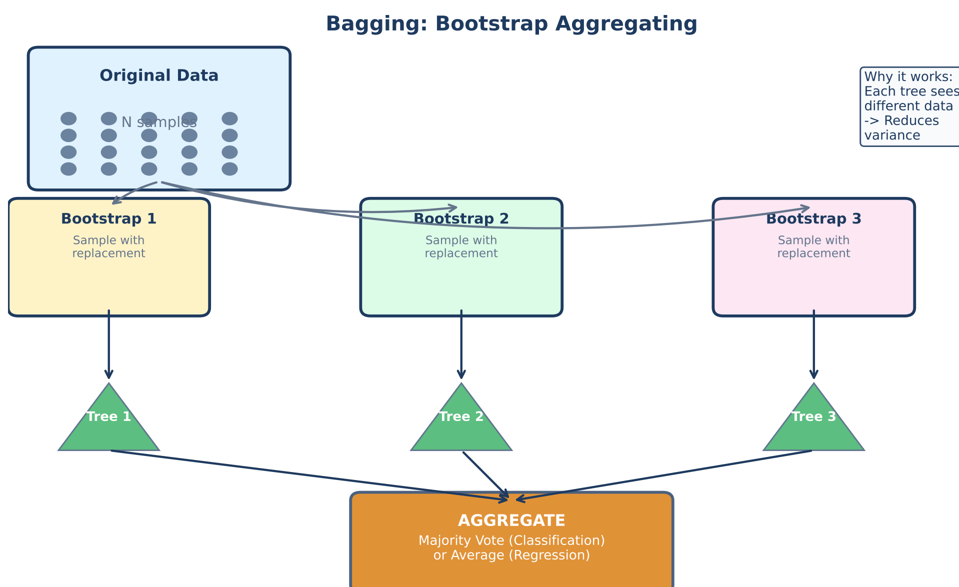

Figure 46: Bagging: each tree sees a different bootstrap sample. Some observations appear multiple times; others are left out entirely (the OOB samples).

Historical Background

Leo Breiman, 2001. He published the Random Forests paper and the provocative "Statistical Modeling: The Two Cultures" essay in the same year. Breiman, a professor at UC Berkeley, challenged the statistics establishment head-on. Traditional statisticians assumed a data-generating model—linear regression, logistic regression, parametric distributions. Breiman called this the "data model" culture. Machine learning practitioners let algorithms discover patterns without assuming a specific model. He called this the "algorithmic model" culture and argued it was the future.

Random forests were his proof of concept. A method with almost no assumptions about the data-generating process, yet it matched or beat carefully specified parametric models across dozens of benchmarks. The paper has over 100,000 citations. The "Two Cultures" essay remains one of the most debated papers in statistics.

Figure 47: Random forest: multiple diverse trees vote together. The ensemble decision boundary is smoother and more stable than any single tree.

Misconception: More trees always improve accuracy

Returns diminish rapidly. Going from 10 trees to 100 typically produces a large accuracy gain. Going from 100 to 1,000 rarely changes accuracy by more than 0.1%. Going from 1,000 to 10,000 is almost always wasted computation. The OOB error curve flattens early (see Figure 48).

Misconception: Feature importance means causal importance

Feature importance measures predictive contribution—how much the model relies on a feature for accurate predictions. A feature merely correlated with the target (but not causal) can rank high. The number of fire trucks at a scene correlates with fire damage. A random forest might rank "number of trucks" as important. Reducing the truck count does not reduce damage.

Misconception: Random forests never overfit

They can. With very deep trees, very few samples, or extremely noisy features, a random forest will overfit. The `max_depth` and `min_samples_leaf` hyperparameters still matter. What *is* true: random forests are far more resistant to overfitting than single trees, and they degrade more gracefully.

5.3 How Many Trees Are Enough?

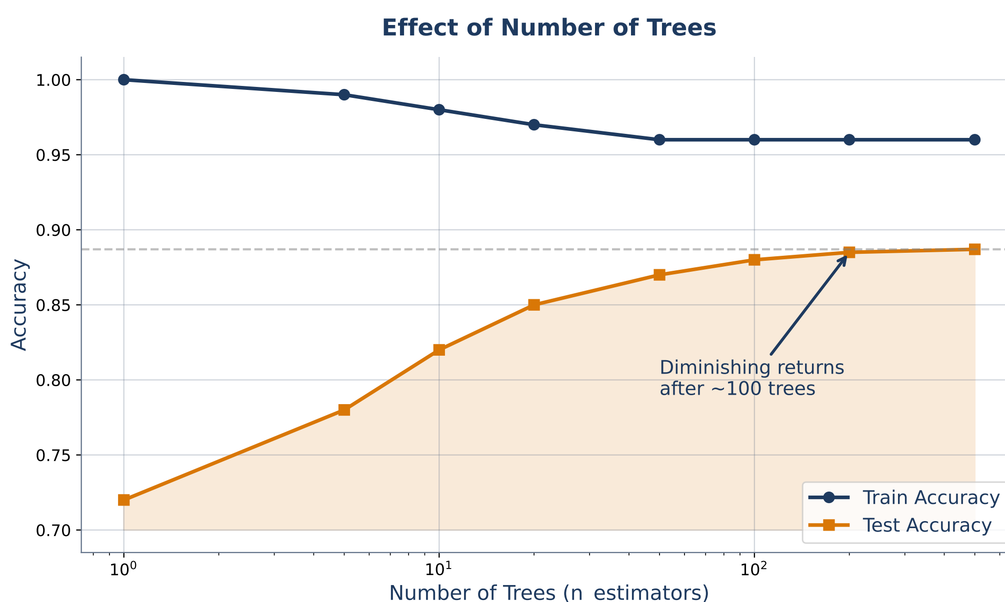

Figure 48: Accuracy vs. number of trees. The curve rises steeply from 1 to 50 trees, then flattens. Beyond 200 trees, improvements are negligible.

5.4 Feature Importance: Gini vs. Permutation

One of the most useful properties of random forests: they rank features by importance. Two common methods exist.

Gini importance (also called mean decrease in impurity) measures how much each feature contributes to reducing impurity across all splits in all trees. It is fast—it falls out of the

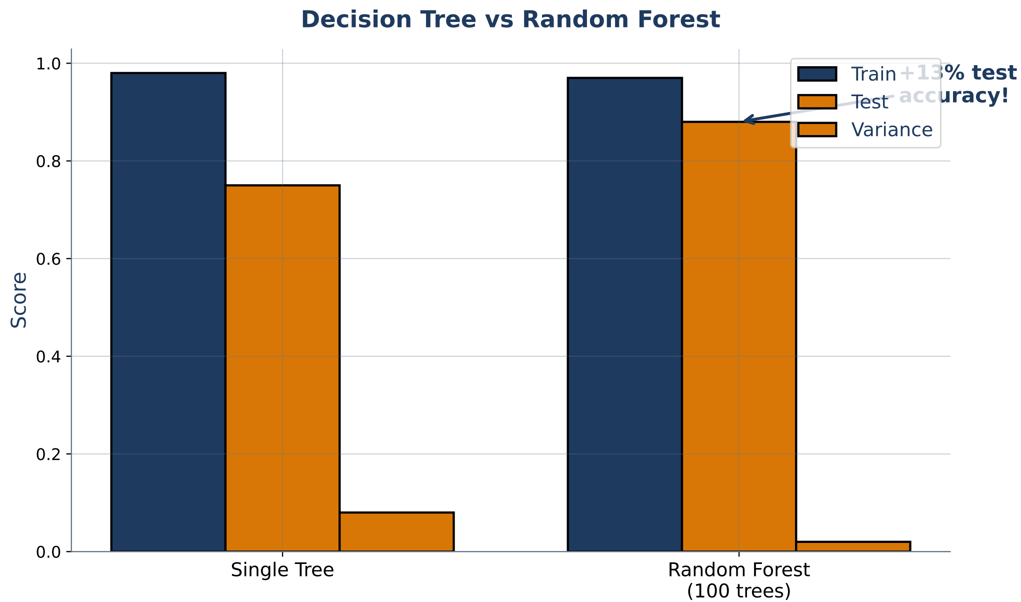

Figure 49: Single tree vs. random forest on the same data. The forest boundary is smoother and generalizes better to unseen points.

training process for free. But it has a bias: features with many unique values or high cardinality get inflated importance scores.

Permutation importance works differently. After training, randomly shuffle one feature's values and measure how much accuracy drops. Big drop? Important feature. Barely any change? Dispensable. Permutation importance is slower but unbiased.

5.5 Finance Application: Backtesting a Forest

In the opening problem, the single tree changed completely when Q2 data arrived. A random forest would not. Individual trees within the forest would shift—but the ensemble prediction would remain stable. The forest absorbs local instability into global robustness. That is the entire point.

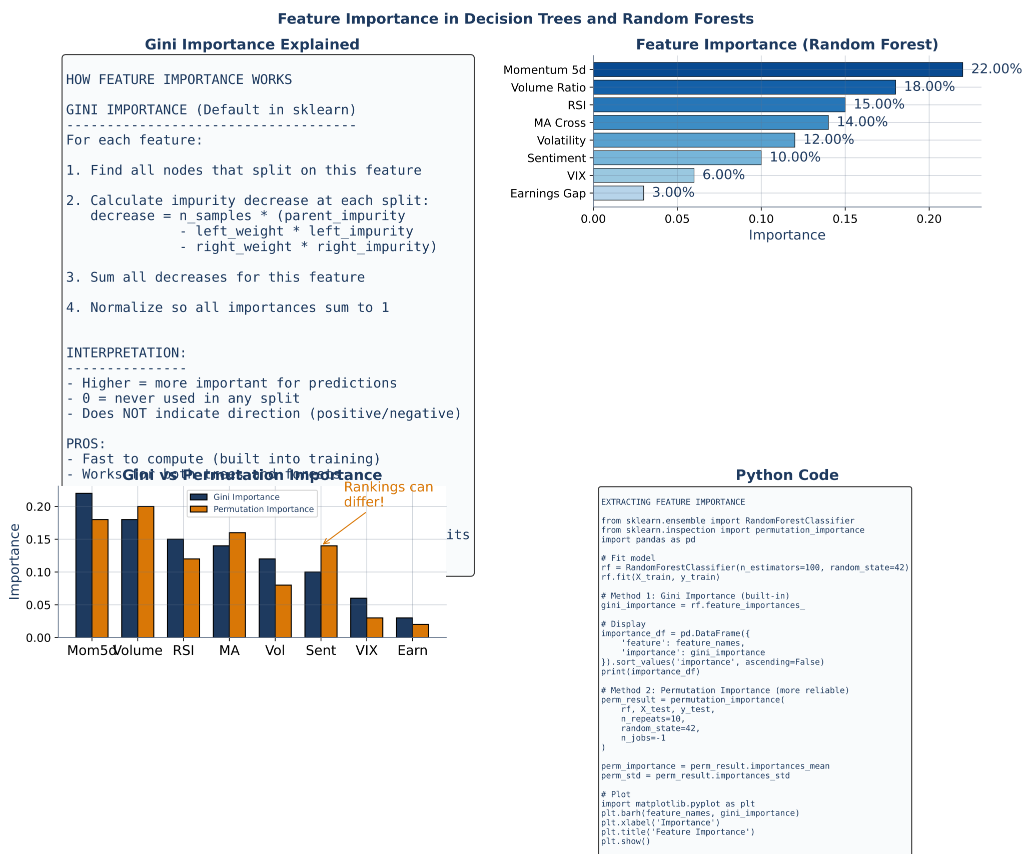

Figure 50: Feature importance concept: which features does the forest rely on most?

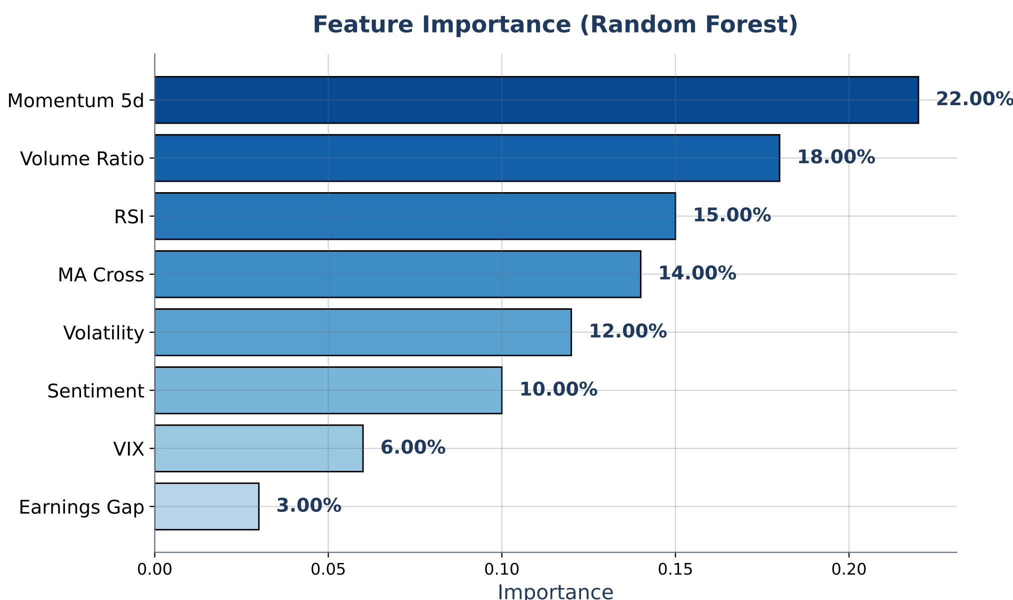

Figure 51: Feature importance bar chart for a credit model. Income and debt ratio dominate; age contributes little.

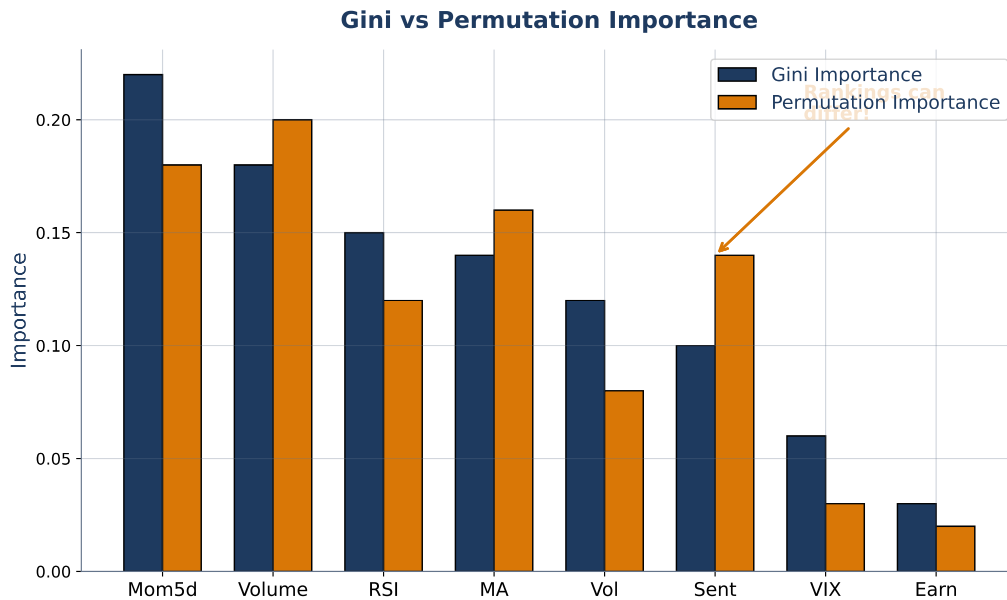

Figure 52: Gini vs. permutation importance. They can disagree, especially for correlated features. Permutation importance is the more honest measure.

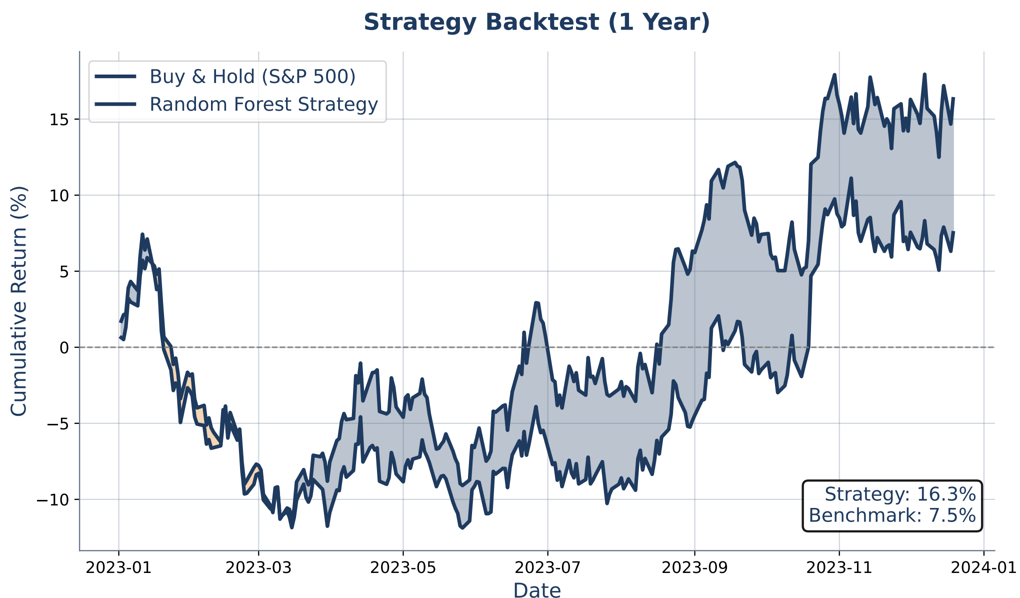

Figure 53: Backtesting a random forest trading model. The ensemble produces more stable out-of-sample performance than any single tree.

Python: Random Forest with Feature Importance

```python
from sklearn.ensemble import RandomForestClassifier

# Train a random forest
rf = RandomForestClassifier(
    n_estimators=200,    # 200 trees (diminishing returns beyond)
    max_depth=8,         # prevent individual trees from
        overfitting
    random_state=42
)
rf.fit(X_train, y_train)

# Feature importance (Gini-based, built-in)
importances = rf.feature_importances_
for name, imp in sorted(zip(feature_names, importances),
                         key=lambda x: x[1], reverse=True):
    print(f"{name:25s} {imp:.4f}")

# OOB score (free cross-validation)
rf_oob = RandomForestClassifier(
    n_estimators=200, max_depth=8,
    oob_score=True, random_state=42
)
rf_oob.fit(X_train, y_train)
print(f"OOB accuracy: {rf_oob.oob_score_:.4f}")
```

Problem 5.1 (Easy)

Explain bagging in your own words, in three sentences or fewer. Why does training on different random subsets of the data reduce variance compared to training on the full dataset once?

Problem 5.2 (Easy)

A random forest reports the following feature importances for a credit scoring model:

Feature	Importance
Income	0.35
Debt ratio	0.28
Employment years	0.20
Age	0.10
Number of accounts	0.07

Which are the top 3 predictors? Do these importance scores tell you whether income *causes* creditworthiness? Why or why not?

(c) Joerg Osterrieder 2026

Problem 5.3 (Medium)

A single decision tree achieves 95% training accuracy and 72% test accuracy. A random forest achieves 89% training accuracy and 84% test accuracy. The single tree is "smarter" on training data. Explain why the forest wins on test data. What phenomenon does the 23-point train/test gap in the single tree indicate?

Problem 5.4 (Medium)

Consider two features in a credit model: `income` and `monthly_spending`. These are highly correlated ($r = 0.82$). Compare Gini importance and permutation importance for these two features. Which method gives a more honest picture of each feature's unique contribution, and why?

Problem 5.5 (Hard)

Design an experiment to select `n_estimators` for a stock trading signal model. Your dataset contains daily features from 2015–2023.

(a) What metric would you use to evaluate the forest? (Hint: accuracy is a poor choice for trading signals.)

(b) Can you use standard k-fold cross-validation? Why is shuffling dangerous for time-series data?

(c) Propose a validation scheme that respects temporal ordering.

