IN402

Machine Learning

Chapter

5

Decision Trees: Theory and Algorithms

Author: Abbas El-Hajj Youssef

University: Lebanese University

Department: Department of Computer Science

These notes extend course materials taught by Prof. Ahmad Faour with additional content from textbooks and supplementary resources.

Disclaimer: *This is not an official course document.*

Contents

Part I: Foundations

1 Introduction: The Art of Asking Questions

In the world of machine learning, few algorithms capture the essence of human decision-making as naturally as decision trees. When we make choices in daily life, we rarely perform complex calculations. Instead, we ask a series of questions, each answer leading us closer to a conclusion. Decision trees formalize this intuitive process into a powerful machine learning algorithm.

1.1 The Power of Sequential Questions

Consider how a doctor diagnoses an illness. They don't immediately order every possible test. Instead, they follow a strategic questioning process:

- "Do you have a fever?" (narrows down to infectious conditions)
- "Is there a cough?" (further refines the possibilities)
- "Is the cough productive?" (distinguishes between conditions)

Each question strategically reduces uncertainty, leading efficiently to a diagnosis. This is precisely how decision trees operate—they learn from data to ask the most informative questions in the most effective order.

Think about playing "20 Questions." To identify an unknown object, you ask strategic yes/no questions. A skilled player starts with broad questions that efficiently split possibilities: "Is it alive?" reveals more than "Is it a toaster?" Decision trees formalize this intuitive strategy, learning from data to construct optimal question sequences.

Why Decision Trees Matter

Decision trees are unique in machine learning because they:

- **Mirror human reasoning:** The question-answer flow matches how experts explain their decisions
- **Provide transparency:** Every prediction can be traced through a clear logical path
- **Handle complexity naturally:** They can capture non-linear patterns without complex mathematics
- **Work with minimal preprocessing:** Unlike many algorithms, they don't require scaled or normalized data
- **Handle mixed data types:** Process categorical and numerical features seamlessly
- **Capture interactions:** Model feature interactions without explicit interaction terms

1.2 What You'll Learn

This chapter takes you on a complete journey through decision tree algorithms. We'll start with the fundamental concepts, build trees from scratch using real examples, explore how different algorithms evolved to address specific challenges, and understand when and how to apply each approach.

By the end, you'll not only understand how decision trees work, but also why they remain one of the most practical and widely-used algorithms in machine learning, forming the foundation for powerful ensemble methods like Random Forests and Gradient Boosting.

2 Anatomy of a Decision Tree

Before we can build trees, we must understand their structure. A decision tree is more than just a flowchart—it's a carefully constructed hierarchy where each component serves a specific purpose in the prediction process.

2.1 Core Components

Decision Tree Structure

A decision tree T consists of:

1. **Root Node:** The starting point containing all training data and the first decision
2. **Internal Nodes:** Decision points that test a single feature and partition the data
3. **Branches:** Paths representing the outcomes of tests (e.g., Yes/No, or categorical values)
4. **Leaf Nodes:** Terminal nodes that provide the final prediction

Root Node. The first and most powerful question, chosen because it best splits the entire dataset into informative groups.

Internal Nodes. After the initial split, these nodes continue refining by asking targeted questions about features to further divide subgroups created by their parent.

Leaf Nodes. Where decisions conclude. A leaf represents a sufficiently pure data subset where a final decision is made. For classification, it assigns the majority class of samples reaching that node.

Branches. Paths that follow from answers. If a node asks "Owns a car?", the 'Yes' branch leads to one set of follow-ups, while 'No' leads to another.

We visualize these components with a concrete example in Figure 1.

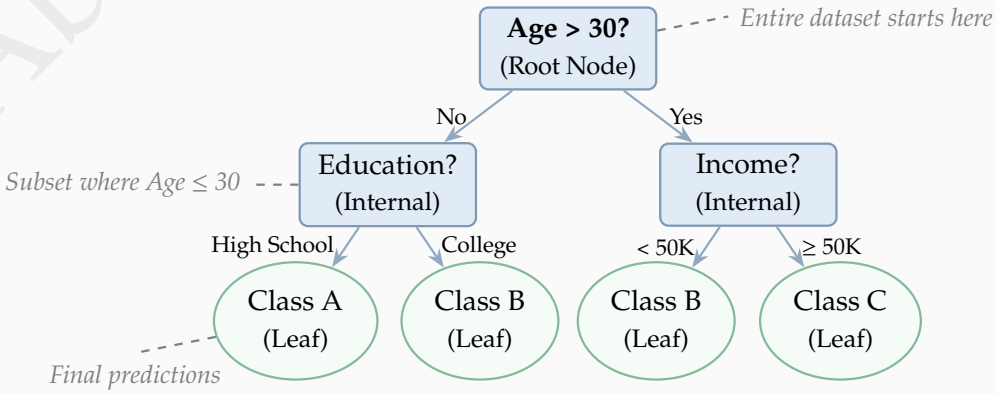

Figure 1: *Anatomy of a decision tree showing how data flows from the root to the leaves through internal decision nodes*

2.2 How Trees Make Predictions

The prediction process is straightforward and deterministic:

Prediction Algorithm

Given a new instance $\mathbf{x}$:

1. Start at the root node
2. Evaluate the test condition using the instance's features
3. Follow the branch corresponding to the test outcome
4. Repeat steps 2–3 until reaching a leaf node
5. Return the leaf's prediction (class label or value)

This process creates what we call a **decision path**—a sequence of tests that leads to a specific prediction. Each path can be interpreted as an IF-THEN rule, making the model's logic transparent and auditable.

