Introduction to Decision Trees

Machine Learning for Classification and Regression

October 29, 2025

What is Classification?

The Task
Given features about something, predict its category:

- Email features $\rightarrow$ Spam or Not Spam
- Patient symptoms $\rightarrow$ Disease A, B, or C
- Image pixels $\rightarrow$ Cat or Dog
- Transaction data $\rightarrow$ Fraud or Legitimate

Formal Definition
Input: Features $\mathbf{x} = (x_1, x_2, \ldots, x_p)$
Output: Class label $y \in \{1, 2, \ldots, K\}$

Learning from Examples

1. Collect labeled training data
2. Learn pattern from examples
3. Predict labels for new data

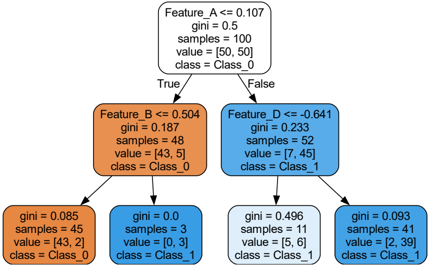
Many Algorithms Exist

- Logistic Regression
- K-Nearest Neighbors
- **Decision Trees** (today!)
- Neural Networks
- And many more...

Classification: Learning to assign categories based on features

What is a Decision Tree?

Visual Structure



A tree of decisions:

- **Root**: Start here
- **Internal nodes**: Ask question
- **Branches**: Answer (yes/no)
- **Leaves**: Final decision

How It Works

Start at top, follow path down:

Example: New loan application

1. Income > 50K? **Yes** → Go right
2. Credit Score > 700? **No** → Go left
3. Reach leaf: **REJECT**

Key Idea

Recursive binary partitioning:

- Split data based on features
- Repeat in each subset
- Until stopping criterion

Decision trees are hierarchical rules: Follow path from root to leaf

Classification Example

Problem: Loan Approval
Features:

- Income
- Credit Score
- Debt
- Employment Years

Decision: Approve or Reject?

Training Data
Learn from past decisions:

- 1000 approved loans
- 500 rejected loans

What We Want
A rule or model that:

- Captures patterns in past decisions
- Generalizes to new applications
- Is interpretable (explain decisions)
- Is accurate (makes good predictions)

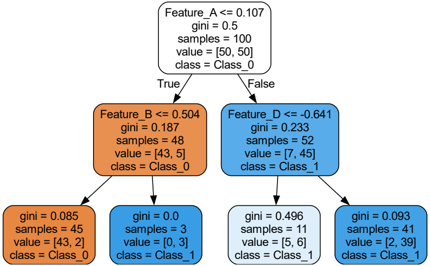
Decision Tree Approach
Create a series of yes/no questions:

- Is Income $>$ 50K?
- If yes, is Credit Score $>$ 700?
- If no, is Debt $<$ 20K?
- Etc.

Decision trees mimic human decision-making: Ask questions to reach conclusion

What is a Decision Tree?

Visual Structure



A tree of decisions:

- **Root**: Start here
- **Internal nodes**: Ask question
- **Branches**: Answer (yes/no)
- **Leaves**: Final decision

How It Works

Start at top, follow path down:

Example: New loan application

1. Income $>$ 50K? **Yes** $\rightarrow$ Go right
2. Credit Score $>$ 700? **No** $\rightarrow$ Go left
3. Reach leaf: **REJECT**

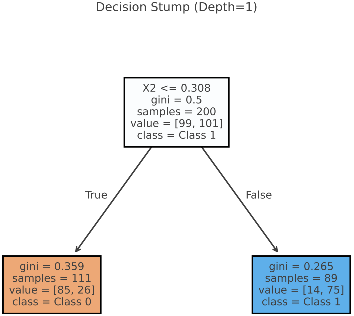
Key Idea

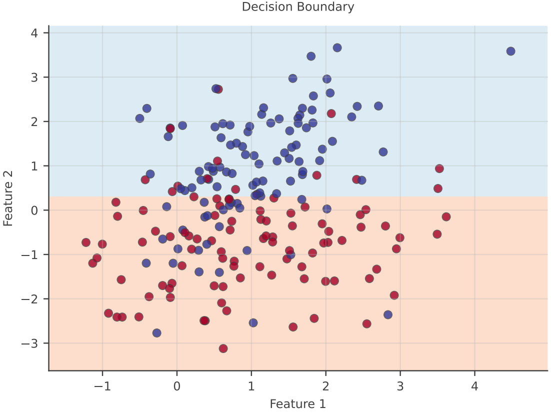
Recursive binary partitioning:

- Split data based on features
- Repeat in each subset
- Until stopping criterion

Decision trees are hierarchical rules: Follow path from root to leaf

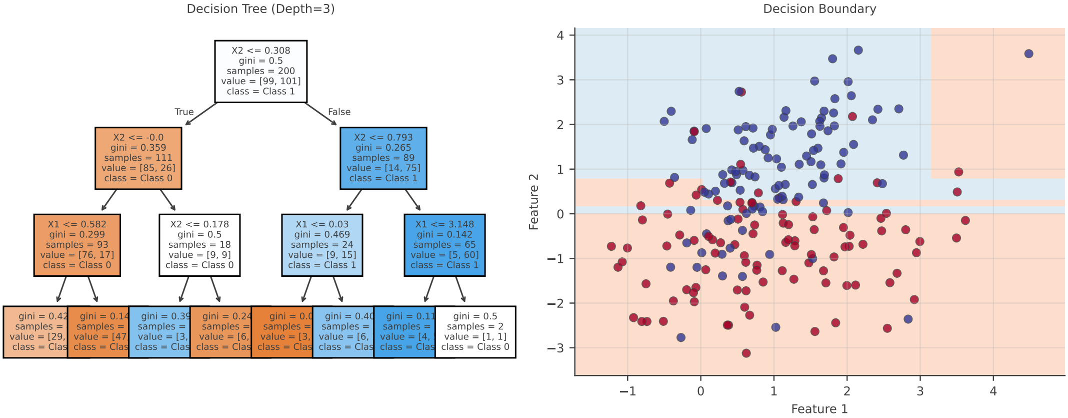
Tree Structure: Simple to Complex





Depth 1 (stump): Single question. Depth increases: More questions, finer decisions

Decision Trees Create Rectangular Regions



Each split creates axis-parallel rectangles: Unique property of decision trees

Splitting Criteria Overview

Goal

Create pure groups:

- All Class A in one group
- All Class B in another group

Impurity Measures

How mixed is a group?

Gini Impurity:

$$G = 1 - \sum_k p_k^2$$

Entropy:

$$H = -\sum_k p_k \log_2(p_k)$$

Lower = more pure

Best Split

1. Try all features
2. Try all thresholds
3. Choose split that:
   - Maximizes purity gain
   - Creates most separated groups

Greedy Algorithm

Make best split now, don't look ahead.

Recurse on left and right children.

Algorithm: Greedily split on feature + threshold that best separates classes

Building the Tree: Recursive Splitting

Algorithm

1. Start with all data at root
2. Find best split
3. Create two children
4. Recurse on each child
5. Stop when:
   - All points same class (pure)
   - Max depth reached
   - Too few points to split

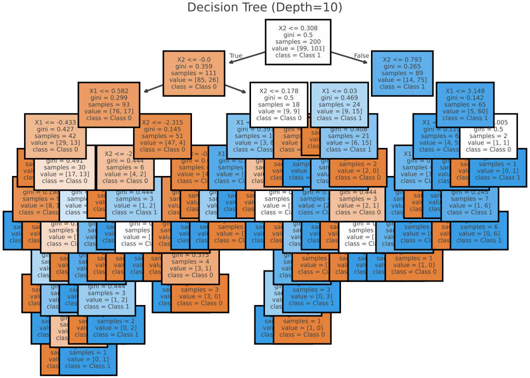
Stopping Criteria

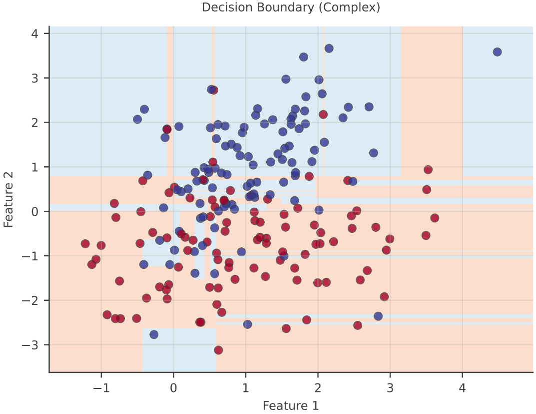
Prevent overfitting:

- `max_depth`: Limit tree depth
- `min_samples_split`: Min points to split
- `min_samples_leaf`: Min points in leaf

Without limits: Tree fits training data perfectly but overfits!

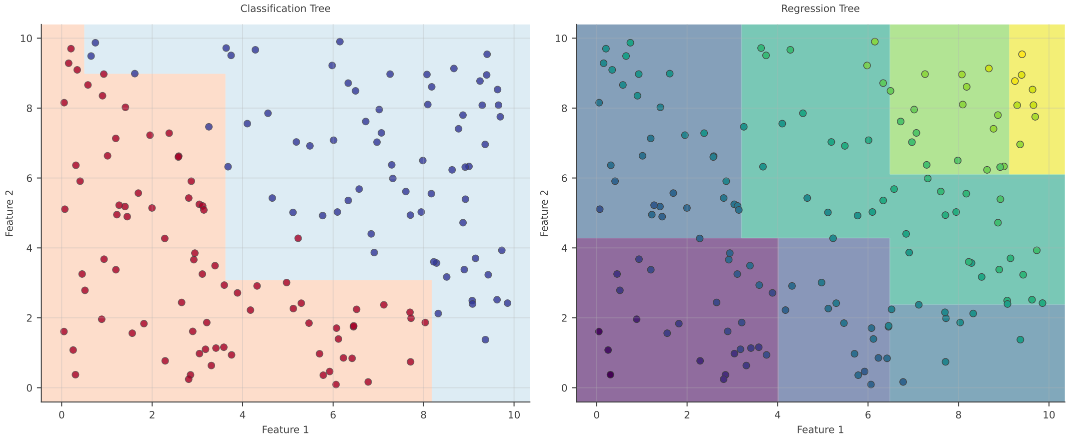
Recursive algorithm: Split, split, split until stopping criteria met





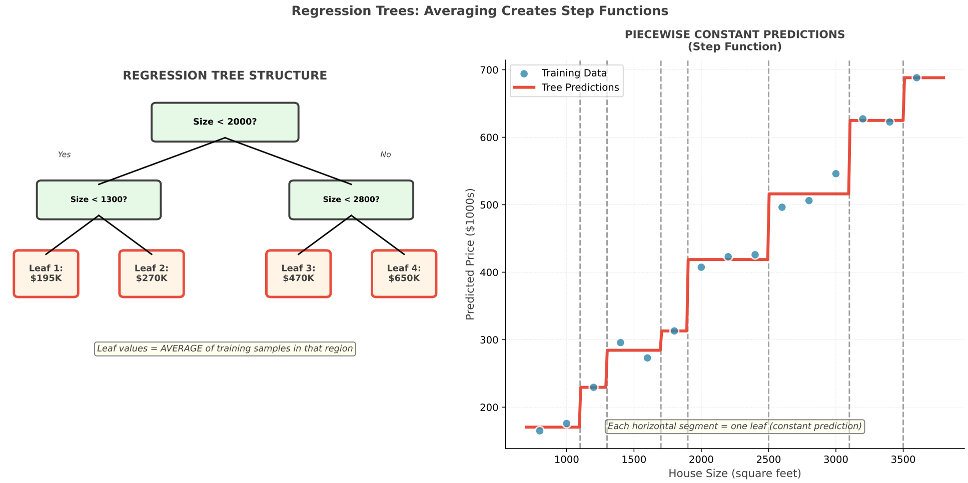
Deeper trees = more complex boundaries = higher risk of overfitting