(d) Sketch the expected relationship between `n_estimators` and your chosen metric.

Connecting Forward

Random forests and logistic regression (Section 2) both output probabilities. A forest might say "probability of default = 0.73." Logistic regression might say "probability of default = 0.68." Both sound confident.

But how *good* are those probabilities? A model can output 0.9 confidence and still be wrong half the time. It can achieve 99% accuracy and catch zero frauds. Measuring classification quality requires metrics beyond accuracy—and those metrics are Section 6.

Key Takeaway: A random forest reduces variance by averaging many diverse trees—ensemble wisdom beats individual expertise.

6 Beyond the Grade – Confusion Matrix, Precision, Recall, and F1

Opening Problem

A junior data scientist presents to the board of a European bank. The slide reads:

Fraud Detection Model — Accuracy: 99.8%

The CFO smiles. The chief risk officer does not. She asks one question: "How many actual frauds did you catch?"
The answer: zero.
The model predicted "not fraud" for every single transaction. Since only 0.2% of transactions are fraudulent, always predicting "not fraud" yields 99.8% accuracy. The model learned the laziest possible pattern: ignore the minority class entirely. And accuracy—the metric the junior analyst was so proud of—rewarded this laziness.
Accuracy is a lie when classes are imbalanced.

Discovery Question

Your model scores 95% accuracy. Your colleague's model scores 60%. You both apply for the same position and present these numbers. Who gets the job?
Now the interviewer adds context: the dataset has 95% negative cases and 5% positive cases. Your colleague's model catches 90% of the positives. Yours catches 0%. Who gets the job now?

6.1 The Grading Analogy

Think of school grades. An A student who only attempts the easy questions vs. a C student who attempts every question, including the hard ones. Who learned more?

The A student has high "accuracy" on attempted problems. But they dodged the difficult material. The C student made more errors but engaged with the full range. In a final exam with only hard questions, the C student outperforms.

Classification metrics face the same tension. Accuracy counts all correct predictions equally. But in fraud detection, cancer screening, and credit default prediction, the "hard questions"—the rare positive cases—are exactly the ones that matter most.

6.2 The Four Outcomes

Every binary classification produces one of four outcomes for each sample. Get these straight—everything in this section builds on them.

True Positive (TP): The model predicted positive, and the sample is actually positive. Example: flagged a transaction as fraud, and it was fraud.

False Positive (FP): The model predicted positive, but the sample is actually negative. Example: flagged a legitimate transaction as fraud. Also called a Type I error, or a false alarm.

False Negative (FN): The model predicted negative, but the sample is actually positive. Example: a fraudulent transaction slipped through as "legitimate." Also called a Type II error, or a miss.

True Negative (TN): The model predicted negative, and the sample is actually negative. Example: correctly classified a legitimate transaction as legitimate.

Now the metrics.

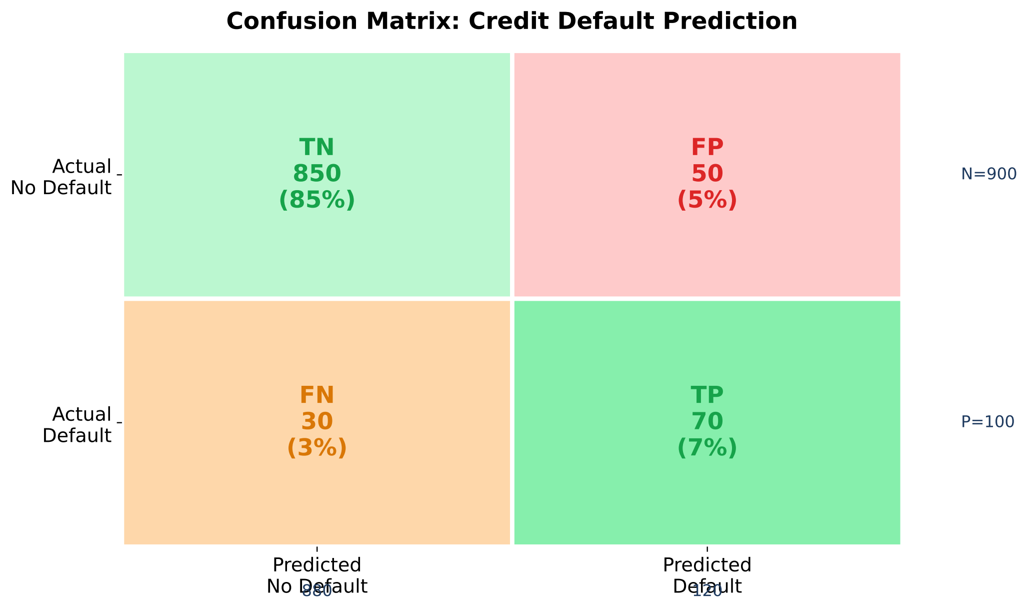

Figure 54: The confusion matrix: a 2×2 table that counts all four outcomes of a binary classifier. Everything—precision, recall, F1, accuracy—is derived from these four numbers.

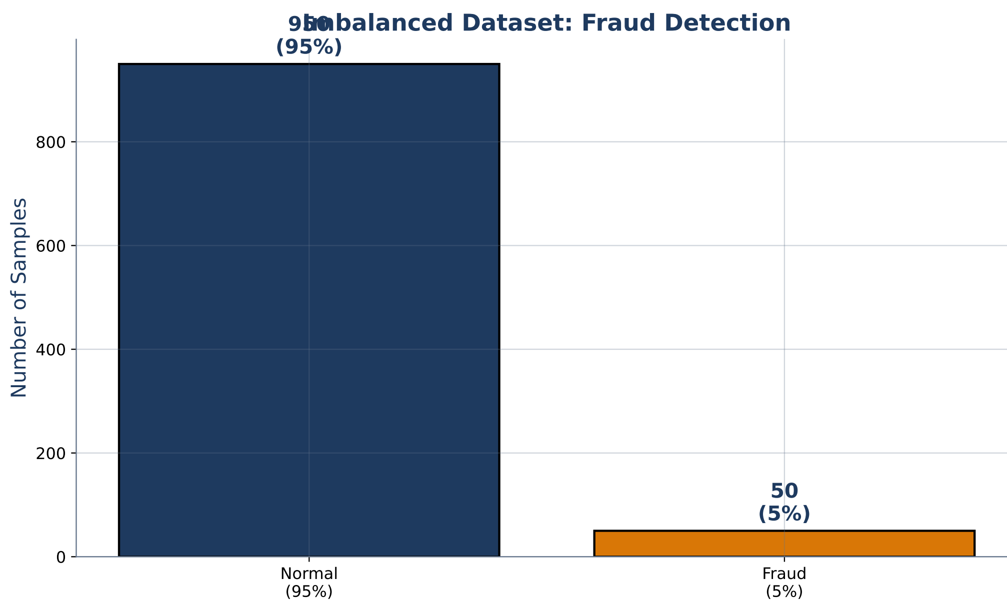

Figure 55: An imbalanced dataset: the positive class (fraud) is a tiny sliver. A model that ignores this sliver achieves near-perfect accuracy.

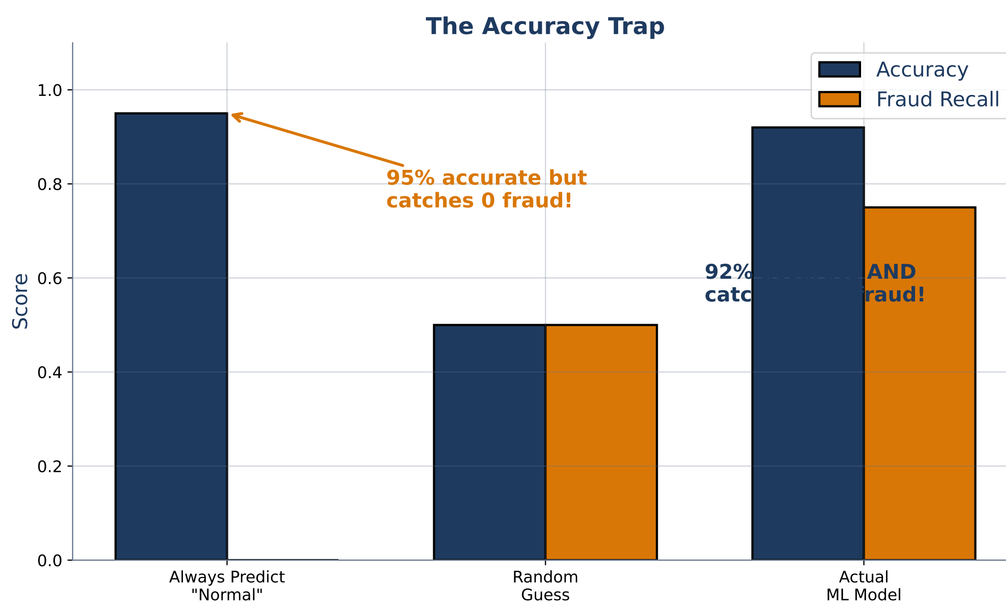

Figure 56: The accuracy trap: a “dumb” model that always predicts the majority class scores high accuracy while providing zero useful information.

Precision

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

Question it answers: Of all the transactions I *flagged* as fraud, how many actually were fraud?
High precision means few false alarms. When you flag something, you are usually right.

Recall (Sensitivity)

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

Question it answers: Of all the actual frauds, how many did I *catch*?
High recall means few missed positives. You rarely let a fraud slip through.

F1 Score

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

The harmonic mean of precision and recall. It is low whenever *either* precision or recall is low—you cannot game it by maximizing one while ignoring the other.

Accuracy (for reference)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

The fraction of all predictions that are correct. Useful for balanced datasets. Misleading for imbalanced ones.

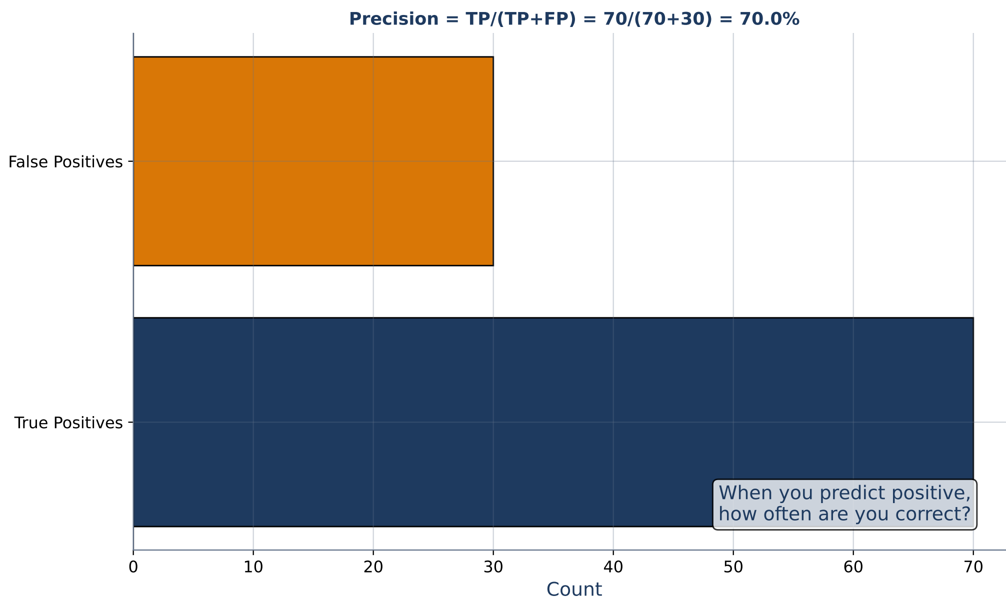

Figure 57: Precision visualized: of everything the model flagged, what fraction is correct?

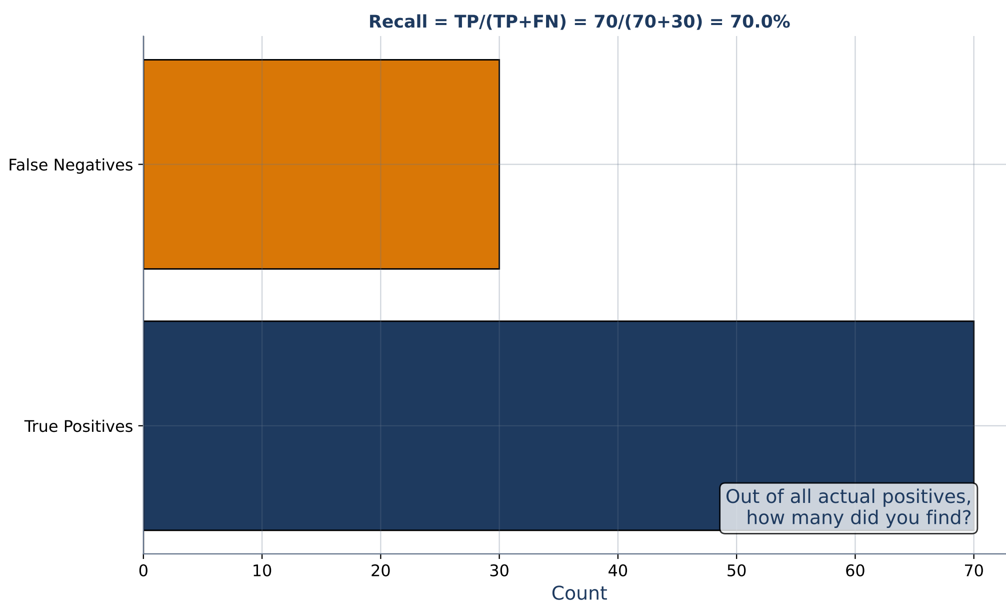

Figure 58: Recall visualized: of everything that is actually positive, what fraction did the model catch?

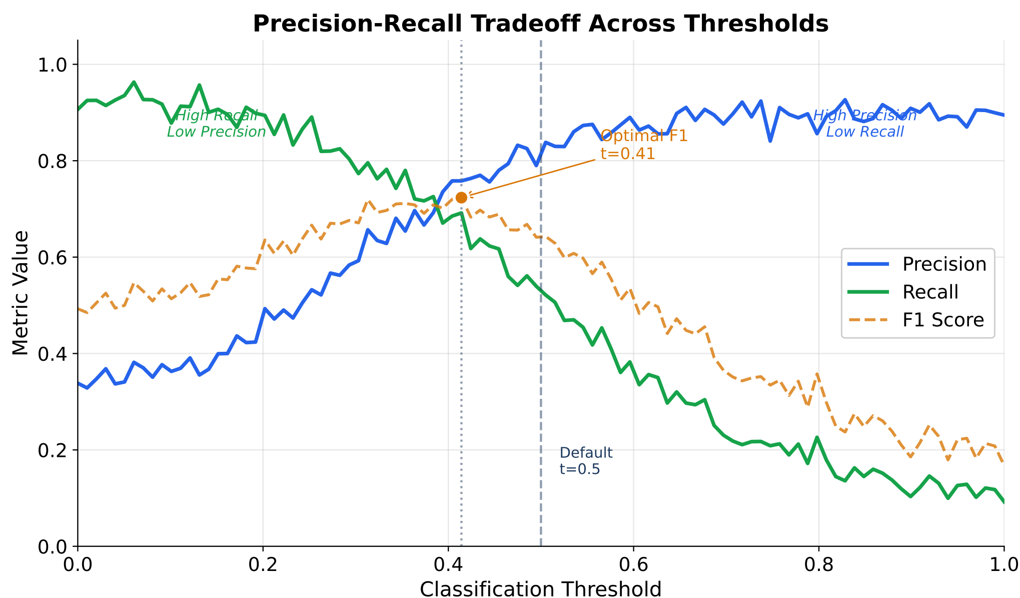

Figure 59: The precision–recall tradeoff: pushing recall higher (catching more fraud) inevitably lowers precision (more false alarms).

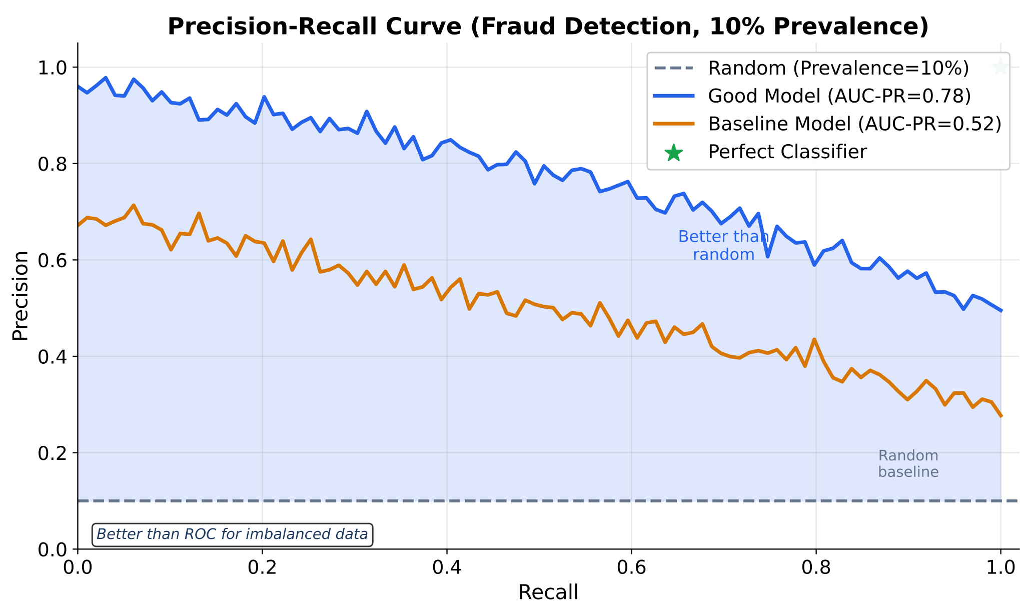

Figure 60: The precision–recall curve traces this tradeoff across all possible thresholds.

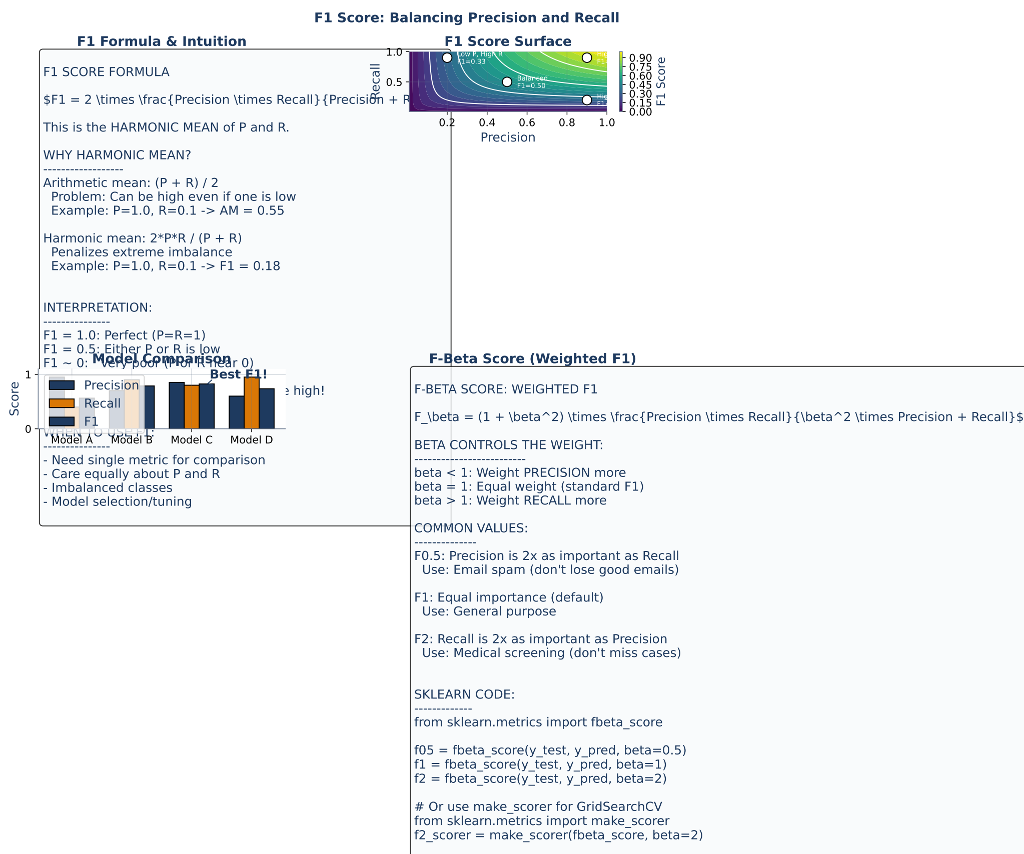

Figure 61: F1 score: the harmonic mean of precision and recall. It penalizes models that sacrifice one for the other.

