Complete Decision Tree Lab Guide: Mushroom Classification

Table of Contents

1. Packages
2. Problem Statement
3. Dataset
4. Decision Tree Theory
5. Implementation
6. Building the Tree
7. Program Flow & Outputs

---

1. Packages

Theory: Essential Libraries for Machine Learning

The lab begins by importing fundamental Python libraries:

```python
import numpy as np
import matplotlib.pyplot as plt
from public_tests import *
from utils import *
```

Library Functions:

- **NumPy**: Provides efficient numerical computing with arrays and matrices
- **Matplotlib**: Creates visualizations and plots for data analysis
- **Public Tests**: Contains unit tests to validate implementations
- **Utils**: Helper functions for visualization and data processing

---

2. Problem Statement

Business Context: Mushroom Classification

Scenario: You're starting a wild mushroom company and need to classify mushrooms as edible or poisonous based on physical attributes.

Objective: Build a decision tree classifier to determine mushroom edibility using features like cap color, stalk shape, and solitary growth.

Real-world Application: This represents a binary classification problem common in:

- Medical diagnosis
- Quality control
- Risk assessment
- Product categorization

---

3. Dataset

3.1 Original Dataset Structure

The dataset contains 10 mushroom samples with the following features:

Cap Color	Stalk Shape	Solitary	Edible
Brown	Tapering	Yes	1
Brown	Enlarging	Yes	1
Brown	Enlarging	No	0
...	...	...	...

3.2 One-Hot Encoded Dataset

For computational efficiency, categorical features are converted to binary (0/1) values:

Brown Cap	Tapering Stalk	Solitary	Edible
1	1	1	1
1	0	1	1
1	0	0	0
...	...	...	...

Dataset Dimensions:

- X_train shape: (10, 3) - 10 samples, 3 features
- y_train shape: (10,) - 10 target labels
- Features: [Brown Cap, Tapering Stalk, Solitary]
- Labels: 1 (edible), 0 (poisonous)

---

4. Decision Tree Theory

4.1 What is a Decision Tree?

A **Decision Tree** is a supervised learning algorithm that makes predictions by learning simple decision rules inferred from data features. It creates a model that predicts target values by learning simple decision rules.

4.2 Key Concepts

Entropy

Definition: Entropy measures the impurity or randomness in a dataset.

Formula: $H(p_1) = -p_1 \log_2(p_1) - (1-p_1) \log_2(1-p_1)$

Where p_1 is the proportion of positive examples.

Interpretation:

- Entropy = 0: Pure node (all samples belong to one class)
- Entropy = 1: Maximum impurity (equal distribution of classes)

Information Gain

Definition: Information Gain measures how much information a feature gives us about the class.

Formula: IG = H(parent) - [w_left × H(left) + w_right × H(right)]

Where:

- H(parent): Entropy of parent node
- w_left, w_right: Proportion of samples in left/right child
- H(left), H(right): Entropy of child nodes

4.3 Decision Tree Algorithm

1. **Start** with all examples at the root node
2. **Calculate** information gain for all possible features
3. **Select** feature with highest information gain
4. **Split** dataset based on selected feature
5. **Repeat** process for child nodes until stopping criteria met

4.4 Stopping Criteria

Common stopping criteria include:

- Maximum depth reached
- Minimum samples per node
- Minimum information gain threshold
- Pure nodes (entropy = 0)

---

5. Implementation

5.1 Exercise 1: Calculate Entropy

Function: `compute_entropy(y)`

Theory: Entropy quantifies uncertainty in a dataset.

```python
def compute_entropy(y):
    if len(y) == 0:
        return 0.
    p1 = np.mean(y)  # Proportion of positive examples
    if p1 == 0 or p1 == 1:
        return 0.  # Pure node
    entropy = -p1 * np.log2(p1) - (1 - p1) * np.log2(1 - p1)
    return entropy
```

Expected Output:

- Root node entropy = 1.0 (maximum impurity with 5 edible, 5 poisonous)

5.2 Exercise 2: Split Dataset

Function: `split_dataset(X, node_indices, feature)`

Theory: Splits data based on binary feature values (0 or 1).

python

```python
def split_dataset(X, node_indices, feature):
    left_indices = []   # Feature value = 1
    right_indices = []  # Feature value = 0

    for i in node_indices:
        if X[i, feature] == 1:
            left_indices.append(i)
        else:
            right_indices.append(i)

    return left_indices, right_indices
```

Example Output (splitting on Brown Cap feature):

