Lecture 18: Decision Trees for Classification

CS109A: Introduction to Data Science

Harvard University

■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 18

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Topics:** Decision Trees, Splitting Criteria, Gini Index, Entropy, Stopping Conditions, Overfitting

Lecture Overview

This lecture introduces **Decision Trees**, a fundamentally different approach to classification compared to logistic regression. Decision trees are intuitive, interpretable, and can model complex decision boundaries.

Key Topics:

1. Why we need decision trees (limitations of logistic regression)
2. How decision trees make predictions (tree traversal)
3. How decision trees learn (splitting criteria)
4. How to prevent overfitting (stopping conditions)

Key Insight: Decision trees work like a game of "20 Questions"—a series of yes/no questions leads to a final classification. This simple idea creates surprisingly powerful models that can handle non-linear decision boundaries.

Contents

1 Motivation: Why Not Just Use Logistic Regression?

1.1 What Logistic Regression Does Well

Logistic regression works beautifully when:

- Classes are **linearly separable** (or close to it)
- The **decision boundary** can be expressed as a straight line (or hyperplane)

Example:

Logistic Regression Success Consider classifying land as "agricultural" (green) vs "dry" (white) based on longitude and latitude.
If the data looks like this:

- Green points clustered on the right
- White points clustered on the left

A simple linear decision boundary works perfectly: latitude $= -0.8 \times$ longitude $+ 0.1$

1.2 Where Logistic Regression Fails

The problem arises when the decision boundary is **not linear**:

Example:

Logistic Regression Failure **Case 1: Circular boundary**

- Green points (agricultural) in the center
- White points (dry land) surrounding them in a ring

No straight line can separate these! You'd need a circle: $x_1^2 + x_2^2 < r^2$

Case 2: Scattered regions

- Green points in the top-left and bottom-right corners
- White points in the top-right and bottom-left corners

Now you'd need diagonal lines or complex polynomial boundaries!

Warning

The Polynomial Solution Has Limits

Yes, you could use polynomial logistic regression (adding x^2, x_1x_2 terms) to create curved boundaries. But:

- It requires manually engineering features
- Complex boundaries need very high-degree polynomials
- You don't know what shape the boundary should be beforehand

1.3 Our Wish List for a New Model

We want a model that can:

1. **Create complex decision boundaries** without manually engineering features

2. Be **easy to interpret**—we can explain why the model made a decision
3. Be **computationally efficient** to train and predict

Key Summary

Enter Decision Trees!

Decision trees satisfy all three criteria:

- Complex boundaries through recursive splitting
- Interpretable through a flowchart-like structure
- Efficient through simple comparison operations

2 The Intuition: Flowcharts and 20 Questions

2.1 Everyday Decision Making

We make tree-like decisions every day:

- "Is it raining? If yes, take umbrella. If no, check temperature..."
- "Did I finish my homework? If yes, go to party. If no, how much time left?..."

Example:

The Engineering Flowchart Classic problem-solving flowchart:
Question 1: Does it move?

- **Yes** → Question 2: Should it move?
  - Yes → *No problem!*
  - No → *Use duct tape*
- **No** → Question 2: Should it move?
  - Yes → *Use WD-40*
  - No → *No problem!*

This is exactly how a decision tree works! Binary questions leading to final decisions.

2.2 Key Properties of This Approach

1. **Binary decisions:** Each question has only two answers (yes/no)
2. **Interpretable:** You can explain any decision by tracing the path
3. **Simple:** No distributions, no gradients—just comparisons
4. **Flexible:** The shape of the final regions can be arbitrarily complex

3 Decision Tree Terminology

Before diving deeper, let's establish the vocabulary:

Definition:

Tree Components

- **Root Node:** The topmost node where the tree starts (first question)
- **Internal Nodes:** Nodes that ask questions and split into children
- **Leaf Nodes (Terminal Nodes):** Final nodes that make predictions (no children)
- **Split:** The act of dividing a node into child nodes based on a question
- **Depth:** The number of splits from root to a node
- **Branch:** The path from root to any node

Key Information

Visualizing the Tree:

Decision trees are typically drawn "upside down":