Historical Background

Jerzy Neyman and Egon Pearson, 1933. Working at University College London, they formalized the hypothesis testing framework that introduced Type I errors (false positives) and Type II errors (false negatives). Their framework was controversial—Ronald Fisher himself opposed it bitterly. Fisher viewed significance testing as a measure of evidence, not a decision procedure. Neyman and Pearson disagreed: they framed testing as a *decision* between two hypotheses, with explicit error rates for each type of wrong answer.

The Neyman–Pearson lemma became the foundation of statistical decision theory. Every precision/recall calculation traces back to their Type I/II error framework. When you compute a confusion matrix, you are applying their 1933 idea to machine learning.

Misconception: High accuracy means the model is good

The accuracy trap. With 99% non-fraud transactions, predicting "no fraud" always gives 99% accuracy while catching nothing. Accuracy counts true negatives equally with true positives, flooding the numerator with easy-to-predict majority cases.

Misconception: F1 is always better than accuracy

F1 ignores true negatives entirely. For perfectly balanced datasets (50/50 split), accuracy is a fine metric. F1 shines specifically when classes are imbalanced and you care about the minority class. Use the right tool for the situation.

Misconception: Precision and recall can both be maximized

They trade off. Catch more positives (higher recall) and you inevitably flag more false alarms (lower precision). The only way to have both at 1.0 is a perfect classifier—which does not exist in practice.

Misconception: The confusion matrix changes only when you change the model

It also changes when you change the *threshold* on the same model. A logistic regression outputs probabilities. At threshold 0.5, you get one confusion matrix. At threshold 0.3, you get a different one—higher recall, lower precision. The model did not change. The decision boundary did. Thresholds are the subject of Section 7.

(c) Joerg Osterrieder 2026

6.3 Worked Example: Fraud Detection Metrics

Computing All Four Metrics

A fraud detection model produces the following confusion matrix on 1,000 test transactions:

	Predicted Fraud	Predicted Legitimate
Actually Fraud	TP = 45	FN = 10
Actually Legitimate	FP = 5	TN = 940

Precision: $\frac{45}{45+5} = \frac{45}{50} = 0.90$. When the model flags fraud, it is correct 90% of the time.
Recall: $\frac{45}{45+10} = \frac{45}{55} = 0.818$. The model catches 82% of actual frauds. Ten slip through.
F1: $\frac{2\times0.90\times0.818}{0.90+0.818} = \frac{1.473}{1.718} = 0.857$.
Accuracy: $\frac{45+940}{1000} = 0.985 = 98.5\%$.
Which metric matters here? For fraud detection, recall is critical—every missed fraud costs real money. The 10 missed frauds (FN = 10) might represent hundreds of thousands of euros in losses. The 5 false alarms (FP = 5) merely trigger manual review.

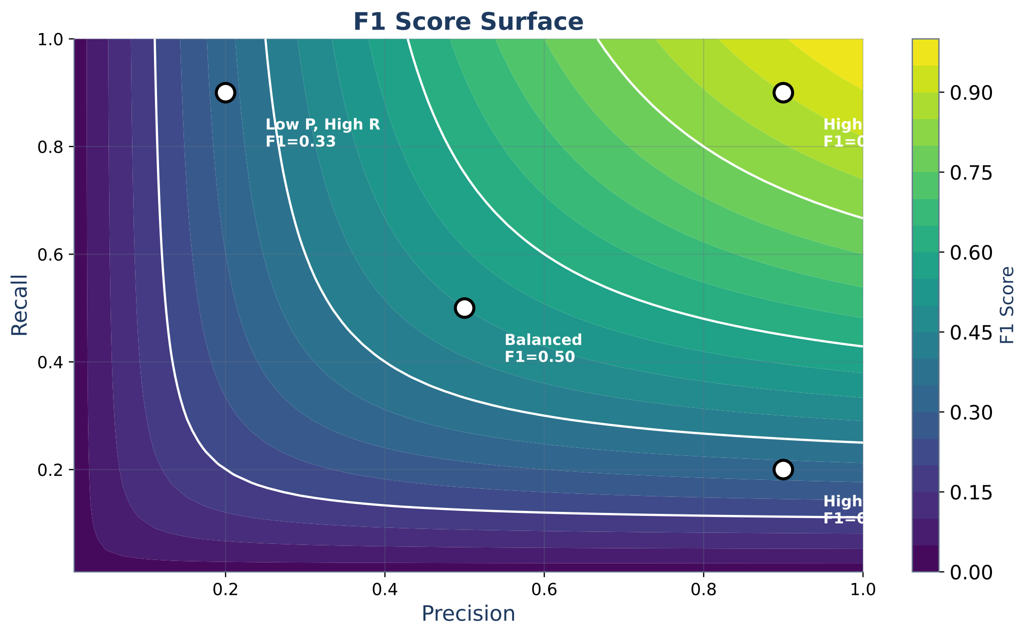

Figure 62: The F1 surface: how F1 changes as precision and recall vary. F1 is low whenever either component is low.

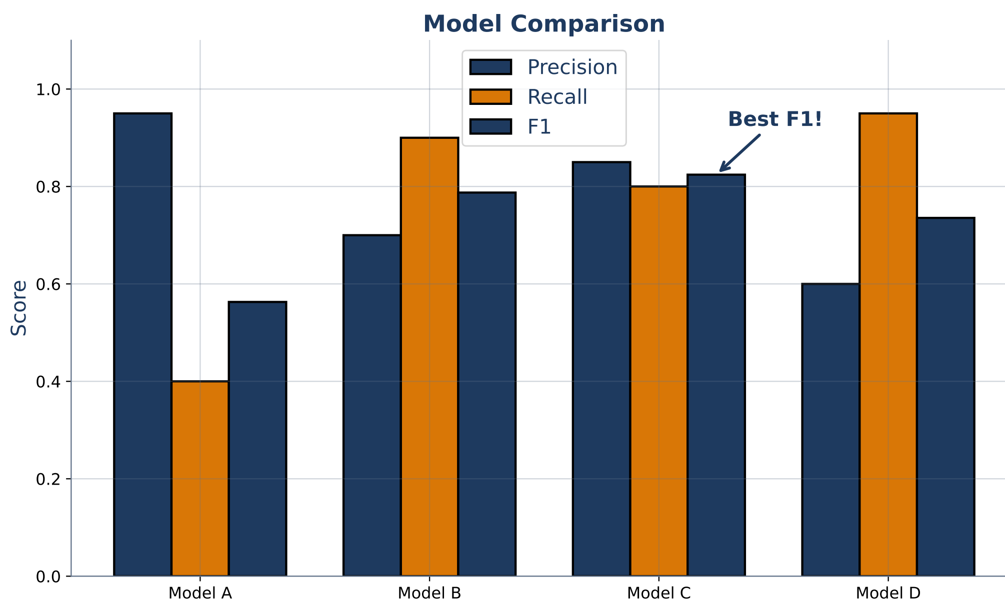

Figure 63: Comparing models on multiple metrics. Model A has higher accuracy; Model B has higher F1. Which is "better" depends entirely on the application.

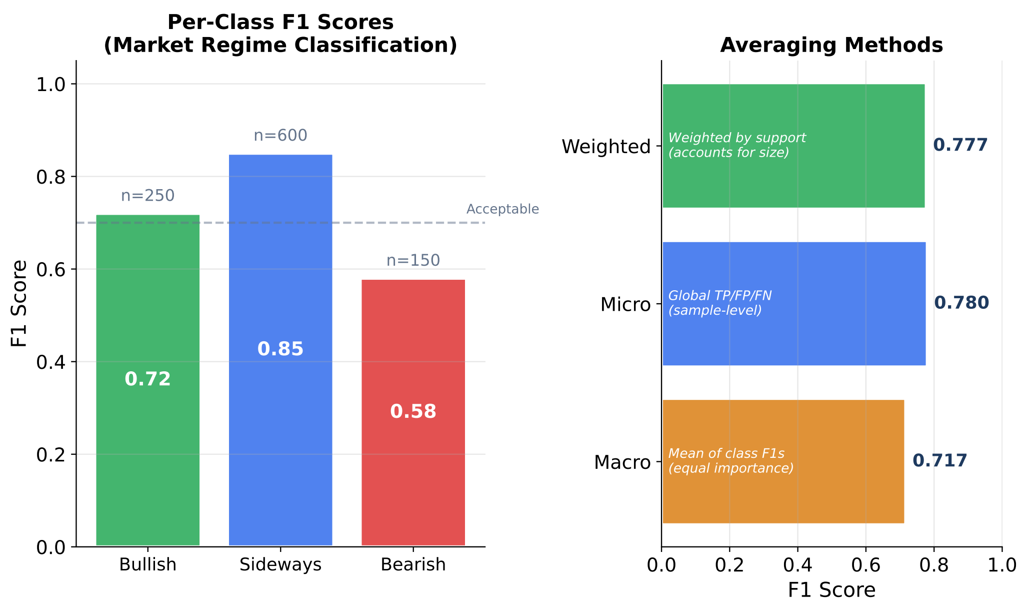

Figure 64: Multiclass extension: precision, recall, and F1 computed per class, then aggregated via macro or weighted averaging.

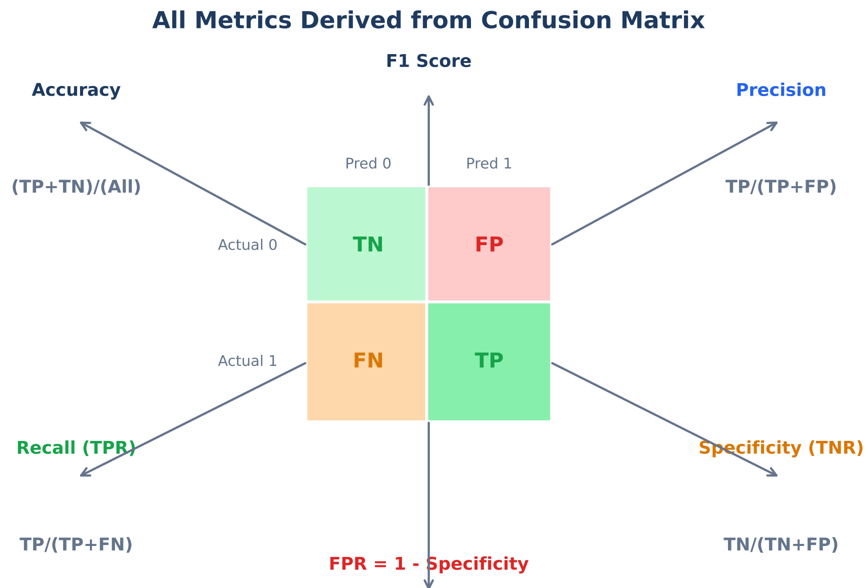

Figure 65: Overview of classification metrics and their relationships.

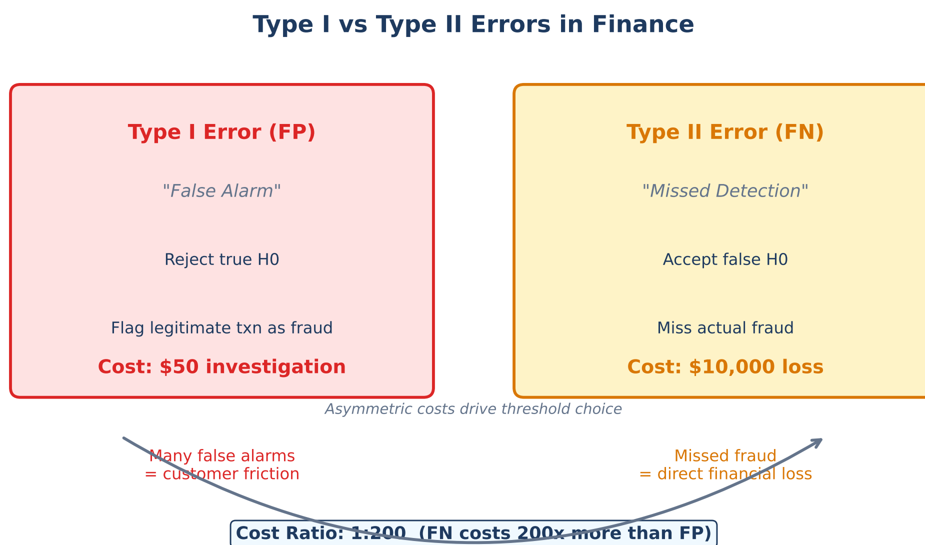

Figure 66: Type I and Type II errors. The relative cost of each determines which metric you optimize.

(c) Joerg Osterrieder 2026

Problem 6.1 (Easy)

Given TP = 45, FP = 5, FN = 10, TN = 940, compute:

(a) Precision

(b) Recall

(c) F1 score

(d) Accuracy

Yes, this repeats the worked example. Do it yourself without looking back. The formulas must become automatic.

Problem 6.2 (Easy)

In one paragraph, explain why 99% accuracy is terrible for a fraud detection model when the fraud rate is 0.5%. What specific metric would reveal the model's failure?

Problem 6.3 (Medium)

A medical screening model produces: TP = 80, FP = 20, FN = 30, TN = 870.

(a) Compute precision, recall, F1, and accuracy.

(b) You can reduce either FP or FN by 10 (not both). Which do you choose for a cancer screening application? Justify by computing the new recall in each case.

Problem 6.4 (Medium)

For each scenario, argue whether **precision** or **recall** matters more. Explain the cost of being wrong in each direction (FP vs. FN).

(a) Spam filter for your personal email

(b) Cancer screening test (initial, not diagnostic)

(c) Credit card fraud alert system

(d) Automated resume screening for job applications

(c) Joerg Osterrieder 2026

Problem 6.5 (Hard)

Design a metric selection strategy for a loan approval system. The bank faces two pressures:

- A regulator (GDPR Article 22) demands explanations for every denial.
- The credit department wants to minimize default losses.

(a) What metric do you report to the regulator? Why?

(b) What metric do you optimize internally? Why might it differ from (a)?

(c) How do you reconcile a model that optimizes for recall (catching defaults) with a regulatory demand for explainability (favoring simpler, more precise models)?

Connecting Forward

Precision and recall are snapshots. They describe model performance at *one specific threshold*—the cutoff above which the model predicts "positive." But what if you want to see performance across *all possible thresholds*? What if you want to compare two models regardless of threshold choice?

The ROC curve sweeps across every threshold from 0 to 1 and plots the tradeoff between true positive rate and false positive rate. Its area—the AUC—provides a single number summarizing discriminative ability across the entire range. That is Section 7.

Key Takeaway: Accuracy lies when classes are imbalanced—precision and recall reveal what accuracy hides.

7 The Whole Picture – ROC Curves, AUC, and Threshold Tuning

Opening Problem

Two teams at a bank build fraud detection models. Both use the same data. Both present to the model validation committee.
Team A: "Our model achieves AUC = 0.92."
Team B: "Our model achieves AUC = 0.88."
The committee picks Team A. Higher AUC, better model. Obvious.
Three months later, Team B's model—the one rejected—catches more fraud in production at another bank. The committee is baffled.
What happened? Team B tuned their threshold to the bank's cost structure: €50 per investigation (false positive) vs. €5,000 per missed fraud (false negative). Team A deployed with the default threshold of 0.5, which was nowhere near optimal for this cost ratio. AUC told part of the story. The threshold told the rest. Nobody asked about the threshold.

Discovery Question

Can a model with *lower* AUC actually outperform a model with higher AUC at a specific operating threshold? Under what conditions would you prefer the "worse" model?

7.1 The Speedometer Analogy

A car speedometer shows capability: 0 to 250 km/h. AUC is like that—it summarizes performance across every possible operating point.

But you do not drive at every speed simultaneously. In a 30 km/h school zone, you care about low-speed control. On the Autobahn, you care about stability at 180 km/h. The speedometer's full range tells you the car's potential, but your driving conditions determine what matters.

AUC summarizes the full ROC curve. Your bank operates at one threshold. The left half of the curve (low FPR, low TPR) might be irrelevant. The right half too. Only a narrow band around your operating point determines production performance.

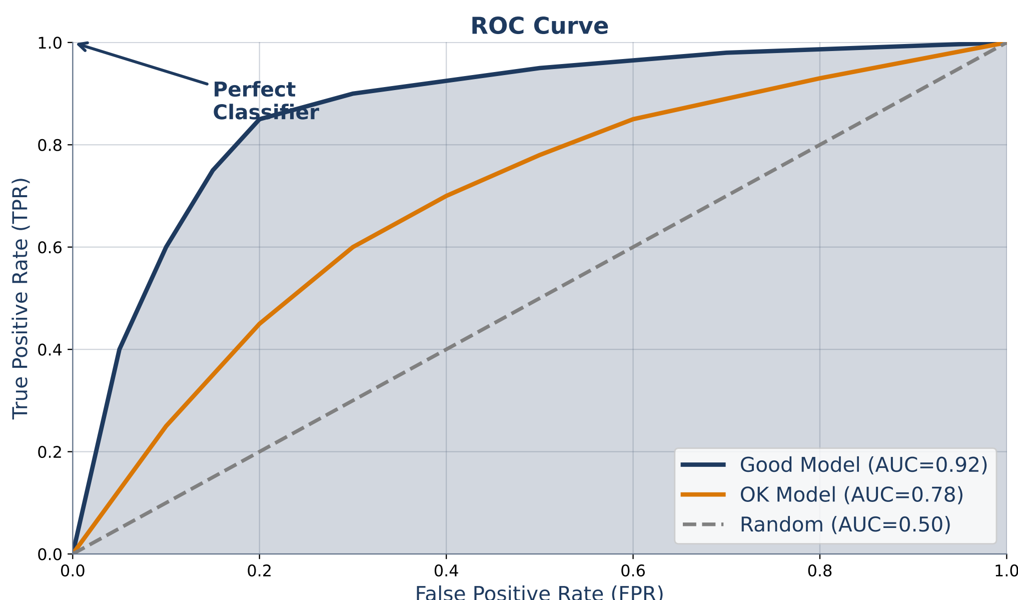

Figure 67: The ROC curve: each point represents a different threshold. Moving right increases both the true positive rate and the false positive rate.

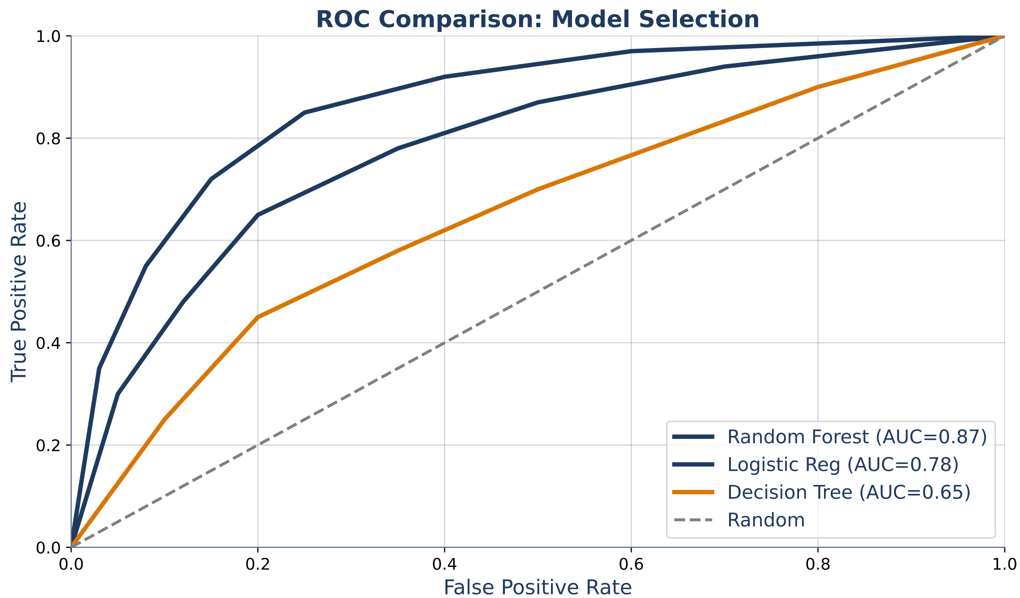

Figure 68: ROC comparison: Model A dominates Model B across most thresholds. But at the specific operating region the bank cares about, the gap may narrow or even reverse.

