COMS4995W31
Applied Machine Learning

Dr. Spencer W. Luo

Columbia University | Spring 2026


MIDTERM

Announcement

Assignment 1

Due Date: Saturday, February 28th, 11:59 PM EST

Ed post: Link

Midterm

- Monday, March 2, 2026, from 4:10 PM to 5:40 PM
- No cell phones or calculators allowed
- Sample questions and answers
- Prep: Focus on the lecture slides, raise questions and ask ChatGPT/Gemini for clarifications

Tree Models & Ensembles

AI of the Week: The Undisputed Kings of Tabular Data 👑

Standard structured data (SQL databases) still runs most of the world's real-world business systems

The Reality in 2026:

- Classical Tree Ensembles (XGBoost, LightGBM etc.) remain the absolute gold standard
- A review of recent Kaggle tabular competitions shows that these gradient boosting frameworks are still the core of almost every single winning solution
- The recent XGBoost 3.1 releases rolled out massive updates
    - introducing blazing-fast GPU-accelerated pipelines and "Native Categorical Re-coding"
    - multimodal support

AI of the Week: The Undisputed Kings of Tabular Data 👑

Ensembles Win

As we will cover today, combining many "weak" decision trees into a powerful ensemble dramatically reduces both bias and variance

Feature Engineering > Raw Compute

Tree models heavily reward clever feature engineering and a deep understanding of data

Agenda

- Motivation
- Decision Trees
- Ensembles
- Summary

Motivation

Why do we need Tree Models? 🌲

In previous:

- Linear Regression → assume linear decision boundary
- Naive Bayes → assume feature independence

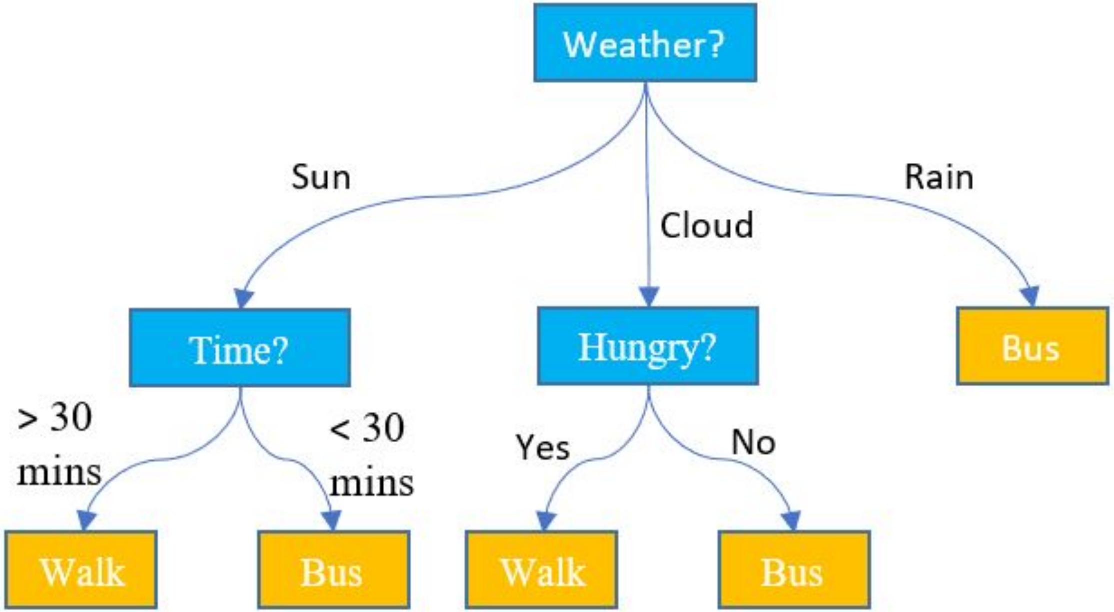
But… real-world data is often non-linear, complex, and mixed

Why do we need Tree Models? 🌲

Decision Trees:

- Flexible → no strict distribution or linearity assumptions
- Versatile → handle both numeric & categorical features
- Non-linear power → capture curved/complex decision boundaries
- Interpretability → rules easy to visualize & explain to non-experts
- Foundation of Ensembles → Random Forests, Boosting, XGBoost 🚀

Why do we need Tree Models? 🌲



Decision Trees

What is a Decision Tree? 🌳

[Inference] A decision tree is like a 20 Questions game 🎲

- At each step → ask a yes/no question about the data
- Each answer leads you further down the tree
- Finally, you reach the final decision / prediction

Example:

Shall we have the mid-term test next week?

