上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 12: Decision Trees & Ensemble Methods

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

1. Decision Trees

Recap: What We Have So Far

Previous lectures — all linear models:

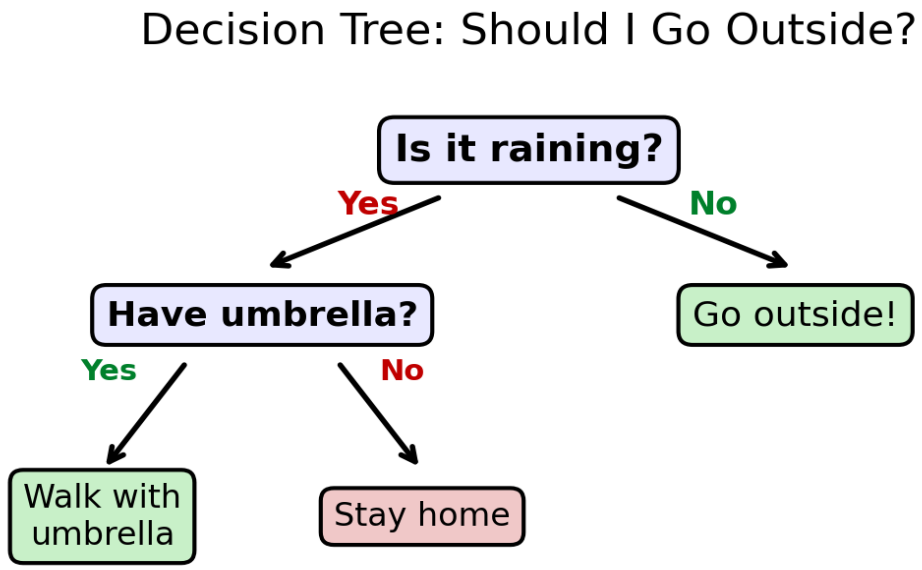
- L11: Linear Regression (squared loss, closed-form)
- L12: Logistic Regression (cross-entropy, sigmoid)
- L13: SVM (hinge loss, margin, kernel trick)

All learn a single global function $w \cdot \phi(x)$

- **Question: What if we want something more interpretable and fundamentally non-linear?**

A Different Way to Think

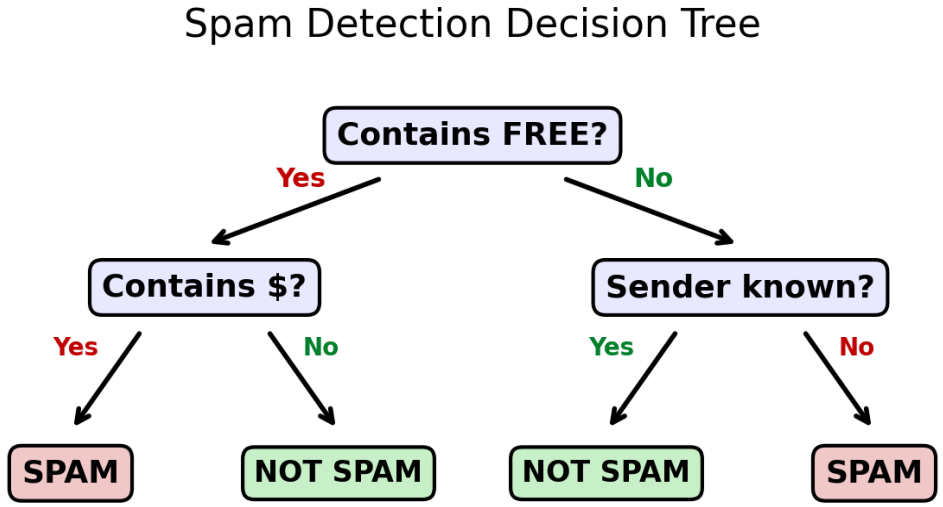
- Linear models: one global boundary $w \cdot \phi(x) = 0$
- **Alternative: make decisions by asking yes/no questions**
- **Example: Should I go outside?**
  - Is it raining? → No → Go outside!
  - Is it raining? → Yes → Do I have umbrella?
    - → Yes → Go outside with umbrella
    - → No → Stay home
- **This is a decision tree — a flowchart for classification.**



Decision Tree: Spam Example

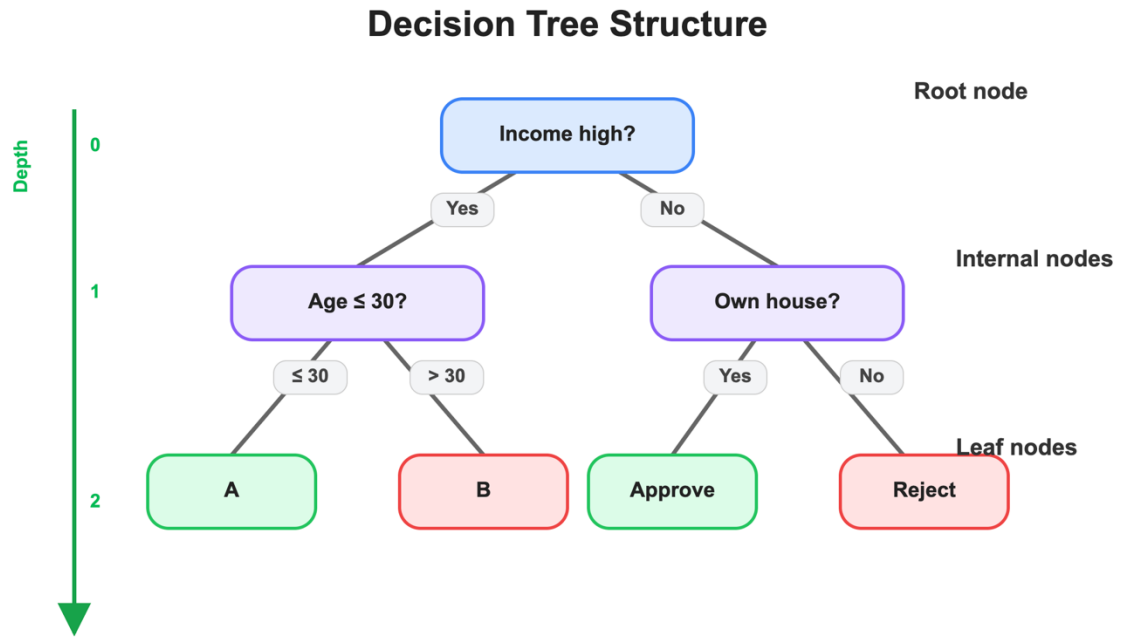
Revisiting spam detection from L8:

- Instead of w·ϕ(x), ask sequential questions:
- Contains FREE?
  - → Yes → Contains $?
    - → Yes → Predict: SPAM
    - → No → Predict: NOT SPAM
  - → No → Sender in contacts?
    - → Yes → Predict: NOT SPAM
    - → No → Predict: SPAM
- **Each leaf gives a final prediction.**



Anatomy of a Decision Tree

- **Root node: first question (most informative feature)**
- **Internal nodes: subsequent feature tests**
- **Branches: outcomes (Yes/No, or thresholds)**
- **Leaf nodes: predictions (class label or value)**
- Depth = number of questions from root to leaf
- A deeper tree can ask more questions → more complex boundaries
  …but also more prone to overfitting!



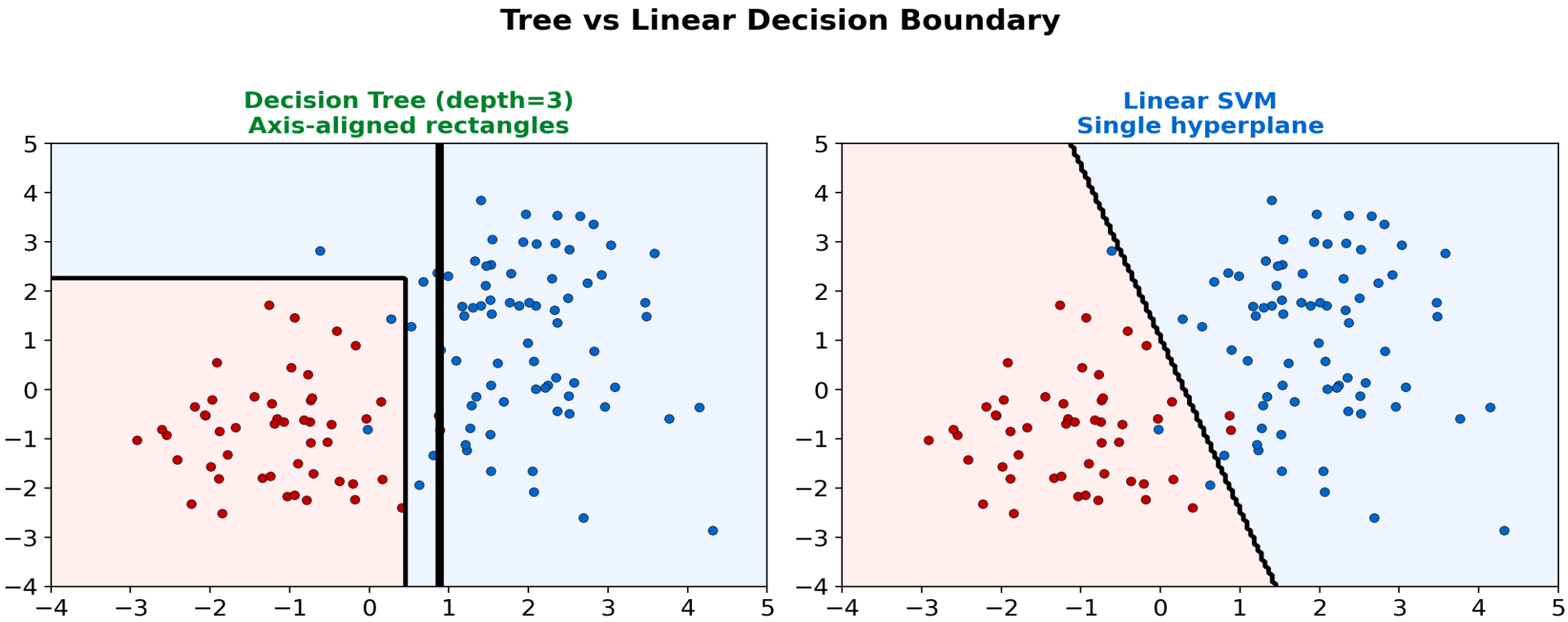
Decision Boundary: Trees vs Linear

- Trees partition the feature space into axis-aligned rectangles.
- Each rectangle maps to one leaf (one prediction).

- Contrast with SVM / Logistic Regression:
  - Linear: single hyperplane boundary
  - Tree: staircase pattern of rectangles

- Trees naturally capture non-linear patterns
  without feature engineering or kernels!

Decision Boundary: Trees vs Linear



🧠 Think: Design a Tree by Hand

Given 6 points with features (x_1, x_2) and labels:

- (+1): (1,3), (2,4), (3,3.5)
- (−1): (1,1), (2,0.5), (3,1.5)

Exercise (1 minute):

1. Which feature do you split on first?
2. What threshold?
3. Draw the resulting tree.

Hint: look at x_2 — is there a clean split?

How to Build a Tree: The Key Question

We need to decide:

- Which feature to split on?
- What threshold value?
- When to stop splitting?

Core idea:

- Choose the split that best separates the classes.

We need a measure of purity / impurity.

- How mixed are the labels in a node?