7.2 TPR, FPR, and the ROC Curve

True Positive Rate (TPR): Same as recall: $\text{TPR} = TP/(TP+FN)$. The fraction of actual positives correctly identified. Also called sensitivity.

False Positive Rate (FPR): $\text{FPR} = FP/(FP+TN)$. The fraction of actual negatives incorrectly flagged as positive. The "false alarm rate."

ROC curve: Receiver Operating Characteristic curve. Plots TPR (y-axis) vs. FPR (x-axis) at every threshold from 0 to 1. A perfect model hugs the top-left corner. A random model follows the diagonal.

AUC: Area Under the ROC Curve. Ranges from 0 to 1. Interpretation: the probability that a randomly chosen positive sample receives a higher predicted score than a randomly chosen negative sample.

AUC Interpretation

$$\text{AUC} = P\big(\hat{p}(\mathbf{x}^+) > \hat{p}(\mathbf{x}^-)\big)$$

where $\mathbf{x}^+$ is a random positive sample and $\mathbf{x}^-$ is a random negative sample.

Scale:

- $\text{AUC} = 0.5$: random guessing (the diagonal line)
- $\text{AUC} \in [0.7, 0.8)$: acceptable discrimination
- $\text{AUC} \in [0.8, 0.9)$: good discrimination
- $\text{AUC} \in [0.9, 1.0)$: excellent discrimination
- $\text{AUC} = 1.0$: perfect separation (never happens with real data)

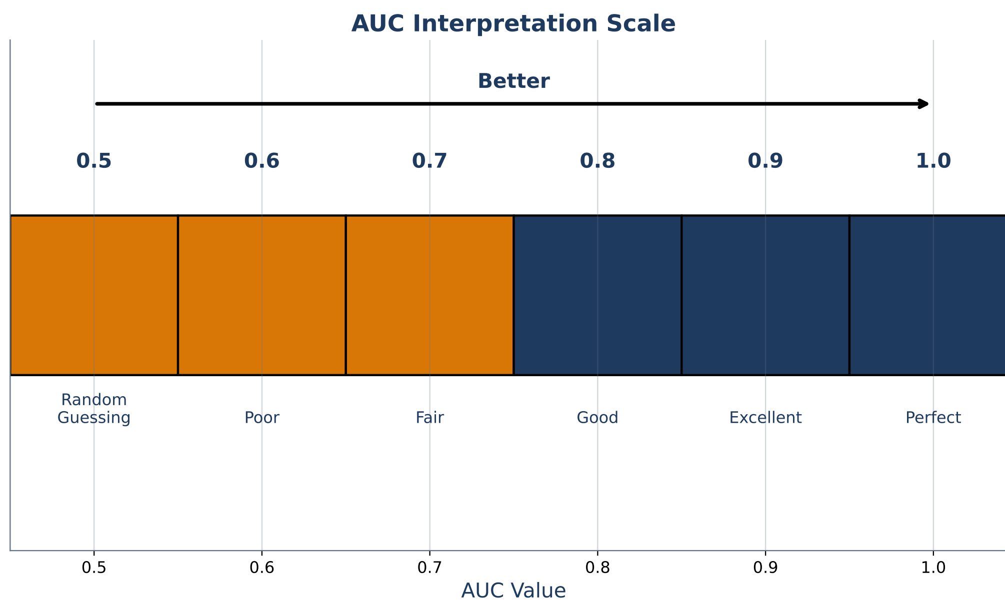

Figure 69: The AUC scale: from random (0.5) to perfect (1.0). Most production models in finance land between 0.7 and 0.9.

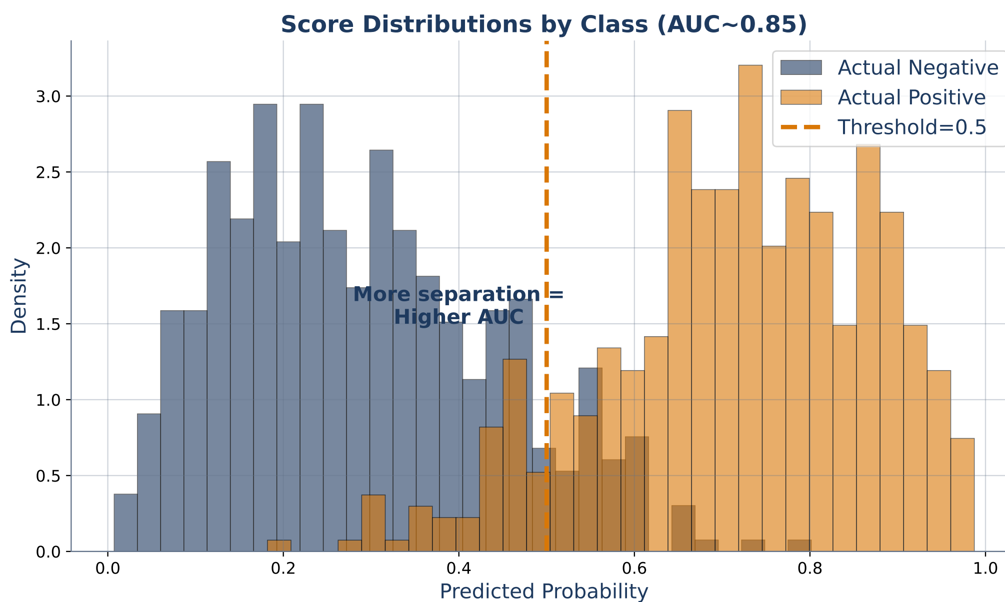

Figure 70: Score distributions for positives and negatives. Heavy overlap means low AUC. Clean separation means high AUC.

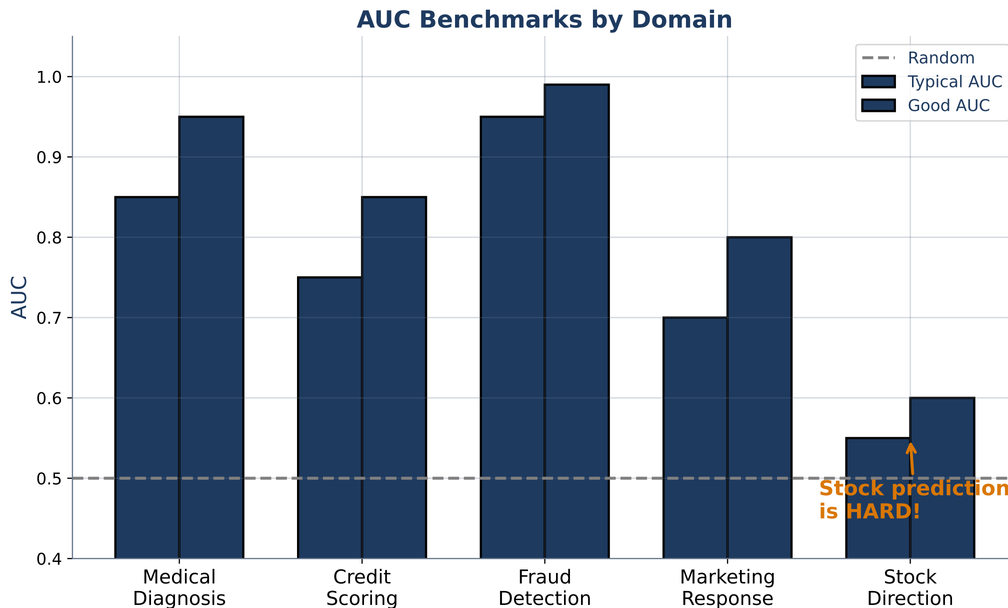

Figure 71: AUC benchmarks across different models and applications.

Historical Background

Tom Fawcett, 2006. Published "An Introduction to ROC Analysis" in *Pattern Recognition Letters*—the definitive tutorial that standardized how machine learning reports and interprets ROC curves. The paper has over 30,000 citations.
But the ROC concept is much older. It originated in World War II radar signal detection, around 1941. Radar operators had to distinguish enemy aircraft (true positives) from noise (false positives). Different receiver settings produced different tradeoffs: more sensitive settings caught more planes but also triggered more false alarms. The name "Receiver Operating Characteristic" comes directly from those radar receiver settings.
For decades, ROC analysis lived in signal processing and medical diagnostics. Fawcett brought it into mainstream machine learning with clear guidelines for plotting, interpreting, and comparing curves. His tutorial made AUC the default model comparison metric.

Misconception: AUC of 0.9 means 90% accuracy

AUC and accuracy measure completely different things. AUC = 0.9 means a randomly chosen positive scores higher than a randomly chosen negative 90% of the time. It says nothing about accuracy at any particular threshold. A model with AUC = 0.9 could have accuracy of 85%, 95%, or 50%—depending on the threshold and class balance.

Misconception: ROC curves are always better than PR curves

For heavily imbalanced data, ROC can paint an overly rosy picture. The FPR denominator includes all true negatives. When negatives dominate (say 9,900 out of 10,000), even 100 false positives give FPR = 0.01—tiny. The ROC curve barely notices. Precision, however, would be TP/(TP+100), which drops fast. PR curves are more informative for imbalanced data. We will see this directly in Section 8.

Misconception: The default threshold of 0.5 is optimal

Almost never. Threshold 0.5 makes sense only when: (a) classes are balanced, (b) false positives and false negatives have equal costs, and (c) the model is well-calibrated. In fraud detection, where cost(FN) is 100× cost(FP), the optimal threshold is much lower—perhaps 0.1 or 0.2.

Misconception: Higher AUC always means the better model

AUC summarizes *all* thresholds. If you operate at one specific threshold, a model with lower AUC might outperform there. Two ROC curves can cross: Model A wins at low FPR, Model B wins at high FPR. Their AUCs might be similar, but the right choice depends on where you operate.

7.3 Threshold Tuning: Where Theory Meets Business

The ROC curve shows all possible operating points. Which one do you pick? That depends on the cost of errors.

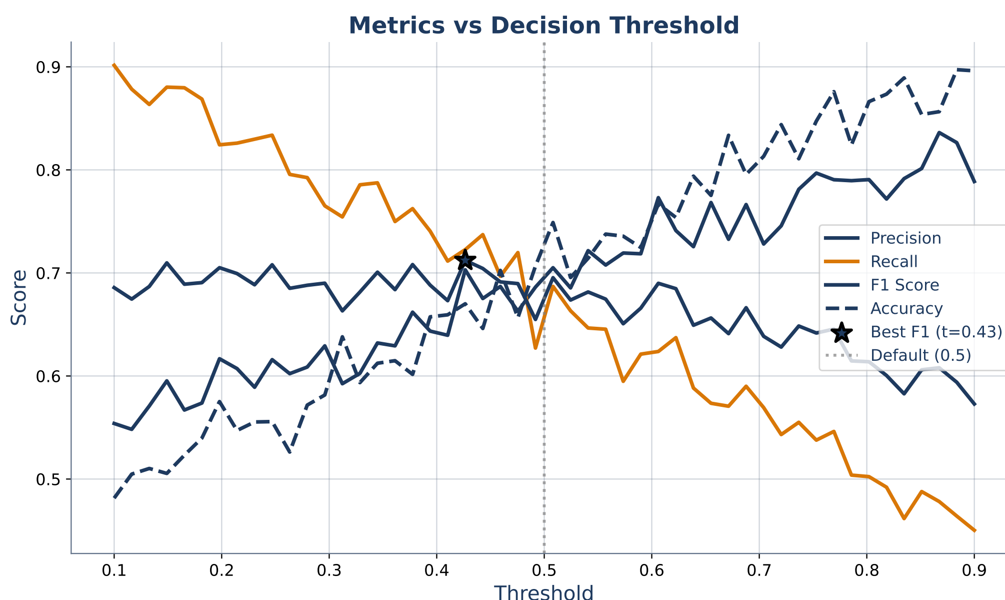

Figure 72: How precision, recall, and F1 change as the threshold sweeps from 0 to 1. There is no single "best" threshold—it depends on what you optimize.

(c) Joerg Osterrieder 2026

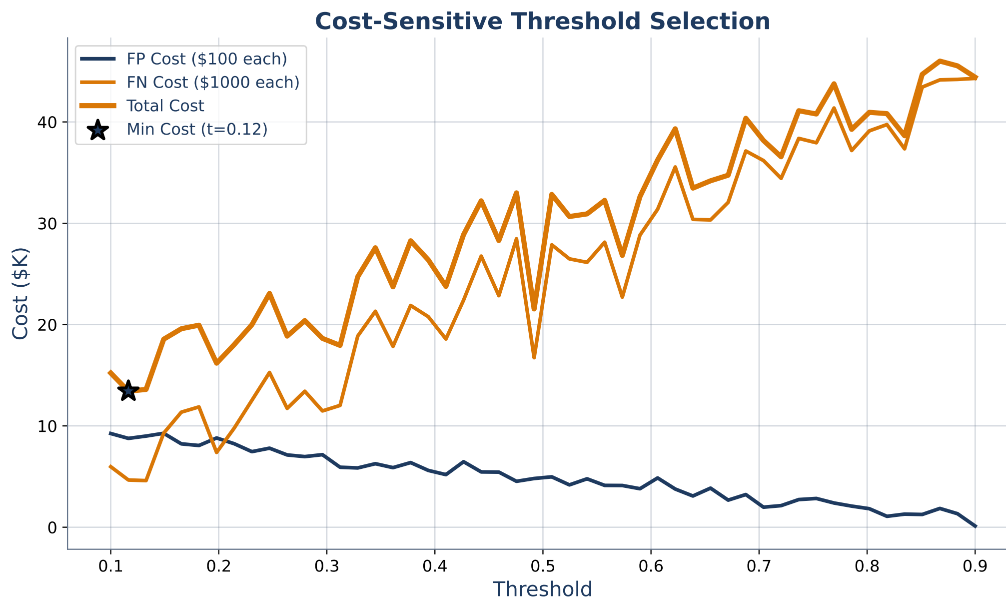

Figure 73: Cost-based threshold selection: the expected cost curve has a minimum. That minimum is your optimal operating point.

Finding the Cost-Optimal Threshold

A fraud detection system processes 10,000 transactions daily. Fifty are fraudulent.
Costs:

- False positive: €50 per investigation
- False negative: €5,000 per missed fraud

At threshold t, let $\text{FPR}(t)$ be the false positive rate and $\text{FNR}(t) = 1 - \text{TPR}(t)$ the false negative rate.
Expected daily cost:

$$C(t) = \text{FPR}(t) \times 9{,}950 \times 50 + \text{FNR}(t) \times 50 \times 5{,}000$$

At $t = 0.5$: $\text{FPR} = 0.02$, $\text{TPR} = 0.70$.

$$C = 0.02 \times 9{,}950 \times 50 + 0.30 \times 50 \times 5{,}000 = 9{,}950 + 75{,}000 = €84{,}950$$

At $t = 0.2$: $\text{FPR} = 0.05$, $\text{TPR} = 0.92$.

$$C = 0.05 \times 9{,}950 \times 50 + 0.08 \times 50 \times 5{,}000 = 24{,}875 + 20{,}000 = €44{,}875$$

Lowering the threshold from 0.5 to 0.2 saves €40,075 per day—despite tripling false alarms. Investigation costs are cheap; missed fraud is expensive.

(c) Joerg Osterrieder 2026

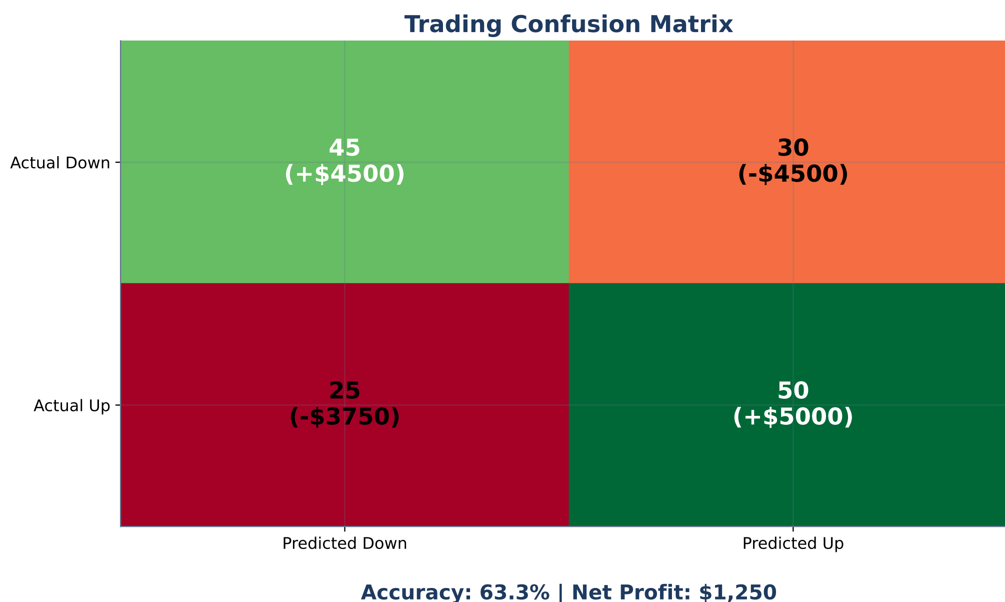

Figure 74: Trading signal confusion matrix: the four outcomes of a buy/sell classifier applied to daily stock returns.

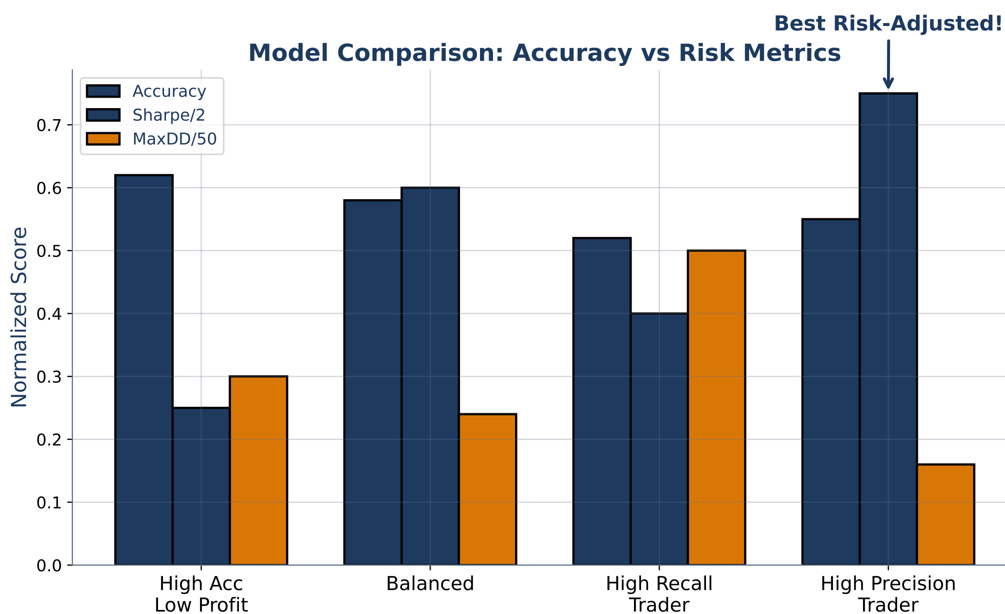

Figure 75: Strategy performance at different thresholds. Aggressive (low threshold) vs. conservative (high threshold) trading.