- Root at the top (like a family tree)
- Leaves at the bottom
- Data flows downward through questions

4 How Decision Trees Make Predictions

4.1 The Lemon vs Orange Example

Suppose we have a trained decision tree that classifies fruits as “Lemon” or “Orange” based on:

- `height`: Height of the fruit
- `width`: Width of the fruit

The tree structure:

```
                 height > 6.5?
                 /           \
               No             Yes
               /                \
        width > 6.0?      width > 9.5?
        /        \         /        \
      No         Yes      No         Yes
       |           |       |           |
    Orange       Lemon   Orange      Lemon
```

4.2 Prediction: Tree Traversal

Definition:

Tree Traversal **Tree traversal** is the process of starting at the root node and following the appropriate branches based on feature values until reaching a leaf node, which provides the prediction.

Example:

Predicting a New Fruit A new fruit arrives with measurements: `height = 5.9`, `width = 5.8`

Step 1: Root node asks: “Is height > 6.5?”

- 5.9 is NOT greater than 6.5 $\rightarrow$ Go **left** (No branch)

Step 2: Next node asks: “Is width > 6.0?”

- 5.8 is NOT greater than 6.0 $\rightarrow$ Go **left** (No branch)

Step 3: We’ve reached a leaf node labeled “Orange”

Prediction: This fruit is an **Orange**

4.3 Decision Boundaries in Feature Space

Each split in the tree creates a boundary that is **parallel to a feature axis**:

- Split on `height > 6.5` creates a **horizontal line** at height $= 6.5$
- Split on `width > 6.0` creates a **vertical line** at width $= 6.0$

The final decision regions are **axis-aligned rectangles**:

- Region 1: height ≤ 6.5 AND width $\leq 6.0 \rightarrow$ Orange
- Region 2: height ≤ 6.5 AND width $> 6.0 \rightarrow$ Lemon

- Region 3: height > 6.5 AND width $\leq 9.5 \rightarrow$ Orange
- Region 4: height > 6.5 AND width $> 9.5 \rightarrow$ Lemon

Key Information

Why Rectangles?

Decision trees always create **axis-aligned rectangular regions** because:

1. Each split is on a single feature
2. Splits create boundaries perpendicular to that feature's axis
3. Multiple splits partition space into nested rectangles

This is different from logistic regression, which creates tilted linear boundaries.

5 How Decision Trees Learn: Splitting Criteria

5.1 The Learning Problem

We've seen how to *use* a decision tree. But how do we *build* one?

The key questions:

1. Which feature should we split on?
2. What threshold value should we use?
3. When should we stop splitting?

5.2 The Goal: Maximize Purity

Definition:

Purity and Impurity

- A node is **pure** if all its data points belong to the same class
- A node is **impure** if it contains a mixture of classes
- **Impurity** measures how "mixed" a node is

The goal of each split is to create child nodes that are **more pure** than the parent.

Example:

Good vs Bad Splits **Parent node:** 6 blue circles, 8 orange triangles (14 total)

Split A:

- Left child: 6 blue, 0 orange (100% pure!)
- Right child: 0 blue, 8 orange (100% pure!)

→ **Excellent split!** Both children are perfectly pure.

Split B:

- Left child: 3 blue, 4 orange (mixed)
- Right child: 3 blue, 4 orange (mixed)

→ **Terrible split!** Children are just as impure as parent.

5.3 Measuring Impurity: Three Approaches

5.3.1 Method 1: Classification Error

The most intuitive measure: what fraction would we get wrong if we predicted the majority class?

$$\text{Classification Error} = 1 - \max_k \hat{p}_k \tag{1}$$

Where $\hat{p}_k$ is the proportion of class k in the node.

Example:

Classification Error Calculation Node with 5 blue circles and 8 orange triangles (13 total):

- $\hat{p}_{\text{blue}} = 5/13 = 0.385$
- $\hat{p}_{\text{orange}} = 8/13 = 0.615$

If we predict "orange" (majority), we get 5 wrong.
Classification Error $= 1 - \max(0.385, 0.615) = 1 - 0.615 = 0.385$

5.3.2 Method 2: Gini Impurity (Index)

The most commonly used measure in practice:

$$\text{Gini} = 1 - \sum_{k=1}^{K} \hat{p}_k^2 \tag{2}$$

Definition:

Interpreting Gini Gini impurity represents the probability that two randomly chosen samples from the node would have **different** class labels.

