Machine Learning

Imperial College London

Nuri Cingillioglu

Part 5: Decisions, Decisions

The art of separation

How we make decisions?

> *"If the patient's systolic blood pressure is > 140 AND age > 50, order an ECG; otherwise, if troponin is elevated, admit immediately."*

Compared to:

- Multiply blood pressure with $w_2 = 1.2$, then age with $w_3 = 1.8$, then add them up and pass it through a sigmoid function.

Okay Steve, how do we let machines make decisions in a similar way?

- **Discrete, conditional, hierarchical, and rule-based $\rightarrow$ Decision Trees**.

Imperial Exam Prep

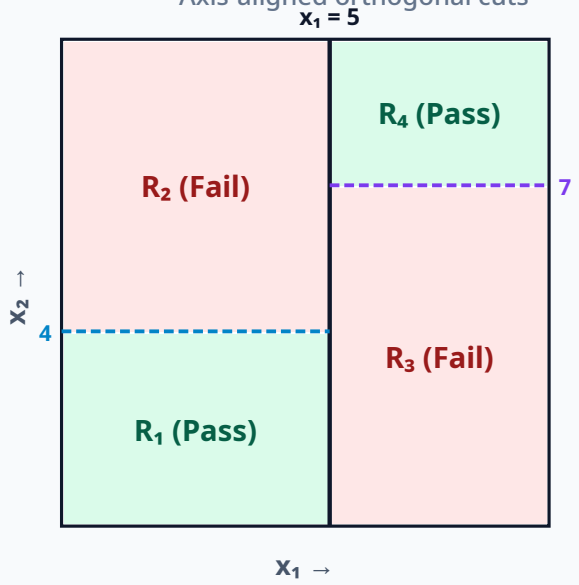
Student	Study Hours (x_1)	Exam Prep (x_2)	Pass Exam (y)
S1	2.0	No	Fail (0)
S2	3.0	No	Fail (0)
S3	4.0	No	Fail (0)
S4	5.0	Yes	Pass (1)
S5	6.0	No	Fail (0)
S6	7.0	Yes	Pass (1)
S7	8.0	Yes	Pass (1)

- Mixed features: continuous (`Study Hours`) + categorical (`Exam Prep`).
- Total: 6 Pass, 4 Fail. How do we build a decision tree to classify them?

Decision Trees

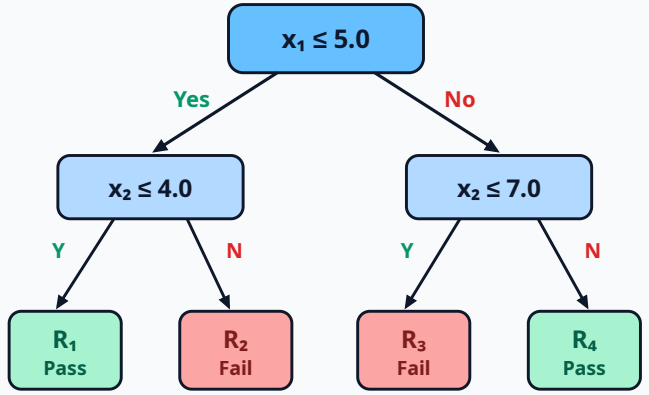
1. Space Partition $\mathbb{R}^2$

Axis-aligned orthogonal cuts



2. Binary Tree Graph

Recursive decision hierarchy



Leaves define disjoint partition:

$\mathbb{R}^2 = R_1 \cup R_2 \cup R_3 \cup R_4$

3. Nested Decision Rules

Disjunctive Normal Form (DNF)

```
if x₁ ≤ 5.0:
    if x₂ ≤ 4.0:
        return R₁ # Pass
    else:
        return R₂ # Fail
else:
    if x₂ ≤ 7.0:
        return R₃ # Fail
    else:
        return R₄ # Pass
```

100% Transparent & Interpretable

Decision Trees mathematically

A decision tree divides feature space into M disjoint axis-aligned bounding boxes $\{R_1, \ldots, R_M\}$:

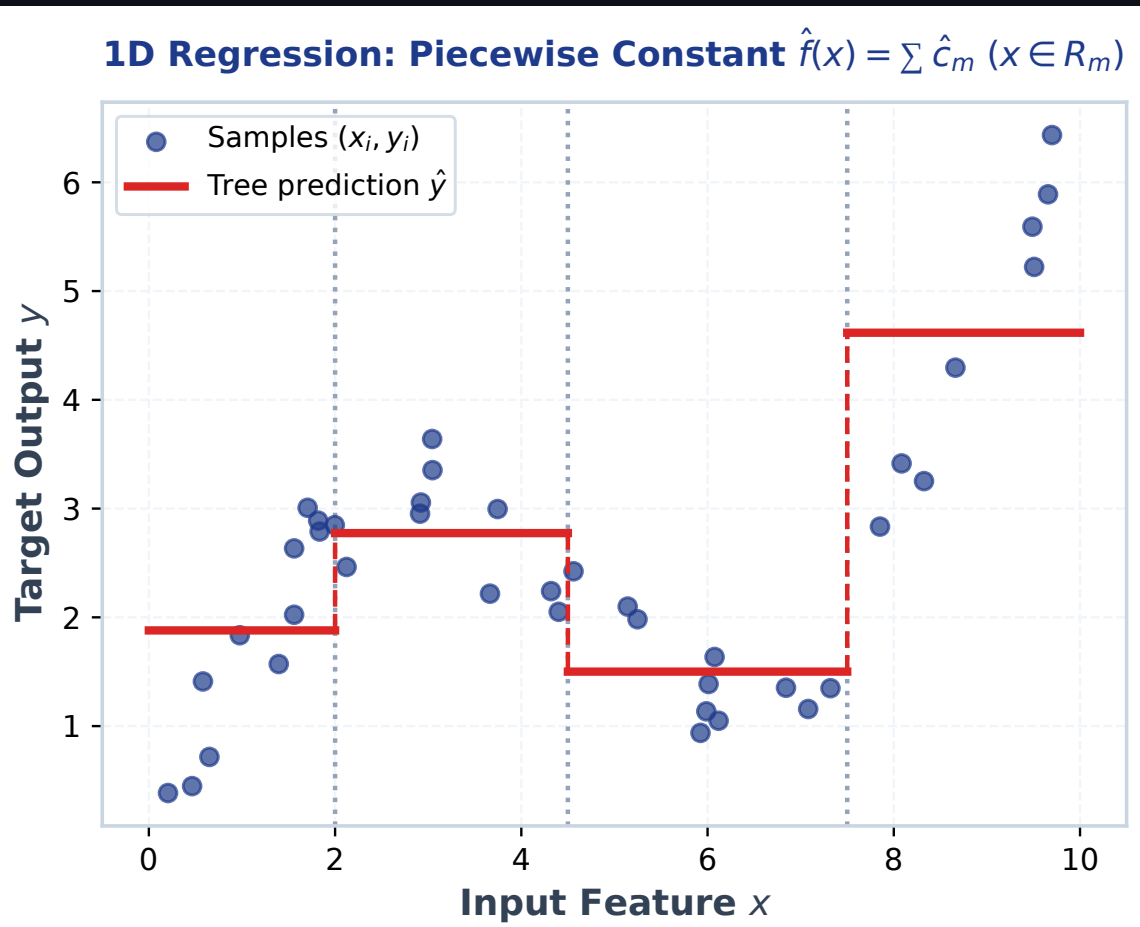
$$\bigcup_{m=1}^{M} R_m = \mathbb{R}^d \quad \text{where} \quad R_m \cap R_{m'} = \emptyset \quad \forall m \neq m'$$

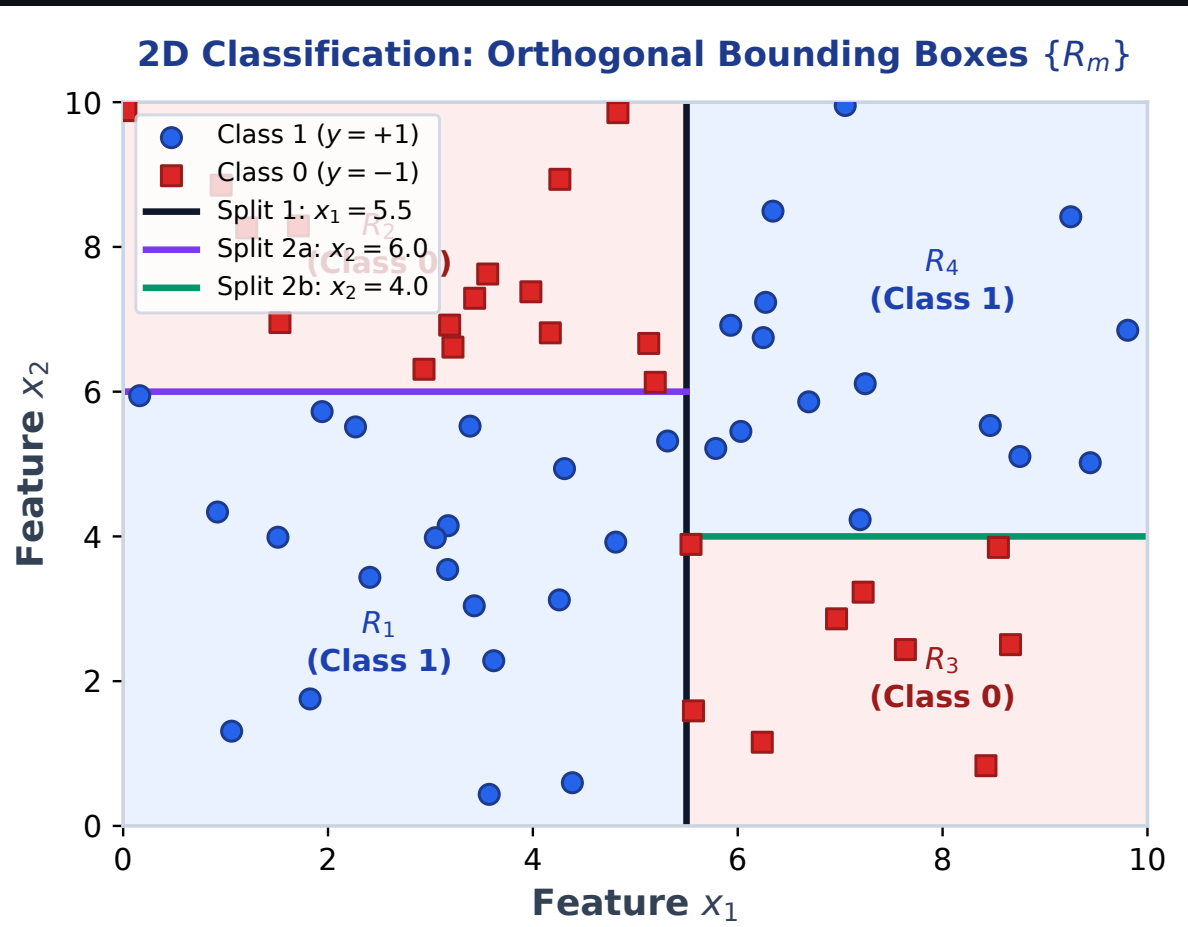
Piecewise-Constant Functions

The tree's prediction for any query $\mathbf{x}$ is a regional constant c_m:

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m \mathbb{I}(\mathbf{x} \in R_m)$$

- **Classification:** Majority vote $\hat{y} = \arg\max_k \hat{p}_{mk}$ where $\hat{p}_{mk} = \frac{1}{|R_m|} \sum_{i \in R_m} \mathbb{I}(y_i = k)$
- **Regression:** Regional sample mean $\hat{y} = \frac{1}{|R_m|} \sum_{i \in R_m} y_i$