Classification vs Regression Trees



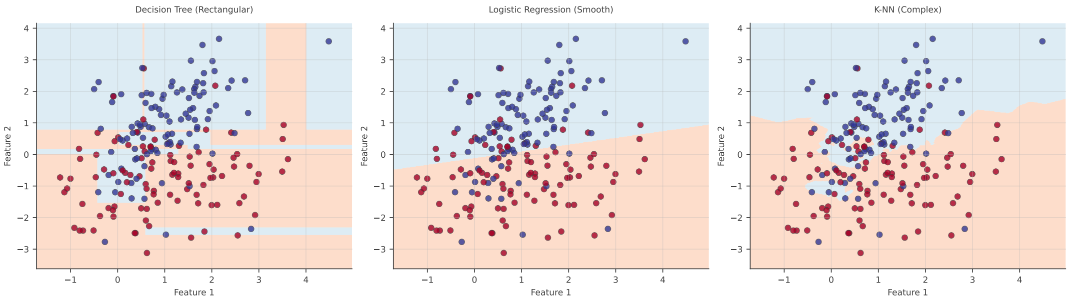
Decision trees work for both classification (majority vote) and regression (average)

Regression Trees in Detail



Left: Tree structure with average values in leaves — Right: Piecewise constant step function predictions

Comparison to Other Methods



Decision trees create rectangular boundaries unlike smooth boundaries of other methods

When to Use Decision Trees

Advantages

- **Interpretable**: Easy to explain
- **Visual**: Can draw the tree
- **No preprocessing**: No scaling needed
- **Handles mixed data**: Numeric + categorical
- **Non-linear**: Captures interactions
- **Fast prediction**: Just follow path

Best For

- Need interpretability
- Mixed feature types
- Non-linear patterns
- Baseline model

Disadvantages

- **Overfitting**: Grows until perfect fit
- **Unstable**: Small data change = different tree
- **Biased**: Favors features with more levels
- **Not smooth**: Rectangular boundaries
- **Greedy**: May miss globally optimal tree

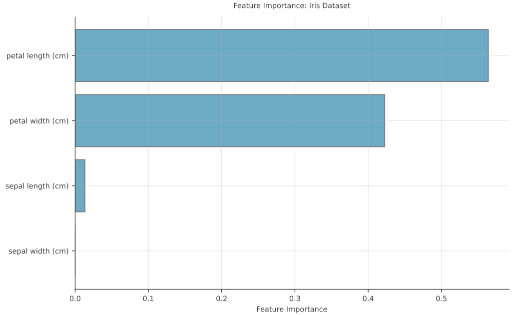
Not Ideal For

- Linear relationships
- Need smooth boundaries
- Very noisy data
- Small datasets

Decision trees: Interpretable and flexible but prone to overfitting

Using Trees to Find Innovations: Feature Importance

Discovery Through Importance
Decision trees reveal which factors matter most:



What This Tells Us

- Most important features listed first
- Focus resources on these factors
- Ignore low-importance features
- Discover unexpected drivers

Innovation Example
Customer churn prediction reveals:

1. Contract Length: 35% importance
2. Support Calls: 28% importance
3. Price: 18% importance
4. Age: 12% importance
5. Location: 7% importance

Innovation Insight
"We didn't realize support calls were the #2 driver of churn! Let's invest in better support."

Trees help discover what truly matters.

Using Trees for Discovery: Pattern Finding

Extract Interpretable Rules
Decision trees create IF-THEN rules:

Top 3 Churn Rules:

1. IF Support Calls $>$ 5 AND Contract $>$ 2 years THEN 85% churn
2. IF Price Increase $>$ 15% THEN 72% churn
3. IF Support Calls $=$ 0 AND Contract $<$ 1 year THEN 9% churn (loyal!)

Innovation Opportunities
From these rules:

- **Insight 1**: Long-term customers with many support calls are at risk - Innovation: Proactive support program for veterans
- **Insight 2**: Price sensitivity is non-linear (15% threshold) - Innovation: Keep increases $<$15%
- **Insight 3**: Low-touch customers are loyal - Innovation: Don't over-contact them

Decision trees reveal actionable patterns that drive business innovations.

Extract rules from trees to discover non-obvious patterns and opportunities

Real-World Applications

Medical Diagnosis
Symptoms → Disease

- Fever > 38°C?
- Cough present?
- Chest pain?
- → Diagnosis

Doctors can understand and verify rules.