Tree Traversal Example

Consider classifying a person with: `Age = 28`, `Education = College`.

Traversal path:

1. Root asks: "Age > 30?" → Answer: No → Follow left branch
2. Internal node asks: "Education?" → Answer: College → Follow right branch
3. Reach leaf: **Class B** → This is our prediction

3 The Mathematics of Good Questions

The key challenge in building a decision tree is determining which questions to ask and in what order. We need a mathematical framework to evaluate the quality of each potential split. This is where information theory provides elegant solutions.

3.1 Measuring Uncertainty: Entropy

When we have a dataset with mixed classes, we have uncertainty about which class a randomly selected instance belongs to. **Entropy** quantifies this uncertainty.

Understanding Entropy Through Examples

Imagine three boxes of colored balls:
Box 1 - No Uncertainty: 100 red balls. Reaching in blindfolded, what will you pull? Red, certainly. Your uncertainty is zero. **Entropy: 0 bits**
Box 2 - Maximum Uncertainty: 50 red, 50 blue. What now? Could be either—a coin flip. Your uncertainty is maximum. **Entropy: 1 bit**

Box 3 - Some Uncertainty: 90 red, 10 blue. What will you pull? Probably red, but uncertain. Some uncertainty remains. **Entropy: 0.47 bits**
Entropy quantifies this uncertainty. Pure groups have zero entropy. Perfectly mixed groups have maximum entropy. Everything else falls between.

Entropy Formula

For a dataset D with c classes, where p_i is the proportion of instances in class i:

$$H(D) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

where we define $0 \log_2 0 = 0$ by convention. Using base 2 gives entropy in **bits**.

Let's verify our intuitive examples mathematically:

Box 1 (pure): $p_{\text{red}} = 1.0$, $p_{\text{blue}} = 0$

$$H = -1.0 \cdot \log_2(1.0) - 0 \cdot \log_2(0) = 0 \text{ bits}$$

Box 2 (balanced): $p_{\text{red}} = 0.5$, $p_{\text{blue}} = 0.5$

$$H = -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) = 1.0 \text{ bit}$$

Box 3 (skewed): $p_{\text{red}} = 0.9$, $p_{\text{blue}} = 0.1$

$$H = -0.9 \cdot \log_2(0.9) - 0.1 \cdot \log_2(0.1) \approx 0.47 \text{ bits}$$

The formula perfectly captures our intuition! The relationship between class distribution and entropy follows a predictable pattern:

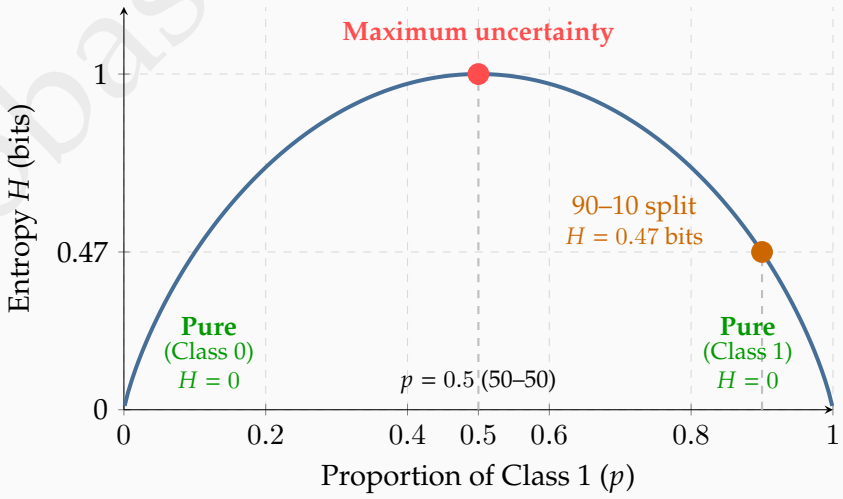

Figure 2: *Entropy curve for binary classification. The curve is symmetric, peaking at equal class distribution (50–50 split = 1 bit of entropy). Pure nodes (all one class) have zero entropy, representing no uncertainty.*

3.2 Entropy Range for Multiple Classes

The entropy range depends on the number of classes. Maximum entropy occurs when all classes are equally likely, with value $H_{\max} = \log_2(c)$ for c classes.

For binary classification, $H_{\max} = \log_2(2) = 1$ bit, giving range $[0, 1]$.

For multi-class problems:

- 3 classes: $H_{\max} = \log_2(3) \approx 1.585$ bits
- 4 classes: $H_{\max} = \log_2(4) = 2$ bits
- c classes: $H_{\max} = \log_2(c)$ bits

3.3 Why Entropy Matters for Decision Trees

When splitting data at a node, we want to reduce entropy:

- **Before split:** High entropy (mixed classes)
- **After split:** Low average entropy (purer child nodes)
- **Success measure:** Total entropy decrease

This decrease is **Information Gain**—our next topic.