- Left indices: [0, 1, 2, 3, 4, 7, 9] (brown cap mushrooms)
- Right indices: [5, 6, 8] (red cap mushrooms)

5.3 Exercise 3: Calculate Information Gain

Function: `compute_information_gain(X, y, node_indices, feature)`

Theory: Measures improvement in purity after splitting.

python

```python
def compute_information_gain(X, y, node_indices, feature):
    left_indices, right_indices = split_dataset(X, node_indices, feature)

    # Calculate entropies
    H_node = compute_entropy(y[node_indices])
    H_left = compute_entropy(y[left_indices])
    H_right = compute_entropy(y[right_indices])

    # Calculate weights
    w_left = len(left_indices) / len(node_indices)
    w_right = len(right_indices) / len(node_indices)

    # Information gain
    information_gain = H_node - (w_left * H_left + w_right * H_right)
    return information_gain
```

Expected Output:

- Brown cap: 0.0349
- Tapering stalk: 0.1245
- Solitary: 0.2781 (highest → best feature)

5.4 Exercise 4: Get Best Split

Function: `get_best_split(X, y, node_indices)`

Theory: Selects feature with maximum information gain.

```python
def get_best_split(X, y, node_indices):
    num_features = X.shape[1]
    best_feature = -1
    max_info_gain = -1

    for feature in range(num_features):
        info_gain = compute_information_gain(X, y, node_indices, feature)
        if info_gain > max_info_gain:
            max_info_gain = info_gain
            best_feature = feature

    return best_feature
```

Expected Output: Best feature = 2 (Solitary)

---

6. Building the Tree

6.1 Recursive Tree Construction

Function: `build_tree_recursive()`

Theory: Recursively builds tree by splitting nodes until maximum depth reached.

Algorithm Flow:

1. Check stopping criteria (max depth)
2. Find best feature to split on
3. Split dataset
4. Recursively build left and right subtrees

6.2 Tree Structure Visualization

```
Root: Split on feature 2 (Solitary)
├── Left (Solitary=1): Split on feature 0 (Brown Cap)
│   ├── Left (Brown Cap=1): [0,1,4,7] → Mostly Edible
│   └── Right (Brown Cap=0): [5] → Poisonous
└── Right (Solitary=0): Split on feature 1 (Tapering Stalk)
    ├── Left (Tapering=1): [8] → Edible
    └── Right (Tapering=0): [2,3,6,9] → Mostly Poisonous
```

---

7. Program Flow & Outputs

7.1 Complete Execution Flow

1. **Data Loading**

   ```
   X_train shape: (10, 3)
   y_train shape: (10,)
   ```

2. **Entropy Calculation**

   ```
   Root entropy: 1.0
   ```

3. **Feature Splitting Analysis**

   ```
   Brown Cap IG: 0.0349
   Tapering Stalk IG: 0.1245
   Solitary IG: 0.2781 (Winner!)
   ```

4. **Tree Construction**

   ```
   Depth 0, Root: Split on feature: 2
   - Depth 1, Left: Split on feature: 0
     -- Left leaf: [0,1,4,7]
     -- Right leaf: [5]
   - Depth 1, Right: Split on feature: 1
     -- Left leaf: [8]
     -- Right leaf: [2,3,6,9]
   ```

7.2 Decision Tree Interpretation

Path Analysis:

- **Solitary mushrooms with brown caps**: Usually edible

- **Solitary mushrooms with red caps**: Poisonous
- **Non-solitary mushrooms with tapering stalks**: Edible
- **Non-solitary mushrooms with enlarging stalks**: Usually poisonous

7.3 Key Insights

1. **Solitary** is the most informative feature (highest information gain)
2. **Brown cap** helps distinguish among solitary mushrooms
3. **Stalk shape** is useful for non-solitary mushrooms
4. Tree achieves good separation with only 2 levels of depth

Summary

This lab demonstrates the complete implementation of a decision tree classifier from scratch, covering:

- **Theoretical foundations**: Entropy, information gain, tree construction
- **Practical implementation**: Four core functions for tree building
- **Real-world application**: Mushroom classification problem
- **Algorithm visualization**: Tree structure and decision paths

The decision tree successfully learns patterns in the mushroom data, providing an interpretable model for classification decisions. The recursive nature of the algorithm and the greedy feature selection based on information gain are key characteristics of decision tree learning.

Final Tree Performance: The constructed tree provides clear decision rules that can be easily interpreted by domain experts, making it valuable for applications requiring explainable AI.