→ Yes → take mid-term

→ No → take mid-term :)

Decision Tree in Action

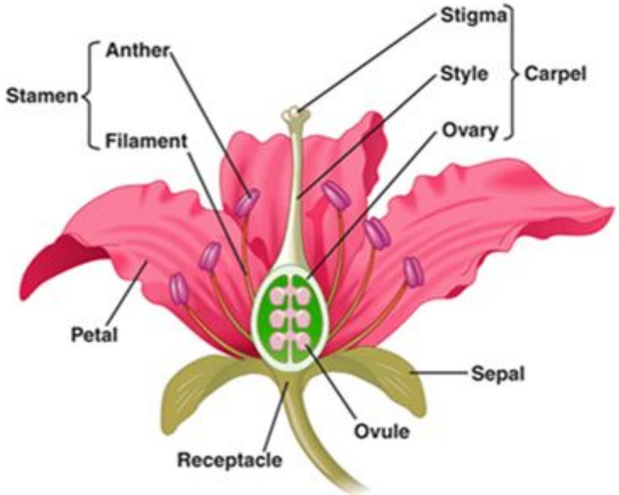
Iris Dataset: 150 flowers → 3 classes

Features:

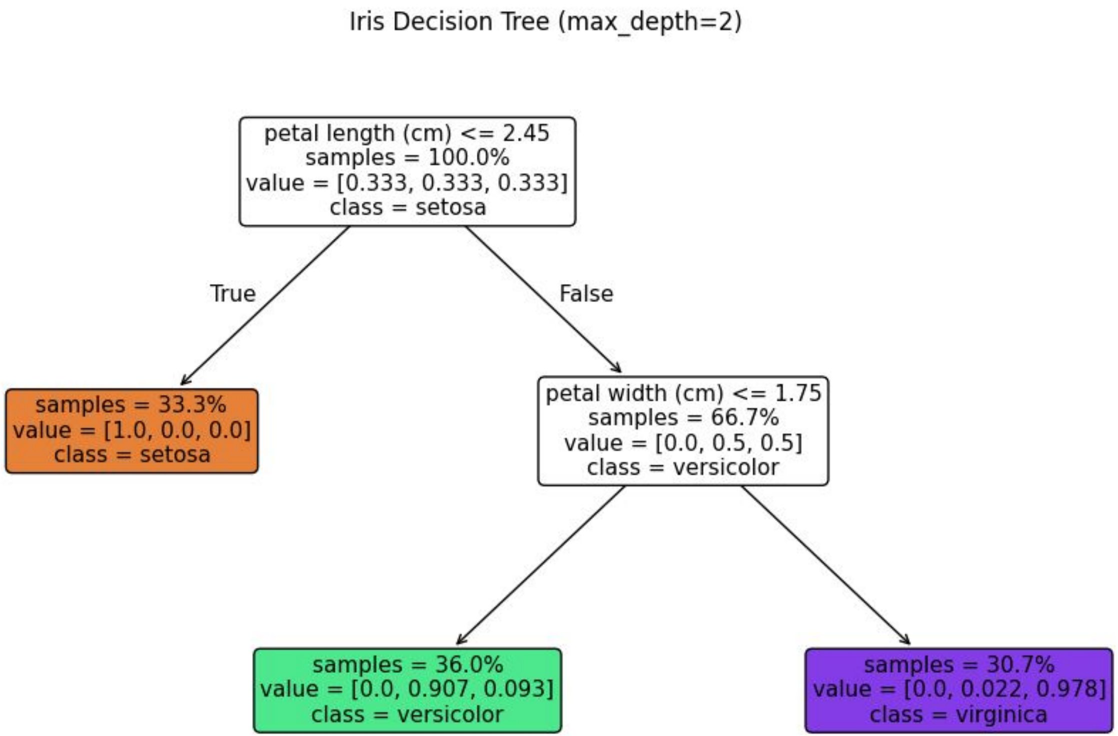
- Sepal length
- Sepal width
- Petal length
- Petal width