3.4 Information Gain: The Value of a Question

Once we can measure uncertainty, we can quantify how much a particular question (split) reduces that uncertainty. This reduction is called **Information Gain**.

Information Gain Intuition

Think of Information Gain as the score for asking a question:

1. Start with initial impurity (parent entropy)
2. Propose a question (split on an attribute like "Owns a car?")
3. Question splits data into new groups (child nodes)
4. Calculate each new group's impurity
5. Calculate average child impurity, weighting by group size
6. **Information Gain** = (Impurity Before) - (Average Impurity After)

Higher Information Gain means the question better created purer subgroups—the quantified "Aha!" moment.

Information Gain

Let a dataset D at a parent node be split on attribute A into k subsets $\{D_1, D_2, \ldots, D_k\}$. The **information gain** is the reduction in entropy:

$$\underbrace{IG(D, A)}_{\text{Gain from split A}} = \underbrace{H(D)}_{\text{Parent Entropy}} - \underbrace{\sum_{j=1}^{k} \frac{|D_j|}{|D|} H(D_j)}_{\text{Weighted Average Child Entropy}}$$

The term $\frac{|D_j|}{|D|}$ is the weight for the j-th child node, representing the fraction of data that goes down that branch. This ensures that creating a tiny, pure node at the cost of leaving a large, impure node is penalized.

The intuition is simple: a good split creates subsets that are purer (lower entropy) than the original dataset. The Information Gain measures this improvement.

Calculating Information Gain

Suppose we have 100 customers (60 buyers, 40 non-buyers) and want to evaluate splitting by "Income > 50K":
Before split: $H(D) = -0.6 \log_2(0.6) - 0.4 \log_2(0.4) = 0.971$ bits
After split:

- High income (30 instances): 25 buyers, 5 non-buyers $\rightarrow H = 0.650$ bits
- Low income (70 instances): 35 buyers, 35 non-buyers $\rightarrow H = 1.000$ bits

Information Gain: $IG = 0.971 - (0.3 \times 0.650 + 0.7 \times 1.000) = 0.076$ bits
This split provides 0.076 bits of information about the target class.

Part II: Building Trees

4 The ID3 Algorithm: A Step-by-Step Journey

Now that we understand entropy and information gain, we can explore how the ID3 (Iterative Dichotomiser 3) algorithm uses these concepts to build decision trees automatically.

4.1 The Core Algorithm

ID3 follows a greedy, top-down approach:

ID3 Algorithm

Input: Training set D, Attribute set A
Process:

1. If all instances in D belong to the same class c:
   - Return a leaf node with label c
2. If A is empty:
   - Return a leaf node with the majority class in D
3. Otherwise:
   - Calculate $IG(D, a)$ for each attribute $a \in A$
   - Select $a^* = \arg\max_{a \in A} IG(D, a)$

- Create a node with test a^*
- For each value v of a^*:
  - Create subset D_v where $a^* = v$
  - Recursively build subtree with $(D_v, A \setminus \{a^*\})$
  - Attach subtree as child via branch v

Output: Decision tree rooted at the created node

The elegance of ID3 lies in its simplicity: at each step, choose the attribute that provides the most information about the target class.

ID3's greedy strategy is now clear: at each node, calculate Information Gain for every possible attribute, then choose the highest:

$$A^* = \arg\max_A IG(D, A)$$

4.2 Why Greedy Works (Usually)

You might wonder: why not look ahead several steps to find the globally optimal tree? The answer involves computational complexity. Finding the optimal decision tree is NP-complete, meaning it's computationally infeasible for all but the smallest datasets.

The greedy approach works surprisingly well because:

- Good initial splits tend to create naturally separable subgroups
- Information gain has strong theoretical foundations in information theory
- The recursive structure allows the algorithm to adapt to local patterns

5 Complete Worked Example: Transportation Choice

Let's build a complete decision tree by hand to solidify our understanding. We'll use a transportation dataset that's small enough to follow each calculation, yet rich enough to demonstrate key concepts.

5.1 The Dataset

We have 10 training examples with four attributes predicting transportation mode:

ID	Gender	Car Own.	Travel Cost	Income	**Transport**
1	M	0	Cheap	Low	Bus
2	M	1	Cheap	Med	Bus
3	F	1	Cheap	Med	Train
4	F	0	Cheap	Low	Bus
5	M	1	Cheap	Med	Bus
6	M	0	Standard	Med	Train
7	F	1	Standard	Med	Train
8	F	1	Expensive	High	Car
9	M	2	Expensive	Med	Car
10	F	2	Expensive	High	Car

Table 1: *Transportation choice training dataset with class distribution: Bus (4), Train (3), Car (3)*

5.2 Step 1: Calculate Initial Entropy

First, we calculate the entropy of the entire dataset:

$$\begin{aligned} H(D) &= -\sum_{i \in \{\text{Bus, Train, Car}\}} p_i \log_2 p_i \\ &= -\left(\frac{4}{10}\log_2\frac{4}{10} + \frac{3}{10}\log_2\frac{3}{10} + \frac{3}{10}\log_2\frac{3}{10}\right) \\ &= -(0.4 \times (-1.322) + 0.3 \times (-1.737) + 0.3 \times (-1.737)) \\ &= -(-0.529 - 0.521 - 0.521) = 1.571 \text{ bits} \end{aligned}$$

This high entropy (close to the maximum of $\log_2(3) \approx 1.585$) indicates significant uncertainty in our dataset.