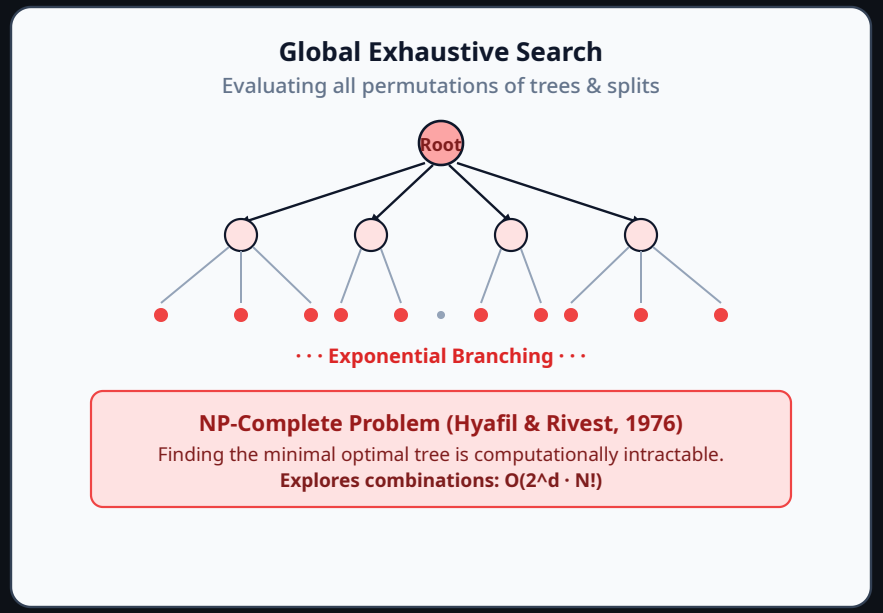

1D Regression: Piecewise Constant $\hat{f}(x) = \sum \hat{c}_m \ (x \in R_m)$
Samples (x_i, y_i)
Tree prediction $\hat{y}$
Target Output y
Input Feature x
0
2
4
6
8
10
1
2
3
4
5
6

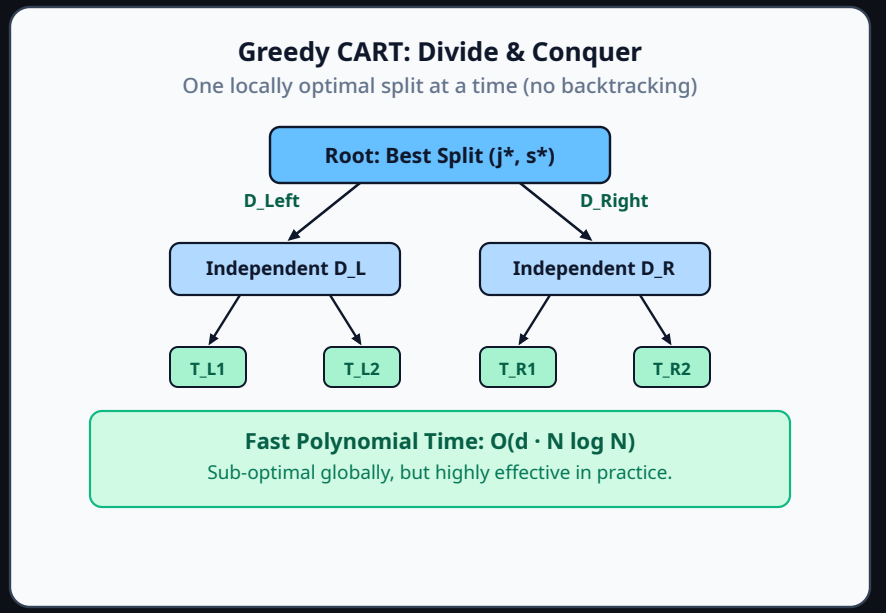

2D Classification: Orthogonal Bounding Boxes $\{R_m\}$
Class 1 ($y = +1$)
Class 0 ($y = -1$)
Split 1: $x_1 = 5.5$
Split 2a: $x_2 = 6.0$
Split 2b: $x_2 = 4.0$
R_1
(Class 1)
R_2
(Class 0)
R_3
(Class 0)
R_4
(Class 1)
Feature x_2
Feature x_1
0
2
4
6
8
10

Okay Steve, build me some trees

Can We Search All Possible Trees?

- For N samples, the number of binary tree topologies grows exponentially...
- **Hyafil & Rivest (1976):** Finding the globally optimal decision tree is **NP-complete**.





Top-Down Greedy Induction

Classification and Regression Trees (CART)

Since global search is impossible, we use a greedy **divide-and-conquer** heuristic:

1. Start at the root node containing all training data D.
2. Scan all features and candidate thresholds.
3. Pick the **single split** that produces the cleanest, most *pure* children.
4. Partition the data into D_L and D_R.
5. Repeat on D_L and D_R until a stopping criterion is met.

> But how do we define "cleanest" or "most pure"?

What Does "Impurity" Mean?

Consider our 10 students at the root node: 6 Pass, 4 Fail.

- If a split creates children that are still 50% Pass and 50% Fail, we gained **zero information**.
- If a split creates a child with 100% Pass and 0% Fail, that child is **pure**!

We define an **Impurity Measure** $I(D)$ that:

- Equals 0 when the node is completely pure (all samples belong to one class).
- Is maximised when classes are evenly mixed.

Impurity Reduction (Gain):

$$\Delta I(D, j, t) = I(D) - \left[\frac{|D_L|}{|D|} I(D_L) + \frac{|D_R|}{|D|} I(D_R) \right]$$

where:

- $I(D)$ is the impurity of the parent node.
- j is the candidate feature, and t is the threshold:

$$D_L = \{(x, y) \in D \mid x_j \leq t\}, \quad D_R = \{(x, y) \in D \mid x_j > t\}$$

- $\frac{|D_L|}{|D|}$ and $\frac{|D_R|}{|D|}$ weight each child's impurity by its sample share.

The Splitting Criteria

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

Shannon Entropy

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

- Grounded in information theory.
- Bits of surprise / uncertainty.

Gini Impurity

$$I_G(D) = 1 - \sum_{k=1}^{K} p_k^2$$

- Probability of misclassifying a random sample.
- No logarithms required.

Misclassification Error

$$E(D) = 1 - \max_k p_k$$

- Training error under majority vote.
- **This fails though**

Why does misclassification error fail?

Suppose a node has $N = 800$ samples (400 Class 1, 400 Class 2). Current error: $E(D) = 0.5$.

Split A

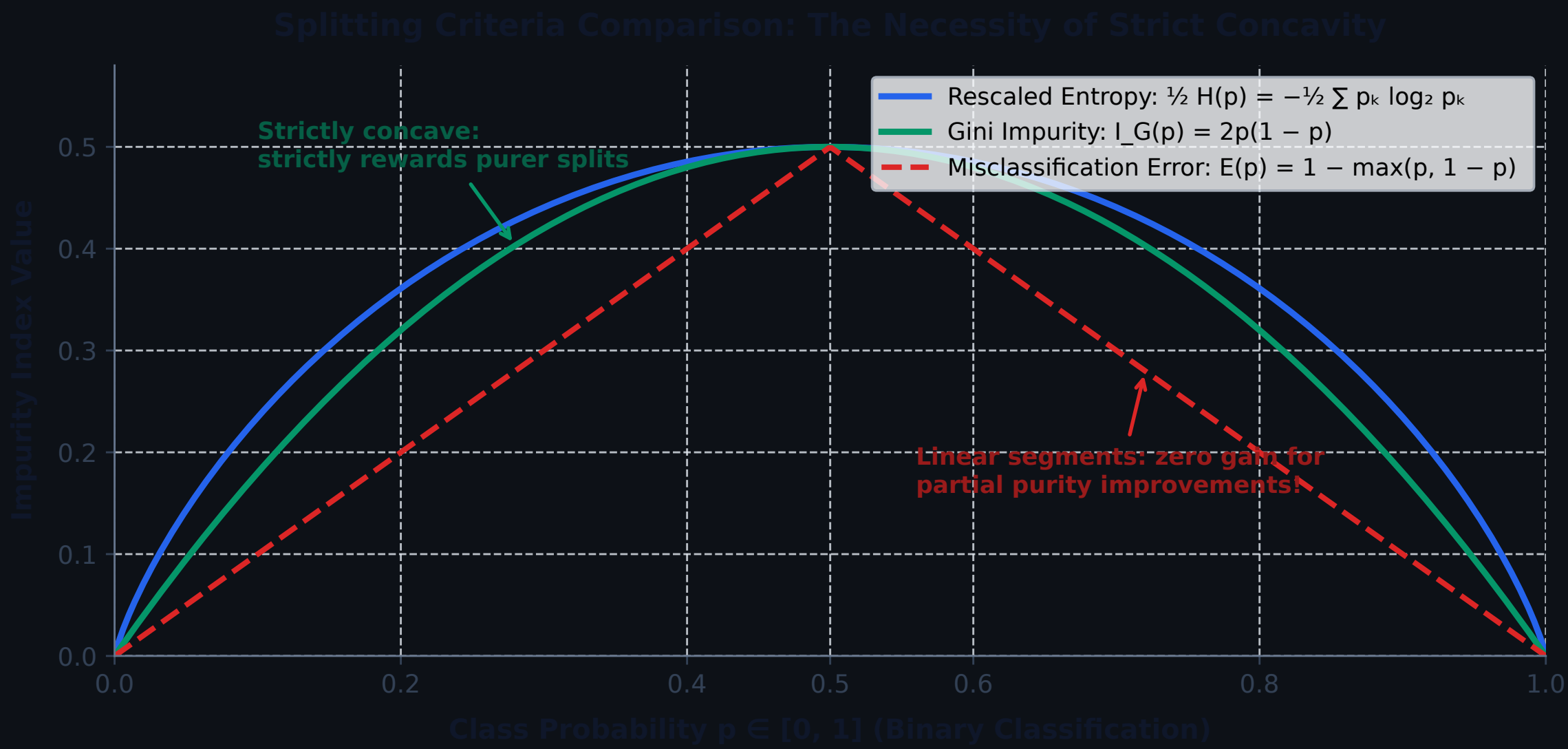
- Left: $(300, 100) \implies p_1 = 0.75$
- Right: $(100, 300) \implies p_1 = 0.25$
- Error: $\frac{400}{800}(0.25) + \frac{400}{800}(0.25) = 0.25$
- **Gain:** $\Delta E = 0.50 - 0.25 = \mathbf{0.25}$

Split B

- Left: $(400, 200) \implies p_1 = 0.67$
- Right: $(0, 200) \implies \mathbf{100\%\ Pure!}$
- Error: $\frac{600}{800}\left(\frac{1}{3}\right) + \frac{200}{800}(0) = 0.25$
- **Gain:** $\Delta E = 0.50 - 0.25 = \mathbf{0.25}$

> Misclassification error is blind to Split B creating a pure terminal leaf. It is not strictly **concave**.

Splitting Criteria



Back to exams

Let's calculate the root impurity for our 10 Imperial students (6 Pass, 4 Fail):

$$p_{\text{Pass}} = \frac{6}{10} = 0.6, \qquad p_{\text{Fail}} = \frac{4}{10} = 0.4$$