- **Gini = 0:** Node is perfectly pure (all same class)
- **Gini = 0.5:** Maximum impurity for 2 classes (50-50 split)
- For K classes: Maximum Gini $= 1 - 1/K$

Example:

Gini Calculation Same node: 5 blue, 8 orange (13 total)
Gini $= 1 - \left[\left(\frac{5}{13}\right)^2 + \left(\frac{8}{13}\right)^2\right]$
$= 1 - \left[\frac{25}{169} + \frac{64}{169}\right] = 1 - \frac{89}{169} = \frac{80}{169} \approx 0.47$
Compare to a pure node (all orange): Gini $= 1 - [0^2 + 1^2] = 0$

5.3.3 Method 3: Entropy

From information theory, entropy measures the "uncertainty" or "disorder":

$$\text{Entropy} = -\sum_{k=1}^{K} \hat{p}_k \log_2(\hat{p}_k) \tag{3}$$

Definition:

Interpreting Entropy Entropy measures the number of bits needed to encode the class of a random sample.

- **Entropy = 0:** Node is perfectly pure (no uncertainty)
- **Entropy = 1:** Maximum impurity for 2 classes (1 bit needed)
- For K classes: Maximum Entropy $= \log_2(K)$

Example:

Entropy Calculation Same node: 5 blue, 8 orange

Entropy $= -\left[\frac{5}{13}\log_2\left(\frac{5}{13}\right) + \frac{8}{13}\log_2\left(\frac{8}{13}\right)\right]$

$= -[0.385 \times (-1.38) + 0.615 \times (-0.70)]$

$= -[-0.53 - 0.43] = 0.96$

Compare to a pure node: Entropy $= -[1 \times \log_2(1)] = -[1 \times 0] = 0$

5.4 Why Squaring in Gini?

The squaring operation in Gini impurity has an important effect:

Key Information

Effect of Squaring:

Squaring the proportions **accentuates the difference** between pure and impure regions:

- Large proportions (majority class) contribute more when squared
- Small proportions (minority classes) are diminished when squared
- This makes Gini more sensitive to "almost pure" vs "quite mixed"

Think of it like **softmax**—it emphasizes the maximum without explicitly computing max!

5.5 Comparing the Three Measures

All three measures:

- Equal 0 when the node is pure
- Reach maximum when classes are evenly split
- Are monotonically related to purity

Key difference: Sensitivity to impurity changes

Table 1: *Impurity Measures Comparison*

Property	Classification Error	Gini Impurity	Entropy
Value at pure node	0	0	0
Max value (2 classes)	0.5	0.5	1.0
Sensitivity to changes	Least sensitive	Moderately sensitive	Most sensitive
Computational cost	Lowest	Low	Slightly higher
Default in sklearn	No	**Yes**	No

Warning

Why Not Use Classification Error?

Classification error is the most intuitive but the **least sensitive** to small improvements in purity.

Consider: A node goes from (50%, 50%) to (60%, 40%)

- Classification error: $0.5 \rightarrow 0.4$ (decrease of 0.1)

- Gini: $0.5 \to 0.48$ (slight decrease)

Gini and Entropy have **curved shapes** that penalize impurity more heavily near 50-50, making them better for finding good splits.

5.6 Evaluating a Split: Weighted Average

When evaluating a split, we must consider the **sizes** of the resulting child nodes.

Definition:

Weighted Impurity For a split creating regions R_1 (with N_1 samples) and R_2 (with N_2 samples):

$$\text{Weighted Impurity} = \frac{N_1}{N} \times \text{Impurity}(R_1) + \frac{N_2}{N} \times \text{Impurity}(R_2) \tag{4}$$

where $N = N_1 + N_2$ is the total samples.