Target labels:

- [0] Iris Setosa 🌱
- [1] Iris Versicolor 🌷
- [2] Iris Virginica 🌼


Stigma
Anther
Style
Carpel
Stamen
Ovary
Filament
Petal
Sepal
Ovule
Receptacle

Visualizing a Simple Decision Tree 👀



Components of a Decision Tree 🌳

Root Node → initial state

Internal Nodes → decision points (questions based on features)

Branches → outcomes of questions (Yes / No, > / = / <)

Leaves → final prediction (class label / value)

Decision Path → sequence of rules from root to leaf

- one classification rule

Tree Depth → longest path from root to leaf (tree complexity)

Two Flavors of Trees 🌲

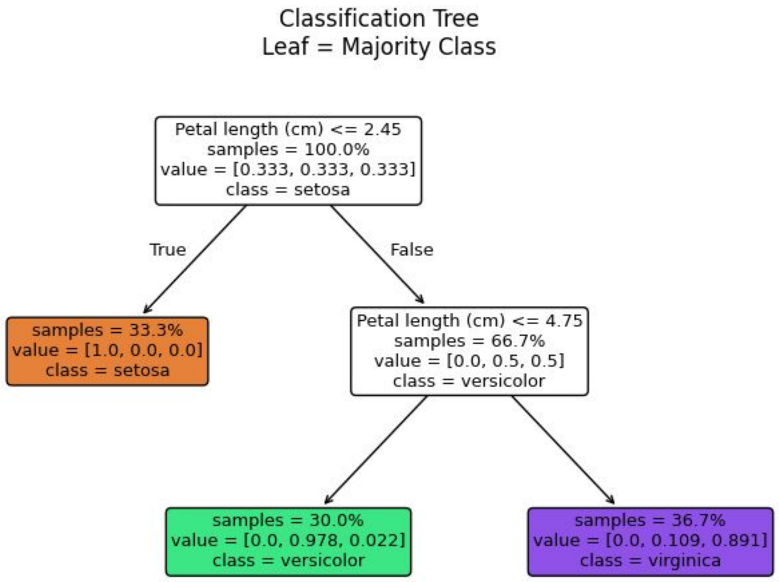
Classification Tree:

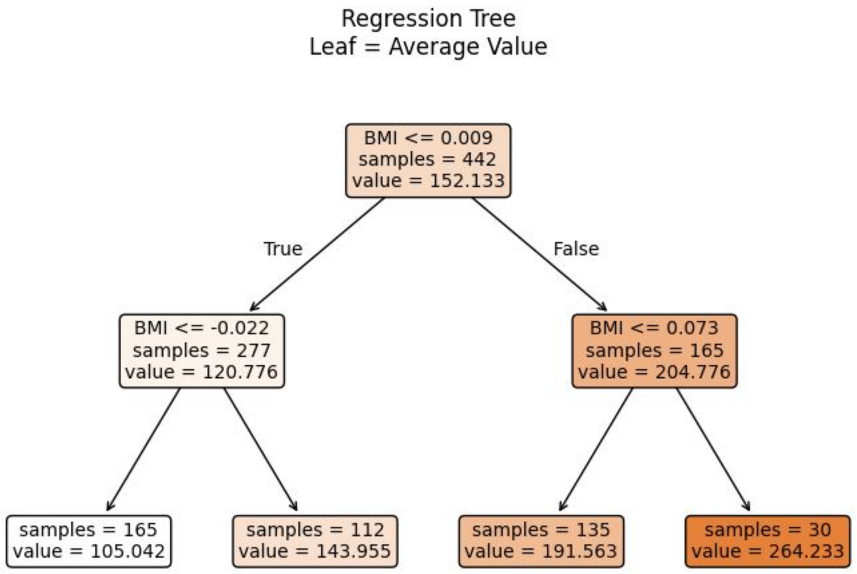
- Output = discrete label
- Leaf prediction = majority class

Regression Tree:

- Output = continuous value
- Leaf prediction = average of samples

Two Flavors of Trees 🌲





How do we train (aka split)? ✂️

💡 Idea: Make children nodes "purer" than the parent

Pure node → samples mostly belong to one class

Example intuition:

- Before split:
  - 50% red 🔴, 50% blue 🔵 (100)
- After split:
  - left node 90% red 🔴 (40)
  - right node 80% blue 🔵 (60)
- Better separation → better classification capability

Mathematical Support 📐

Pure definition:

Gini Impurity

$$Gini(S) = 1 - \sum_{i=1}^{C} p_i^2$$

Entropy

$$H(S) = -\sum_{i=1}^{C} p_i \log p_i$$

p_i : proportion of samples in node S that belong to class i

Smaller → Better

Entropy vs Gini

Gini Impurity: How mixed the classes are in a node

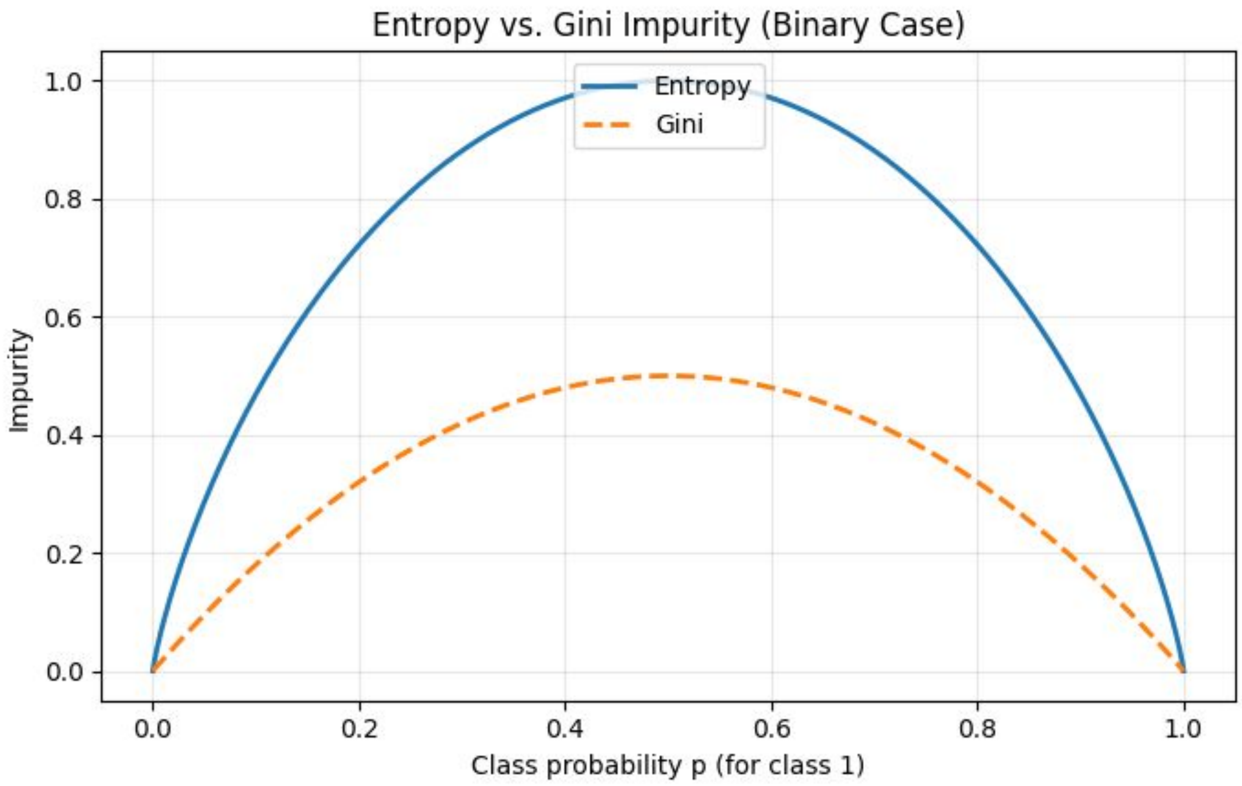
Entropy: What the uncertainty degree is in a node

0 = pure, higher = more mixed

Both are similar in shape

Used interchangeably in practice

Entropy vs Gini



Decision Tree Algorithm

```
func build_tree(data) → current node:
    if stopping_condition(data):
        return Leaf(prediction=data.label_majority)
    best_feature, threshold = choose_best_split(data)
    left_data, right_data = split(data, best_feature, threshold)
    node = Node(feature=best_feature, threshold=threshold)
    node.left  = build_tree(left_data)
    node.right = build_tree(right_data)
    return node
```