Gini Impurity at Root

$$I_G(D) = 1 - (p_{\text{Pass}}^2 + p_{\text{Fail}}^2)$$

$$I_G(D) = 1 - (0.6^2 + 0.4^2)$$

$$I_G(D) = 1 - (0.36 + 0.16) = \mathbf{0.480}$$

Shannon Entropy at Root

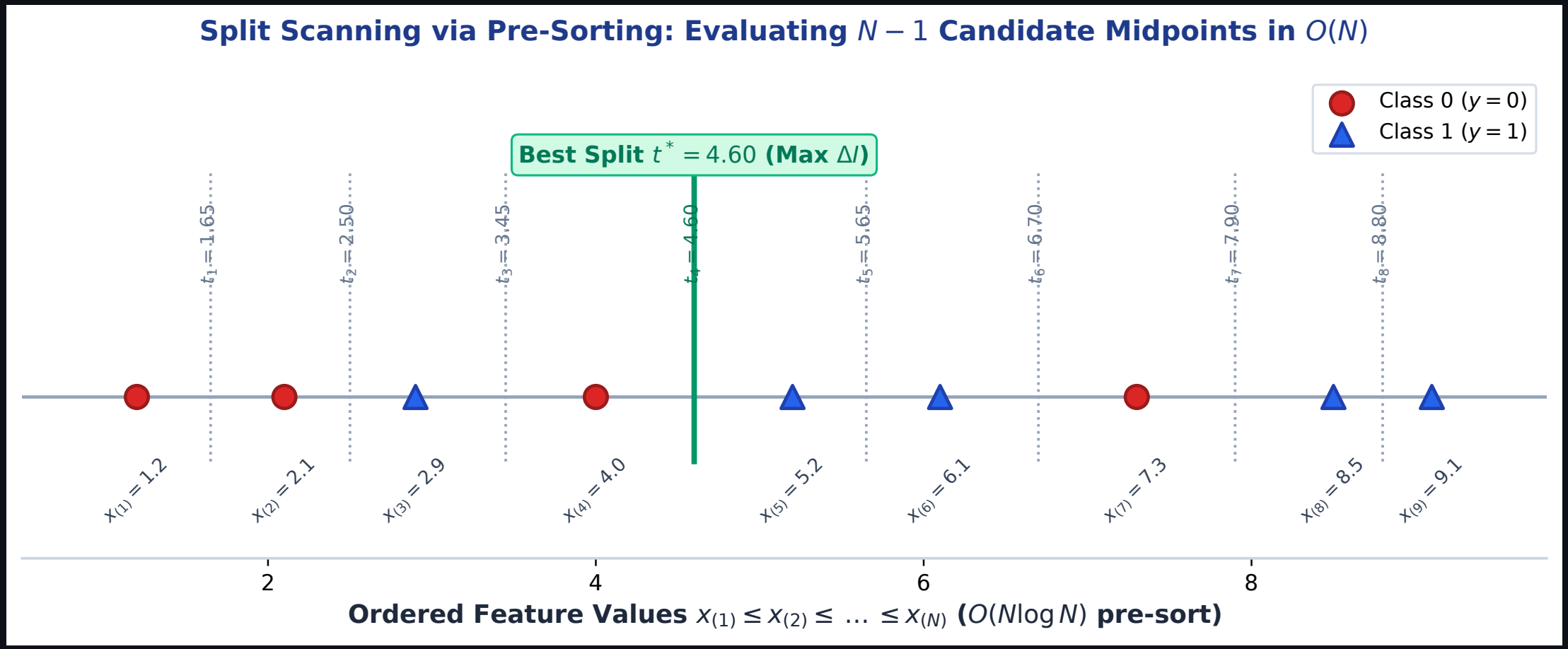
$$H(D) = -(0.6 \log_2 0.6 + 0.4 \log_2 0.4)$$

$$H(D) = -(0.6 \cdot (-0.737) + 0.4 \cdot (-1.322))$$

$$H(D) = 0.442 + 0.529 = \mathbf{0.971} \text{ bits}$$

Scanning...

For continuous feature `Study Hours`: $x_1 \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$.
Candidate split thresholds are the midpoints: $t \in \{2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5\}$.



Comparing all candidate splits

Let's evaluate the candidate splits using Gini impurity ($I_G(D) = 0.480$):

Candidate Split	Left Child (N_L)	Right Child (N_R)	Left I_G	Right I_G	Weighted I_G	Gain ΔI_G
Study ≤ 2.5	0 Pass, 1 Fail	6 Pass, 3 Fail	0.000	0.444	0.400	+0.080
Study ≤ 3.5	0 Pass, 2 Fail	6 Pass, 2 Fail	0.000	0.375	0.300	+0.180
Study ≤ 4.5	**0 Pass, 3 Fail**	**6 Pass, 1 Fail**	**0.000**	**0.245**	**0.171**	**+0.309 ★**
Study ≤ 5.5	1 Pass, 3 Fail	5 Pass, 1 Fail	0.375	0.278	0.317	+0.163
Exam Prep =						

$$\Delta I_G(\text{Study} \leq 4.5) = 0.480 - \left[\frac{3}{10}(0.0) + \frac{7}{10}(0.245)\right] = 0.480 - 0.171 = +\mathbf{0.309}$$

> **Winner:** The algorithm chooses `Study Hours <= 4.5`, well done Steve.

Building the Tree: The Next Step

Following the winning split `Study Hours <= 4.5`:

- **Left Child ($N = 3$):** Students S1, S2, S3 (all Fail).
  - Impurity $I_G = 0.000 \implies$ **Pure leaf node! Prediction: Fail.**
- **Right Child ($N = 7$):** Students S4 through S10 (6 Pass, 1 Fail).
  - Impurity $I_G = 0.245$.
  - The algorithm now recurses on this subset:
  - Scanning candidate splits on the remaining 7 students isolates S5 (who did not attend Exam Prep, Study = 6.0, Fail).
  - Splitting on `Exam Prep == Yes` separates S5 ($N = 1$, Fail) from the rest ($N = 6$, 100% Pass)!

Decision Tree Demo

DT Decision Tree Induction Explorer

Step 0 / 2 | DATASET: Student Admissions (Continuous & Cat) | CRITERION: Gini Impurity | ◀ Prev | Next Split ▶ | ▶ Auto-Play | ↺ Reset

NODE TRAINING SAMPLES — Node N = 10

#	GPA	Interview	Projects	Admit
1	3.8	Pass	4	Yes
2	3.7	Pass	3	Yes
3	3.6	Pass	2	Yes
4	3.4	Pass	5	Yes
5	3.5	Fail	1	No
6	3.1	Pass	1	No
7	2.9	Fail	3	No
8	2.8	Fail	0	No
9	3.9	Fail	4	Yes
10	3	Fail	2	No

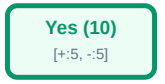
EVALUATED CANDIDATE SPLITS — Gini(D) = 0.500

Testing: GPA ≤ 3.25 — ΔGini = +0.333

ΔI = 0.500 - [(4/10)×0.000 + (6/10)×0.278]

CONDITION	LEFT SUBTREE	RIGHT SUBTREE	GAIN (ΔI)	STATUS
GPA ≤ 3.25 ★ Best	4 pts (I=0.00)	6 pts (I=0.28)	+0.333	Selected
GPA ≤ 3.55	6 pts (I=0.28)	4 pts (I=0.00)	+0.333	Suboptimal
GPA ≤ 3.05	3 pts (I=0.00)	7 pts (I=0.41)	+0.214	Suboptimal
GPA ≤ 3.65	7 pts (I=0.41)	3 pts (I=0.00)	+0.214	Suboptimal
Projects ≤ 1.5	3 pts (I=0.00)	7 pts (I=0.41)	+0.214	Suboptimal
Projects ≤ 3.5	7 pts (I=0.41)	3 pts (I=0.00)	+0.214	Suboptimal
GPA ≤ 3.45	5 pts (I=0.32)	5 pts (I=0.32)	+0.180	Suboptimal
Interview = 'Pass'	5 pts (I=0.32)	5 pts (I=0.32)	+0.180	Suboptimal
Interview = 'Fail'	5 pts (I=0.32)	5 pts (I=0.32)	+0.180	Suboptimal
Projects ≤ 2.5	5 pts (I=0.32)	5 pts (I=0.32)	+0.180	Suboptimal
GPA ≤ 2.95	2 pts (I=0.00)	8 pts (I=0.47)	+0.125	Suboptimal
GPA ≤ 3.75	8 pts (I=0.47)	2 pts (I=0.00)	+0.125	Suboptimal
GPA ≤ 2.85	1 pts (I=0.00)	9 pts (I=0.49)	+0.056	Suboptimal
GPA ≤ 3.85	9 pts (I=0.49)	1 pts (I=0.00)	+0.056	Suboptimal

CONSTRUCTED TREE — Active: Root



Left Child (Condition True) | Right Child (Condition False) | Optimal Split (Max ΔI)

Tip: Click any candidate row in the centre table to preview its partition.

What About Regression Trees?

Suppose target y is continuous (e.g. `Exam Score` $\in [0, 100]$ or `Salary` in £k).

Toy Example: Experience vs. Salary (£k)

Consider 5 software engineers:

Experience (x)	**1 yr**	**2 yrs**	**3 yrs**	**7 yrs**	**8 yrs**
Salary (y)	£30k	£35k	£40k	£80k	£95k

- **Parent Node:** Mean $\bar{y} = \frac{30+35+40+80+95}{5} = \mathbf{£56k}$.
- **Parent Variance:** $\sigma^2(D) = \frac{1}{5}\sum(y_i - 56)^2 = \mathbf{694.0}$.
- **Candidate Split at $x \leq 5.0$:**
  - Left child ($x \leq 5$): $\{30, 35, 40\} \implies \bar{y}_L = £35\text{k},\ \sigma^2_L = \mathbf{16.7}$.
  - Right child ($x > 5$): $\{80, 95\} \implies \bar{y}_R = £87.5\text{k},\ \sigma^2_R = \mathbf{56.3}$.

Regression = Variance Reduction

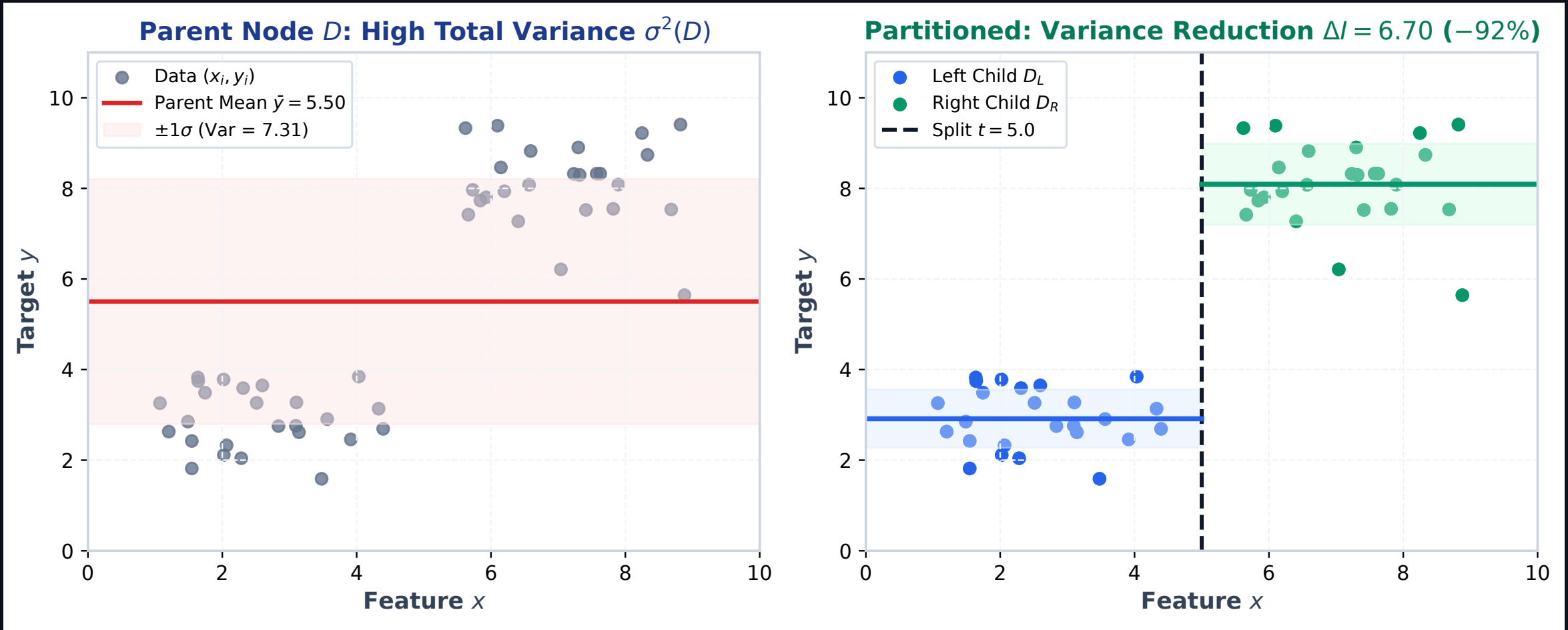
Weighted child variance:

$$\mathrm{Var}_{\text{split}} = \frac{3}{5}(16.7) + \frac{2}{5}(56.3) = 10.0 + 22.5 = \mathbf{32.5}$$

Variance reduction (gain):