Example:

Why Weighted Average Matters Consider two splits of a node with 18 samples:

Split A:

- R_1: 1 sample, Gini $= 0$ (pure)
- R_2: 17 samples, Gini $= 0.45$ (impure)

Weighted: $\frac{1}{18}(0) + \frac{17}{18}(0.45) = 0.425$

Split B:

- R_1: 9 samples, Gini $= 0.2$
- R_2: 9 samples, Gini $= 0.2$

Weighted: $\frac{9}{18}(0.2) + \frac{9}{18}(0.2) = 0.2$

Split B is better! Even though Split A has a "pure" child, it only contains 1 sample.

6 The Learning Algorithm: Greedy Approach

6.1 Why Can't We Find the "Best" Tree?

Finding the globally optimal decision tree is **NP-complete**—computationally infeasible for any reasonable dataset.

Why? Consider:

- p features, each with many possible split points
- At each node, we choose one split
- The number of possible trees grows **exponentially**

6.2 The Greedy Solution

Definition:

Greedy Algorithm A **greedy algorithm** makes the locally optimal choice at each step, without considering future consequences. It doesn't guarantee a globally optimal solution but is computationally tractable.

Decision Tree Learning Algorithm:

1. **Start:** All training data in root node
2. **For each node:**
   - For each feature p
   - For each possible threshold t
   - Calculate weighted impurity of split (p, t)
3. **Choose:** The (p^*, t^*) that **minimizes weighted impurity**
4. **Split:** Create two child nodes based on (p^*, t^*)
5. **Recurse:** Repeat steps 2-4 for each child
6. **Stop:** When stopping criterion is met

Key Information

Key Insight:
Unlike linear/logistic regression where we minimize a single loss function over all data, decision trees:

- Don't have a global loss function
- Make locally optimal decisions at each split
- This greedy approach is fast but may miss globally better trees

7 Preventing Overfitting: Stopping Conditions

7.1 The Overfitting Problem

If we let the tree grow without limits, it will keep splitting until:

- Every leaf node contains exactly **one** training sample
- Training accuracy is 100%

This is **severe overfitting**:

- The tree memorizes every training point, including noise
- It creates tiny rectangular regions around individual points
- Test accuracy will be poor

Warning

Visualizing Overfitting:
Imagine the true pattern is a circle of green points surrounded by white points.
Shallow tree (max_depth=4):

- Creates a rough square approximating the circle
- High bias, low variance
- Underfitting

Deep tree (max_depth=100):

- Creates tiny squares around every green point
- Even captures isolated green points in "white" regions (noise!)
- Low bias, high variance
- Overfitting

7.2 Stopping Conditions (Hyperparameters)

To prevent overfitting, we **intentionally limit** tree growth:

Definition:

Common Stopping Conditions

1. `max_depth`: Maximum number of splits from root to any leaf
   - `max_depth=3` means at most 3 questions to reach a prediction
   - Most commonly used regularization
2. `min_samples_leaf`: Minimum samples required in a leaf node
   - `min_samples_leaf=5` means stop if split would create a leaf with < 5 samples
   - Prevents very specific rules based on few points
3. `max_leaf_nodes`: Maximum total number of leaf nodes
   - `max_leaf_nodes=8` limits the tree to 8 final regions
   - Controls overall model complexity

4. **min_impurity_decrease**: Minimum impurity improvement required
   - Only split if it reduces impurity by at least this amount
   - Prevents splits that barely improve purity
5. **Pure nodes**: Automatically stop if a node is already 100% pure

7.3 Tree Growth Strategies

7.3.1 Level-Order Growth (Default)

The standard approach:

- Split all nodes at depth 1
- Then split all nodes at depth 2
- Continue level by level

Used with `max_depth` and most stopping conditions.

7.3.2 Best-First Growth

Alternative approach (used when `max_leaf_nodes` is set):

- At each step, consider ALL current leaf nodes
- Split the one that gives the **greatest impurity reduction**
- Regardless of its depth

Warning

Computational Cost of Best-First:
Best-first growth is more computationally expensive because:
- Must evaluate all possible splits at all leaf nodes
- Compare across different depths
- Number of comparisons grows with tree size

Use level-order when possible for efficiency.