How to Split?

```
func choose_best_split(data) → (feature, threshold):
    best_split = None; best_gain  = -inf
    for feature in features:
        for threshold in possible_thresholds(feature):
            left, right = split(data, feature, threshold)
            if left or right is empty: continue
            gain = impurity(parent) - weighted_impurity(left, right)
            if gain > best_gain:
                best_gain = gain;  best_split = (feature, threshold)
    return best_split
```

When to Stop?

```
func stopping_condition(data, depth):
    if all_same_label(data):
        return True
    if depth >= MAX_DEPTH:
        return True
    if split_gain(data) < MIN_GAIN:
        return True
    # Otherwise, continue splitting
    return False
```

Deep Trees = Overfitting ⚠️

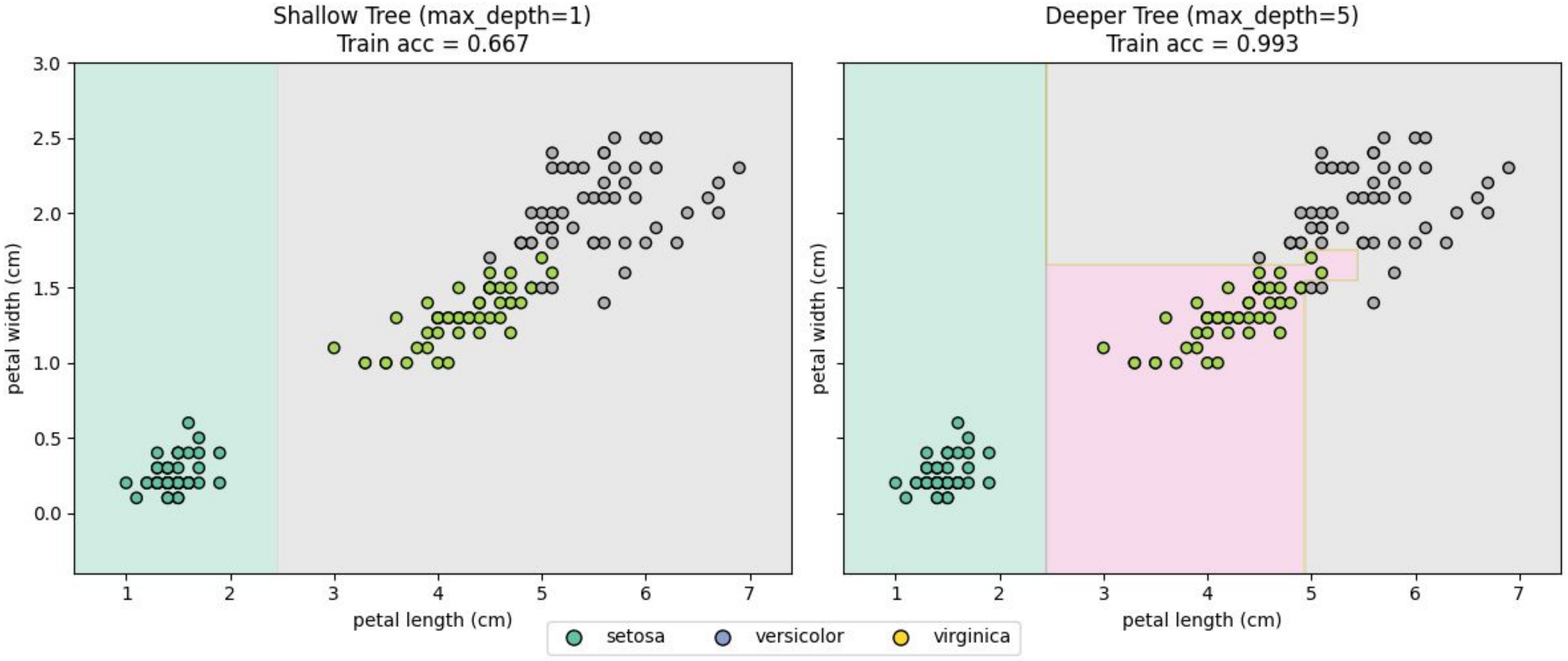
Deep tree memorizes training data → high variance