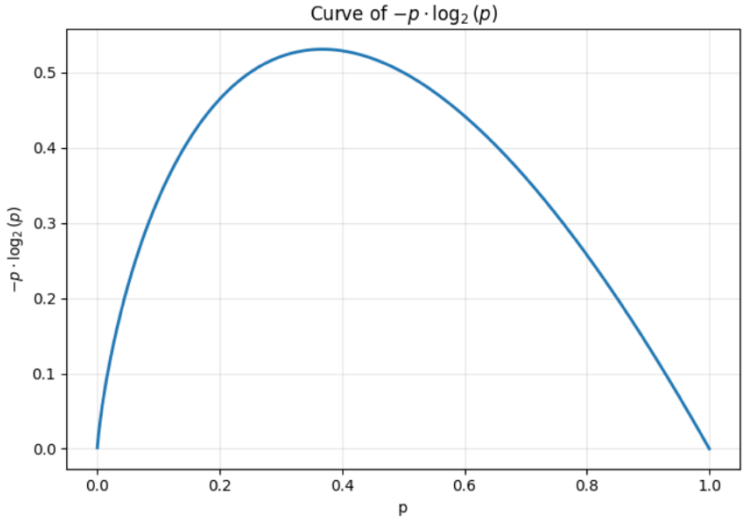
Measuring Impurity: Entropy

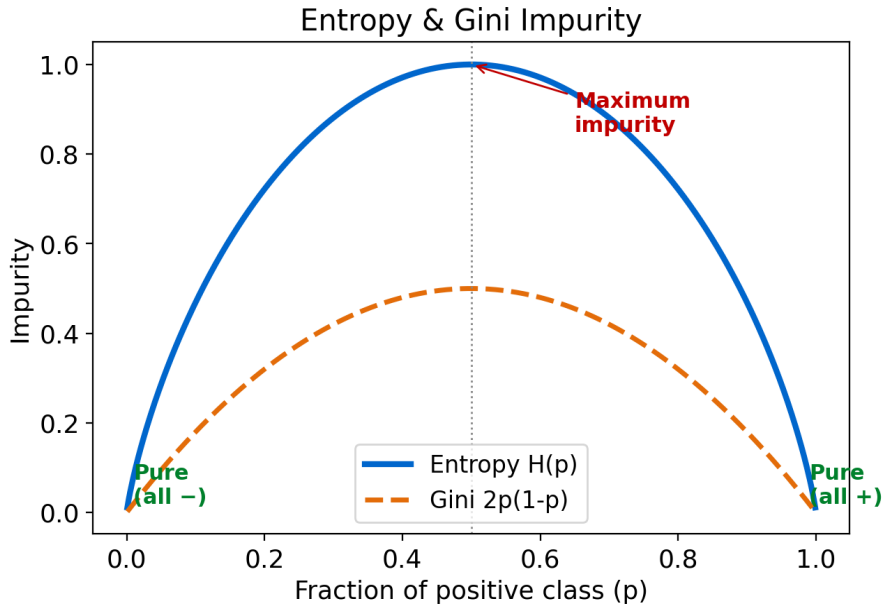
Given a set S with fraction p_+ positive, p_- negative:

$$H(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

- H = 0: perfectly pure (all same class)
- H = 1: maximum impurity (50/50 split)

- Lower entropy = more pure = better!



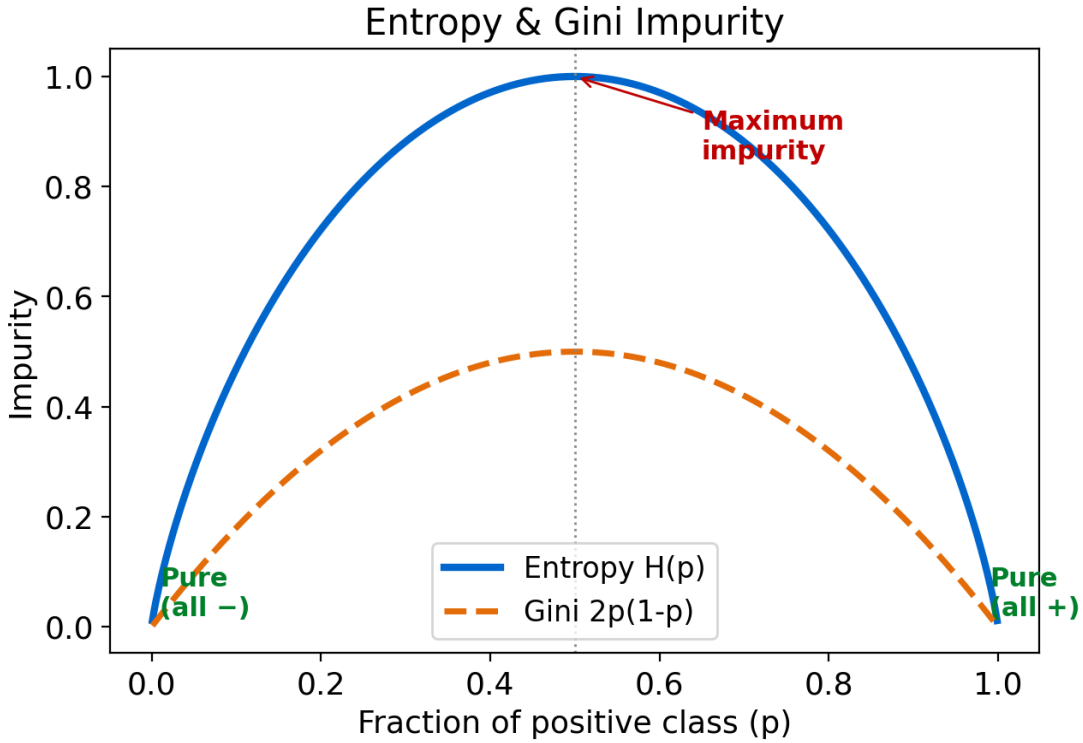


Alternative: Gini Impurity

$$Gini(S) = 1 - p_+^2 - p_-^2$$

- Also 0 when pure, maximum at 50/50
- Default in scikit-learn's DecisionTreeClassifier
- Slightly faster to compute (no logarithm)