5.3 Step 2: Evaluate All Possible Splits

Now we calculate the information gain for each attribute. Let's work through Travel Cost in detail:

5.3.1 Travel Cost Split

Partitioning by Travel Cost creates three subsets:

- **Cheap** (5 instances): Bus(4), Train(1)

$$H_{\text{cheap}} = -\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5} = 0.722 \text{ bits}$$

- **Standard** (2 instances): Train(2) — Pure node!

$$H_{\text{standard}} = 0 \text{ bits}$$

- **Expensive** (3 instances): Car(3) — Pure node!

$$H_{\text{expensive}} = 0 \text{ bits}$$

Weighted average entropy after split:

$$H(D|\text{Travel Cost}) = \frac{5}{10}(0.722) + \frac{2}{10}(0) + \frac{3}{10}(0) = 0.361$$

Information Gain:

$$IG(D, \text{Travel Cost}) = 1.571 - 0.361 = 1.210 \text{ bits}$$

5.3.2 Summary of All Splits

Following similar calculations for other attributes:

Attribute	Weighted Entropy	Information Gain
Gender	1.446	0.125
Car Ownership	1.036	0.535
Travel Cost	**0.361**	**1.210**
Income Level	0.875	0.696

Table 2: *Information gain for all attributes at the root*

Travel Cost provides the highest information gain, so it becomes our root split.

5.4 Step 3: Build the First Level

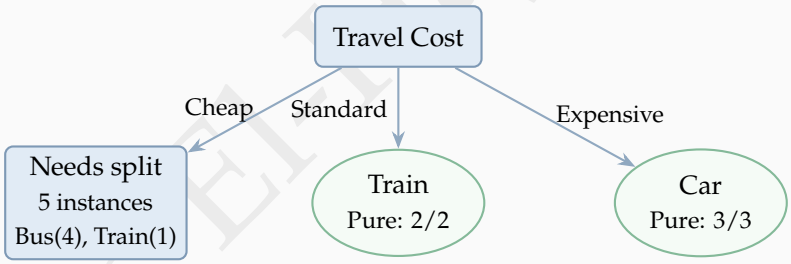

Figure 3: *Tree after first split. Two branches are already pure!*

5.5 Step 4: Refine the "Cheap" Branch

The "Cheap" node still has mixed classes. We evaluate the remaining attributes on these 5 instances:

Attribute	Information Gain
Gender	**0.322**
Car Ownership	0.171
Income Level	0.171

Table 3: *Information gains for the "Cheap" subset*

Gender wins, creating:

- Male (3 instances): All Bus → Pure!
- Female (2 instances): Bus(1), Train(1) → Needs further splitting

5.6 Step 5: Complete the Tree

For the final split (Female + Cheap), any remaining attribute will create pure leaves. We choose Car Ownership:

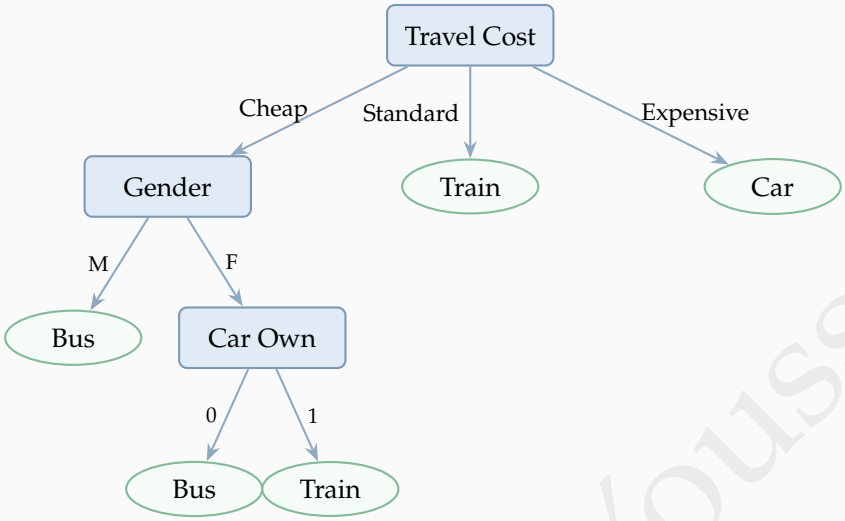

Figure 4: *Final decision tree achieving 100% training accuracy*

5.7 Insights from Our Tree

Our completed tree reveals several important patterns:

What the Tree Teaches Us

1. **Feature Dominance:** Travel Cost alone explains 77% of the entropy (1.21/1.571 bits).
2. **Asymmetric Structure:** Different branches have different depths, reflecting varying complexity in different regions of the feature space.
3. **Conditional Relevance:** Gender only matters for cheap travel, and car ownership only for females choosing cheap travel. This illustrates how decision trees naturally capture *interaction effects*.
4. **Implicit Feature Selection:** Income never appears in the tree despite having reasonable individual correlation (IG = 0.696). It was always outperformed at each decision point.
5. **Clear Decision Rules:** The tree yields interpretable rules:

```
IF travel_cost = Standard THEN Train
IF travel_cost = Expensive THEN Car
IF travel_cost = Cheap AND gender = Male THEN Bus
IF travel_cost = Cheap AND gender = Female AND car = 0 THEN Bus
IF travel_cost = Cheap AND gender = Female AND car ≥ 1 THEN Train
```

6. **Perfect Training Accuracy:** The tree correctly classifies all 10 training examples—but this might indicate overfitting! Real applications require validation and pruning strategies.

Part III: Evolution of Algorithms

6 The Challenge: ID3's Hidden Weakness

While ID3 elegantly builds decision trees, it suffers from a critical flaw that wasn't immediately obvious. Let's discover this limitation through a revealing example.

6.1 The Bias Toward High-Cardinality Attributes

Consider a modified version of our dataset where we add a "Student ID" attribute:

Student ID	Age Group	Income	Purchase
001	Teen	Low	No
002	Teen	High	No
003	Adult	Low	Yes
004	Adult	High	Yes
005	Senior	Low	No
006	Senior	High	Yes

Table 4: *Student laptop purchase dataset with ID column*

If we split on Student ID, each student gets their own leaf node—perfectly pure! The information gain would be maximal (1.0 bits). But what predictive value does "If StudentID = 003, then Purchase = Yes" provide for new students?

The Overfitting Trap

ID3's use of raw Information Gain creates a systematic bias toward attributes with many unique values. These attributes can always create pure partitions by essentially memorizing the training data, leading to trees that don't generalize.

This isn't just a theoretical concern. Real datasets often contain:

- ID numbers or codes
- Timestamps with fine granularity
- Free-text fields with many unique values
- Continuous variables treated as categorical

Any of these could dominate the tree-building process, creating models that perform perfectly on training data but fail on new instances.

6.2 Comparing Meaningful and Meaningless Splits

Splitting by Student ID

With 6 unique values creating perfectly pure leaves:

$$\text{IG(StudentID)} = 1.0 - 0 = 1.0 \text{ bits (maximum)}$$

But this merely memorizes: "If StudentID = 3, then Purchase = Yes" tells us nothing about new students.

Splitting by Age Group

Creates three subsets with meaningful patterns:

- **Teen** (2 instances): No (2) — Pure!
- **Adult** (2 instances): Yes (2) — Pure!
- **Senior** (2 instances): No (1), Yes (1) — Mixed

Weighted entropy:

$$H(D|\text{Age Group}) = \frac{2}{6}(0) + \frac{2}{6}(0) + \frac{2}{6}(1) = \frac{1}{3}$$

Information Gain:

$$\text{IG}(\text{Age Group}) = 1.0 - \frac{1}{3} = 0.667 \text{ bits}$$

Despite Age Group being clearly more meaningful for prediction, ID3 would choose Student ID due to its higher Information Gain (1.0 vs 0.667). This is the **high-cardinality bias problem**.

7 C4.5: Refining the Approach

Ross Quinlan, ID3's creator, recognized this limitation and developed C4.5 with several key improvements. The most important innovation was the Gain Ratio, which penalizes excessive fragmentation.

7.1 The Gain Ratio Solution

Gain Ratio

C4.5 normalizes Information Gain by the intrinsic information of the split:

Split Information:

$$\text{SplitInfo}(D, A) = -\sum_{v \in \text{Values}(A)} \frac{|D_v|}{|D|} \log_2\left(\frac{|D_v|}{|D|}\right)$$

Gain Ratio:

$$\text{GainRatio}(D, A) = \frac{\text{IG}(D, A)}{\text{SplitInfo}(D, A)}$$

The Split Information measures how broadly and evenly an attribute splits the data. Attributes that create many small partitions have high Split Information, which reduces their Gain Ratio.

7.2 Comparing ID3 and C4.5 on Our Example

Let's see how Gain Ratio solves the Student ID problem:

Student ID:

- Information Gain: 1.0 bits (maximum)
- Split Information: $-6 \times \frac{1}{6}\log_2\left(\frac{1}{6}\right) = \log_2(6) = 2.585$ bits
- Gain Ratio: $1.0/2.585 = 0.387$

Age Group:

- Information Gain: 0.667 bits
- Split Information: $-3 \times \frac{2}{6} \log_2\left(\frac{2}{6}\right) = \log_2(3) = 1.585$ bits
- Gain Ratio: $0.667/1.585 = 0.421$

Now Age Group correctly wins! The normalization successfully penalizes the overfragmentation of Student ID while preserving Age Group's meaningful patterns.