Small perturbation in data → different splits

Example: depth=1 vs depth=5

Need control: `max_depth`, `min_samples_split`

Deep Trees = Overfitting ⚠️



Why Pruning? 🌳✂️

Without limits, trees grow very deep → low training error, overfitting 😵‍💫

Pruning = controlling tree growth to improve generalization

2 perspectives:

- When to prune (Pre vs Post)
- How much to prune (Moderate vs Strong)

Pre-pruning (Early Stopping) ⏸️ When

Put pruning rules **inside** `stopping_condition()`:

- Max depth, min samples, min gain…

Stops growth before overfitting appears

✅ Fast, simple

❌ Risk of underfitting if too strict

Post-pruning (Cost-Complexity) 🔄 When

Steps:

- First grow a big tree 🌳
- Then cut back unnecessary branches
- Use cross-validation to decide how much to prune

CART (Classification and Regression Trees)

- balance accuracy vs simplicity

✅ Improves generalization

❌ Extra computation

Moderate vs Strong Pruning ⚖️ How much

Moderate pruning:

- Keep useful sub-branches
- Balance accuracy & simplicity 🙂

Strong pruning:

- Aggressively cut → very shallow tree
- Easier to interpret, but higher bias 😬

Works for both Pre & Post:

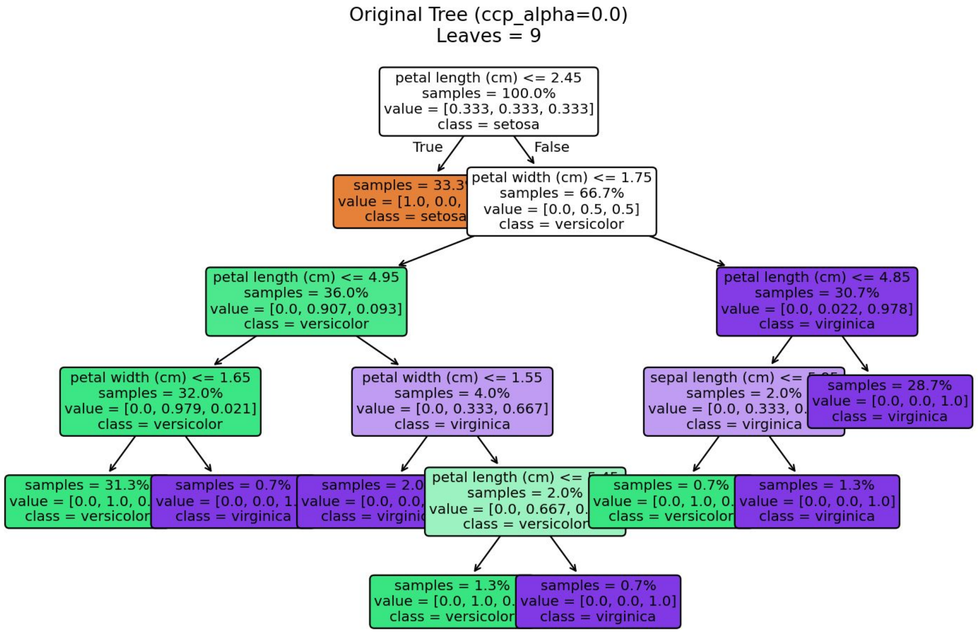
- Pre: adjust thresholds (strict vs loose)
- Post: choose small vs large

How to Implement Pruning ⚖️

- In any decision tree libraries, there is a built-in parameter called `ccp_alpha`
- It controls how much the tree is pruned
- Think of it like regularization in linear models:
- Best value is usually chosen by empirical study or cross-validation ⚖️