7.4 Bias-Variance Tradeoff

Tree depth directly affects the bias-variance tradeoff:

7.5 Finding Optimal Hyperparameters

How do we choose the right `max_depth` or `min_samples_leaf`?

Answer: Cross-Validation!

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score

```

Table 2: *Tree Depth and Bias-Variance*

	Shallow Tree	**Deep Tree**
Complexity	Low (simple model)	High (complex model)
Bias	High (underfits)	Low (fits training data well)
Variance	Low (stable across datasets)	High (changes with different data)
Training Error	Higher	Lower (approaching 0)
Test Error	May be high (underfitting)	May be high (overfitting)
Interpretability	High (few rules)	Low (many rules)

```python
# Try different max_depth values
for depth in [2, 3, 4, 5, 6, 8, 10, None]:
    tree = DecisionTreeClassifier(max_depth=depth, random_state=42)
    scores = cross_val_score(tree, X, y, cv=5)
    print(f"max_depth={depth}: CV accuracy = {scores.mean():.3f}")

# Choose the depth with highest CV accuracy
```

8 Decision Trees in Python

8.1 Basic Usage

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Create and train model
tree = DecisionTreeClassifier(
    criterion='gini',       # 'gini' or 'entropy'
    max_depth=5,            # Maximum depth
    min_samples_leaf=5,     # Minimum samples in leaf
    random_state=42
)
tree.fit(X_train, y_train)

# Predict
y_pred = tree.predict(X_test)
y_prob = tree.predict_proba(X_test)  # Probability estimates

# Evaluate
from sklearn.metrics import accuracy_score
print(f"Accuracy: {accuracy_score(y_test, y_pred):.3f}")
```

8.2 Visualizing the Tree

```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Plot the tree
plt.figure(figsize=(20, 10))
plot_tree(tree,
          feature_names=X.columns,
          class_names=['Orange', 'Lemon'],
          filled=True,
          rounded=True)
plt.title("Decision Tree Visualization")
plt.show()
```

8.3 Feature Importance

```python
# Get feature importances
importances = tree.feature_importances_
```

```python

# Display
for name, importance in zip(X.columns, importances):
    print(f"{name}: {importance:.3f}")

# Most important feature = used in highest splits or most impurity reduction
```

9 Key Takeaways

Key Summary

Decision Trees Summary

What they are:

- Flowchart-like models that make binary decisions at each node
- Predictions by traversing from root to leaf
- Decision boundaries are axis-aligned rectangles

How they learn:

- Greedy algorithm: find best split at each step
- Goal: maximize purity (minimize impurity)
- Criteria: Gini impurity (default), Entropy, or Classification Error
- Evaluate splits using weighted average impurity

How to prevent overfitting:

- Stopping conditions: `max_depth`, `min_samples_leaf`, etc.
- Find optimal values via cross-validation

Advantages:

- Highly interpretable ("white box" model)
- No feature scaling needed
- Handle nonlinear boundaries
- Fast training and prediction

Limitations:

- Can only create axis-aligned boundaries
- Prone to overfitting without proper constraints
- Greedy algorithm may miss global optimum
- High variance (sensitive to training data)

10 Learning Checklist

☐ Can you explain why logistic regression fails for certain decision boundaries?

☐ Can you define: root node, internal node, leaf node, split, depth?

☐ Can you walk through tree traversal to make a prediction?

☐ Do you understand why decision boundaries are rectangular?

☐ Can you calculate Gini impurity for a given node?

☐ Can you calculate Entropy for a given node?

☐ Do you know why we use weighted average when evaluating splits?

☐ Can you explain why Gini/Entropy are preferred over classification error?

☐ Do you understand what "greedy" means in this context?

☐ Can you list 3+ stopping conditions and explain their purpose?

☐ Do you understand the bias-variance tradeoff as tree depth changes?

☐ Do you know how to find optimal hyperparameters using cross-validation?

11 Looking Ahead

In the next lectures, we'll extend decision trees to:

1. **Regression Trees**: Predicting continuous values instead of classes
2. **Random Forests**: Combining many trees to reduce variance
3. **Bagging and Boosting**: Ensemble methods for better performance
4. **Gradient Boosting**: XGBoost, LightGBM—state-of-the-art for tabular data