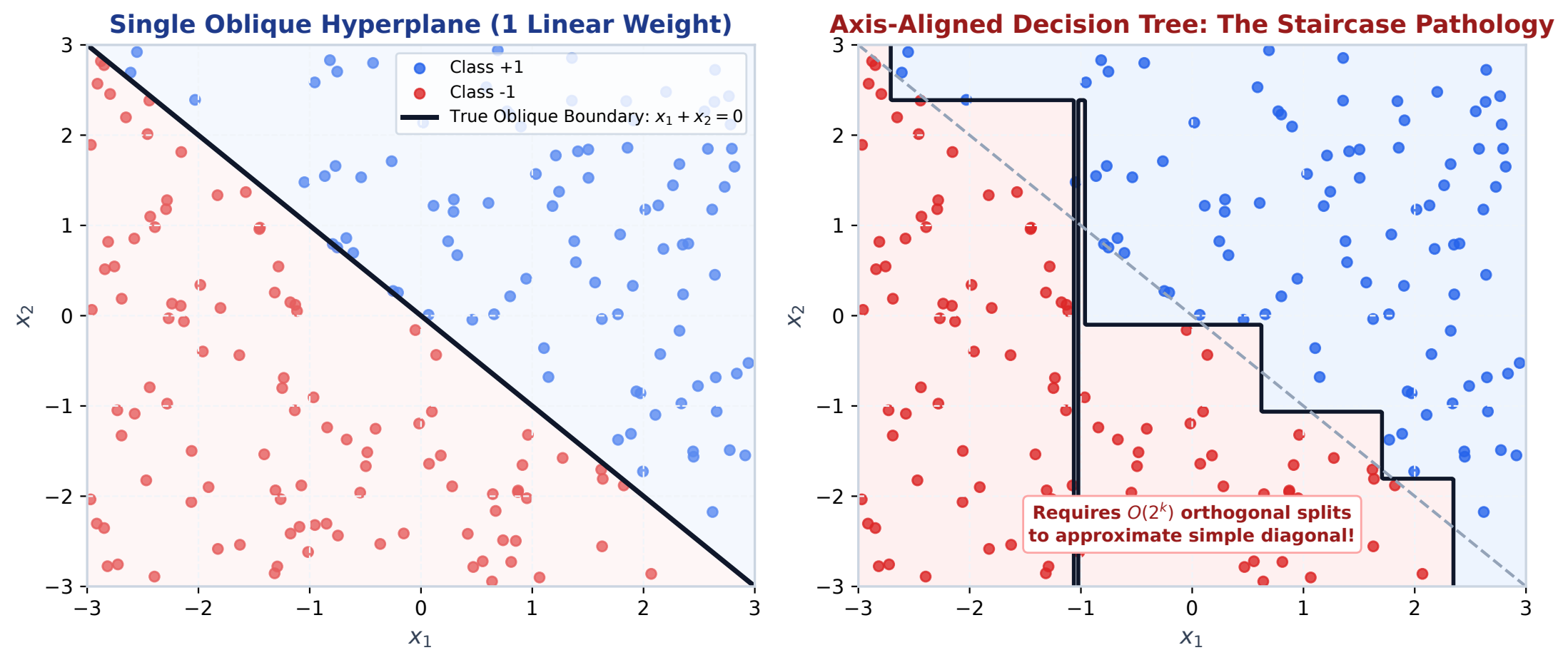
$$\Delta I = \sigma^2(D) - \mathrm{Var}_{\text{split}} = 694.0 - 32.5 = \mathbf{+661.5} \quad (\mathbf{-95.3\%})$$

> Here variance is used as a measure of impurity.


Parent Node D: High Total Variance $\sigma^2(D)$
Data (x_i, y_i)
Parent Mean $\bar{y} = 5.50$
$\pm 1\sigma$ (Var = 7.31)
Target y
Feature x
Partitioned: Variance Reduction $\Delta I = 6.70$ (−92%)
Left Child D_L
Right Child D_R
Split $t = 5.0$
Target y
Feature x

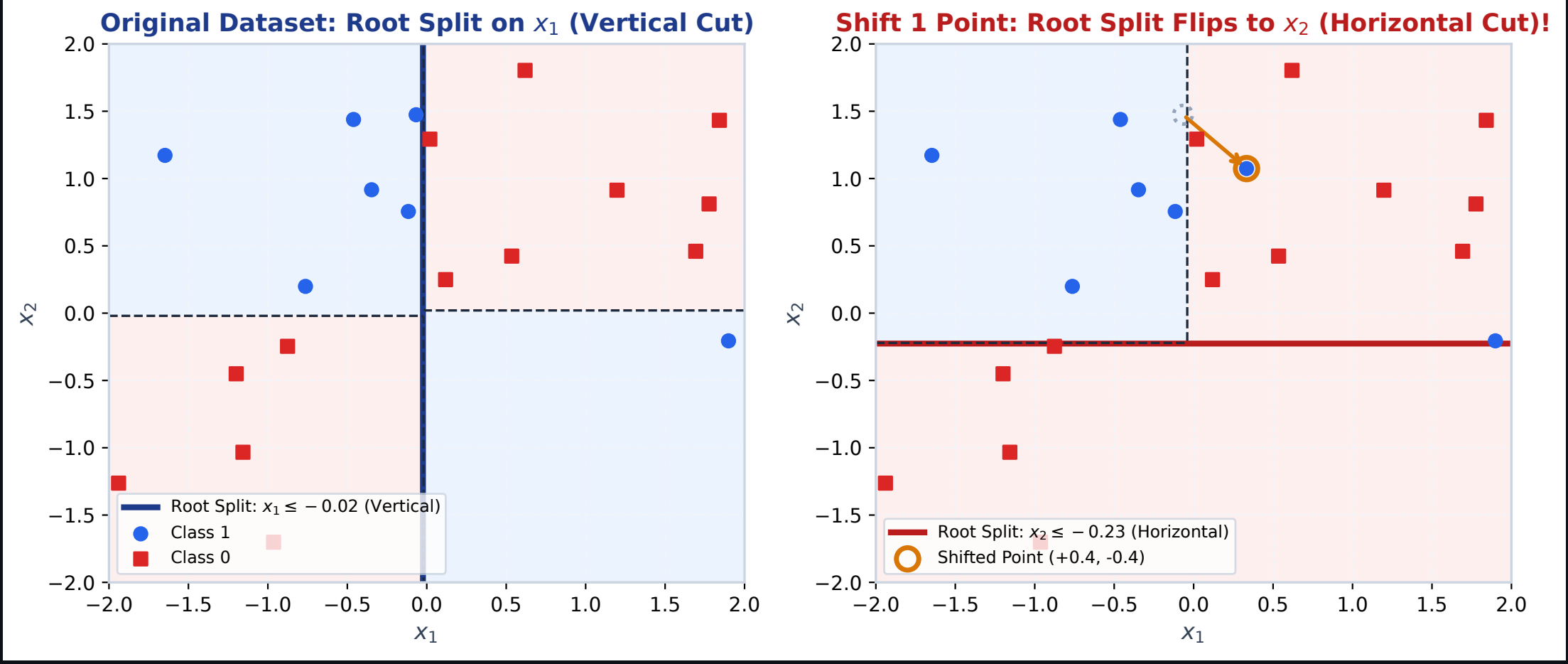
Failure Mode 1: The Diagonal Boundary

If the true boundary is diagonal (not axis-aligned). A decision tree becomes a **staircase**:



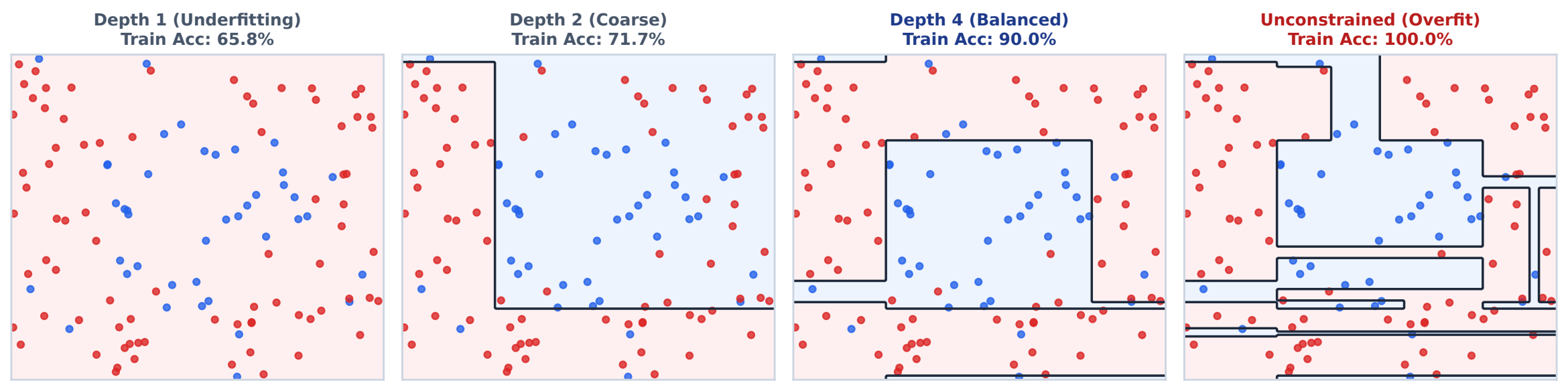
Failure Mode 2: High Variance & Instability

When competing splits have similar gain, shifting just **1 data point** flips the root cut from vertical (x_1) to horizontal (x_2)—rotating the entire partition and altering downstream decisions:



Failure Mode 3: Overfitting Dynamics

If unconstrained, trees split until every leaf contains potentially 1 sample.



Pre-Pruning & The Horizon Effect

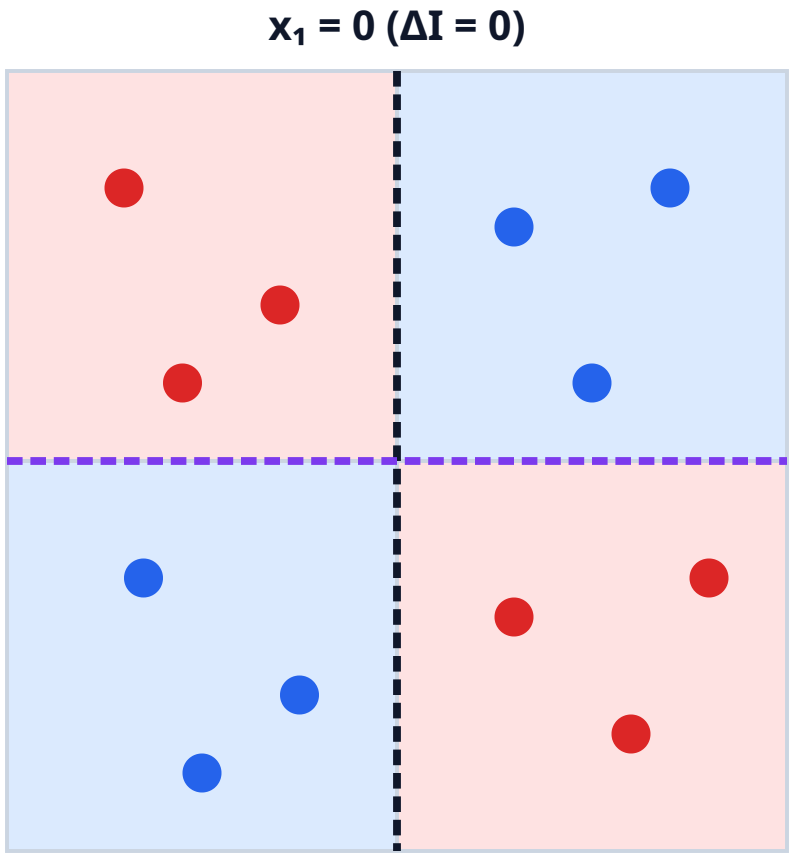
To prevent overfitting, can we simply stop growing when $\Delta I \leq \varepsilon$?

The XOR Trap

Consider 4 points: $(0,0) \to \text{Red}$, $(1,1) \to \text{Red}$, $(0,1) \to \text{Blue}$, $(1,0) \to \text{Blue}$.

Candidate 1-Step Cut	Left Child	Right Child	Gain ΔI
Split on $x_1 = 0.5$	1 Red, 1 Blue	1 Red, 1 Blue	**0.000**
Split on $x_2 = 0.5$	1 Red, 1 Blue	1 Red, 1 Blue	**0.000**

- Every single 1-step split yields **zero impurity decrease**!
- **Greedy** early stopping halts immediately at the root.
- Yet, splitting to depth 2 achieves **100% purity**.



Greedy 1-Step Gain = 0

Common Pre-Pruning Heuristics

- **Minimum Impurity Decrease ($\Delta I \leq \varepsilon$):**
  Stops when single-step information gain vanishes.
- **Maximum Depth (`max_depth`):**
  Imposes a hard cap on overall tree depth.
- **Minimum Leaf Samples (`min_samples_leaf`):**
  Ensures support at every prediction leaf.