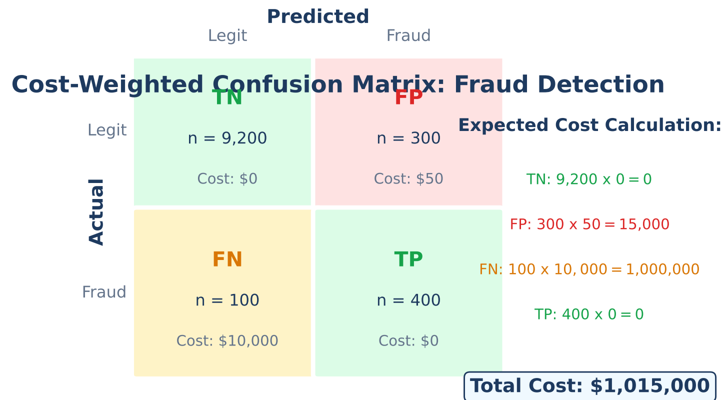

Figure 76: Fraud cost matrix: the asymmetry between investigation costs and fraud losses drives threshold selection far below 0.5.

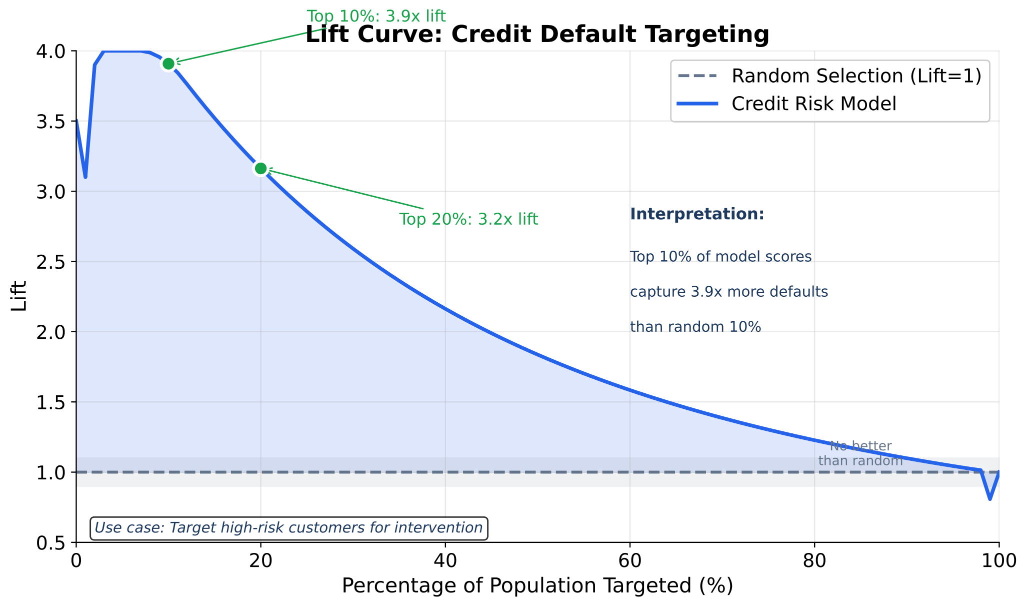

Figure 77: Lift curve: how much better is the model than random at each fraction of the population? Useful for prioritizing which cases to investigate first.

7.4 Finance Applications

Python: ROC Curve and Cost-Optimal Threshold

```python
from sklearn.metrics import roc_curve, roc_auc_score
import numpy as np

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
auc_score = roc_auc_score(y_test, y_proba)
print(f"AUC: {auc_score:.4f}")

# Cost-optimal threshold
cost_fp = 50      # investigation cost per false alarm
cost_fn = 5000    # fraud loss per missed fraud
n_pos = y_test.sum()
n_neg = len(y_test) - n_pos

# Expected cost at each threshold
costs = fpr * n_neg * cost_fp + (1 - tpr) * n_pos * cost_fn
optimal_idx = np.argmin(costs)
optimal_threshold = thresholds[optimal_idx]
print(f"Optimal threshold: {optimal_threshold:.3f}")
print(f"TPR={tpr[optimal_idx]:.3f}, FPR={fpr[optimal_idx]:.3f}")
```

Problem 7.1 (Easy)

Interpret each AUC value in plain English, using the “randomly chosen positive vs. randomly chosen negative” framing:

(a) $\text{AUC} = 0.5$

(b) $\text{AUC} = 0.85$

(c) $\text{AUC} = 1.0$

Problem 7.2 (Medium)

A ROC curve has a clear “elbow” at $\text{FPR} = 0.10$, $\text{TPR} = 0.85$. Explain why this point might make a good operating threshold. What does it mean to operate to the *left* of this point? To the *right*?

Problem 7.3 (Medium)

AUC is called “threshold-independent” because it summarizes performance across all thresholds. Is this always an advantage? Describe a concrete scenario where you would prefer a threshold-specific metric (e.g., precision at recall $= 0.9$) over AUC.

Problem 7.4 (Medium)

A fraud detection system handles 1,000 transactions, 5 of which are fraudulent. Costs: FP = €10 (investigation), FN = €100 (fraud loss). The model reports:

Threshold	Precision	Recall	Predicted Positives
0.3	0.05	1.00	100
0.4	0.08	0.80	50
0.5	0.15	0.60	20
0.6	0.25	0.40	8
0.7	0.50	0.20	2

At each threshold, compute: (a) TP and FP counts, (b) FN count, (c) cost from FP and cost from FN, (d) total expected cost. Which threshold minimizes total cost?
Hint: TP = Recall × 5 and FP = Predicted Positives − TP.

Problem 7.5 (Hard)

Consider a dataset with 1% positive class (100 positives, 9,900 negatives).

(a) A model achieves ROC-AUC = 0.95 but PR-AUC = 0.30. Explain how this is possible. What does each metric "see" that the other misses?

(b) Which metric is more honest for stakeholders who care about catching the positive class?

(c) Under what class distribution would ROC-AUC and PR-AUC agree closely?

Connecting Forward

In Sections 6 and 7 we assumed the test set reflects reality. We computed metrics on held-out data and trusted the numbers.

But what if 99.9% of your data is one class? Standard training fails. The optimizer finds the path of least resistance: predict the majority class always. The confusion matrix is nearly empty in the positive quadrant. Precision is undefined (0/0). Recall is zero.

The solutions—SMOTE, class weights, cost-sensitive learning—modify either the data or the loss function so that the model cannot ignore the minority class. That is Section 8.

Key Takeaway: AUC summarizes a model's overall discrimination—but the threshold you deploy determines the actual impact.

8 Finding Needles in Haystacks – Class Imbalance and Cost-Sensitive Learning

Opening Problem

A European bank processes 50,000 credit card transactions daily. Of these, about 50 are fraudulent—a 0.1% fraud rate. The data science team trains a gradient boosted classifier on six months of historical data.

The model predicts "legitimate" for every single transaction.

Accuracy: 99.9%. Fraud caught: zero. Revenue protected: nothing.

The model learned the laziest pattern: always predict the majority class. The loss function penalized all errors equally, and the class with 49,950 samples overwhelmed the class with 50 samples. From the model's perspective, ignoring fraud *minimized total error*. A perfectly rational but perfectly useless result.

This is the class imbalance problem. It afflicts nearly every real-world classification task in finance: fraud detection, credit default, money laundering, insider trading, market manipulation. The events you most need to catch are exactly the ones rarest in the data.

Discovery Question

If you duplicate all 50 fraud cases to create 50,000 fraud samples + 50,000 legitimate samples, the model now sees balanced classes. But does it actually learn *better*? Or does it just memorize the same 50 fraud patterns repeated 1,000 times each?

8.1 Needles, Haystacks, and Three Strategies

The metaphor is exact. You have a haystack of 50,000 pieces of straw and 50 needles hidden inside. Your detector was calibrated for haystacks with equal amounts of straw and needles. So it learned what straw looks like and classified everything as straw.

Three strategies for finding those needles:

1. **Add more needles** (oversampling): duplicate existing needles or synthesize new ones so the detector sees them more often.
2. **Remove straw** (undersampling): discard most straw so needles become a larger fraction.
3. **Make needles heavier** (class weights): keep the data unchanged, but tell the detector that missing a needle costs 1,000 times more than misclassifying straw.

Each strategy has tradeoffs. We examine all three.

8.2 Resampling Strategies

Class imbalance: When one class significantly outnumbers the other in the training data (e.g., 99:1 or 999:1 ratio). The majority class dominates the loss function, causing the model to ignore the minority class.

Random oversampling duplicates minority samples until classes are balanced. Simple and fast. The problem: the model sees the same fraud examples repeatedly. It does not learn new fraud patterns—it memorizes existing ones. This risks overfitting to the specific minority samples in the training set.

Random undersampling removes majority samples until classes are balanced. Also simple and fast. The problem: you throw away data. With 50 fraud and 50,000 legitimate transactions, undersampling to 50:50 means keeping only 50 legitimate transactions out of 50,000. You discard 99.9% of your negative examples.

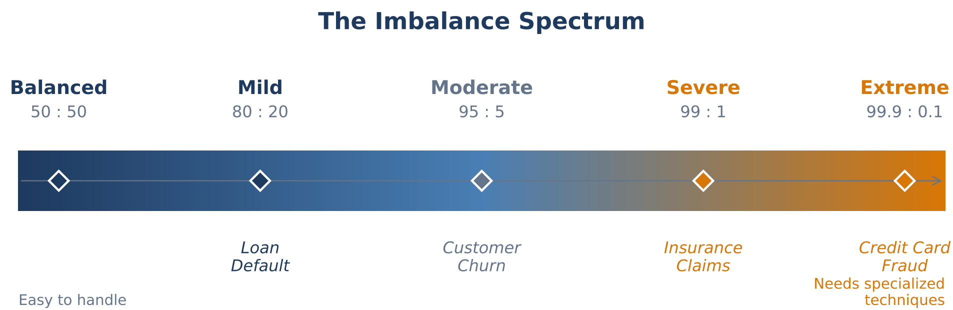

Figure 78: The imbalance spectrum: from mildly imbalanced (70:30) to severely imbalanced (99.9:0.1). Most real-world finance problems live at the severe end.

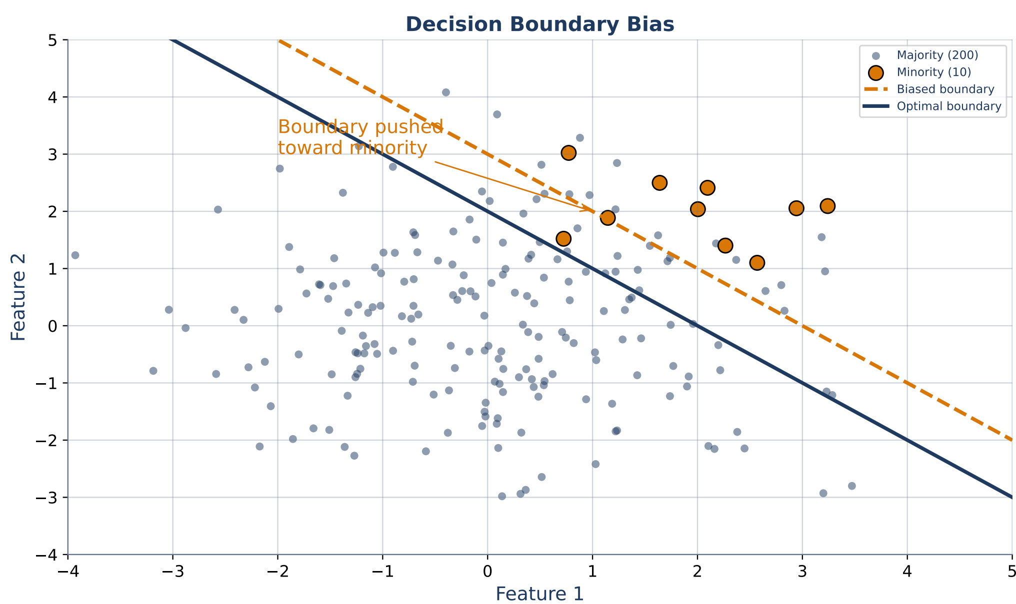

Figure 79: Decision boundary bias: when one class dominates, the boundary shifts toward the minority, effectively ignoring it.

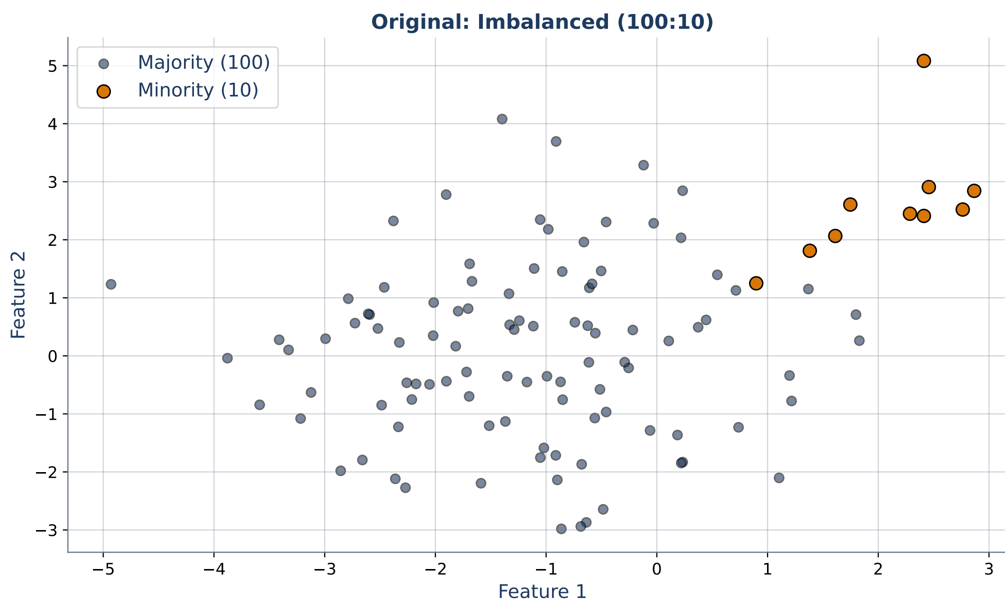

Figure 80: The original imbalanced dataset: 100 majority samples, 10 minority samples. The decision boundary is pulled toward the majority class.

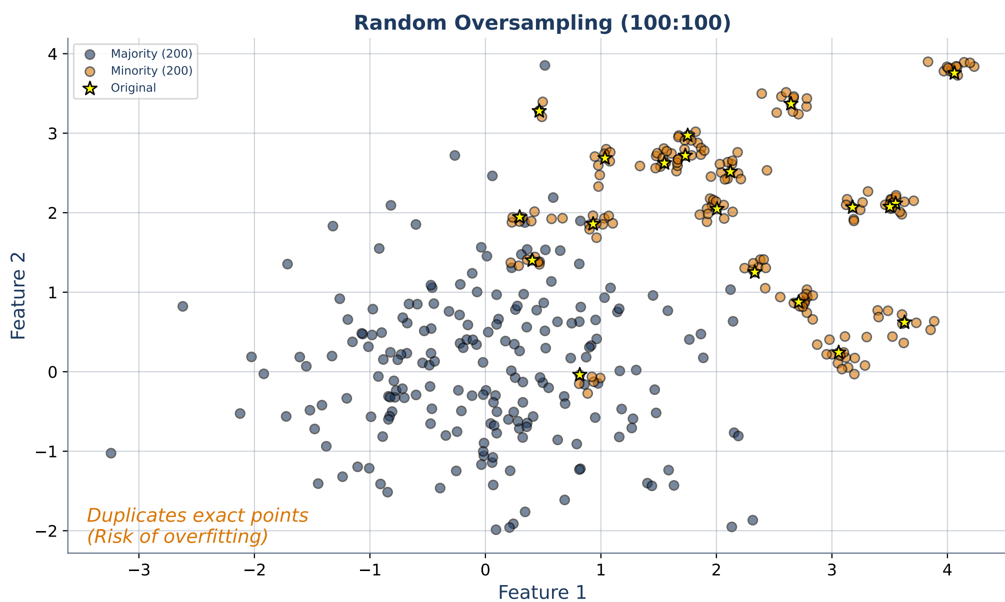

Figure 81: After random oversampling: the minority class is duplicated to match the majority. The same 10 points now appear multiple times—no new information.

SMOTE: Synthetic Minority Over-sampling Technique. Instead of duplicating existing minority samples, SMOTE creates *new* synthetic samples by interpolating between existing minority neighbors. This generates genuine new data points.

SMOTE Interpolation

To generate a synthetic sample $\mathbf{x}_{\text{new}}$:

1. Pick a minority sample $\mathbf{x}_i$.
2. Find its k nearest minority-class neighbors ($k = 5$ by default).
3. Randomly select one neighbor $\mathbf{x}_j$.
4. Generate: $\mathbf{x}_{\text{new}} = \mathbf{x}_i + \lambda \cdot (\mathbf{x}_j - \mathbf{x}_i)$, where $\lambda \sim \text{Uniform}(0, 1)$.

The new sample lies *between* two existing minority samples in feature space. SMOTE fills in the minority region rather than stacking copies on existing points.

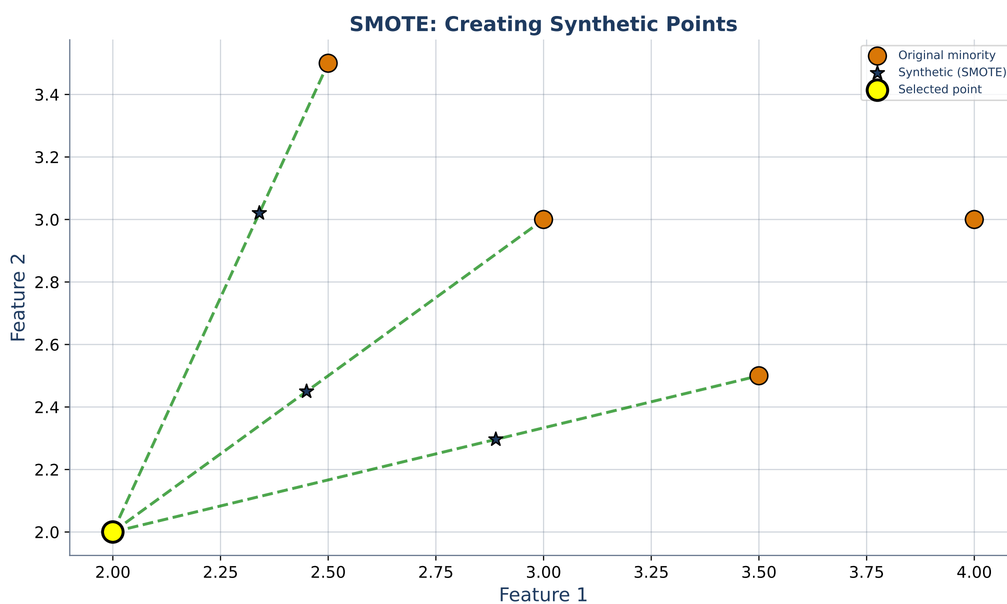

Figure 82: SMOTE in action: synthetic samples (stars) appear along line segments between existing minority neighbors, expanding the minority class's coverage of feature space.

8.3 Class Weights: Changing the Loss, Not the Data

Class weights: Instead of modifying the data, modify the loss function. Assign higher penalty to misclassifying minority samples. The model sees the original data but "feels" minority errors more acutely.

Weighted Loss and Balanced Weights

The weighted loss assigns class-specific multipliers:

$$\mathcal{L}_{\text{weighted}} = \sum_{i=1}^{n} w_{y_i} \cdot \mathcal{L}(y_i, \hat{y}_i)$$

With `class_weight='balanced'` in scikit-learn, the weight for class k is:

$$w_k = \frac{n}{K \cdot n_k}$$

where n = total samples, K = number of classes, n_k = count of class k.
Example: 9,500 legitimate + 500 fraud ($n = 10{,}000$, $K = 2$).