Credit Scoring
Applicant data → Approve/Reject
Explainable to customers and regulators.

Customer Segmentation
Behavior → Customer Type

- Purchase frequency
- Average spend
- Product categories
- → Segment

Manufacturing Quality
Sensor readings → Defect/OK
Easy to implement on production line.

Decision trees excel where interpretability and explainability are crucial

Summary: Key Takeaways

What
Hierarchical tree of IF-THEN rules

How

1. Find best split (greedy)
2. Create children
3. Recurse
4. Stop when pure or criteria met

Impurity
Gini or Entropy measure purity

Advantages
Interpretable, visual, no preprocessing

Disadvantages
Overfitting, unstable, greedy

Solutions

- Limit depth (pruning)
- Use ensembles (Random Forest)
- Cross-validation

Decision trees are one of the most interpretable machine learning algorithms, making them ideal for applications requiring explainability.

Decision trees: Simple concept, powerful tool, foundation for ensemble methods

Questions?

See appendix for details

Appendix A1: Real-World Applications

Medical Diagnosis
Symptoms → Disease

- Fever > 38°C?
- Cough present?
- Chest pain?
- → Diagnosis

Doctors can understand and verify rules.

Credit Scoring
Applicant data → Approve/Reject
Explainable to customers and regulators.

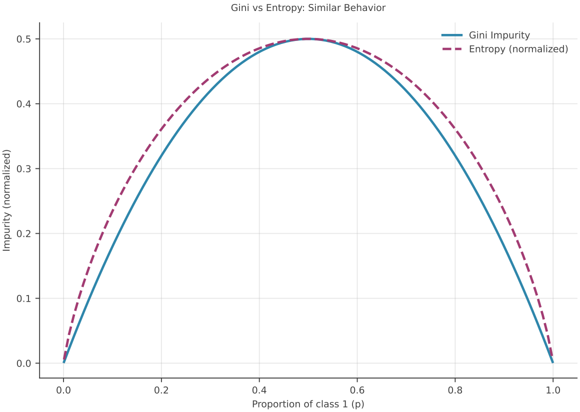
Customer Segmentation
Behavior → Customer Type

- Purchase frequency
- Average spend
- Product categories
- → Segment

Manufacturing Quality
Sensor readings → Defect/OK
Easy to implement on production line.

Decision trees excel where interpretability and explainability are crucial

Appendix A2: Impurity Measures Compared



Gini and Entropy give very similar results in practice

Appendix A3: Gini Impurity Formula

Definition

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

where p_k is proportion of class k

Properties

- $G = 0$ when pure (all one class)
- $G = 0.5$ when maximally impure (binary, 50/50)
- Lower is better

Example

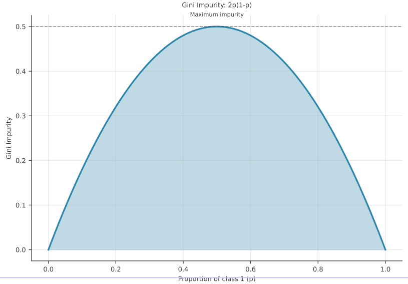
Node with 100 samples:

- 80 Class A
- 20 Class B

$$p_A = 0.8, \quad p_B = 0.2$$

$$G = 1 - (0.8^2 + 0.2^2)$$

$$G = 1 - 0.68 = 0.32$$



Gini: Fast to compute, commonly used in practice

Appendix A4: Entropy Formula

Definition

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

Information Theory

Entropy measures uncertainty or "surprise"

- $H = 0$ when certain (pure)
- $H = 1$ when maximally uncertain (binary, 50/50)
- Lower is better

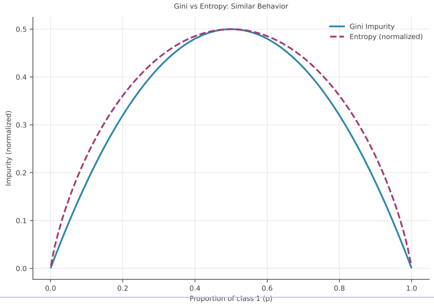
Same Example

Node with 100 samples:

- 80 Class A, 20 Class B

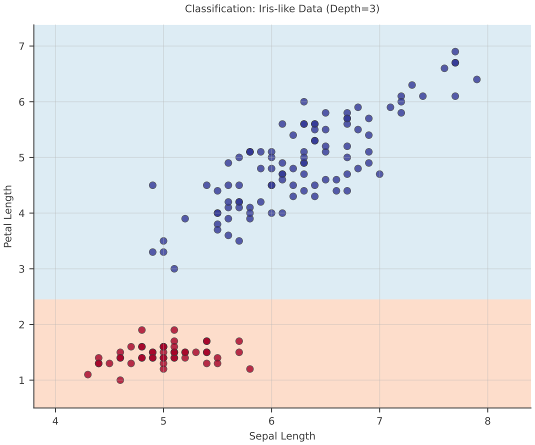
$$H = -(0.8 \log_2 0.8 + 0.2 \log_2 0.2)$$

$$H \approx 0.72$$

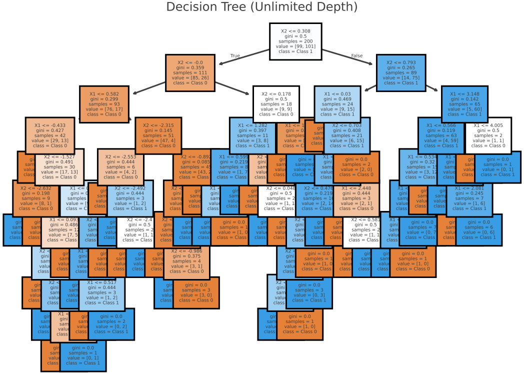


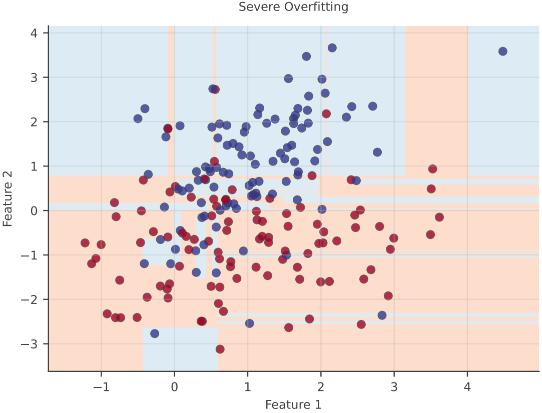
Entropy: Information-theoretic measure, similar results to Gini

Appendix A5: More Tree Examples



Example: Iris flower classification using petal/sepal measurements





Unlimited depth: Fits training data perfectly but creates isolated islands (overfitting)

Appendix A7: Hyperparameter Tuning

Key Hyperparameters

`max_depth`

- Maximum tree depth
- Default: None (unlimited)
- Typical: 3-10

`min_samples_split`

- Min samples to split node
- Default: 2
- Increase to prevent overfitting

`min_samples_leaf`

- Min samples in leaf
- Default: 1
- Increase for smoother model

How to Choose

1. Start with defaults
2. If overfitting:
   - Decrease `max_depth`
   - Increase `min_samples`
3. If underfitting:
   - Increase `max_depth`
   - Decrease `min_samples`
4. Use cross-validation

Quick Rule

Start with `max_depth=5`
Adjust based on validation performance.

Hyperparameters control tree complexity and prevent overfitting

Appendix A8: Using Decision Trees in Python

Scikit-learn

Very simple to use:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(
    max_depth=5
)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

That's it!

Visualization

Can export tree structure:

```
from sklearn.tree import plot_tree
plot_tree(model)
```

Or export to Graphviz for publication-quality diagrams.