- **In practice, entropy and Gini give very similar splits.**



Information Gain

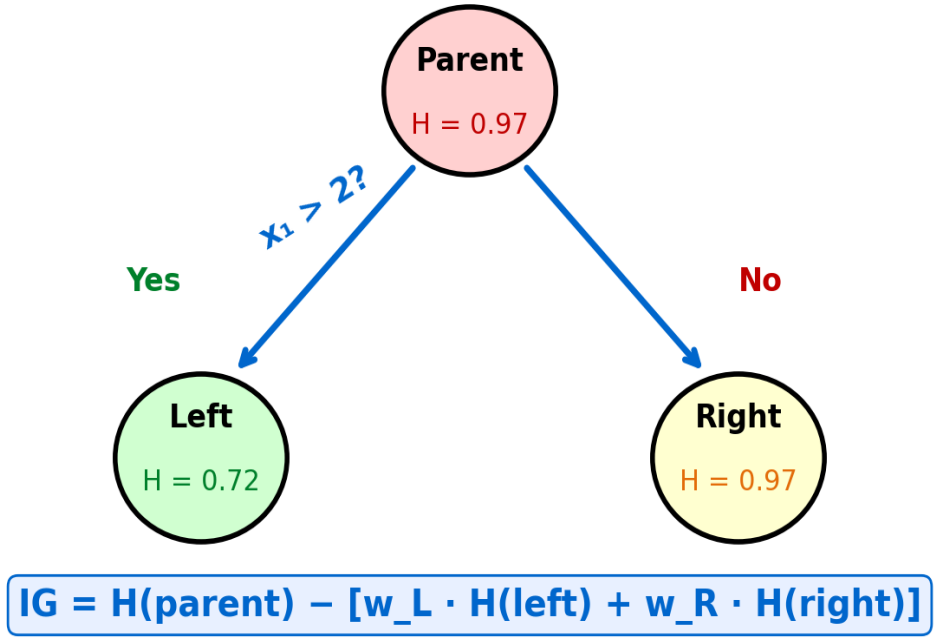
Split data S into S_left and S_right:

$$IG = H(S) - \left[\frac{|S_L|}{|S|} \cdot H(S_L) + \frac{|S_R|}{|S|} \cdot H(S_R)\right]$$

= reduction in impurity after the split

Algorithm:

- try all features and all thresholds, **pick the split with maximum information gain.**
- Greedy: locally optimal at each node.



Example: Information Gain

Dataset: 10 samples, 6 positive (+), 4 negative (−)

- Feature: contains FREE? Yes: (4+, 1−) No: (2+, 3−)
- **Step 1: H(parent)**

  $= -(6/10)\log_2(6/10) - (4/10)\log_2(4/10) \approx 0.971$
- **Step 2: H(Yes)**

  $= -(4/5)\log_2(4/5) - (1/5)\log_2(1/5) \approx 0.722$
- **Step 3: H(No)**

  $= -(2/5)\log_2(2/5) - (3/5)\log_2(3/5) \approx 0.971$
- **Step 4: IG**

  $= 0.971 - [5/10 \cdot 0.722 + 5/10 \cdot 0.971]$

  $= 0.971 - 0.847 = 0.125$
- Compare with other features to pick the best split!

The Recursive Algorithm: ID3 / CART

BuildTree(S, Features):

1. If all labels same → return leaf(label)
2. If no features left → return leaf(majority)
3. Find best feature + threshold (max IG)
4. Split data into S_left, S_right
5. left_child = BuildTree(S_left, Features)
6. right_child = BuildTree(S_right, Features)
7. Return node(feature, threshold, left, right)

- **This is a greedy, recursive algorithm.**

Overfitting in Decision Trees

- Without limits, the tree perfectly classifies every training point.
- Each leaf contains exactly one sample → memorization!

Pre-pruning (stop early):

- Maximum depth
- Minimum samples per leaf
- Minimum information gain threshold

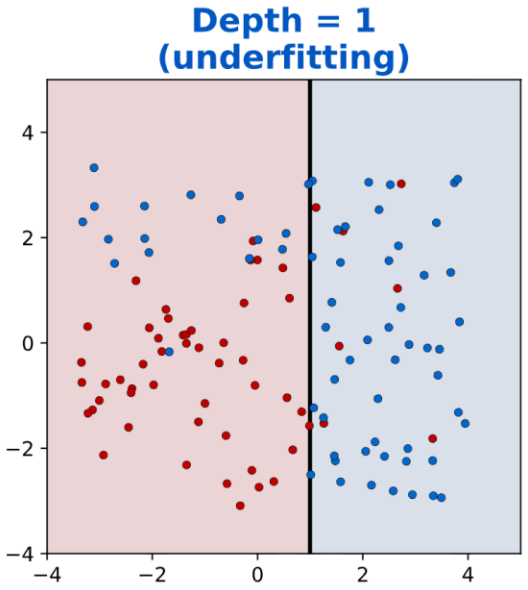
Post-pruning:

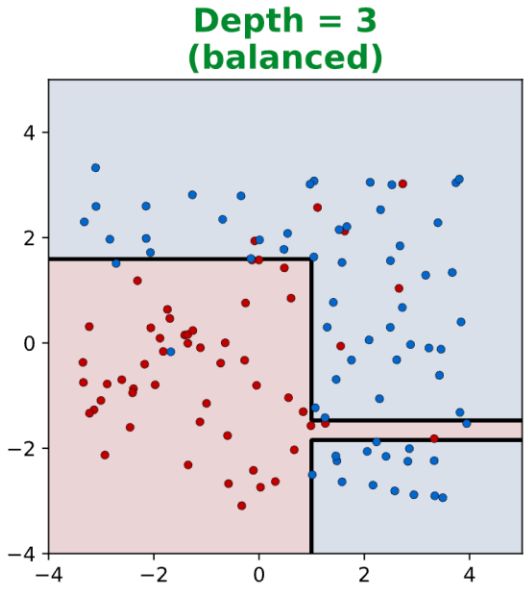
- Grow full tree, then cut branches that hurt validation

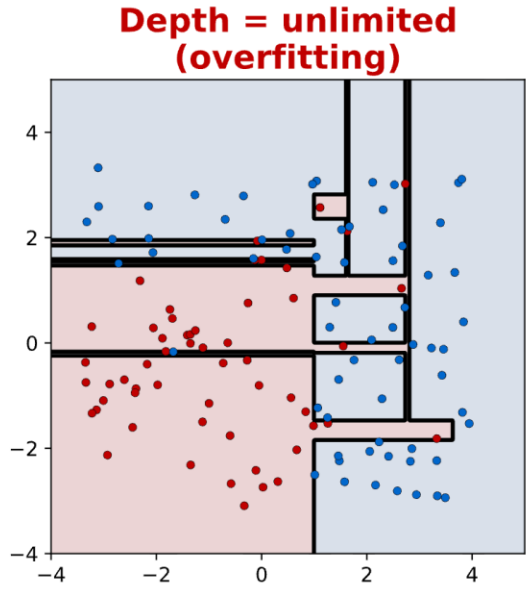
Experiment: Effect of Tree Depth

Same dataset, different maximum depths:

- **Depth = 1 (decision stump)**
  Too simple, underfitting
- **Depth = 3**
  Reasonable boundary, good generalization
- **Depth = unlimited**
  Perfectly fits training data, overfits!
- **Same pattern as C in SVM and γ in RBF kernel!**







Pros and Cons of Decision Trees

Pros:

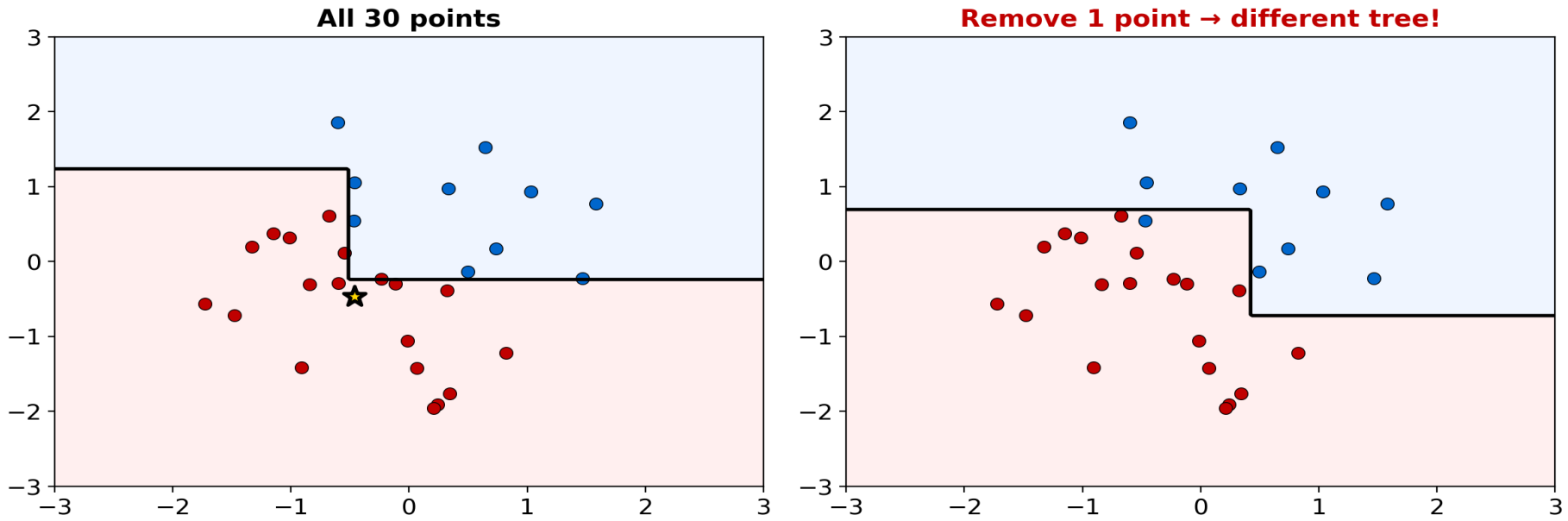
- Highly interpretable (visualize and explain)
- No feature scaling needed
- Handles numerical and categorical features
- Captures nonlinear boundaries naturally

Cons:

- High variance (unstable — small changes → different tree)
- Prone to overfitting
- Greedy splits may miss global structure
- Axis-aligned boundaries only

The Instability Problem

- Remove one training point → completely different tree!
- High variance = different runs give wildly different models.
- **This is the fundamental weakness of single trees.**
- **Question: Can we reduce variance without increasing bias?**
- **Answer: Yes! → Ensemble methods.**