- $w_{\text{legit}} = \frac{10{,}000}{2\times 9{,}500} = 0.526$
- $w_{\text{fraud}} = \frac{10{,}000}{2\times 500} = 10.0$

Each fraud misclassification now hurts 19× more than a legitimate one. The model cannot afford to ignore fraud.

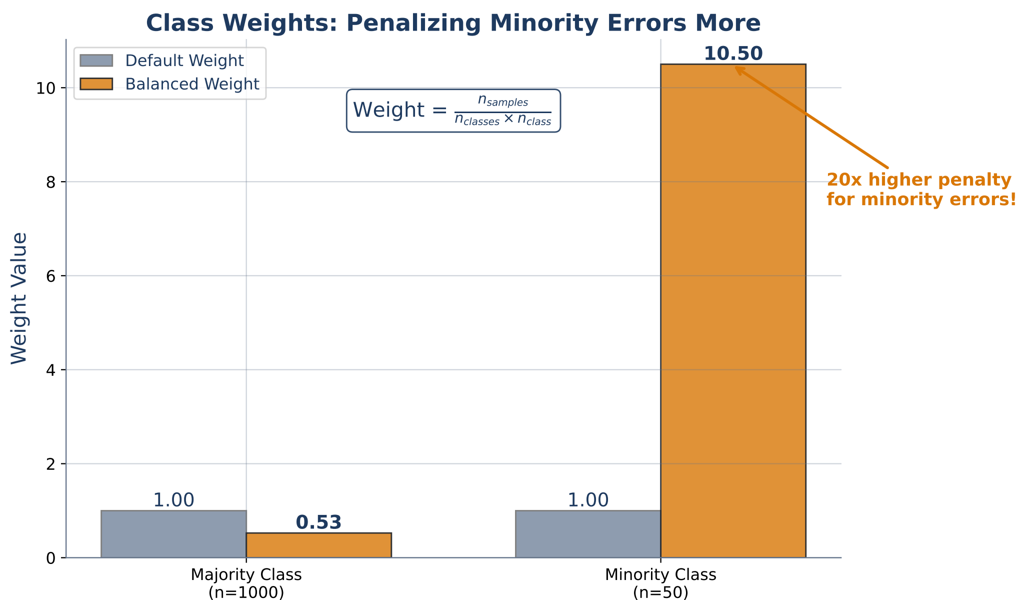

Figure 83: Class weights shift the decision boundary: minority errors are penalized more heavily, pulling the boundary away from the minority class.

8.4 Stratified Cross-Validation

Stratified K-Fold: Cross-validation that preserves class proportions in each fold. If the dataset has 1% fraud, each fold also has approximately 1% fraud. Without stratification, some folds might have zero fraud examples, making evaluation meaningless.

This is non-negotiable for imbalanced data. Standard k-fold with random shuffling can produce folds where the minority class is entirely absent.

Historical Background

Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and W. Philip Kegelmeyer, 2002. Published “SMOTE: Synthetic Minority Over-sampling Technique” in the *Journal of Artificial Intelligence Research*. The problem they solved had plagued fraud detection, medical diagnosis, and text classification for years: how do you train a model when positive examples are extremely rare?

The key insight was deceptively simple. Random oversampling adds no new information—it just stacks copies. SMOTE generates genuinely new samples by interpolating between neighbors. The synthetic points fill in the minority class’s feature space, giving the model a richer picture of what “positive” looks like.

The paper now has over 45,000 citations, making it one of the most impactful machine learning papers ever published. Variants like Borderline-SMOTE, ADASYN, and SMOTE-ENN followed, but the original algorithm remains the most widely used.

Misconception: Oversampling and SMOTE are the same

Random oversampling duplicates existing minority samples verbatim. Fifty fraud samples oversampled to 5,000 means each original appears 100 times. No new information. SMOTE creates *synthetic* samples by interpolating between neighbors—each one is a new point in feature space. SMOTE adds information; duplication does not.

Misconception: You should balance the test set too

Never. The test set must reflect real-world conditions. If fraud is 0.1% in production, your test set should have 0.1% fraud. Balancing the test set inflates performance metrics that collapse at deployment. Only modify the *training* set.

Misconception: Class weights and resampling produce identical results

Different mechanisms, different effects. Weights modify the loss function—the model sees original data but penalizes minority errors more. Resampling modifies the data itself—the model sees a different distribution. In practice they produce similar but not identical boundaries. Weights are simpler and avoid overfitting to duplicated samples. Resampling can work better when the minority class has complex structure that benefits from SMOTE’s interpolation.

8.5 Why PR Curves Matter More Under Imbalance

In Section 7 we noted that ROC curves can be optimistic for imbalanced data. Here is why, concretely.

The FPR denominator includes all true negatives. With 9,900 negatives, 100 false positives produce FPR $= 100/9{,}900 \approx 0.01$. The ROC curve barely registers. But precision = TP/(TP + 100). If TP is also 100, precision is 0.50. The PR curve catches this immediately.

For imbalanced data, **always report PR-AUC alongside ROC-AUC**. The gap between them reveals how much the class imbalance is flattering the ROC metric.

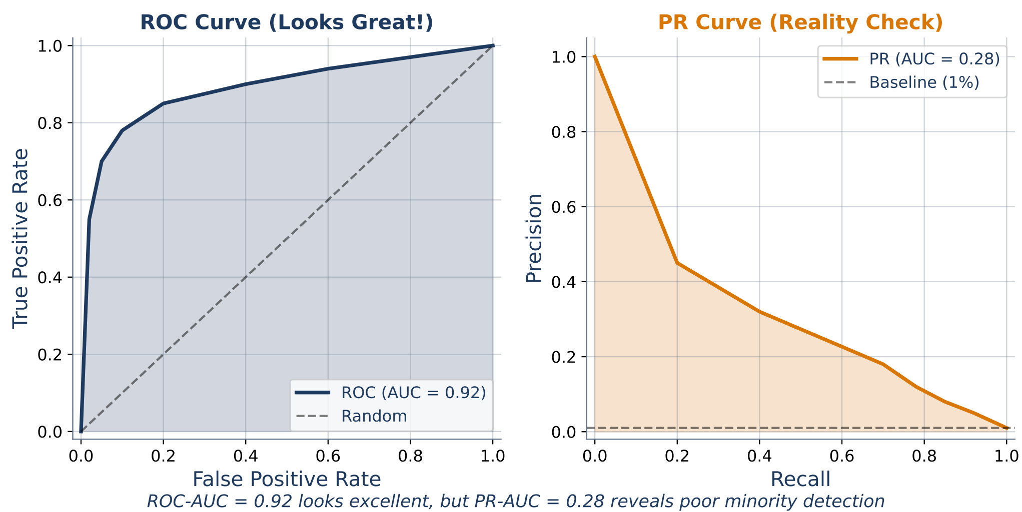

Figure 84: PR vs. ROC for the same model on imbalanced data. The ROC curve looks excellent (AUC near 0.95). The PR curve tells a different story—precision drops sharply as recall rises. PR-AUC is much lower than ROC-AUC.

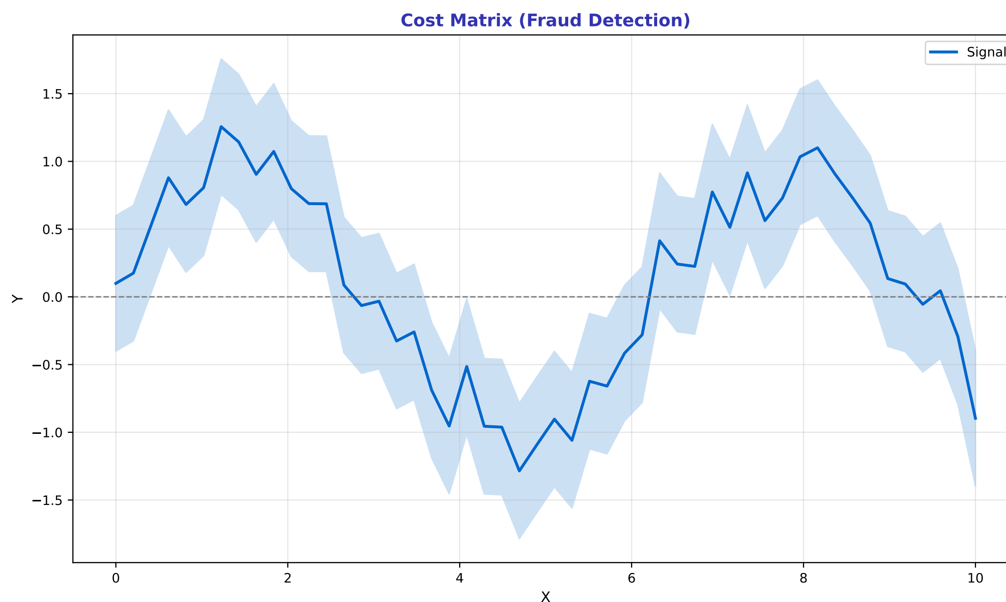

Figure 85: Fraud cost matrix: the asymmetry between investigation cost (€10) and fraud loss (€5,000) drives threshold selection far below 0.5.

8.6 Cost-Sensitive Fraud Detection

Full Pipeline: Fraud Detection with SMOTE and Threshold Tuning

Dataset: 10,000 transactions, 10 fraudulent (0.1% positive rate).
Step 1 — Split. Stratified train/test split preserving 0.1% fraud in both. Result: 8,000 train (8 fraud), 2,000 test (2 fraud).
Step 2 — SMOTE (training only). Synthesize fraud samples until training is 50:50. Result: 7,992 legitimate + 7,992 synthetic fraud. Test set is untouched.
Step 3 — Train with class weights as backup.

```
1 model = LogisticRegression(class_weight='balanced', max_iter=1000)
2 model.fit(X_train_smote, y_train_smote)
```

Step 4 — Tune threshold. Using cost(FP) = €10, cost(FN) = €5,000, sweep thresholds from 0.05 to 0.95 and pick the one minimizing expected cost on a validation set.
Step 5 — Evaluate on original test set. Report PR-AUC, ROC-AUC, precision, recall, F1 at the chosen threshold. Do *not* report accuracy—it is meaningless at 0.1% positive rate.

Python: SMOTE + Class Weights + Stratified CV

```
1 from imblearn.over_sampling import SMOTE
2 from sklearn.model_selection import StratifiedKFold
3 from sklearn.linear_model import LogisticRegression
4
5 # SMOTE: create synthetic minority samples
6 smote = SMOTE(random_state=42)
7 X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
8
9 # Alternative: class weights (no resampling needed)
10 model = LogisticRegression(class_weight='balanced', max_iter
      =1000)
11 model.fit(X_train, y_train)
12
13 # Always use StratifiedKFold for imbalanced data
14 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
15 for train_idx, val_idx in cv.split(X_train, y_train):
16     X_fold_train = X_train.iloc[train_idx]
17     y_fold_train = y_train.iloc[train_idx]
18     # Apply SMOTE inside the fold, never on full data
19     X_smote, y_smote = smote.fit_resample(
20         X_fold_train, y_fold_train
21     )
```

Problem 8.1 (Easy)

A dataset has 9,500 legitimate and 500 fraudulent transactions.

(a) What is the imbalance ratio (majority : minority)?

(b) If a model predicts "legitimate" for every transaction, what is its accuracy?

(c) What is its recall for the fraud class?

Problem 8.2 (Medium)

Describe step by step how SMOTE generates **one** synthetic fraud sample. Use this concrete example with 2 features:

- Existing fraud sample A: amount = €450, time = 02:30
- Nearest fraud neighbor B: amount = €520, time = 03:15
- Random $\lambda = 0.4$

Compute the synthetic sample's feature values and explain what λ controls.

Problem 8.3 (Medium)

For a 99:1 imbalanced dataset with 10,000 samples (9,900 majority, 100 minority), compare three strategies:

(a) Random oversampling to 50:50 (duplicate minority to 9,900)

(b) Random undersampling to 50:50 (reduce majority to 100)

(c) `class_weight='balanced'` (no data change)

For each: What is the resulting training set size? What are the main risks? Which preserves the most information?

Problem 8.4 (Medium)

A fraud detection system processes 10,000 transactions with 50 frauds. Costs: FP = €10 (investigation), FN = €5,000 (fraud loss). The model reports:

Threshold	Precision	Recall
0.3	0.15	0.90
0.5	0.45	0.60
0.7	0.80	0.30

For each threshold, compute:

(a) TP count (= Recall × 50) and FP count (= TP/Precision − TP)

(b) FN count (= 50 − TP) and its cost

(c) Total expected cost (FP × €10 + FN × €5,000)

Which threshold minimizes cost?

(c) Joerg Osterrieder 2026

Problem 8.5 (Hard)

Design a complete pipeline for credit card fraud detection at a bank processing 100,000 daily transactions with a 0.05% fraud rate. Address each component:

(a) **Data splitting:** Why is random train/test splitting dangerous for fraud data? Propose a better approach.

(b) **Resampling:** Choose between SMOTE, random oversampling, and class weights. Justify.

(c) **Model:** Logistic regression, random forest, or gradient boosting? Why?

(d) **Metrics:** Which primary metric? Which secondary?

(e) **Threshold:** How do you select it? What cost information do you need from the business?

(f) **Monitoring:** The fraud rate shifts from 0.05% to 0.15% over three months. How do you detect this drift and when do you retrain?

Looking Back

This handout started with a loan officer who could not be consistent (Section 1). We gave her the sigmoid and logistic regression (Section 2). We taught her to explain decisions through coefficients and odds ratios (Section 3). We offered decision trees that split the problem into interpretable yes/no questions (Section 4). We built forests that resist the instability of individual trees (Section 5). We learned that accuracy lies when classes are imbalanced—precision and recall reveal the truth (Section 6). We discovered that the threshold you deploy matters more than the model you choose (Section 7). And here we confronted the uncomfortable reality that the rare events finance cares about most—fraud, default, market manipulation—are exactly the ones standard machine learning ignores.

The classification toolkit is now in your hands: logistic regression for interpretability, trees and forests for flexibility, metrics for honesty, and resampling for fairness to rare events. Every real-world deployment uses some combination of these ideas.

Key Takeaway: Class imbalance is the norm in finance—the rare events (fraud, default, anomalies) are exactly the ones you cannot afford to miss.

Solutions to Practice Problems

Section 1: When Numbers Become Decisions

Solution 1.1:

(a) **Predicting tomorrow's stock price** — *Regression.* The target is a continuous number (e.g., $142.37). There is no finite set of categories.

(b) **Detecting spam emails** — *Classification (binary).* The target is one of two categories: spam or not-spam.

(c) **Estimating a house's market value** — *Regression.* The target is a continuous dollar amount.

(d) **Flagging fraudulent credit card transactions** — *Classification (binary).* Each transaction is either fraudulent or legitimate.

(e) **Forecasting a customer's lifetime spending** — *Regression.* The target is a continuous monetary value accumulated over time.

Solution 1.2:

When you fit a straight line $\hat{y} = \beta_0 + \beta_1 x$ to binary data ($y \in \{0, 1\}$), the points cluster at the top ($y = 1$) and bottom ($y = 0$) of the plot. The best-fit line cuts through the cloud at an angle. The problem becomes visible at the extremes: for very small x, the line predicts values below 0 (e.g., $\hat{y} = -0.3$), and for very large x, it predicts values above 1 (e.g., $\hat{y} = 1.4$). Neither value makes sense as a probability. You cannot be "negative 30% likely" to default, nor "140% likely." Furthermore, around the decision boundary the line gives values like 0.49 and 0.51, which are treated as opposite decisions despite being nearly identical scores. The sigmoid function solves this by squashing all outputs into the interval $(0, 1)$.

Solution 1.3:

(a) **Features:** `annual_income`, `employment_years`, `num_credit_cards`, `loan_amount`. **Target:** `defaulted` (0/1).

(b) **Binary classification.** The target takes exactly two values: 0 (no default) and 1 (default).

(c) The feature most likely to have the strongest association with default is `annual_income` (or `loan_amount`). Income directly determines a borrower's capacity to service debt. Higher income means more disposable cash flow to make payments. Of course, the *ratio* of loan amount to income (a derived feature) would be even more predictive, but among the raw features listed, income is the most fundamental driver of repayment ability.

Solution 1.4:

(a) `predict()` for a `LinearRegression` model returns a continuous real number for each sample—the predicted value of y. For example, $[0.73, -0.15, 1.42]$.

(b) `predict_proba()` for a `LogisticRegression` model returns an array of shape $(n, 2)$ where each row contains $[P(\text{class} = 0), P(\text{class} = 1)]$. For example, $[[0.27, 0.73], [0.85, 0.15]]$. Both columns sum to 1 for each row.

(c) The classifier outputs probabilities because the *decision threshold* should be chosen by the user, not hard-coded into the model. A probability of 0.6 might justify different actions in different contexts: approve a credit card but deny a mortgage. By returning probabilities, the model separates estimation ("how likely is default?") from the decision ("should we approve?").

Solution 1.5:

(a) **Features to engineer:**

- *Volume anomaly:* Ratio of current trading volume to the 30-day average for the same security.
- *Timing features:* Number of days between the trade and the next material corporate announcement (earnings, M&A, regulatory filings).
- *Profit features:* Abnormal return earned within a short window (e.g., 5 days) after the trade.
- *Network features:* Whether the trader has professional or personal connections to company insiders (shared board memberships, family ties, prior employer overlap).
- *Pattern features:* Frequency of trades in the same security before prior announcements; whether the trader has a history of well-timed trades across multiple companies.

(b) **Why this is harder than credit default:** (1) Ground truth is scarce—insider trading is illegal, rarely prosecuted, and confirmed cases are few, so labeled positive examples are extremely rare. (2) The signal is deliberately hidden; insiders use intermediaries, spread trades across accounts, and time their activity to avoid detection. (3) The feature space is relational (networks of people), not tabular like credit data. (4) Normal trading can look identical to insider trading (a lucky bet), making the false positive cost high.

(c) **Class balance:** The vast majority of trades are legitimate. Insider trading might represent fewer than 0.01% of all trades. This extreme imbalance means a model predicting "legitimate" for every trade achieves >99.99% accuracy while catching zero insiders. We need precision/recall-based metrics and resampling strategies (Section 8).

Section 2: The Sigmoid and Logistic Regression

Solution 2.1:

The sigmoid function is $\sigma(z) = \dfrac{1}{1+e^{-z}}$.

For $z = -2$:

$$e^{-(-2)} = e^2 = 7.3891, \quad 1 + e^2 = 8.3891, \quad \sigma(-2) = \frac{1}{8.3891} = 0.1192$$

For $z = 0$:

$$e^0 = 1, \quad 1 + 1 = 2, \quad \sigma(0) = \frac{1}{2} = 0.5000$$

For $z = 2$:

$$e^{-2} = 0.1353, \quad 1 + 0.1353 = 1.1353, \quad \sigma(2) = \frac{1}{1.1353} = 0.8808$$

For $z = 5$:

$$e^{-5} = 0.0067, \quad 1 + 0.0067 = 1.0067, \quad \sigma(5) = \frac{1}{1.0067} = 0.9933$$

Verification: $\sigma(-2) + \sigma(2) = 0.1192 + 0.8808 = 1.0000$. This holds in general: $\sigma(-z) = 1 - \sigma(z)$, because

$$\sigma(-z) = \frac{1}{1+e^z} = \frac{e^{-z}}{e^{-z}+1} = 1 - \frac{1}{1+e^{-z}} = 1 - \sigma(z).$$

Solution 2.2:

(c) Joerg Osterrieder 2026

(a) Compute the linear score:

$$z = \beta_0 + \beta_{\text{income}} \cdot 5 + \beta_{\text{debt}} \cdot 0.4 = -1.5 + 0.3 \times 5 + (-2.1) \times 0.4$$

$$z = -1.5 + 1.5 - 0.84 = -0.84$$

(b) Compute the probability:

$$P(\text{default}) = \sigma(-0.84) = \frac{1}{1 + e^{0.84}} = \frac{1}{1 + 2.3164} = \frac{1}{3.3164} = 0.3016$$

(c) At a threshold of 0.5, we predict "default" if $P(\text{default}) > 0.5$. Since $0.3016 < 0.5$, we predict **no default**. The customer would be **approved**.