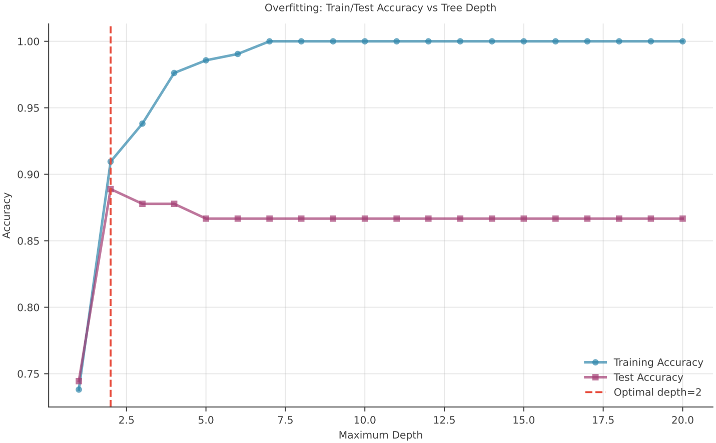
Feature Importance

`model.feature_importances_`

Shows which features matter most.

Python implementation is straightforward with scikit-learn

Appendix A9: The Overfitting Problem



Training accuracy keeps improving, test accuracy degrades: Classic overfitting

Pre-Pruning
Stop early:

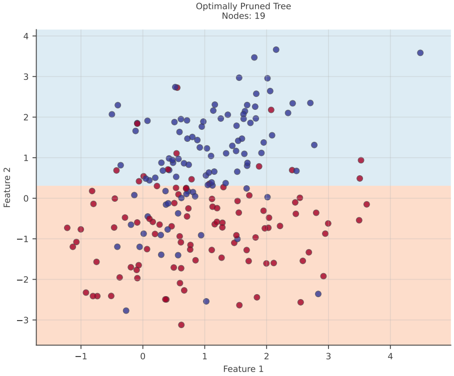
- Set `max_depth`
- Set `min_samples`
- Prevent growth

Simple and fast.

Post-Pruning
Grow full tree, then cut:

- Cost-complexity pruning
- Remove least important branches
- More optimal but slower

Comparison



Optimal pruning balances complexity and accuracy

Pruning is essential to prevent overfitting in decision trees

Appendix A11: Beyond Single Trees - Random Forest

The Problem
Single trees:

- High variance
- Unstable
- Overfit easily

The Solution
Random Forest:

- Train many trees
- On different data samples
- With random feature subsets
- Average predictions

Wisdom of Crowds
Many weak predictors $\rightarrow$ One strong predictor

Benefits

- Reduces overfitting
- More stable
- Better accuracy
- Still interpretable (feature importance)

Random Forest is one of the most powerful ML algorithms, built on decision trees!

Single trees are foundation for powerful ensemble methods like Random Forest

Appendix A12: How Feature Importance Works

Calculation Method

For each feature:

1. Find all splits using that feature
2. Calculate impurity reduction at each split
3. Weight by number of samples
4. Sum across all splits
5. Normalize to sum to 100%

Formula

$$\text{Importance}_j = \frac{\sum_{k \in \text{nodes}} n_k \cdot \Delta I_k}{\sum_{\text{all nodes}} n_k \cdot \Delta I_k}$$

where n_k = samples at node, ΔI_k = impurity decrease

Example

Tree with 3 splits on 100 samples:

- Income split: $0.20 \times 100 = 20$
- Credit split: $0.25 \times 60 = 15$
- Income split: $0.25 \times 40 = 10$

Total: 45

Importances:

- Income: $(20+10)/45 = 66.7\%$
- Credit: $15/45 = 33.3\%$

Key Points

- Root splits most important (affect all samples)
- Unused features: 0% importance
- Sum to 100%
- RF averages across trees (more stable!)

Feature importance: Weighted impurity reduction reveals which features drive decisions

Appendix A13: Further Learning

Key Concepts Covered

- Classification task
- Decision tree structure
- Splitting criteria (Gini/Entropy)
- Recursive algorithm
- Overfitting and pruning
- Hyperparameter tuning

Next Steps

- Practice with real datasets
- Try different hyperparameters
- Visualize your trees
- Compare to other algorithms
- Explore Random Forests

Resources

Books:

- "Introduction to Statistical Learning"
- "Pattern Recognition and Machine Learning"

Online:

- Scikit-learn documentation
- Decision tree visualizers
- Interactive tutorials

Practice:

- Kaggle datasets
- UCI ML Repository

Decision trees are a gateway to understanding machine learning