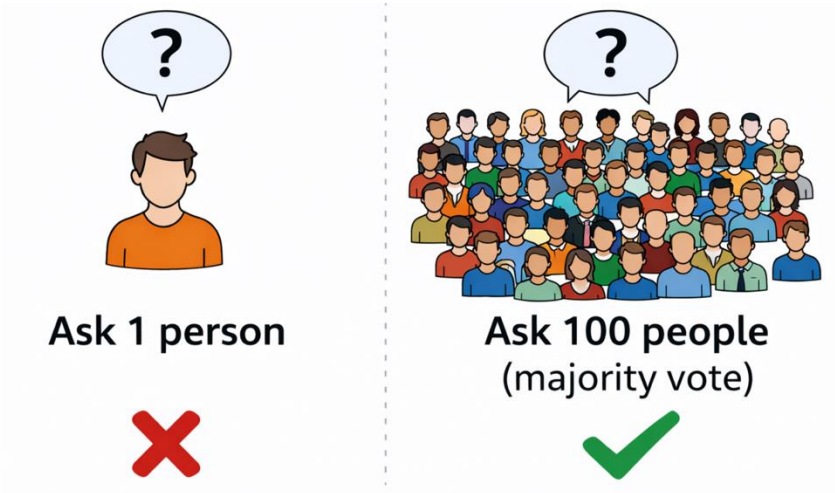
2. Ensemble Methods

The Wisdom of Crowds

- Ask 1 person a question: might be wrong.
- Ask 100 people, take majority vote: usually right!

- **Condition: individuals must be diverse**

  (they make different errors)



Ensemble methods:

- Train multiple models, combine their predictions.
  - Classification: majority vote
  - Regression: average

Ensemble Strategy

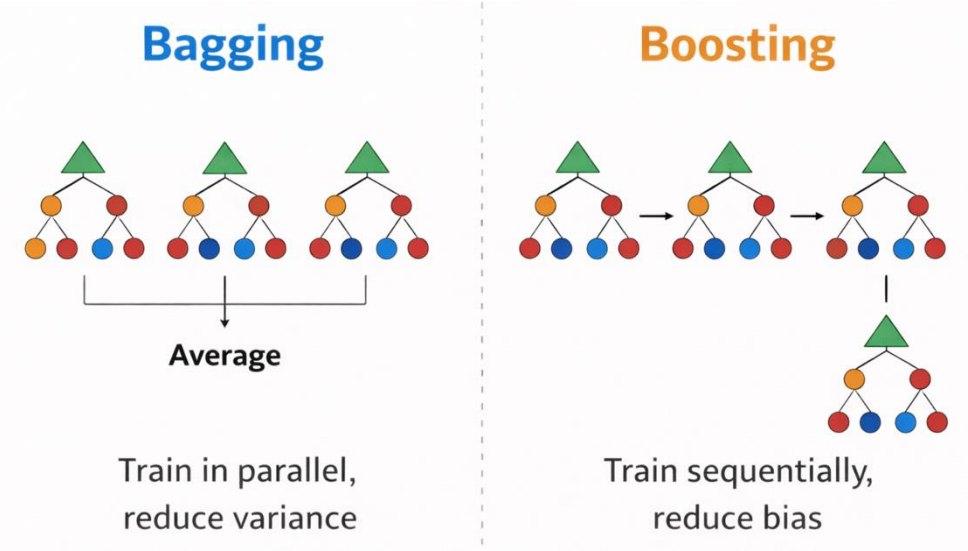
- Train multiple base learners (weak learners).
- Combine their predictions for a stronger result.

Key requirements:

1. Each model is better than random guessing
2. Models make different errors (diversity!)

Two main approaches:

- Bagging: train in parallel, reduce variance
- Boosting: train sequentially, reduce bias

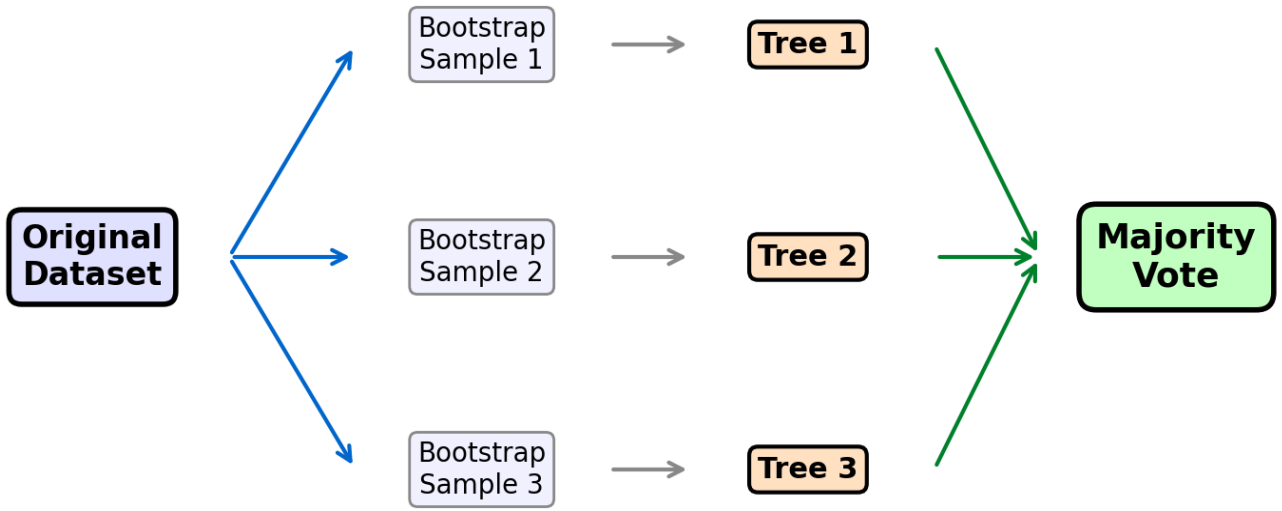


Bagging (Bootstrap Aggregating)

Algorithm:

1. Create B bootstrap samples (random with replacement)
2. Train one tree on each bootstrap sample
3. Aggregate: majority vote or average

- Each tree sees different data → different trees → diversity!
- **Effect: reduces variance, keeps bias roughly the same.**



Random Forests

Random Forest = Bagging + Random Feature Selection

- At each split, only consider m random features
  (typically $m \approx \sqrt{d}$ for classification)

Why random features?