Solution 2.3:

(a) The decision boundary is defined by $P(\text{positive}) = 0.5$, which occurs when $\sigma(z) = 0.5$, i.e., $z = 0$. The equation $z = 0$ becomes:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

This is the equation of a straight line in the (x_1, x_2) plane. Any linear combination of features set equal to a constant defines a hyperplane (a line in 2D), so the boundary is always a straight line.

(b) The slope of the boundary line is $-\beta_1/\beta_2$. Rearranging $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$ gives $x_2 = -(\beta_1/\beta_2)x_1 - \beta_0/\beta_2$. The ratio of the two feature coefficients determines the angle.

(c) The intercepts are determined by β_0. The x_2-intercept is $-\beta_0/\beta_2$ (set $x_1 = 0$) and the x_1-intercept is $-\beta_0/\beta_1$ (set $x_2 = 0$). Changing β_0 shifts the line parallel to itself without changing its slope.

Solution 2.4:
The log loss for a single sample is $\mathcal{L} = -[y \log p + (1 - y) \log(1 - p)]$.
Case (a): $y = 1$, $p = 0.9$.

$$\mathcal{L}_a = -[1 \cdot \log(0.9) + 0 \cdot \log(0.1)] = -\log(0.9) = -(-0.1054) = 0.1054$$

Case (b): $y = 1$, $p = 0.01$.

$$\mathcal{L}_b = -[1 \cdot \log(0.01) + 0 \cdot \log(0.99)] = -\log(0.01) = -(-4.6052) = 4.6052$$

Ratio:

$$\frac{\mathcal{L}_b}{\mathcal{L}_a} = \frac{4.6052}{0.1054} \approx 43.7$$

The loss in case (b) is approximately **44 times larger**. This demonstrates that log loss penalizes confident wrong predictions catastrophically. A model that says "1% chance" when the true label is 1 gets punished far more than a model that says "90% chance" gets rewarded for being right. This asymmetric penalty is what makes log loss effective for training: it forces the model away from overconfident errors.

Solution 2.5:
We start from the log loss for a single sample:

$$\mathcal{L} = -\big[y \log \sigma(z) + (1 - y) \log(1 - \sigma(z))\big]$$

We need $\frac{\partial \mathcal{L}}{\partial z}$. Apply the chain rule to each term.

First term: $\frac{\partial}{\partial z}[-y \log \sigma(z)]$.

$$= -y \cdot \frac{1}{\sigma(z)} \cdot \sigma'(z) = -y \cdot \frac{\sigma(z)(1-\sigma(z))}{\sigma(z)} = -y(1-\sigma(z))$$

Second term: $\frac{\partial}{\partial z}[-(1-y)\log(1-\sigma(z))]$.

$$= -(1-y) \cdot \frac{1}{1-\sigma(z)} \cdot (-\sigma'(z)) = (1-y) \cdot \frac{\sigma(z)(1-\sigma(z))}{1-\sigma(z)} = (1-y)\,\sigma(z)$$

Combine:

$$\frac{\partial \mathcal{L}}{\partial z} = -y(1-\sigma(z)) + (1-y)\,\sigma(z) = -y + y\,\sigma(z) + \sigma(z) - y\,\sigma(z) = \sigma(z) - y$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial z} = \sigma(z) - y}$$

This result is remarkably simple: the gradient is just the *prediction error* (predicted probability minus true label). When $y = 1$ and $\sigma(z) = 0.9$, the gradient is $0.9 - 1 = -0.1$ (a small push to increase z). When $y = 1$ and $\sigma(z) = 0.1$, the gradient is $0.1 - 1 = -0.9$ (a large push to increase z). The sigmoid and the log loss were designed for each other: the log is the natural inverse of the exponential inside the sigmoid, and their composition cancels to leave only the residual. This is not a coincidence—it is why logistic regression uses log loss rather than squared error.

Section 3: Coefficients, Odds Ratios, and Regularization

Solution 3.1:

The odds ratio for a one-unit increase is $\text{OR} = e^{\beta}$. With $\beta = 0.5$:

$$\text{OR} = e^{0.5} = 1.6487$$

Complete sentence: "For each one-unit increase in X, the odds of default are multiplied by **1.6487**."

In practical terms, the odds increase by about 65%. If the original odds of default were 1:4 (20% probability), after a one-unit increase in X the odds become $1.6487 \times (1/4) = 0.412$, corresponding to a probability of $0.412/1.412 \approx 29.2\%$.

Solution 3.2:

(a) **Debt-to-income (OR = 3.2)** most strongly predicts default. For each one-unit increase in debt-to-income, the odds of default are multiplied by 3.2—more than tripled. The other features have ORs below 1 (protective) but their magnitudes are smaller. To compare fairly: income's "distance" from 1 is $|1 - 0.85| = 0.15$, employment's is $|1 - 0.70| = 0.30$, and debt-to-income's is $|3.2 - 1| = 2.2$, making it by far the strongest.

(b) $\text{OR} = 0.85$ for income means that for each one-unit increase in income, the odds of default are multiplied by 0.85, i.e., the odds *decrease* by 15%. Higher income is protective against default.

(c) For a 0.2-unit change in debt-to-income, the multiplicative effect on the odds is:

$$\text{OR}^{0.2} = 3.2^{0.2}$$

Computing: $\ln(3.2) = 1.1632$, so $0.2 \times 1.1632 = 0.2326$, and $e^{0.2326} = 1.262$.

The odds of default are multiplied by approximately **1.26**, a 26% increase, for a 0.2-unit increase in debt-to-income.

(c) Joerg Osterrieder 2026

Solution 3.3:

In scikit-learn, $C = 1/\lambda$ where λ is the regularization strength. Larger C means *weaker* regularization.

(a) **Coefficient magnitudes:** At $C = 0.01$ (strong regularization, $\lambda = 100$), the penalty for large coefficients is severe, so the optimizer shrinks all coefficients toward zero. At $C = 100$ (weak regularization, $\lambda = 0.01$), coefficients are nearly unconstrained and can grow large to fit the training data closely.

(b) **Decision boundary complexity:** With $C = 0.01$, the small coefficients produce a boundary that is nearly flat—the model underfits, ignoring subtle patterns. With $C = 100$, large coefficients produce a boundary that closely follows the training data's shape, capturing noise along with signal.

(c) **Overfitting risk:** $C = 100$ has high overfitting risk because the model fits training noise. $C = 0.01$ has high underfitting risk. This is the bias-variance tradeoff: low C increases bias (oversimplified model) but reduces variance (stable across different training sets); high C reduces bias but increases variance. The optimal C is found via cross-validation, typically searching a logarithmic grid (e.g., $C \in \{0.01, 0.1, 1, 10, 100\}$).

Solution 3.4:

(a) The predicted class is **Finance**, because 0.72 is the largest probability among $[0.15, 0.72, 0.13]$.

(b) The model's confidence is **72%**. This is reasonably high—the model assigns nearly three-quarters of its probability mass to Finance.

(c) Yes, softmax probabilities always sum to 1.00 by construction. The softmax function is defined as:

$$P(k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Summing over all k: $\sum_{k=1}^{K} P(k) = \frac{\sum_k e^{z_k}}{\sum_j e^{z_j}} = 1$. The denominator ensures normalization regardless of the input scores.

(d) With probabilities $[0.34, 0.35, 0.31]$, you should **not trust this prediction**. The model is nearly uniformly uncertain across all three classes. The "winner" (Finance at 0.35) beats the runner-up by only 0.01. In practical terms, this observation lies near the intersection of all three decision boundaries. A responsible system would flag this prediction as low-confidence and either request human review or abstain from classifying.

Solution 3.5:

(a) **L1 drives small coefficients exactly to zero; L2 does not.** L2 (Ridge) shrinks all coefficients proportionally toward zero but never reaches exactly zero. L1 (Lasso) produces a "diamond-shaped" constraint region whose corners lie on the coordinate axes, so the optimal solution often hits a corner where one or more coefficients are exactly zero.

(b) L1 produces a sparse model when: (1) the regularization strength is sufficiently high (small C), and (2) many features are redundant or irrelevant. With correlated features, L1 tends to pick one and zero out the others. The sparser the true signal (few features truly matter), the better L1 works.

(c) Joerg Osterrieder 2026

(c) Use **L1 regularization**. The regulator requires at most 5 features, which means you need feature selection built into the model. L1 automatically eliminates irrelevant features by setting their coefficients to zero. You tune C until at most 5 coefficients remain nonzero. L2 cannot do this—it keeps all features with small but nonzero coefficients, violating the 5-feature constraint.

(d) In scikit-learn:

- L1: `LogisticRegression(penalty='l1', solver='saga')`
- L2: `LogisticRegression(penalty='l2')` (default)

Note that L1 requires a solver that supports it (`'saga'` or `'liblinear'`). The default solver `'lbfgs'` only supports L2.

Section 4: Decision Trees

Solution 4.1:

The 6-row dataset:

ID	Income ($k)	Credit Score	Default?
1	35	620	Yes
2	80	750	No
3	45	580	Yes
4	70	700	No
5	30	640	Yes
6	90	720	No

Split on Income ≤ 50:

- **Left leaf** (Income ≤ 50): IDs 1, 3, 5. Classes: Yes, Yes, Yes. Majority class: **Default** (3/3).
- **Right leaf** (Income > 50): IDs 2, 4, 6. Classes: No, No, No. Majority class: **No Default** (3/3).

All 6 samples are correctly classified: the 3 defaulters land in the left leaf (predicted Default) and the 3 non-defaulters land in the right leaf (predicted No Default). The split achieves **6 out of 6 correct** (100% training accuracy). This is a perfect split for this small dataset.

Solution 4.2:

(a) The proportion of positives: $p = 30/100 = 0.3$.

(b) Gini impurity:

$$G = 2p(1-p) = 2 \times 0.3 \times 0.7 = 0.42$$

(c) This node is *not* pure. A pure node has $G = 0$ (all samples belong to one class). The maximum Gini impurity for a binary problem is $G_{\max} = 2 \times 0.5 \times 0.5 = 0.5$, occurring at a 50/50 split. Our value of 0.42 is close to the maximum, indicating substantial mixing of the two classes. The node needs further splitting.

Solution 4.3:

(a) **Parent Gini:** $p = 60/100 = 0.6$.

$$G_{\text{parent}} = 2 \times 0.6 \times 0.4 = 0.48$$

(b) **Left child Gini:** 35 positive out of 40, so $p_L = 35/40 = 0.875$.

$$G_{\text{left}} = 2 \times 0.875 \times 0.125 = 0.21875$$

Right child Gini: 25 positive out of 60, so $p_R = 25/60 = 0.4167$.

$$G_{\text{right}} = 2 \times 0.4167 \times 0.5833 = 0.4861$$

(c) **Weighted average Gini after split:**

$$G_{\text{split}} = \frac{40}{100} \times 0.21875 + \frac{60}{100} \times 0.4861 = 0.0875 + 0.2917 = 0.3792$$

(d) **Gini gain (reduction in impurity):**

$$\Delta G = G_{\text{parent}} - G_{\text{split}} = 0.48 - 0.3792 = 0.1008$$

(e) This is a **moderately good split**. The Gini dropped from 0.48 to 0.3792, a reduction of about 21%. The left child is fairly pure ($G = 0.22$) with 87.5% positives, but the right child remains impure ($G = 0.49$) with a near-even 42/58 split. The tree would continue splitting the right child to further separate the classes. For comparison, a perfect split would reduce Gini to 0 (gain of 0.48), so this split captures about $0.1008/0.48 = 21\%$ of the possible improvement.

Solution 4.4:

(a) **Tree B is overfitting.** The hallmark of overfitting is a large gap between training accuracy and test accuracy. Tree B achieves 99% on training data but only 55% on test data—a 44-percentage-point gap. It has memorized the training set, including its noise, and fails to generalize. Tree A has a modest 2-point gap (72% vs. 70%), indicating it generalizes nearly as well as it trains.

(b) **Deploy Tree A.** In production, only test accuracy matters (because the model sees new, unseen data). Tree A's 70% test accuracy is far better than Tree B's 55%. A simple model that generalizes beats a complex model that memorizes.

(c) Hyperparameters to tune between the extremes of depth 2 and depth 20:

- `max_depth`: Try values 3–10 using cross-validation.
- `min_samples_split`: Require at least 10–50 samples to justify a split.
- `min_samples_leaf`: Require at least 5–20 samples in each leaf.
- `max_leaf_nodes`: Directly limit the tree's number of leaves.

The goal is a depth (or complexity) that balances bias and variance—somewhere between the underfitting of depth 2 and the overfitting of depth 20.

Solution 4.5:

(a) Each split in a decision tree tests a *single feature* against a threshold: "Is $x_1 \leq t$?" This creates a boundary that is perpendicular to the x_1-axis—a vertical or horizontal line in 2D. The tree never tests a combination like "Is $x_1 + x_2 \leq t$?". Since every split is axis-aligned, the overall boundary (the union of all splits) is also axis-aligned—a collection of horizontal and vertical line segments.

(b) For the diagonal boundary $x_2 = x_1$, the tree must approximate the diagonal with many axis-aligned cuts:

- First split: $x_1 \leq 0.5$ (vertical line).
- Then: $x_2 \leq 0.5$ in each child (horizontal lines).
- Then: finer splits on x_1, then x_2, alternating.

The result is a staircase pattern that zigzags along the diagonal. The more steps, the closer the approximation, but each step requires an additional split (and more data to support it). A depth-d tree can create at most 2^d rectangular regions, so approximating a smooth diagonal requires d to be large.

(c) **Logistic regression** from Section 2 handles a diagonal boundary with a single line. It learns $\beta_1 x_1 + \beta_2 x_2 + \beta_0 = 0$. For the boundary $x_2 = x_1$ (equivalently $x_1 - x_2 = 0$), logistic regression sets $\beta_1 = 1$, $\beta_2 = -1$, $\beta_0 = 0$—one equation, done.

(d) **Feature engineering** lets a tree capture diagonal boundaries. Create a new feature $x_3 = x_1 - x_2$. Now the diagonal boundary $x_2 = x_1$ becomes $x_3 = 0$, which is a single axis-aligned split on the new feature. More generally, creating interaction features (sums, differences, ratios of existing features) can transform non-axis-aligned boundaries into axis-aligned ones in the augmented feature space.

Section 5: Random Forests

Solution 5.1:

Bagging (bootstrap aggregating) trains many decision trees, each on a different random sample drawn with replacement from the training data. Because each tree sees a slightly different version of the data, each tree makes different errors. When you average their predictions (or take a majority vote), the individual errors tend to cancel out. A single tree trained on the full dataset is highly sensitive to which specific points are included—change a few points and the tree can look completely different. By training on many bootstrap samples and combining the results, the ensemble's prediction is much more stable (lower variance) than any single tree.

Solution 5.2:

The top 3 predictors are:

1. **Income** (importance = 0.35)
2. **Debt ratio** (importance = 0.28)
3. **Employment years** (importance = 0.20)

These importance scores do *not* tell us whether income *causes* creditworthiness. Feature importance measures how much a feature helps the model make accurate predictions—it captures *predictive association*, not *causation*. Income might be correlated with other unmeasured factors (wealth, education, family support) that are the true causal drivers. Observational data alone cannot establish causation; that requires controlled experiments or careful causal inference techniques.

Solution 5.3:

The single tree's 23-point gap (95% train, 72% test) indicates **overfitting**. The tree has memorized the training data, including its noise, and fails to generalize. The forest achieves lower training accuracy (89%) because each individual tree sees only a bootstrap sample and a random subset of features, so no single tree can memorize the full dataset. However, this "impaired" training translates to much better generalization (84% test) because:

- Each tree overfits in a *different direction* (different data subset, different feature subset).

- When you average many diverse overfitting trees, the overfitting components cancel and the shared signal reinforces.
- The ensemble's variance is approximately Var(single tree)$/n$ (reduced by the number of trees), while the bias stays roughly the same.

The forest trades a small increase in bias (89% vs. 95% train) for a large reduction in variance, yielding better test performance.

Solution 5.4:

Gini importance measures how much a feature contributes to reducing impurity across all splits in the forest. When two features are highly correlated ($r = 0.82$), the tree may split on either one interchangeably. The importance gets *shared* between `income` and `monthly_spending`—each appears important in some trees but not others. Both get moderate importance scores, even if only one truly matters. Gini importance also has a bias toward high-cardinality features.

Permutation importance measures how much test accuracy drops when a feature's values are randomly shuffled. For correlated features, permuting `income` while leaving `monthly_spending` intact causes little accuracy drop, because the model can still extract the same information from the correlated partner. Both features appear *less* important than they truly are.

Permutation importance gives a more honest picture of each feature's *unique* contribution, because it answers: "What information does this feature provide that no other feature provides?" However, it can underestimate the joint importance of a correlated group. To get the full picture, one can permute both correlated features simultaneously to measure their combined contribution.

Solution 5.5:

(a) **Metric:** Use **precision at a fixed recall level** (e.g., precision@recall=0.5) or **PR-AUC**. Accuracy is a poor choice for trading signals because: (1) profitable signals are rare (class imbalance), and (2) a model predicting "no signal" always achieves high accuracy but zero trading profit. A profit-based metric like Sharpe ratio of the resulting portfolio could also work, but it requires a full backtesting pipeline.

(b) **Standard k-fold cross-validation shuffles data randomly**, breaking the temporal ordering. This means the model trains on future data and predicts on past data (data leakage). Financial time series contain autocorrelation, regime changes, and trends that make future information genuinely informative about the past—but in a way that cannot be exploited in real trading.

(c) **Time-series cross-validation** (expanding or sliding window):
   - Fold 1: Train on 2015–2017, validate on 2018.
   - Fold 2: Train on 2015–2018, validate on 2019.
   - Fold 3: Train on 2015–2019, validate on 2020.
   - Continue until the final fold tests on 2023.

   This ensures the model never trains on data from after the validation period.

(d) **Expected relationship:** As `n_estimators` increases from 1 to perhaps 500, the chosen metric improves rapidly at first (each new tree adds diversity) and then plateaus. Unlike a single tree, random forests do *not* overfit with more trees—the metric curve flattens but does not decline. The point of diminishing returns (where adding more trees yields negligible improvement) is typically between 100 and 500 trees. Choose the smallest `n_estimators` where the metric has stabilized, to balance performance against computational cost.

Section 6: Precision, Recall, and the Confusion Matrix

Solution 6.1:
Given: TP = 45, FP = 5, FN = 10, TN = 940.

(a) **Precision** $= \frac{\text{TP}}{\text{TP}+\text{FP}} = \frac{45}{45+5} = \frac{45}{50} = 0.90$ (90%)

(b) **Recall** $= \frac{\text{TP}}{\text{TP}+\text{FN}} = \frac{45}{45+10} = \frac{45}{55} = 0.8182$ (81.8%)

(c) **F1 score** $= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.90 \times 0.8182}{0.90 + 0.8182}$

$$= 2 \times \frac{0.7364}{1.7182} = 2 \times 0.4286 = 0.8571 \approx 0.857 \text{ (85.7\%)}$$

(d) **Accuracy** $= \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{FN}+\text{TN}} = \frac{45+940}{45+5+10+940} = \frac{985}{1000} = 0.985$ (98.5%)

Note the gap: accuracy is 98.5% (sounds excellent) but recall is only 81.8% (the model misses 10 out of 55 positives). This is why accuracy alone is misleading when classes are imbalanced.

Solution 6.2:
If the fraud rate is 0.5%, then in 10,000 transactions there are 50 frauds and 9,950 legitimate ones. A model that predicts "legitimate" for *every* transaction achieves accuracy = 9,950/10,000 = 99.5%. This exceeds 99%, yet the model catches zero frauds—every single fraud goes undetected. The model is useless for its intended purpose.