7.3 Additional C4.5 Improvements

Beyond Gain Ratio, C4.5 introduced several practical enhancements:

C4.5's Complete Feature Set

1. **Handling Missing Values:**
   - Distributes instances with missing values proportionally across branches
   - Weights instances based on the probability of taking each branch
2. **Continuous Attributes:**
   - Automatically finds optimal split thresholds
   - Sorts values and evaluates splits between consecutive values
3. **Post-Pruning:**
   - Builds complete tree then prunes back
   - Uses pessimistic error estimation
4. **Rule Generation:**
   - Converts tree to if-then rules
   - Simplifies rules by removing unnecessary conditions

These enhancements make C4.5 significantly more practical for real-world applications than ID3.

8 CART: A New Philosophy

While C4.5 evolved from ID3, the Classification and Regression Trees (CART) algorithm took a fundamentally different approach, introducing several innovative concepts that remain influential today.

8.1 Binary Splits: Simplicity Through Constraint

CART's most visible difference is its exclusive use of binary splits:

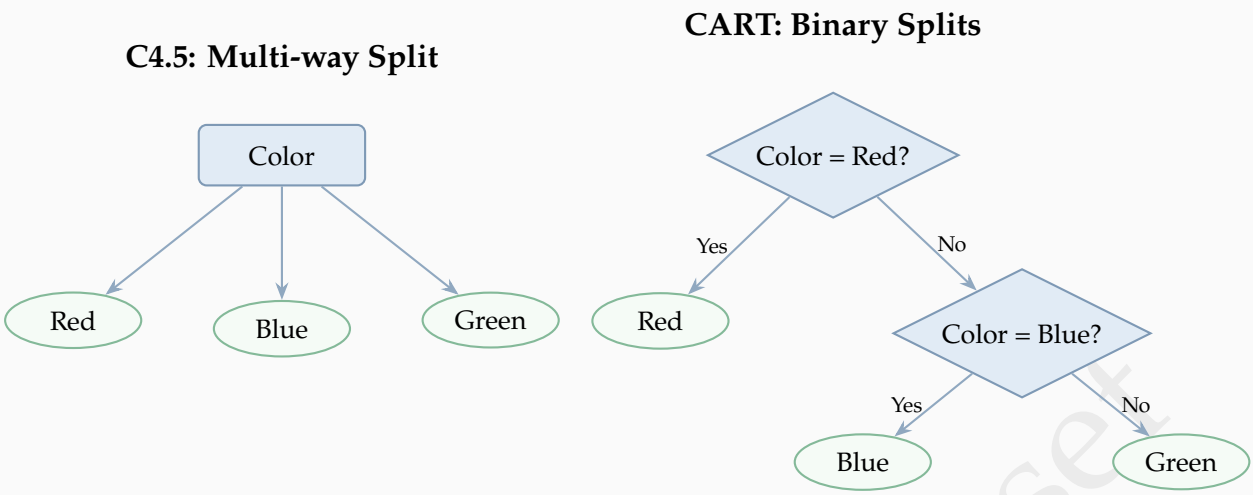

Figure 5: *Comparison of splitting strategies: C4.5 creates one three-way split; CART creates a cascade of binary decisions*

Why Limit to Binary Splits?

Binary splits offer several advantages:

- **Flexibility:** Any partition can be created through multiple binary splits
- **No fragmentation bias:** Every split creates exactly two children, avoiding the bias toward high-cardinality features
- **Better for continuous variables:** Natural threshold representation (e.g., $x \leq 5.3$)
- **Computational advantages:** Simpler to optimize and implement
- **Foundation for ensembles:** Most modern ensemble methods (Random Forests, Gradient Boosting) use binary trees

8.2 Gini Impurity: An Alternative to Entropy

CART typically uses Gini Impurity instead of entropy as its splitting criterion.

Gini Impurity

For a dataset D with class proportions p_i for $i = 1, \ldots, c$:

$$\text{Gini}(D) = 1 - \sum_{i=1}^{c} p_i^2$$

Interpretation: The probability of misclassifying a randomly chosen element if it were randomly labeled according to the class distribution.

Why Gini Works

Consider a node with 100 instances: 70 Class A, 30 Class B.
If you randomly pick an instance and randomly assign it a label (70% chance A, 30% chance B), what is the probability you are wrong?

- Pick A, label B: $0.7 \times 0.3 = 0.21$
- Pick B, label A: $0.3 \times 0.7 = 0.21$
- Total misclassification probability: $0.21 + 0.21 = 0.42$

Using the Gini formula:

$$\text{Gini} = 1 - (0.7^2 + 0.3^2) = 1 - 0.58 = 0.42 \quad \checkmark$$

Comparing Gini and Entropy

Both metrics serve similar purposes but have subtle differences:

- **Computational efficiency:** Gini is faster to compute (no logarithms required)
- **Ranking consistency:** Both rank splits similarly in practice
- **Range:** Gini ranges from 0 (pure) to $1 - \frac{1}{c}$ for c classes
- **Tree structure:** Entropy tends to produce slightly more balanced trees
- **Performance:** Differences in predictive performance are usually negligible

8.3 Unified Classification and Regression

CART's most significant innovation was treating classification and regression uniformly. The same algorithmic framework, with different splitting criteria and prediction rules, handles both tasks seamlessly.