Post-Pruning: Minimal Cost-Complexity

Pre-pruning can stop too early (horizon effect). So we take a two-stage approach:

1. **Grow Deep:** Grow the full unconstrained tree $T_{\max}$ down to purity (uncovering multi-step feature interactions).
2. **Prune Back:** Retrospectively collapse subtrees that don't justify their complexity.

The Regularised Objective

For any subtree $T \subseteq T_{\max}$, define the cost:

$$\mathrm{Cost}_\alpha(T) = \mathrm{Error}(T) + \alpha|T|$$

- $\mathrm{Error}(T)$: Training error (misclassification rate or MSE).
- $|T|$: Number of terminal leaf nodes (model complexity).
- $\alpha \geq 0$: Complexity parameter (the **price per leaf**).

The Intuition: What is a Branch Worth?

We cannot search all subtrees (combinatorial explosion). So we ask a local question:

> *"If we collapse an internal branch T_t back into a single leaf t, how much accuracy do we lose per leaf saved?"*

- **Keeping T_t:** Contributes error $\mathrm{Error}(T_t)$ and complexity $\alpha|T_t|$.
- **Collapsing T_t to leaf t:** Contributes error $\mathrm{Error}(t)$ and complexity $\alpha \times 1$.

They have identical cost when $\mathrm{Error}(t) + \alpha = \mathrm{Error}(T_t) + \alpha|T_t|$, which gives:

$$\alpha = \frac{\mathrm{Error}(t) - \mathrm{Error}(T_t)}{|T_t| - 1} \equiv g(t)$$

- If a branch uses 4 leaves to correct 1 noisy sample, $g(t)$ is tiny $\implies$ **cheap to prune**.
- If a branch separates 50 real samples, $g(t)$ is huge $\implies$ **expensive to prune (keep it!)**.

Calculating $g(t)$

An internal node t contains 20 training samples (12 Pass, 8 Fail) out of $N = 100$ total samples:

If Collapsed to Leaf t

- Predicts majority: Pass.
- Misclassifies 8 Fail samples.
- Training error:

$$\text{Error}(t) = \frac{8}{100} = \mathbf{0.08}$$

- Leaves: $|t| = 1$

If Kept as T_t

- Subtree has $|T_t| = \mathbf{4}$ leaves.
- Across all 4 leaves, it only misclassifies 2 samples:

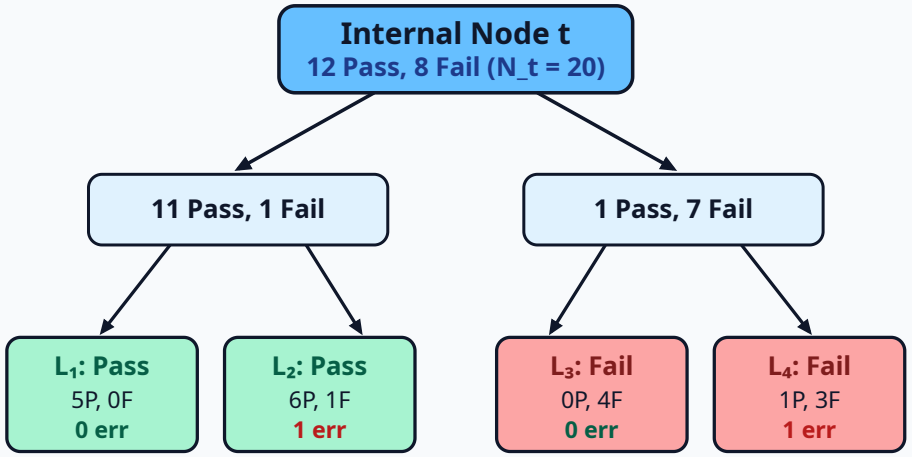
$$\text{Error}(T_t) = \frac{2}{100} = \mathbf{0.02}$$

- Leaves saved if pruned: $4 - 1 = \mathbf{3}$

Weakest Link Collapse: $T_t \rightarrow t$

Subtree Branch T_t (|T_t| = 4 leaves)

Error(T_t) = 2/100 = 0.02 (2 errors across 20 samples)



Total leaves: 4 • Total misclassified: 2

Collapse

ΔError = +0.06
ΔLeaves = 3
g(t) = 0.02

Pruned Leaf Node t (|t| = 1 leaf)

Error(t) = 8/100 = 0.08 (8 errors across 20 samples)

Single Leaf Node t

12 Pass, 8 Fail (N_t = 20)

Majority Prediction: Pass

8 Fail samples misclassified

Leaves saved: 4 − 1 = 3 • Error increase: 8 − 2 = 6

Calculating $g(t)$

$$g(t) = \frac{\text{Error}(t) - \text{Error}(T_t)}{|T_t| - 1} = \frac{0.08 - 0.02}{4 - 1} = \frac{0.06}{3} = \mathbf{0.02}$$

- When $\alpha < 0.02$: Keeping the 4 leaves is worth the complexity penalty.
- When $\alpha \geq 0.02$: The penalty outweighs the gain $\implies$ Subtree T_t collapses into leaf t!

```
# 1. Start with the fully grown tree
T = T_max
trees = [T]
alphas = [0.0]

while len(internal_nodes(T)) > 0:
    # 2. Compute g(t) for every internal node currently in T
    #    g(t) = (error(t) - error(T_t)) / (|T_t| - 1)
    g_scores = { t: compute_effective_alpha(t, T) for t in internal_nodes(T) }

    # 3. Find the weakest link (smallest g(t))
    t_star = min(g_scores, key=g_scores.get)
    alpha_next = g_scores[t_star]

    # 4. Collapse subtree T_t* into a single leaf node
    T = collapse_subtree(T, t_star)

    trees.append(clone(T))
    alphas.append(alpha_next)

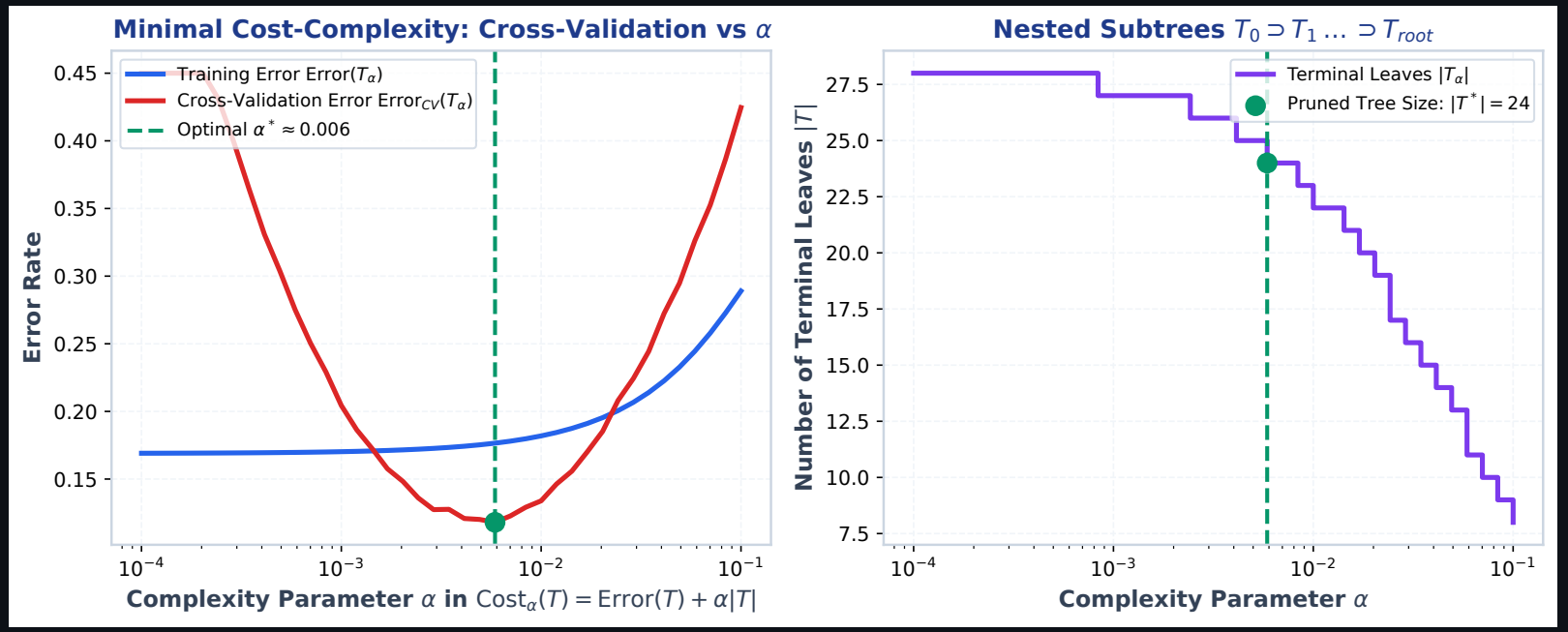
# Produces a finite nested sequence: T_0 ⊃ T_1 ⊃ T_2 ⊃ ... ⊃ T_root
```

When to stop?

Optimal Tree via Cross-Validation

Pruning yields a sequence of nested subtrees: $T_0 \supset T_1 \supset \cdots \supset T_{\text{root}}$ and intervals $[0, \alpha_1), [\alpha_1, \alpha_2), \ldots$

- **Why not use training error?** Training error strictly favours $T_{\max}$ ($\alpha = 0$).
- We evaluate each candidate subtree T_k on a held-out validation set (or via K-fold cross-validation) to find the optimal α^*:



The Bridge to Ensembles

Single decision trees have a dilemma:

- Unpruned: **Low bias, high variance** $\Longrightarrow$ Overfitting
- Pruned: **High bias, low variance** $\Longrightarrow$ Underfitting

Can we combine multiple diverse models to produce a single superior model?

Intuition: Tree Bias vs. Tree Variance

Imagine predicting whether a student passes based on `Study Hours` and `Exam Prep`:

1 Node Tree (Pruned)

- A single crude cut: *Study* ≤ 4.5.
- Completely ignores `Exam Prep`.
- If we resample students tomorrow, the split stays roughly the same ($\downarrow$ **low variance**)…
- …but it consistently misclassifies borderline students ($\uparrow$ **high bias**).

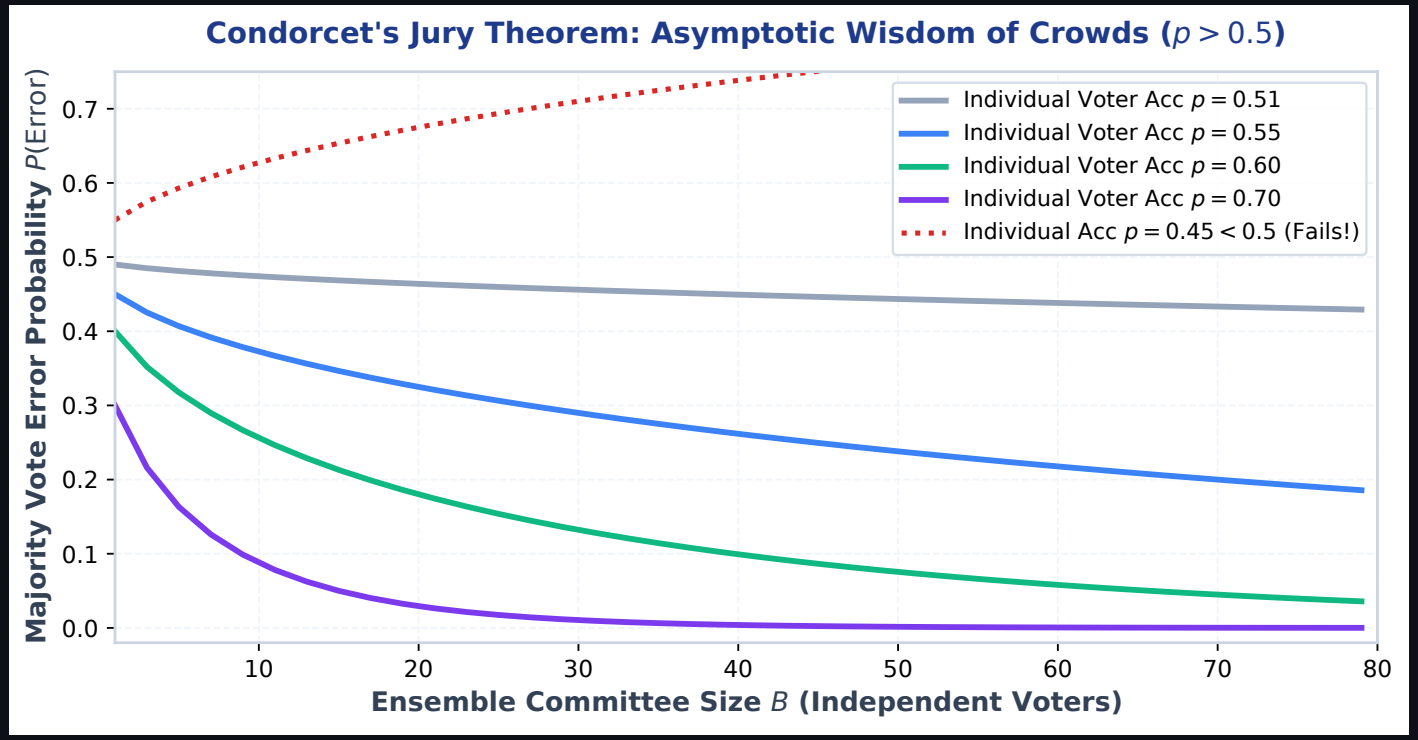
Deep Unpruned Tree

- Tiny custom boxes around every single training sample ($\downarrow$ **low bias**).
- If one student with 6 hours had the flu and failed, the tree memorises it.
- Resample students tomorrow, and the entire branch structure reshuffles ($\uparrow$ **high variance**)!