Recall would reveal the failure immediately: recall $= 0/(0+50) = 0$. The model has zero recall for the fraud class. Precision is undefined (0/0) because the model never predicts positive. The F1 score is also 0. Any metric that considers the positive class (recall, F1, PR-AUC) would expose this model as a failure, while accuracy hides it behind the overwhelming majority class.

Solution 6.3:
Given: TP = 80, FP = 20, FN = 30, TN = 870. Total = 1,000.

(a)
- Precision $= \frac{80}{80+20} = \frac{80}{100} = 0.80$ (80%)
- Recall $= \frac{80}{80+30} = \frac{80}{110} = 0.7273$ (72.7%)
- F1 $= 2 \times \frac{0.80 \times 0.7273}{0.80 + 0.7273} = 2 \times \frac{0.5818}{1.5273} = 2 \times 0.3809 = 0.7619$ (76.2%)
- Accuracy $= \frac{80+870}{1000} = \frac{950}{1000} = 0.95$ (95%)

(b) For cancer screening, you should **reduce FN by 10** (from 30 to 20). Here is why:

Option 1: Reduce FP by 10 $\rightarrow$ TP = 80, FP = 10, FN = 30. New recall $= 80/(80+30) = 0.7273$ (unchanged).

Option 2: Reduce FN by 10 $\rightarrow$ TP = 80, FP = 20, FN = 20. New recall $= 80/(80+20) = 0.80$ (improved from 72.7% to 80%).

In cancer screening, a false negative means a patient with cancer is told they are healthy—they miss early treatment, potentially with fatal consequences. A false positive means a healthy patient gets an unnecessary follow-up test—stressful but not dangerous. The cost of FN (missed cancer) vastly exceeds the cost of FP (extra biopsy), so reducing FN is the right choice.

(c) Joerg Osterrieder 2026

Solution 6.4:

(a) **Spam filter: Precision matters more.** A false positive means a legitimate email is sent to spam—you miss an important message from a colleague or client. A false negative means a spam email reaches your inbox—annoying but harmless (you just delete it). The cost of missing real email outweighs the cost of seeing spam, so you want high precision (few false positives).

(b) **Cancer screening: Recall matters more.** A false negative means a cancer patient is missed and goes untreated—potentially fatal. A false positive means a healthy person gets a follow-up biopsy—uncomfortable but not life-threatening. You must catch as many true cases as possible, so you maximize recall.

(c) **Credit card fraud alert: Recall matters more (with precision awareness).** A false negative means fraud goes undetected—the bank loses money and the customer's account is compromised. A false positive means a legitimate transaction is flagged and the customer gets a verification call—inconvenient but resolvable. However, if precision is too low, customers get frustrated by constant false alerts and may abandon the card. The primary metric is recall, with a precision floor (e.g., precision ≥ 0.05) to keep false alerts manageable.

(d) **Resume screening: Precision matters more.** A false positive means an unqualified candidate gets an interview—wasting interviewer time and resources. A false negative means a qualified candidate is rejected—bad, but less visible and less costly per incident (the company never knows what it missed). The HR team's scarce interview slots should go to genuinely qualified candidates, so precision is prioritized.

Solution 6.5:

(a) **Report to the regulator: Precision (or, more broadly, a clear confusion matrix).** GDPR Article 22 gives individuals the right to meaningful explanations of automated decisions. When a loan is denied, the bank must explain why. High precision means that when you deny someone (predict "default"), you are usually right. This makes explanations credible: "We denied your application because your debt-to-income ratio of 0.85 places you in a high-default-risk category, and 90% of applicants in this category have historically defaulted." Low precision (many false denials) undermines regulatory trust.

(b) **Optimize internally: Recall for the default class (or F1).** The credit department wants to minimize losses, which means catching as many actual defaults as possible before they happen. This is a recall objective. However, pure recall maximization would deny loans to too many creditworthy applicants (lost revenue). In practice, the bank optimizes F1 or a cost-weighted metric that balances missed defaults (FN cost) against lost good customers (FP opportunity cost).

(c) **Reconciliation:** Use a two-stage approach. First, train the most accurate model possible (e.g., a random forest optimized for F1 or a cost-sensitive metric). Second, for every denial, generate a post-hoc explanation using the top contributing features (e.g., SHAP values or the logistic regression coefficients if the model is linear). The model optimizes for recall/F1 to catch defaults, but the explanation layer uses interpretable features to satisfy the regulator. If the regulator requires the model itself to be interpretable (not just the explanations), use logistic regression with L1 regularization to select a small number of features, and accept a modest reduction in recall for the sake of full transparency.

(c) Joerg Osterrieder 2026

Section 7: ROC Curves and AUC

Solution 7.1:

The AUC can be interpreted as: "If you randomly pick one positive sample and one negative sample, the AUC is the probability that the model scores the positive higher than the negative."

(a) **AUC = 0.5:** The model is no better than random guessing. If you pick a random defaulter and a random non-defaulter, the model ranks the defaulter higher only 50% of the time—the same as flipping a coin. The model has learned nothing useful about what distinguishes the two classes.

(b) **AUC = 0.85:** The model is good. A randomly chosen positive sample scores higher than a randomly chosen negative sample 85% of the time. In 15% of random pairs, the model gets the ranking wrong. This is a solid, production-viable model for many applications.

(c) **AUC = 1.0:** The model achieves perfect discrimination. Every positive sample receives a higher score than every negative sample. There exists a threshold that perfectly separates the two classes with zero errors. This is rare in real-world data (usually indicates data leakage or a trivially separable problem).

Solution 7.2:

The "elbow" at (FPR = 0.10, TPR = 0.85) represents the point of maximum curvature on the ROC curve, where the tradeoff between true positives and false positives shifts. At this point:

- You catch 85% of all positives (TPR = 0.85).
- You falsely flag 10% of negatives (FPR = 0.10).

This is a good operating threshold because the ratio of benefit (TPR gained) to cost (FPR incurred) is at its best. Geometrically, this is the point closest to the ideal top-left corner $(0, 1)$.

Operating to the left (lower FPR, e.g., FPR = 0.02) means you are more conservative: fewer false alarms, but you also miss more positives (TPR might drop to 0.50). This is appropriate when false positives are very expensive (e.g., unnecessary surgical biopsies).

Operating to the right (higher FPR, e.g., FPR = 0.30) means you are more aggressive: you catch nearly all positives (TPR might rise to 0.95), but at the cost of many false alarms. This is appropriate when false negatives are catastrophic (e.g., missing a terrorist at airport security).

Solution 7.3:

AUC's threshold-independence is not always an advantage. It summarizes performance across *all* thresholds equally, including thresholds you would never use in practice.

Concrete scenario: A hospital screening system for a rare disease must operate at recall ≥ 0.90 (you cannot miss more than 10% of cases—ethical and legal requirement). Model A has AUC = 0.92 but achieves only precision = 0.02 at recall = 0.90. Model B has AUC = 0.88 but achieves precision = 0.15 at recall = 0.90. AUC prefers Model A, but at the *operationally required* threshold, Model B is far better (7.5 times more precise). The hospital cares about performance at a specific recall level, not across all possible thresholds.

In general, prefer threshold-specific metrics when: (1) the operating point is fixed by business or regulatory constraints, (2) class imbalance is severe (AUC can be optimistic), or (3) the costs of FP and FN are very different (which pins the threshold).

Solution 7.4:

We have 1,000 transactions, 5 fraudulent. FP cost = €10, FN cost = €100.

For each threshold, compute TP, FP, FN, and total cost:

Threshold 0.3: Recall = 1.00, Precision = 0.05, Predicted Positives = 100.

- TP $= 1.00 \times 5 = 5$. FP $= 100 - 5 = 95$. FN $= 5 - 5 = 0$.

- Cost $= 95 \times 10 + 0 \times 100 =$ €950 + €0 = €950.

Threshold 0.4: Recall = 0.80, Precision = 0.08, Predicted Positives = 50.

- TP $= 0.80 \times 5 = 4$. FP $= 50 - 4 = 46$. FN $= 5 - 4 = 1$.
- Cost $= 46 \times 10 + 1 \times 100 =$ €460 + €100 = €560.

Threshold 0.5: Recall = 0.60, Precision = 0.15, Predicted Positives = 20.

- TP $= 0.60 \times 5 = 3$. FP $= 20 - 3 = 17$. FN $= 5 - 3 = 2$.
- Cost $= 17 \times 10 + 2 \times 100 =$ €170 + €200 = €370.

Threshold 0.6: Recall = 0.40, Precision = 0.25, Predicted Positives = 8.

- TP $= 0.40 \times 5 = 2$. FP $= 8 - 2 = 6$. FN $= 5 - 2 = 3$.
- Cost $= 6 \times 10 + 3 \times 100 =$ €60 + €300 = €360.

Threshold 0.7: Recall = 0.20, Precision = 0.50, Predicted Positives = 2.

- TP $= 0.20 \times 5 = 1$. FP $= 2 - 1 = 1$. FN $= 5 - 1 = 4$.
- Cost $= 1 \times 10 + 4 \times 100 =$ €10 + €400 = €410.

Threshold	TP	FP	FN	FP Cost	FN Cost	Total Cost
0.3	5	95	0	€950	€0	€950
0.4	4	46	1	€460	€100	€560
0.5	3	17	2	€170	€200	€370
0.6	2	6	3	€60	€300	**€360**
0.7	1	1	4	€10	€400	€410

Threshold 0.6 minimizes total cost at €360. The cost ratio (FN/FP = 100/10 = 10) is moderate enough that the optimal point balances investigation costs against fraud losses. Note that threshold 0.3 (catch all fraud) costs €950 due to the flood of false alarms, while threshold 0.7 (few alerts) costs €410 because 4 out of 5 frauds slip through.

Solution 7.5:

(a) **How ROC-AUC = 0.95 but PR-AUC = 0.30 is possible:**

ROC-AUC measures the model's ability to rank a random positive above a random negative. With 9,900 negatives, even a small FPR (e.g., 1%) means $0.01 \times 9{,}900 = 99$ false positives. The ROC curve "sees" FPR, which stays small in percentage terms. So the ROC curve looks excellent.

PR-AUC measures precision versus recall. At high recall, precision = TP/(TP + FP). If you catch 90 of the 100 positives (recall = 0.90) but generate 99 false positives, precision $= 90/(90 + 99) = 0.476$. As you try to catch the last few positives, FP grows further and precision drops. PR-AUC is low because the model cannot achieve high precision *and* high recall simultaneously—the 99:1 imbalance makes the denominator of precision large.

What each metric "sees": ROC-AUC is influenced by the large TN count (9,900 negatives correctly classified keep FPR low). PR-AUC ignores TN entirely and focuses only on what happens among the predicted positives. With severe imbalance, ROC-AUC is optimistic; PR-AUC is realistic.

(b) **PR-AUC is more honest** for stakeholders who care about catching the positive class. It directly answers: "When the model says 'positive,' how often is it right?" (precision) and "Of all actual positives, how many does the model find?" (recall). ROC-AUC can look excellent even when precision is poor, which misleads stakeholders into thinking the model is deployment-ready.

(c) **ROC-AUC and PR-AUC agree closely when the class distribution is approximately balanced** (e.g., 50:50 or even 70:30). In balanced settings, a small FPR in percentage terms also means a small absolute number of false positives, so precision stays high. As the positive class becomes rarer, the two metrics diverge: ROC-AUC remains optimistic while PR-AUC drops.

Section 8: Class Imbalance

Solution 8.1:

(a) **Imbalance ratio:**

$$\frac{\text{Majority}}{\text{Minority}} = \frac{9{,}500}{500} = 19 : 1$$

There are 19 legitimate transactions for every fraudulent one.

(b) **Accuracy of always predicting "legitimate":**

$$\text{Accuracy} = \frac{9{,}500}{9{,}500 + 500} = \frac{9{,}500}{10{,}000} = 0.95 = 95\%$$

The model achieves 95% accuracy by doing nothing useful.

(c) **Recall for the fraud class:**

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{0}{0 + 500} = 0$$

The model catches zero frauds. Every single fraud is a false negative.

Solution 8.2:

SMOTE (Synthetic Minority Over-sampling TEchnique) generates synthetic samples by interpolating between existing minority samples and their nearest minority neighbors.

Step-by-step for one synthetic sample:

1. **Pick an existing minority sample.** We select fraud sample A: (amount = €450, time = 02:30).
2. **Find its nearest minority neighbor.** The nearest fraud neighbor is B: (amount = €520, time = 03:15).
3. **Draw a random number $\lambda \in [0, 1]$.** We are given $\lambda = 0.4$.
4. **Interpolate each feature:**

$$\text{amount}_{\text{new}} = 450 + 0.4 \times (520 - 450) = 450 + 0.4 \times 70 = 450 + 28 = €478$$

$$\text{time}_{\text{new}} = 02{:}30 + 0.4 \times (03{:}15 - 02{:}30) = 02{:}30 + 0.4 \times 45 \text{ min} = 02{:}30 + 18 \text{ min} = 02{:}48$$

5. **Result:** The synthetic fraud sample is (amount = €478, time = 02:48).

(c) Joerg Osterrieder 2026

The parameter λ controls where on the line segment between A and B the new point falls. $\lambda = 0$ places it exactly at A; $\lambda = 1$ places it at B; $\lambda = 0.4$ places it 40% of the way from A to B. By drawing λ uniformly from $[0, 1]$, SMOTE fills the feature space between existing minority samples, creating plausible new examples rather than exact duplicates.

Solution 8.3:
Starting dataset: 10,000 samples (9,900 majority, 100 minority).

(a) **Random oversampling to 50:50 (duplicate minority to 9,900):**
- Resulting training set size: $9{,}900 + 9{,}900 = 19{,}800$ samples.
- Main risk: **Overfitting to the minority class.** The 100 original minority samples are duplicated $\sim$99 times each. The model memorizes these exact points rather than learning the underlying pattern. Any noise in the minority examples is amplified.

(b) **Random undersampling to 50:50 (reduce majority to 100):**
- Resulting training set size: $100 + 100 = 200$ samples.
- Main risk: **Massive information loss.** You discard 9,800 out of 9,900 majority samples (99%). The model sees only 2% of the original data. Patterns in the majority class that require large sample sizes to detect are lost. The resulting model has high variance.

(c) **Class weights (`class_weight='balanced'`):**
- Resulting training set size: 10,000 (unchanged).
- Main risk: **No data-level risk**—all original data is preserved. The optimizer assigns a weight of $9{,}900/100 = 99$ to each minority sample's loss contribution, making minority misclassifications 99 times more expensive. The risk is a noisier gradient (the loss landscape is dominated by a few heavily weighted samples) and potentially unstable convergence.

Which preserves the most information? `class_weight='balanced'` preserves all original data. It is generally the safest starting point because it avoids both the memorization risk of oversampling and the data-loss risk of undersampling.

Solution 8.4:
We have 10,000 transactions with 50 frauds. FP cost = €10, FN cost = €5,000.
Threshold 0.3: Precision = 0.15, Recall = 0.90.

- TP $= 0.90 \times 50 = 45$.
- Total predicted positives = TP/Precision $= 45/0.15 = 300$. FP $= 300 - 45 = 255$.
- FN $= 50 - 45 = 5$.
- Cost $= 255 \times 10 + 5 \times 5{,}000 =$ €2,550 + €25,000 = €27,550.

Threshold 0.5: Precision = 0.45, Recall = 0.60.

- TP $= 0.60 \times 50 = 30$.
- Total predicted positives $= 30/0.45 = 66.67 \approx 67$. FP $= 67 - 30 = 37$.
- FN $= 50 - 30 = 20$.
- Cost $= 37 \times 10 + 20 \times 5{,}000 =$ €370 + €100,000 = €100,370.

Threshold 0.7: Precision = 0.80, Recall = 0.30.

- TP $= 0.30 \times 50 = 15$.
- Total predicted positives $= 15/0.80 = 18.75 \approx 19$. FP $= 19 - 15 = 4$.
- FN $= 50 - 15 = 35$.
- Cost $= 4 \times 10 + 35 \times 5{,}000 =$ €40 + €175,000 = €175,040.

Threshold	TP	FP	FN	FP Cost + FN Cost	Total Cost
0.3	45	255	5	€2,550 + €25,000	**€27,550**
0.5	30	37	20	€370 + €100,000	€100,370
0.7	15	4	35	€40 + €175,000	€175,040

Threshold 0.3 minimizes total cost at €27,550. The cost ratio here is extreme: each missed fraud costs €5,000 while each false alarm costs only €10. That is a 500:1 ratio. With such asymmetric costs, it is far cheaper to investigate hundreds of false alarms than to miss even a few frauds. This is why fraud detection systems operate at low thresholds with high recall, accepting many false positives to minimize the devastating cost of false negatives.

Solution 8.5:

(a) **Data splitting:** Random train/test splitting is dangerous because fraud patterns evolve over time (new attack vectors, seasonal patterns, concept drift). A random split lets the model "see" future fraud patterns during training, inflating test performance. **Better approach:** Time-based splitting. Train on the first 3–6 months of data, validate on the next month, test on the most recent month. This mirrors the production setting where the model must predict future transactions using only past data.

(b) **Resampling:** Use `class_weight='balanced'` as the baseline. At a 0.05% fraud rate (50 frauds per 100,000), even SMOTE has limited material to interpolate—50 points in high-dimensional space produce a thin manifold of synthetic samples that may not represent real fraud diversity. Random oversampling at this extreme ratio ($\sim$2,000$\times$ duplication) guarantees severe overfitting. Class weights avoid these problems entirely: all original data is preserved, no synthetic artifacts, and the implementation is a single parameter change. If class weights prove insufficient, try SMOTE *applied only to the training fold* (never the validation or test set).

(c) **Model: Gradient boosting** (e.g., XGBoost or LightGBM). Reasons: (1) It handles tabular data with mixed feature types well. (2) It natively supports `scale_pos_weight` for class imbalance. (3) It provides feature importance for interpretability. (4) It outperforms logistic regression on nonlinear patterns (fraud often involves complex interactions between amount, time, location, and merchant type). (5) Random forest is also strong but gradient boosting typically achieves higher recall at a given precision level. Logistic regression is too simple for the feature interactions typical in fraud, though it could serve as a baseline or a component in an ensemble.

(d) **Metrics:** Primary: **PR-AUC** (area under precision-recall curve). At 0.05% fraud rate, ROC-AUC is misleadingly optimistic because TN dominates. PR-AUC focuses on what matters: among the transactions the model flags, how many are actually fraud (precision), and of all actual frauds, how many does the model catch (recall)? Secondary: **Recall at precision $\geq$ 0.05** (or a business-specified precision floor) to ensure the alert volume is operationally feasible.

(c) Joerg Osterrieder 2026

(e) **Threshold selection:** Use cost-sensitive threshold optimization. From the business, you need: (a) average fraud loss per missed transaction (FN cost), (b) cost to investigate a flagged transaction (FP cost—analyst salary, customer friction), (c) maximum daily alert volume the investigation team can handle. Compute total expected cost at each threshold using the validation set, and select the threshold minimizing cost subject to the alert-volume constraint. The optimal threshold will be low (high recall) because FN cost $\gg$ FP cost.

(f) **Monitoring and drift detection:** Track the following weekly:

- *Prediction distribution:* Monitor the average predicted fraud probability. A shift from 0.05% to 0.15% in the data means the model's calibration degrades.
- *Alert rate:* If the number of daily alerts increases steadily, the fraud rate may be rising.
- *Precision of investigated alerts:* If confirmed fraud among flagged transactions drops, the model is losing discriminative power.
- *Population Stability Index (PSI):* Compare feature distributions monthly between training data and recent production data. PSI > 0.2 signals significant drift.

A tripling of the fraud rate (0.05% to 0.15%) changes the prior probability substantially. **Retrain** when: (1) PR-AUC on a rolling validation window drops below a predefined threshold (e.g., 80% of the original), or (2) PSI exceeds 0.2 for key features, or (3) on a fixed schedule (e.g., monthly) regardless of drift, using a rolling 6-month training window. Retrain on recent data that includes the new fraud patterns.

(c) Joerg Osterrieder 2026