Task	Splitting Criterion	Leaf Prediction
Classification	Gini Impurity or Cross-Entropy	Majority class
Regression	Mean Squared Error (MSE)	Mean of target values

Table 5: *CART's unified framework for classification and regression*

This unified approach was groundbreaking, as it demonstrated that tree-based learning could be formalized as a consistent optimization problem regardless of the target variable type.

Part IV: Advanced Topics

9 CART for Regression: Beyond Classification

Let's explore how CART handles regression through a concrete example, building a tree to predict house prices.

9.1 The Regression Setting

Our dataset contains houses with features and prices:

Table 6: *House price prediction dataset*

House	Size (sqft)	Bedrooms	Age (years)	Price ($K)
h_1	1500	2	5	200
h_2	2000	3	10	250
h_3	1200	2	20	150
h_4	1800	2	15	180
h_5	2200	4	8	300

9.2 The Splitting Criterion: Minimizing MSE

For regression, CART finds splits that minimize the mean squared error:

CART Regression Splitting

For a potential split creating partitions D_L and D_R:

1. Calculate mean target value in each partition:

$$\bar{y}_L = \frac{1}{|D_L|} \sum_{i \in D_L} y_i, \quad \bar{y}_R = \frac{1}{|D_R|} \sum_{i \in D_R} y_i$$

2. Calculate MSE within each partition:

$$\text{MSE}_L = \frac{1}{|D_L|} \sum_{i \in D_L} (y_i - \bar{y}_L)^2$$

3. Calculate weighted average MSE:

$$\text{MSE}_{\text{split}} = \frac{|D_L|}{|D|} \text{MSE}_L + \frac{|D_R|}{|D|} \text{MSE}_R$$

4. Choose split minimizing $\text{MSE}_{\text{split}}$

9.3 Building the Regression Tree

To illustrate CART's regression capabilities, consider a house price prediction problem. Let us evaluate the split "Bedrooms ≤ 2' :

Left Partition (Bedrooms ≤ 2): Houses $\{h_1, h_3, h_4\}$

- Prices: $\{200, 150, 180\}$ (in thousands)
- Mean: $\bar{y}_L = 176.67$
- Mean Squared Error:

$$\text{MSE}_L = \frac{(200 - 176.67)^2 + (150 - 176.67)^2 + (180 - 176.67)^2}{3} = 422.23$$

Right Partition (Bedrooms > 2): Houses $\{h_2, h_5\}$

- Prices: $\{250, 300\}$ (in thousands)

- Mean: $\bar{y}_R = 275$
- Mean Squared Error:

$$\text{MSE}_R = \frac{(250 - 275)^2 + (300 - 275)^2}{2} = 625$$

Weighted MSE: The overall quality of this split is measured by the weighted average:

$$\text{MSE}_{\text{weighted}} = \frac{3}{5}(422.23) + \frac{2}{5}(625) = 503.34$$

After evaluating all possible splits on all features, CART constructs the following tree:

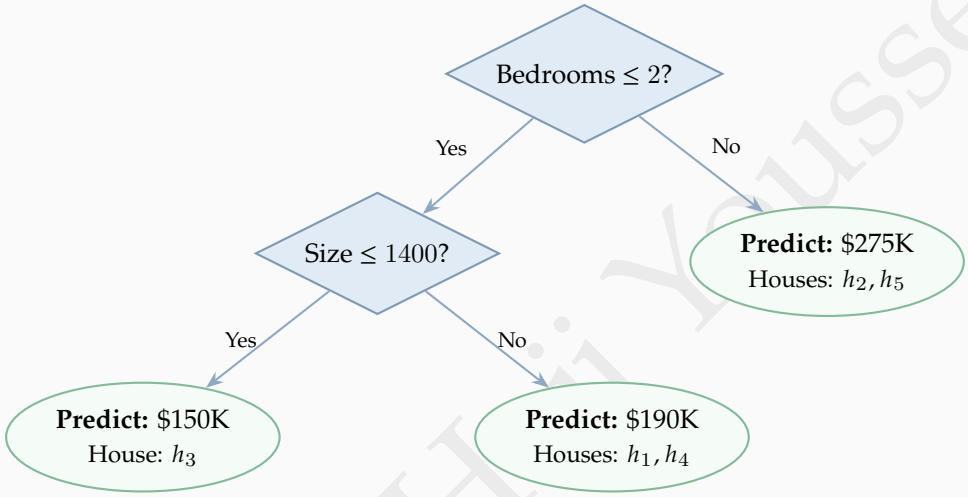

Figure 6: *CART regression tree for house price prediction. Each leaf node predicts the mean price of its training instances, minimizing the squared error within that region.*

9.4 Making Predictions

For new houses, we traverse the tree and output the corresponding leaf's prediction:

- 1600 sqft, 2 bedrooms: Root→Left→Right → Predict $190K
- 2100 sqft, 3 bedrooms: Root→Right → Predict $275K
- 1100 sqft, 2 bedrooms: Root→Left→Left → Predict $150K

10 Controlling Complexity: The Art of Pruning

All decision tree algorithms face a fundamental tension: trees that fit training data perfectly often generalize poorly. Pruning addresses this challenge by simplifying trees to improve generalization performance.