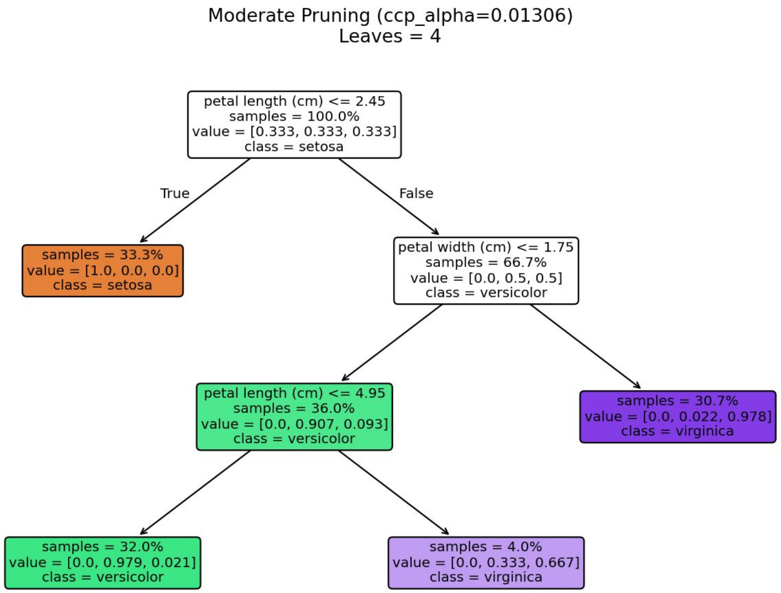
Original Tree 🌳

- Large tree grown fully
- Perfect fit to training data
- Risk: overfitting



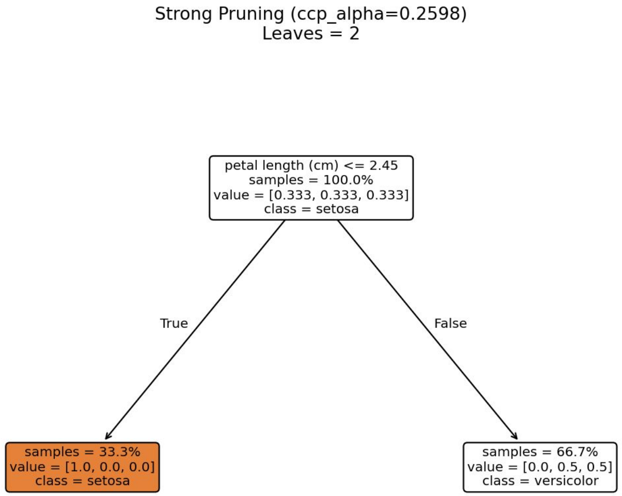
Moderate Pruning 🌱

- Remove weak branches
- Tree becomes smaller
- Slight accuracy loss on train, better test generalization



Strong Pruning 🌿

- Aggressive reduction of branches
- Tree is much simpler
- Easier to interpret



Decision Trees: Pros & Cons ⚖️

✅ Pros:

- Easy to understand & explain
- Works with tabular data
- Captures nonlinear interactions

❌ Cons:

- Prone to overfitting (high variance)
- Small changes in data → very different tree
- Weak standalone performance (needs ensembles)

Ensembles

Why Combine Models? 🤝

Single decision tree = unstable, high variance

Small changes in data → very different trees

💡 Idea: aggregate multiple trees to get more robust predictions

Analogy: 🌲 → 🌳🌳🌳

Bagging = Bootstrap Aggregation 👜

Train multiple models on bootstrap samples (resampling with replacement)

Average predictions (regression) or majority vote (classification)

Reduces variance without increasing bias much

Mathematical View 📐

For regression: average across K models

$$f^{\text{bag}}(x) = \frac{1}{K}\sum_{k=1}^{K} f_k(x)$$

For classification: majority vote

Works best with unstable models (like decision trees 🌳)

Random Forest = Bagging + Random Features 🌲🌲🌲

At each split, we don't look at all features, only a random few

This makes each tree different → expert on some small domains

When we average many such trees → predictions are more stable

That's why Random Forest is a strong baseline in practice ✅

Simple Decision Tree Split Revisit

```
func choose_best_split(data) → (feature, threshold):
  best_split = None; best_gain  = -inf
  for feature in features:
    for threshold in possible_thresholds(feature):
      left, right = split(data, feature, threshold)
      if left or right is empty: continue
      gain = impurity(parent) - weighted_impurity(left, right)
      if gain > best_gain:
        best_gain = gain;  best_split = (feature, threshold)
  return best_split
```

Random Forest Pros & Cons

✅ Pros:

- Great out-of-the-box performance
- Handles tabular data well
- Robust to overfitting

❌ Cons:

- Slower than single trees
- Less interpretable
- Still limited for very high-dimensional sparse data

Boosting = Sequential Learning 🚀

Train models sequentially

Each new model focuses on errors of previous ones

Combines many weak learners → strong learner

Analogy: Coach correcting mistakes step by step

Idea of Gradient Boosting 🌱 → 🌳

Build trees sequentially, each new tree fixes errors of the previous ones.