- If one feature is very strong, all bagged trees split on it
- Result: correlated trees → less variance reduction
- Random selection forces trees to discover alternative patterns

- **Random features decorrelate the trees!**

🧠 Think: Why Random Features Help

- Imagine 20 features where feature #1 is extremely predictive.

Bagging alone:

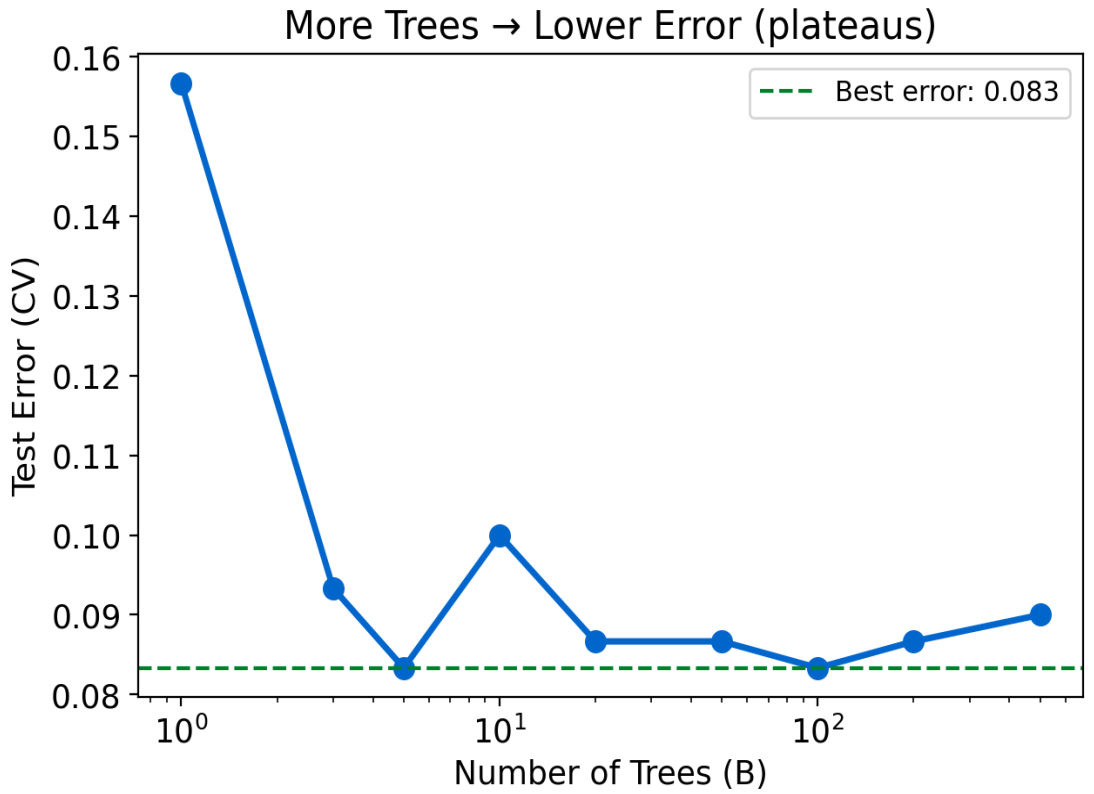
- All trees split on feature #1 first
  → Very similar trees → correlated errors
- **Bagging + random features:**
- Some trees forced to use features #5, #12, #17…
  → Diverse trees → errors cancel out!

- **Diversity is the key ingredient of ensembles.**

Random Forest: Hyperparameters

- **Number of trees (B):**
  More is (almost) always better. No overfitting from adding trees!
- **Max features per split (m):**
  $\sqrt{d}$ for classification, d/3 for regression (rules of thumb)
- **Max depth / min samples per leaf:**
  Controls individual tree complexity

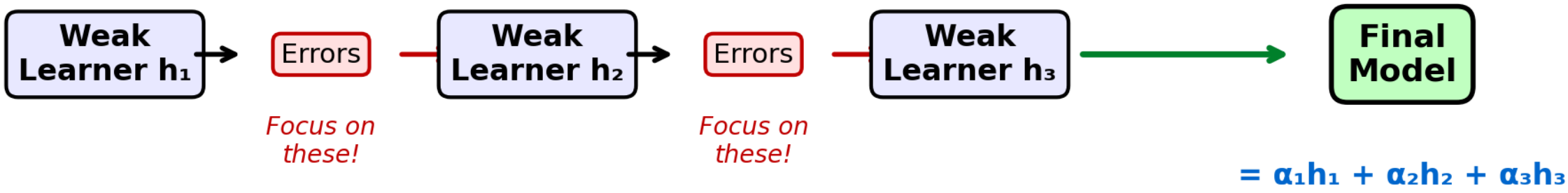
- Practical strength: Random Forests are hard to overfit
- by adding more trees. Just set B large enough.



3. Boosting

A Different Philosophy: Boosting

- Bagging: train models in parallel, independently
- Boosting: train models sequentially, each one fixing mistakes of the previous.
- **Analogy — studying for an exam:**
  - Bagging: ask 10 friends to study independently, combine their answers
  - Boosting: study, take practice test, focus on what you got wrong, repeat



AdaBoost (Adaptive Boosting)

1. Initialize sample weights: $w_i = 1/n$
2. For t = 1, …, T:
   a. Train weak learner h_t on weighted data
   b. Compute weighted error ε_t
   c. Learner weight: $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
   d. Update: increase weight of misclassified points
3. Final: $H(x) = \text{sign}\left(\sum_t \alpha_t h_t(x)\right)$

AdaBoost Intuition

Key ideas:

- Misclassified points get higher weights
  → next learner focuses on hard examples