10.1 Why Trees Overfit

Decision trees are particularly prone to overfitting due to several inherent characteristics:

- They can create arbitrarily complex decision boundaries
- They continue splitting until achieving purity (or near-purity)
- They are sensitive to small variations in training data

- They can memorize noise as if it were genuine signal

The Overfitting Phenomenon

Imagine teaching a child to identify dogs. If shown only three dogs—a golden retriever, a poodle, and a beagle—the child might create overly specific rules:

"Dogs have golden fur, or curly white fur, or brown spots with floppy ears."

This memorizes the examples but fails on a German Shepherd. Similarly, unpruned trees create overly specific rules that memorize training data but fail to generalize to new instances.

10.2 Pre-Pruning: Stopping Early

Pre-pruning prevents overfitting by stopping tree growth before reaching complete purity. This approach sets constraints during the tree-building process.

Pre-Pruning Strategies

Common pre-pruning hyperparameters include:

- **Maximum Depth:** Stop splitting when reaching a specified tree depth
- **Minimum Samples for Split:** Require a minimum number of instances to justify splitting a node
- **Minimum Samples per Leaf:** Ensure leaf nodes have sufficient support (minimum instances)
- **Minimum Impurity Decrease:** Only split if the impurity improvement exceeds a threshold
- **Maximum Leaf Nodes:** Limit the total number of leaf nodes in the tree

Pre-pruning is computationally efficient but risks stopping too early, known as the **horizon effect**, potentially missing important patterns that would emerge deeper in the tree.

10.3 Post-Pruning: Growing Then Trimming

Post-pruning takes a different approach: first grow a complete tree, then systematically remove nodes that do not improve validation performance. This strategy avoids the horizon effect by examining the full tree structure before making pruning decisions.

Cost-Complexity Pruning (CART)

The CART algorithm uses the following procedure:

1. Grow a complete tree T_0 until all leaves are pure (or meet stopping criteria)
2. Define the cost-complexity criterion:

$$R_\alpha(T) = R(T) + \alpha|T|$$

where $R(T)$ is the training error rate, $|T|$ is the number of leaf nodes, and $\alpha \geq 0$ is the

complexity parameter

3. For increasing values of α, find the optimal pruned subtree T_α that minimizes $R_\alpha(T)$
4. Use a validation set (or cross-validation) to select the best value of α

The parameter α controls the trade-off between accuracy and simplicity:

- $\alpha = 0$: No pruning penalty (retains full tree)
- $\alpha \to \infty$: Maximum pruning penalty (collapses to single leaf)
- Optimal α: Balances bias-variance trade-off for best generalization

10.4 Pruning in Practice

Different algorithms employ different pruning strategies, reflecting their design philosophies and historical development.

Algorithm	Pruning Method
ID3	None (prone to overfitting)
C4.5	Pessimistic error pruning
CART	Cost-complexity pruning

Table 7: *Pruning strategies by algorithm*

Modern implementations typically combine both methods: pre-pruning speeds up tree construction, while post-pruning refines the final model.

11 Choosing the Right Tool

Understanding the strengths and trade-offs of each algorithm guides practical decision-making.

11.1 Algorithm Comparison

Aspect	ID3	C4.5	CART
Split Type	Multi-way	Multi-way	Binary only
Split Criterion	Information Gain	Gain Ratio	Gini/MSE
Handles Regression	No	No	Yes
Missing Values	No	Probabilistic	Surrogates
Continuous Features	No	Yes	Yes
Pruning	None	Pessimistic	Cost-complexity
Cardinality Bias	Severe	Corrected	N/A

Table 8: *Key algorithmic differences*

11.2 Selection Guidelines

When to Use Each Algorithm

ID3: Educational purposes, small categorical datasets, maximum interpretability.
C4.5: Varying feature cardinalities, missing data, automatic rule generation.
CART: Classification and regression, ensemble methods, computational efficiency.
Ensembles: Prediction accuracy paramount, sufficient data, production systems.

Modern Practice

Contemporary applications typically employ ensemble methods—Random Forests, Gradient Boosting, XGBoost—which combine multiple trees for superior predictive performance. These sacrifice interpretability for accuracy but rely fundamentally on the principles of individual decision trees.

12 Concluding Remarks

Core Principles

Information theory guides splitting: Entropy and information gain provide mathematically principled criteria for constructing decision boundaries.

Evolutionary improvements: ID3 introduced greedy information gain; C4.5 corrected cardinality bias; CART unified classification and regression through binary splits.

Overfitting management is essential: Trees naturally memorize training patterns. Pruning strategies—whether pre-pruning constraints or post-pruning refinement—are critical for generalization.

Context determines selection: Algorithm choice depends on data characteristics, interpretability requirements, and computational constraints.

Decision trees balance interpretability with modeling power, serving both as standalone interpretable models and as building blocks for state-of-the-art ensemble methods. The principles explored here—measuring uncertainty, constructing informative partitions, managing complexity—extend beyond trees to form conceptual foundations for understanding modern machine learning.