Start with a simple model (seed prediction).

Each step: predict residuals/errors, fit a new weak learner.

Final model = weighted sum of all trees.

XGBoost / LightGBM ⚡

Engineering optimizations for speed and scalability

Key ideas:

- Regularization (to prevent overfitting)
- Handling missing values
- Parallel training

Popular in Kaggle competitions 🏆

Intuition 🎯

Think of a student learning over time:

- Day 1 → learns basics, makes many mistakes.
- Day 2 → focuses on yesterday's mistakes.
- Day 3 → focuses again on remaining mistakes.

Step by step, performance improves.

Gradient Boosting = Residual Learning + Gradient Descent

Key Features & Benefits 🚀

- Learning rate: controls step size, prevents overfitting.
- Number of trees: more trees = lower bias, risk of overfit.
- Depth of trees: shallow = weak learners (better generalization).
- Regularization: subsampling, shrinkage, pruning.

Boosting vs Bagging:

- Boosting = sequential, reduce bias.
- Bagging = parallel, reduce variance.

Decision Trees vs Ensembles 📊

Method	**Strengths ✅**	**Weaknesses ❌**
Decision Tree 🌳	Simple, interpretable, fast	Overfits, unstable
Bagging 🎲	Reduces variance, robust	Less interpretable
Random Forest 🌲🌲	Strong baseline, feature importance	Slower, less transparent
Boosting 🚀	High accuracy, flexible losses	Sensitive to parameters, less interpretable

Trees vs Deep Learning 🤖

Tree excels at tabular data (structured, mixed features)

Deep learning shines at unstructured data (images, text, audio)

In practice → ensembles wins on Kaggle tabular competitions 🏆

Rule of thumb 👍

- Tabular → Random Forest / XGBoost
- Image/Text → Neural Networks

Industry Case

The "Unfair" Match-Up 🥊

Why does the world's most advanced AI struggle with a simple Excel spreadsheet?

Say, a major NYC fintech company needs to predict loan defaults. The dataset is a classic relational database table containing heterogeneous features:

- Age (Integer), Income (Continuous), Employment Type (Categorical) etc.

The Contenders:

🤖 Deep Neural Network (DNN): 100 layers, millions of parameters, trained on massive GPU clusters

🌲 XGBoost Ensemble: 500 relatively shallow trees, trained on a standard laptop CPU

Intuition: Smooth Gradients vs. Hard Rules

It is all about the nature of the data and the "Inductive Bias"

Neural Networks Expect "Smoothness":

- NNs are essentially massive composite functions optimized by gradient descent
- They thrive on homogeneous data where spatial/temporal proximity implies mathematical similarity
- Tabular Data is "Jagged" & Unstructured
  - What is the mathematical "distance" between the categories Teacher and Engineer
  - A Zip Code of 10027 is numerically close to 10028, but demographically completely different

The Struggle: NN waste huge amounts of representational power trying to force these irregular, completely independent columns into a smooth, continuous geometric space (a manifold)

The Verdict: Trees are "Tabular Native" 🌲

Tree ensembles naturally mirror the logic of structured human data

Rule-Based Decision Boundaries:

- Trees don't care about mathematical distance or gradient flow
- They simply ask orthogonal, hard questions: "Is Income > $50k?" AND "Is Profession == Teacher?"
- This perfectly matches the conditional logic of tabular data

Built-in Feature Selection:

- Tabular data often contains noisy, irrelevant columns (e.g., user IDs)
- While a Neural Net might overfit to this noise, a Tree model naturally ignores features that don't reduce Information Gini Impurity

The Takeaway:

For tabular data, Feature Engineering + Tree Ensembles > Raw Compute + Deep Learning

Summary 📖

Decision Trees → splitting, pruning

Ensembles → Bagging, Random Forests, Boosting

Takeaway: ensembles make weak learners strong 💪