Condorcet's Jury Theorem (1785)

Suppose B independent voters each have probability $p > 0.5$ of being correct. The majority vote accuracy converges to 1 as $B \to \infty$:

$$\mathbb{P}(\text{Majority Correct}) = \sum_{k=\lceil B/2 \rceil}^{B} \binom{B}{k} p^k (1-p)^{B-k}$$



Bagging (Bootstrap Aggregation)

We don't have infinite voters. We have one dataset D of size N:

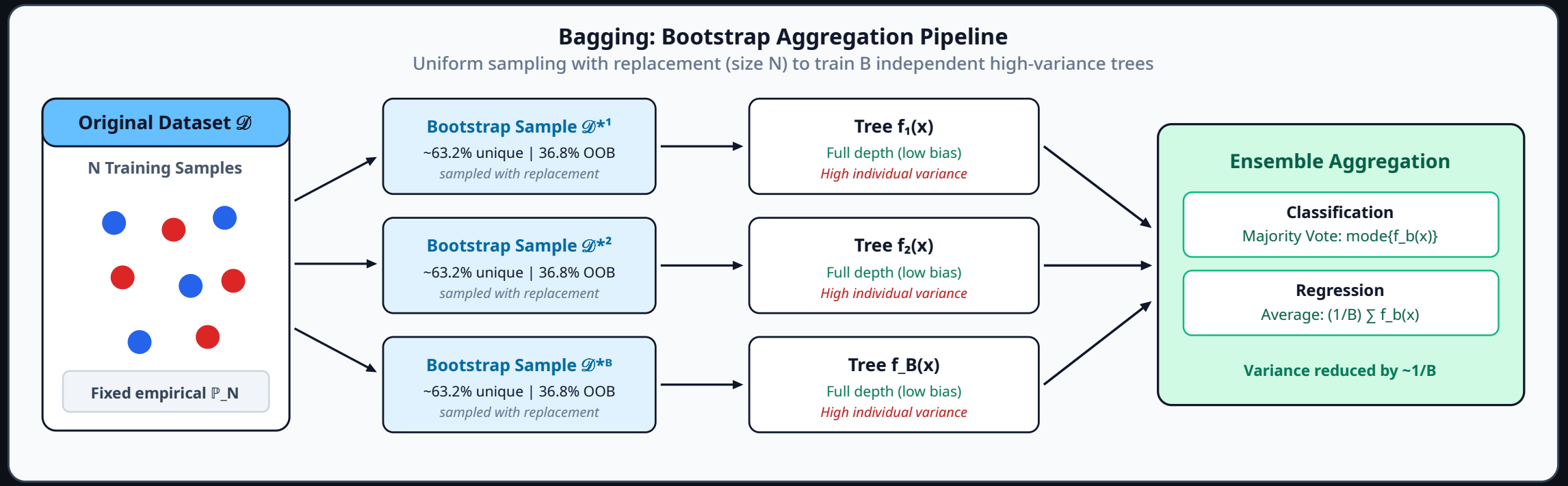
1. Draw B bootstrap datasets $D^{*1}, \ldots, D^{*B}$ by uniformly sampling N points **with replacement**.
2. Train an unpruned deep decision tree $f_b(\mathbf{x})$ on each bootstrap sample.
3. Aggregate predictions:
    - **Classification:** Majority vote.
    - **Regression:** Average $\hat{f}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x})$.

Why it works

The probability of omitting a specific sample across N draws is:

$$\lim_{N \to \infty} \left(1 - \frac{1}{N}\right)^N = \frac{1}{e} \approx 36.8\%$$

- **In-Bag ($\approx 63.2\%$):** Each tree trains on a distinct random subset $\implies$ ensures diversity.
- **Out-of-Bag (OOB, $\approx 36.8\%$):** Over a third of samples are held out for any given tree.



By forcing each tree to see only $\approx 63.2\%$ of unique points, the trees learn slightly different split boundaries. This drives the correlation between trees (ρ) down

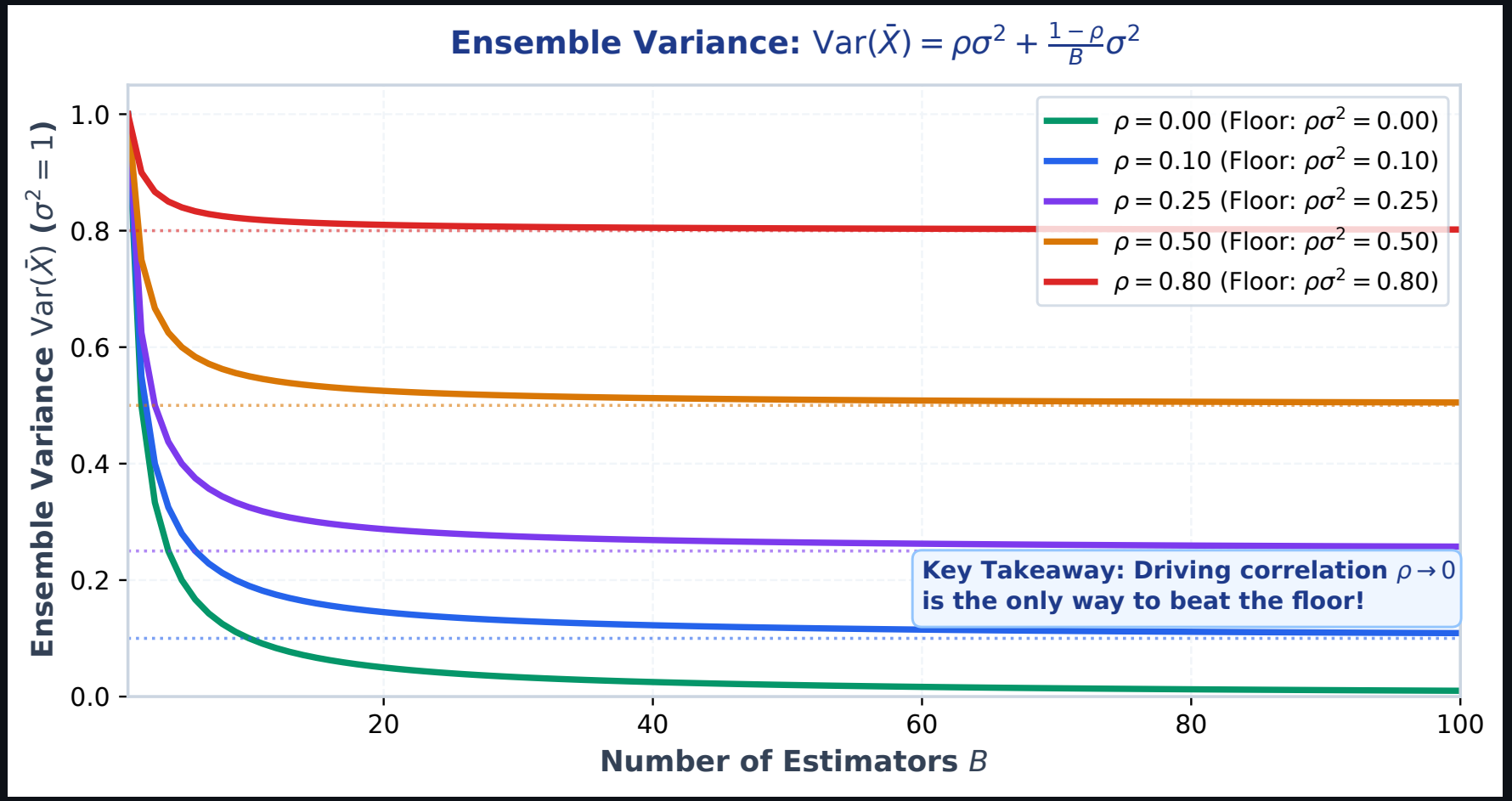
The Limit of Bagging

Why doesn't Bagging reduce variance to zero?

For B estimators with individual variance σ^2 and pairwise correlation $\rho = \text{Corr}(f_i, f_j)$:

$$\text{Var}\left(\frac{1}{B}\sum_{i=1}^{B} f_i\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

- As $B \to \infty$, the second term $\frac{1-\rho}{B}\sigma^2$ drops to 0.
- But the first term $\rho\sigma^2$ **remains as is**!
- **The issue:** If a dataset has a dominant feature (e.g. `Study Hours`), every bagged tree splits on it at the root $\implies$ ρ remains high.



> Intuition: decorrelation means they predict different things. So they can correct each other's mistakes.

Random Forests: De-Correlating Trees

At **each split** in **every tree**:

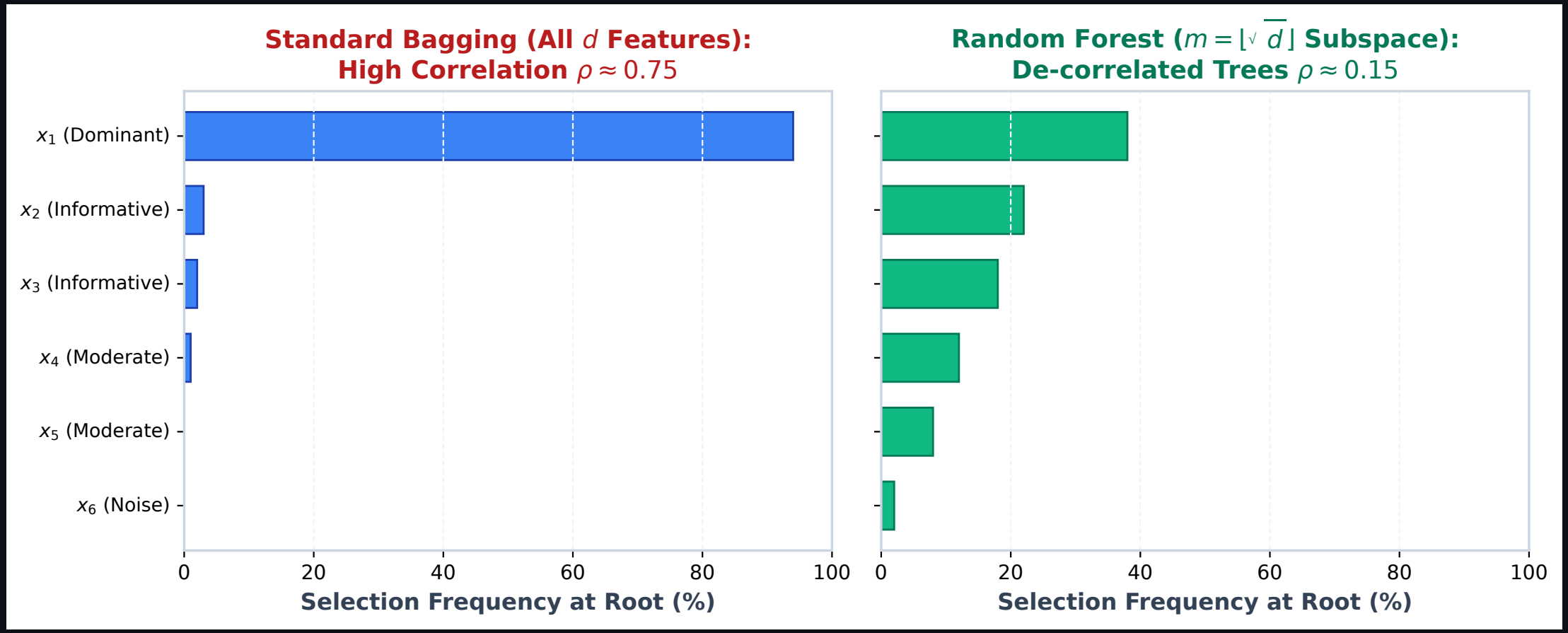
- Instead of scanning all d features, randomly select a subset of $m < d$ features.
- Best split is chosen **only from this restricted subset**.