- Better learners (lower error) get higher α_t
  → more voting power

- Each round, the hard examples get emphasized

- **Final model = weighted vote of all weak learners.**

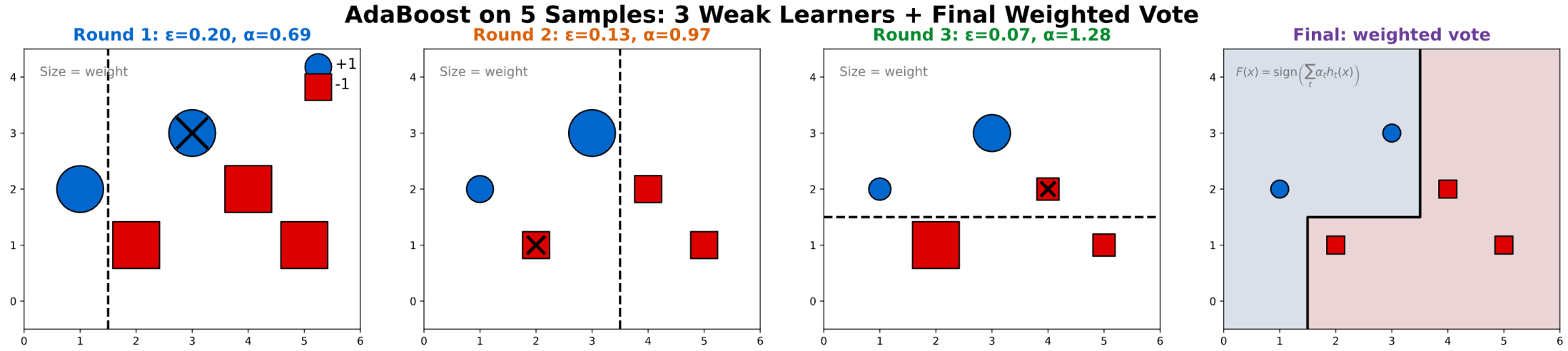
AdaBoost Step by Step

5 points, 3 rounds with decision stumps:

- **Round 1: equal weights, stump on x_1, misclassifies pt 4**

  → pt 4 weight increases
- **Round 2: new stump, focuses on pt 4, misclassifies pt 2**

  → pt 2 weight increases
- **Round 3: fixes remaining errors**
- **Final: weighted combination classifies all correctly!**
- One stump is weak. Three stumps together are strong.



Gradient Boosting (GBM Intuition)

Generalization of boosting:

- Fit each new model to the residual errors.

Iteration:

- $f_0(x) = 0$
- $f_t(x) = f_{t-1}(x) + \eta \cdot h_t(x)$
- h_t is trained to predict the negative gradient (residual)
- η is a learning rate (shrinkage)

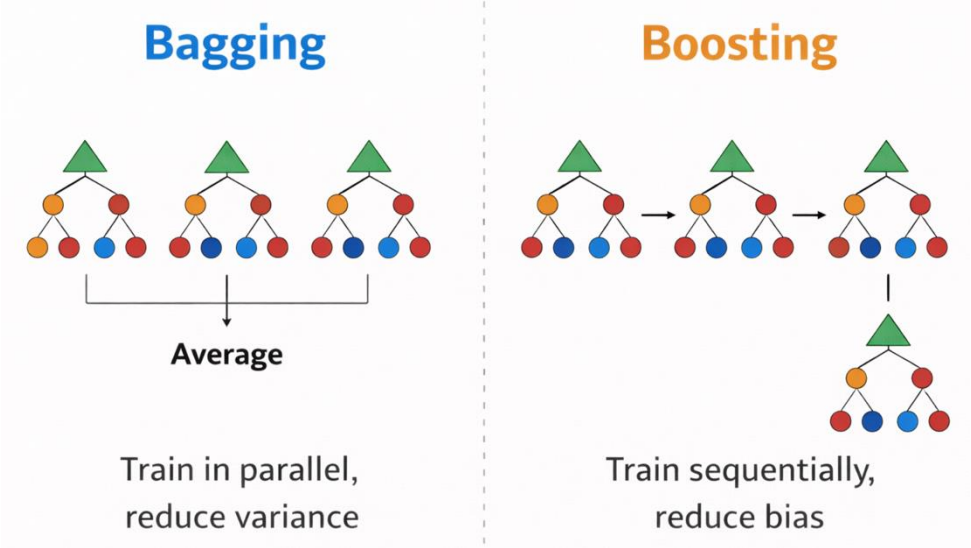
Connection to L10: gradient descent in function space!

Popular: XGBoost, LightGBM, CatBoost

Bagging vs Boosting

Bagging:

- Parallel training, independent models
- Reduces variance
- Hard to overfit
- Base learner: deep trees



Boosting:

- Sequential training, models depend on each other
- Reduces bias + variance
- Can overfit if too many rounds
- Base learner: shallow trees (stumps)

🧠 When to Use What?

- Single Decision Tree: interpretability critical
  (medical diagnosis, legal decisions)
- Random Forest: robust default, works out of the box
  General-purpose, hard to mess up
- Gradient Boosting (XGBoost): maximum accuracy
  Kaggle champion, best for tabular data
- SVM / LR: small data, need probabilities, simple baselines
- Deep Learning: images, text, very large datasets (Next module)

Summary

5 Key Concepts:

- Information Gain — pick the split that reduces impurity most
- Overfitting in Trees — depth control, pruning
- Bagging — parallel training on bootstrap samples
- Random Forests — bagging + random features
- Boosting — sequential focus on hard examples

- Single tree: interpretable but unstable.
- **Ensemble: powerful and robust.**

Next Lecture: Model Evaluation

- How do we fairly compare all the models we have built?

Topics:

- Cross-validation
- Metrics: accuracy, precision, recall, F1, ROC/AUC
- Hyperparameter tuning

We built the tools. Now we learn to evaluate them.