That's it...

Standard Heuristics in Practice:

- **Classification:** $m = \lfloor \sqrt{d} \rfloor$ (e.g., $d = 100 \implies$ pick from only 10 random candidate features per split).
  - *Why so few?* Majority voting is resilient to individual noisy trees; aggressive feature restriction forces exploration, driving correlation $\rho \to 0$.
- **Regression:** $m = \lfloor d/3 \rfloor$ (e.g., $d = 100 \implies$ pick from 33 random candidate features per split).
  - *Why more features?* Numerical averaging requires each tree to maintain reasonable predictive accuracy; severe restriction degrades regression fits.

> **Trade-off (m):** Smaller $m \implies$ lower correlation ρ (lower floor), but higher individual tree bias σ^2. These standard heuristics strike a practical balance.


Standard Bagging (All d Features):
High Correlation $\rho \approx 0.75$
x_1 (Dominant)
x_2 (Informative)
x_3 (Informative)
x_4 (Moderate)
x_5 (Moderate)
x_6 (Noise)
0
20
40
60
80
100
Selection Frequency at Root (%)
Random Forest ($m = \lfloor \sqrt{d} \rfloor$ Subspace):
De-correlated Trees $\rho \approx 0.15$
0
20
40
60
80
100
Selection Frequency at Root (%)

Feature Importance: MDI vs. Permutation

How do we interpret an ensemble of 500 trees?

- **The Motivation (Black Box Dilemma):** A single tree is transparent, but a 500-tree forest has tens of thousands of splits. In healthcare, lending, or scientific research, we need to know: *which features genuinely drive the predictions?*

Mean Decrease Impurity (MDI)

- Sum of all impurity drops ΔI on feature j across all trees, weighted by sample counts.
- **Fast & Built-in** (`rf.feature_importances_`).
- **Pitfall:** Computed on **training data**; heavily biased towards continuous or high-cardinality noise features!

> A feature that offers many splits can have a chance of creating a predictive split more often than it is truly important.

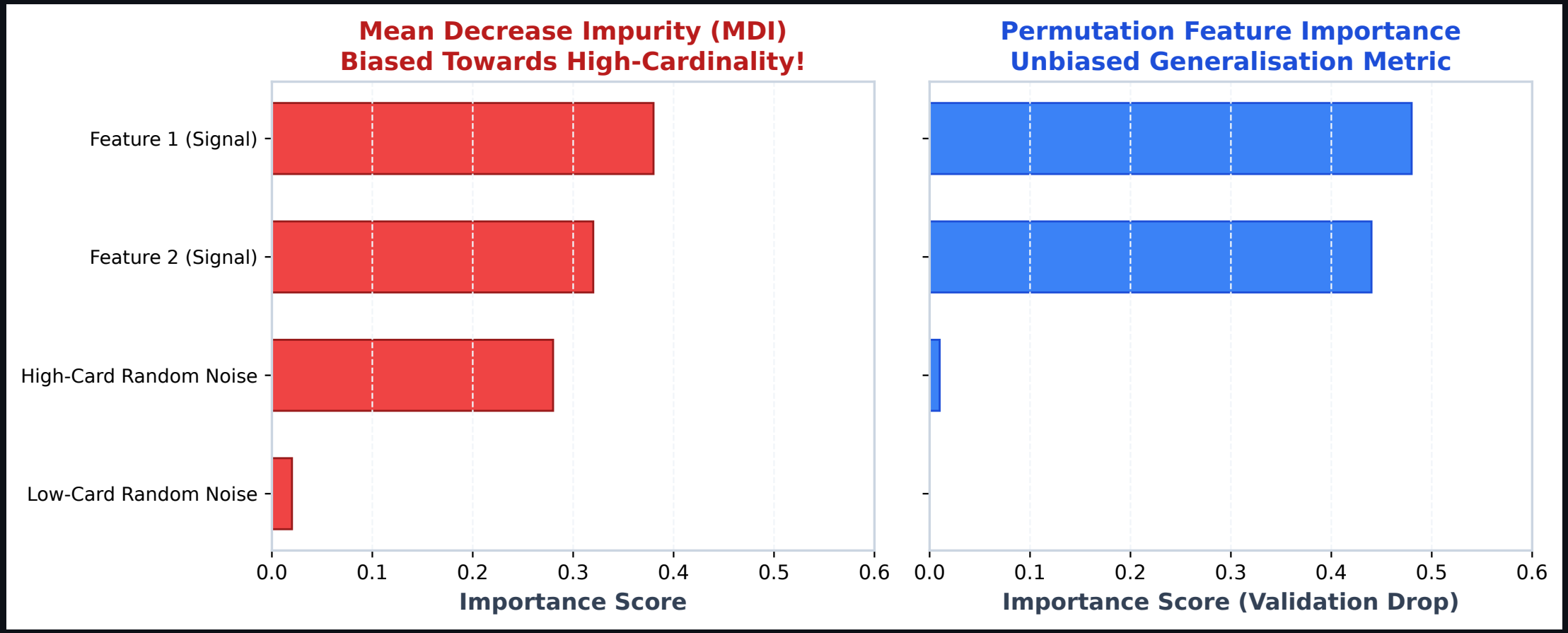
Permutation Importance (PFI)

- Randomly shuffle the feature on **test data**, breaking its relationship with y.
- Measure drop in model score: $\Delta\text{Score} = \text{Score}_{\text{orig}} - \text{Score}_{\text{shuffled}}$.

```
Baseline Test Set:
[ Feature A | Feature B | Feature C ]  ──▶ Predict ──▶ Score: 0.92

Permuting Feature B (Shuffling column values across rows):
[ Feature A | Feature B (shuffled) | Feature C ]  ──▶ Predict ──▶ Score: 0.74

Importance of Feature B = 0.92 - 0.74 = +0.18
```



> E.g. high cardinality features such as `zip-code`, `transaction_id` will score high on MDI but low on PFI.

Bagging vs Boosting

So far we looked at bagging.

Bagging Philosophy (Variance)

- **Parallel Committee of Experts.**
- Build deep, low-bias trees independently.
- Aggregate predictions to **reduce variance**.

Boosting Philosophy (Bias)

- **Sequential Relay of Apprentices.**
- Build shallow, high-bias trees (depth 1–3).
- Train each learner on the **leftover errors (residuals)** of the team to **reduce bias**.

The Additive Model

$$\hat{y}^{(T)} = \hat{y}^{(0)} + \sum_{t=1}^{T} \eta \cdot T_t(\mathbf{x}) \quad \iff \quad \hat{y}^{(t)} = \hat{y}^{(t-1)} + \eta \cdot T_t(\mathbf{x})$$

- $\hat{y}^{(0)}$: **Initial base prediction**, typically the constant target mean $\bar{y}$.
- $T_t(\mathbf{x})$: **Weak decision tree** trained to predict current residuals $r^{(t)} = y - \hat{y}^{(t-1)}$.
- $\eta \in (0, 1]$: **Shrinkage learning rate**, scales each tree's update to prevent overfitting.

```
# Initialise with baseline guess
y_pred = mean(y)

# Sequentially correct errors
for t in range(1, T + 1):
    residual = y - y_pred
    tree = fit_tree(X, target=residual)
    y_pred += eta * tree.predict(X)
```

Toy Example: Boosting on Residuals

Suppose we want to predict exam scores for 4 students:

Student	True Score (y)	Initial Base $\hat{y}^{(0)} = \bar{y}$	Residual $r^{(1)} = y - \hat{y}^{(0)}$	Stump $T_1(\mathbf{x})$	Updated $\hat{y}^{(1)} = \hat{y}^{(0)} + 0.5T_1$
A	20	50	**-30**	-20	$50 - 10 = \mathbf{40}$
B	40	50	**-10**	-20	$50 - 10 = \mathbf{40}$
C	60	50	**+10**	+20	$50 + 10 = \mathbf{60}$
D	80	50	**+30**	+20	$50 + 10 = \mathbf{60}$

- Notice the errors after Step 1: $y - \hat{y}^{(1)} = [-20, 0, 0, +20]$.
- The residual magnitude has halved! Next stump T_2 will fit these remaining errors.

Residuals Are Gradients in Disguise

In our toy example, we nudged predictions by fitting errors $r = y - \hat{y}$.

Notice what happens if we take the derivative of **Squared Error Loss**:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2 \implies -\frac{\partial \mathcal{L}}{\partial \hat{y}} = y - \hat{y}$$

- The residual $y - \hat{y}$ is **literally the negative gradient**!
- Nudging predictions by residuals is **Gradient Descent in prediction space**:

$$\hat{y} \leftarrow \hat{y} - \eta \frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} + \eta(y - \hat{y})$$

- To generalise to unseen points $\mathbf{x}$, we train a tree $T_t(\mathbf{x})$ to predict these gradients:

$$\hat{y}^{(t)}(\mathbf{x}) = \hat{y}^{(t-1)}(\mathbf{x}) + \eta \cdot T_t(\mathbf{x})$$

Gradient Boosted Decision Trees (GBDT)

Because residuals are just negative gradients, **this works for any differentiable loss $\mathcal{L}$**!

Jerome Friedman (2001) generalised boosting by replacing residuals with negative gradients:

$$r_i^{(t)} = -\left[\frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial \hat{y}_i}\right]_{\hat{y}_i = \hat{y}_i^{(t-1)}}$$

Squared Error (Regression)

$$-\frac{\partial \mathcal{L}}{\partial \hat{y}} = y - \hat{y}$$

Fits the standard residual error.

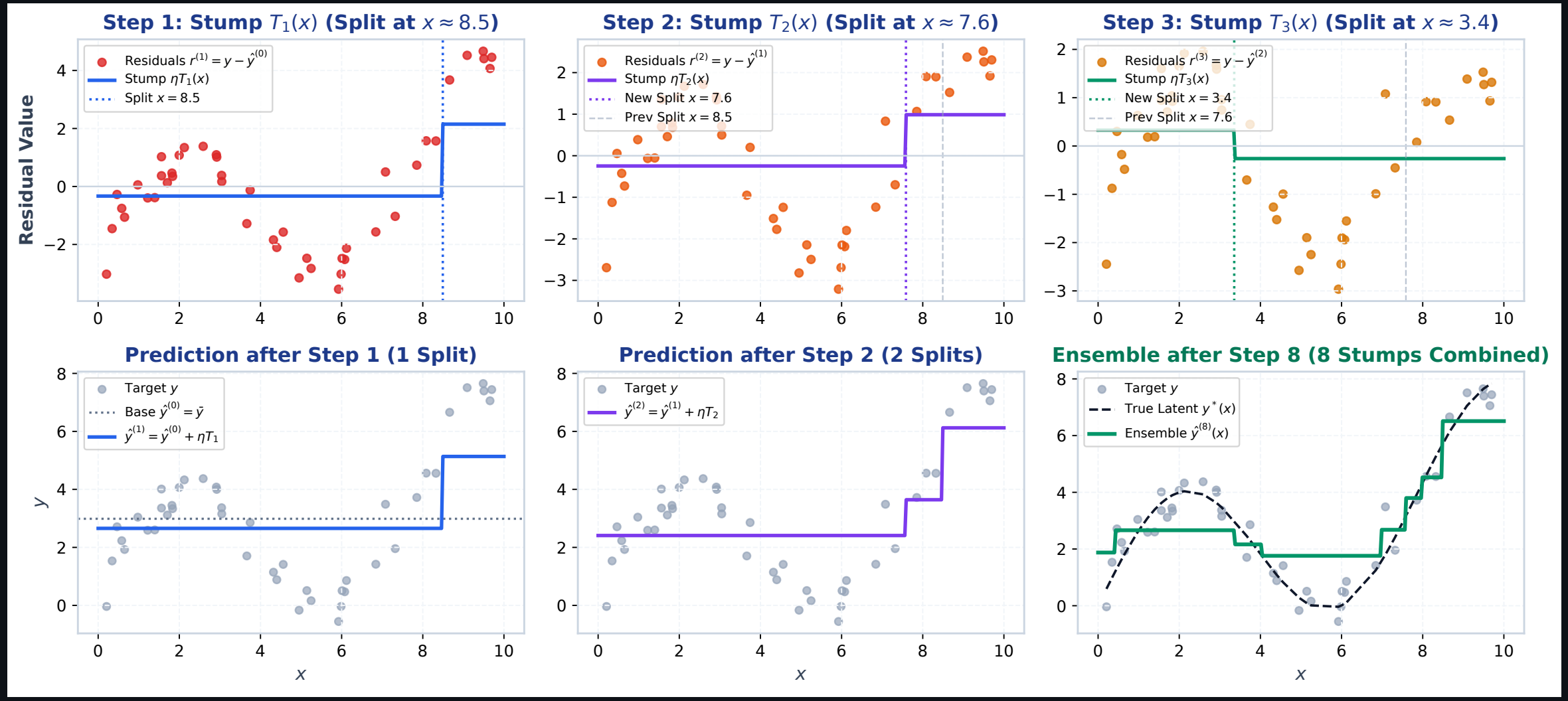
Cross-Entropy (Classification)

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

Fits true label minus predicted probability!

> **The GBDT Principle:** We don't differentiate the trees; we differentiate the loss with respect to our predictions, and train trees to point us downhill!

Visualising GBDT for Regression



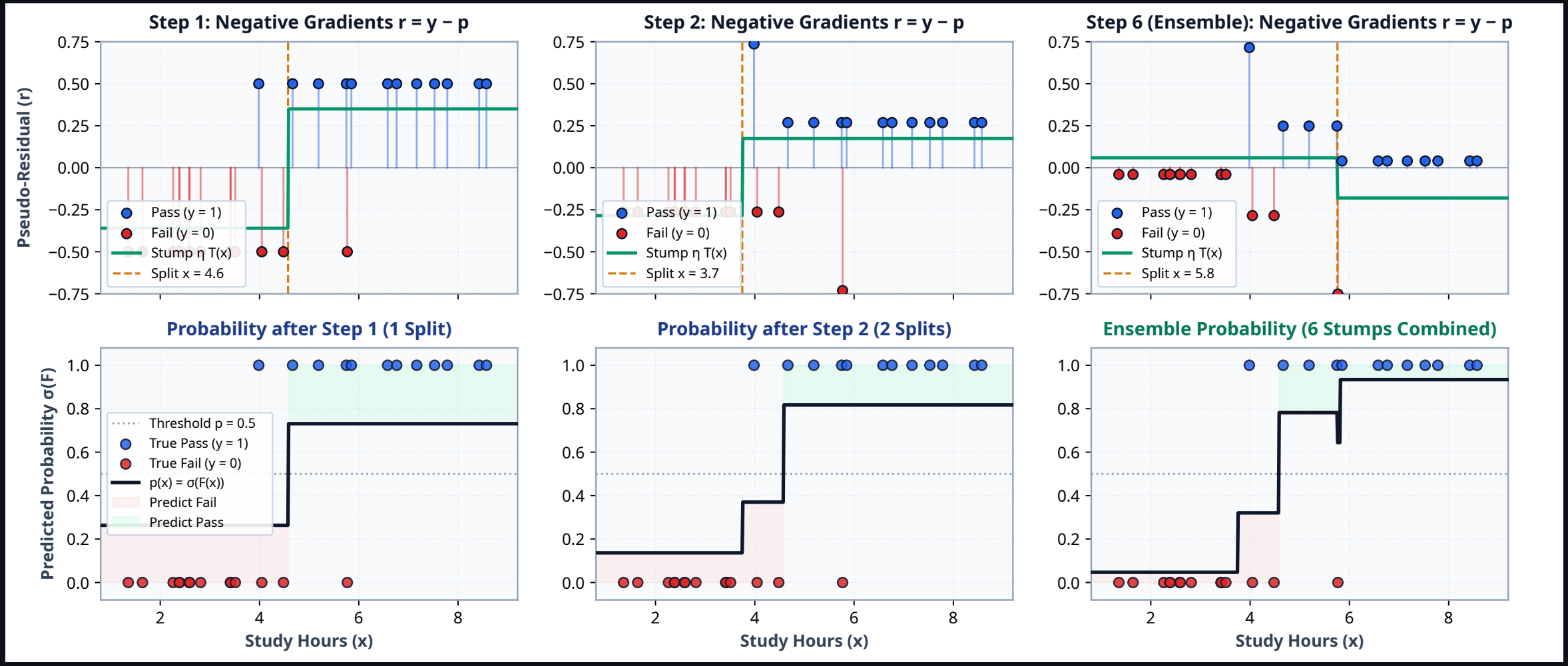
Toy Example: GBDT for Binary Classification

Suppose we want to predict exam success ($y \in \{0, 1\}$) from study hours. Initialise with baseline $\hat{y}^{(0)} = 0 \implies p^{(0)} = \sigma(0) = \mathbf{0.50}$ (50/50 prior):

ID	Study (x)	True y	Prob $p^{(0)}$	Residual $r^{(1)} = y - p^{(0)}$	Stump $T_1(x)$	Logits $\hat{y}^{(1)} = \hat{y}^{(0)} + 0.5T_1$	Updated $p^{(1)} = \sigma(\hat{y}^{(1)})$
A	2h	0	0.50	**-0.50**	-2.0	$0 + 0.5(-2.0) = \mathbf{-1.0}$	$\sigma(-1.0) = \mathbf{0.27}$
B	4h	0	0.50	**-0.50**	-2.0	$0 + 0.5(-2.0) = \mathbf{-1.0}$	$\sigma(-1.0) = \mathbf{0.27}$
C	6h	1	0.50	**+0.50**	+2.0	$0 + 0.5(+2.0) = \mathbf{+1.0}$	$\sigma(+1.0) = \mathbf{0.73}$
D	8h	1	0.50	**+0.50**	+2.0	$0 + 0.5(+2.0) = \mathbf{+1.0}$	$\sigma(+1.0) = \mathbf{0.73}$

- Stump T_1 splits at $x = 5.0$, predicting optimal leaf step $\gamma = \frac{\sum r_i}{\sum p_i(1-p_i)} = \pm 2.0$.

Visualising GBDT for Classification



Modern Variants

In practice, competitive tabular ML uses three popular GBDT toolkits:

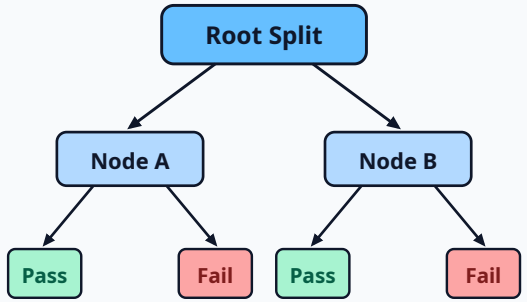
- **XGBoost (2016):** Hardware speedups and regularisation; famous for winning Kaggle competitions.
- **LightGBM (2017):** Feature binning for blazing-fast training on millions of rows.
- **CatBoost (2018):** Automatic handling of categorical variables with minimal tuning.

> Beneath the hood, all three run **Gradient Boosting**.

Decisions Decisions

1. Single CART Tree

Deep unpruned greedy tree

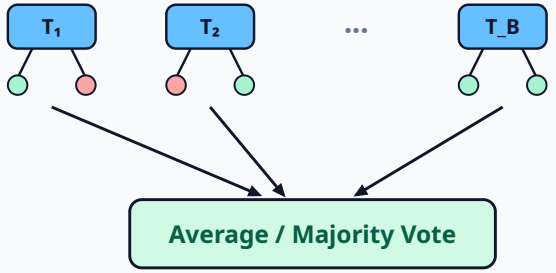


Low Bias, High Variance

- Fits complex boundaries easily
- Highly unstable to data perturbations

2. Random Forest

Parallel bagging + feature subsampling



Variance Reduction (ρ → 0)

- Trees grown independently in parallel
- Random feature subsets m ≈ √d decorrelate

3. Gradient Boosting

Sequential additive descent on residuals



Additive Expansion

$F_M(x) = F_0(x) + \sum \eta\, T_m(x)$

Bias Reduction (Residuals)

- Sequentially corrects prior errors
- Minimises loss $\mathscr{L}$ via gradient steps

Comparison Matrix

Property	Single CART Tree	Random Forest	Gradient Boosting (GBDT)
Primary Goal	Interpretability	**Variance Reduction**	**Bias Reduction**
Base Estimator	Variable depth	**Deep unpruned** (low bias)	**Shallow stumps** (depth 1–4)
Training Flow	Single greedy pass	**Embarrassingly parallel**	**Strictly sequential**
Key Mechanism	Recursive binary splits	Bootstrap + Subspace $m = \sqrt{d}$	Functional gradient residuals
Overfitting Tendency	High	**Low** (Safe as $B \to \infty$)	**Moderate** (Needs early stopping)
Tuning Burden	Low	**Very low** (Works out-of-the-box)	Medium (Tune η, depth, trees)

Implementation in Python

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# 1. Single Decision Tree with Cost-Complexity Pruning
tree = DecisionTreeClassifier(ccp_alpha=0.02, random_state=42)
tree.fit(X_train, y_train)

# 2. Random Forest (Parallel Variance Reduction)
rf = RandomForestClassifier(
    n_estimators=300,
    max_features="sqrt",      # Subspace m = sqrt(d)
    oob_score=True,           # Built-in out-of-bag validation
    n_jobs=-1,                # Parallel CPU cores
    random_state=42
)
rf.fit(X_train, y_train)
print(f"OOB Accuracy: {rf.oob_score_:.3f}")

# 3. Gradient Boosted Trees (Sequential Bias Reduction)
gbdt = GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.05,       # Shrinkage eta
    max_depth=3,              # Shallow weak learners
    subsample=0.8,            # Stochastic gradient boosting
    random_state=42
)
gbdt.fit(X_train, y_train)
```

Practical Guidelines for Tabular ML

1. **Need Complete Interpretability / Regulatory Audit:**
   - Use a **Single Pruned Decision Tree** (or shallow tree with `max_depth <= 4`).
2. **Tabular Data with Minimal Tuning Required:**
   - Use a **Random Forest**. Robust to default hyperparameters, immune to catastrophic overfitting with large B.
3. **Maximum Competitive Predictive Accuracy on Tabular Data:**
   - Use **Gradient Boosting (LightGBM / XGBoost / CatBoost)**.
   - Tune learning rate $\eta \in [0.01, 0.1]$, tree depth ($3 - 6$), and early stopping on validation loss.
4. **Perceptual Data (Images, Audio, Natural Text):**
   - Use **Deep Neural Networks** (CNNs, Transformers - Parts 4 & 6). Trees cannot exploit translation equivariance or continuous spatial topologies.

Where is my artificial brain?

We wanted to make decisions like humans do, but...

AI $\longrightarrow$ ML $\longrightarrow$ Numbers game

Next Part: We'll get rid of labels y to see if we can learn hidden structures in the data.

Appendix

Bagging Derivation

Expand variance of a sum into all pairs:

$$\mathrm{Var}\left(\frac{1}{B}\sum_{i=1}^{B} f_i\right) = \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B}\mathrm{Cov}(f_i, f_j)$$

Split into B diagonal and $B(B-1)$ off-diagonal pairs:

- Diagonal ($i = j$): B terms with variance $\mathrm{Cov}(f_i, f_i) = \sigma^2$
- Off-diagonal ($i \neq j$): $B(B-1)$ terms with covariance $\mathrm{Cov}(f_i, f_j) = \rho\sigma^2$

Simplify algebraically:

$$\frac{1}{B^2}\left[B\sigma^2 + B(B-1)\rho\sigma^2\right] = \frac{\sigma^2}{B} + \frac{B-1}{B}\rho\sigma^2 = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$