ML LAB-1

(ID3 Algorithm)

Comparative Analysis Report

This report analyzes the performance of the ID3 decision tree algorithm across three different datasets: Mushroom, Nursery, and Tic-Tac-Toe. The analysis is based on the provided accuracy metrics, tree complexity data, and general characteristics of each dataset.

a) Algorithm Performance

1. **Which dataset achieved the highest accuracy and why?**

The **Mushroom** dataset achieved a perfect accuracy of **100.00%**, which is the highest among the three. This is likely due to the dataset's high level of predictability and the strong discriminatory power of its features. The attributes in the Mushroom dataset have very clear relationships with the target variable (edible or poisonous), with a few key features, such as "odor," being almost entirely sufficient for perfect classification. This allows the ID3 algorithm, which greedily selects the best attribute at each step, to quickly find a path to a pure leaf node.

2. **How does dataset size affect performance?**

In this specific comparison, dataset size seems to have a mixed impact.

- The **Mushroom** dataset (8,124 samples) achieved perfect accuracy, suggesting that a sufficient number of relevant samples is available for the tree to learn the classification rules.
- The **Nursery** dataset (12,960 samples) is larger than Mushroom but had a slightly lower accuracy of 98.67%.
- The **Tic-Tac-Toe** dataset, which is the smallest with 958 samples, had the lowest accuracy at 89.36%.

A larger number of samples generally provides more data for the algorithm to learn from, potentially leading to a more robust model. However, the sheer size of a dataset is less important than the quality and predictability of its features. The Nursery dataset's lower score compared to Mushroom, despite being larger, highlights that a simple, clear relationship between features and classes (like in the Mushroom dataset) is more crucial for the ID3 algorithm's performance than just a large volume of data.

3. **What role does the number of features play?**

The number of features directly influences the complexity of the decision space and the potential for overfitting.

- **Mushroom** has 22 features and achieves perfect accuracy, demonstrating that when features are highly predictive, the algorithm can build an efficient tree with a shallow depth (4 in this case) and a high number of leaves, indicating many simple, direct rules.

- **Nursery** has 8 features and a higher tree depth (7) and total nodes (952) for a very large dataset, suggesting that the features are less individually powerful, requiring more splits and a deeper tree to achieve high accuracy.
- **Tic-Tac-Toe** has 9 features, similar in number to Nursery. However, the game's complexity and the lower number of samples lead to a larger tree relative to the data size, a lower accuracy, and potentially more nuanced decision patterns.

A higher number of features doesn't automatically mean better performance; rather, the information gain and discriminatory power of those features are what matter most.

b) Data Characteristics Impact

1. **How does class imbalance affect tree construction?**

The provided metrics don't give a direct view of class imbalance, but we can infer its impact. The ID3 algorithm, by its nature, can be sensitive to class imbalance. If one class is much more common than others, the algorithm might favor splitting on attributes that lead to this majority class, as this can still provide a good information gain, even if it doesn't build a well-generalized model for the minority classes. This is reflected in the difference between weighted and macro metrics. For a balanced dataset like Mushroom (51.8% poisonous vs. 48.2% edible), weighted and macro scores are very close. For a dataset with more imbalance, like Nursery, the macro F1-score (0.7628) and macro recall (0.7654) are noticeably lower than their weighted counterparts (0.9872 and 0.9867, respectively), which suggests the model struggles to perform as well on the less-frequent classes.

2. **Which types of features (binary vs. multi-valued) work better?**

ID3 and Information Gain work optimally with categorical, multi-valued features. The datasets here are all categorical.

- **Mushroom** has multi-valued features.
- **Tic-Tac-Toe** features are multi-valued (x, o, b).
- **Nursery** features are highly multi-valued.

The ID3 algorithm's core strength is calculating information gain for each possible split on a categorical feature, which works well for these types of features. In this case, features with a high number of unique values that correlate strongly with the target variable will be selected early in the tree-building process.

c) Practical Applications

1. **For which real-world scenarios is each dataset type most relevant?**
   - **Mushroom Dataset**: This is highly relevant for scenarios where classification is based on clear, well-defined characteristics that lead to a small, understandable set of rules. For example, medical diagnosis (diagnosing a disease based on a set of symptoms) or quality control in manufacturing (identifying a defective product based on specific, visible faults). The output of the model would be highly trusted because the decision-making process is transparent.
   - **Tic-Tac-Toe Dataset**: This dataset represents problems with a finite state space and a clear set of rules, such as game AI or simple expert systems. The model learns to classify based on board state configurations, which is applicable to other turn-based games or rule-based systems.
   - **Nursery Dataset**: This is a great example for a decision-making process involving a large number of subjective or complex categorical

factors. Relevant applications would be customer segmentation, loan application approval, or university admissions, where a decision is made based on many contributing factors. The model's interpretability can provide valuable insights into which factors are most important in the decision.

2. **What are the interpretability advantages for each domain?**

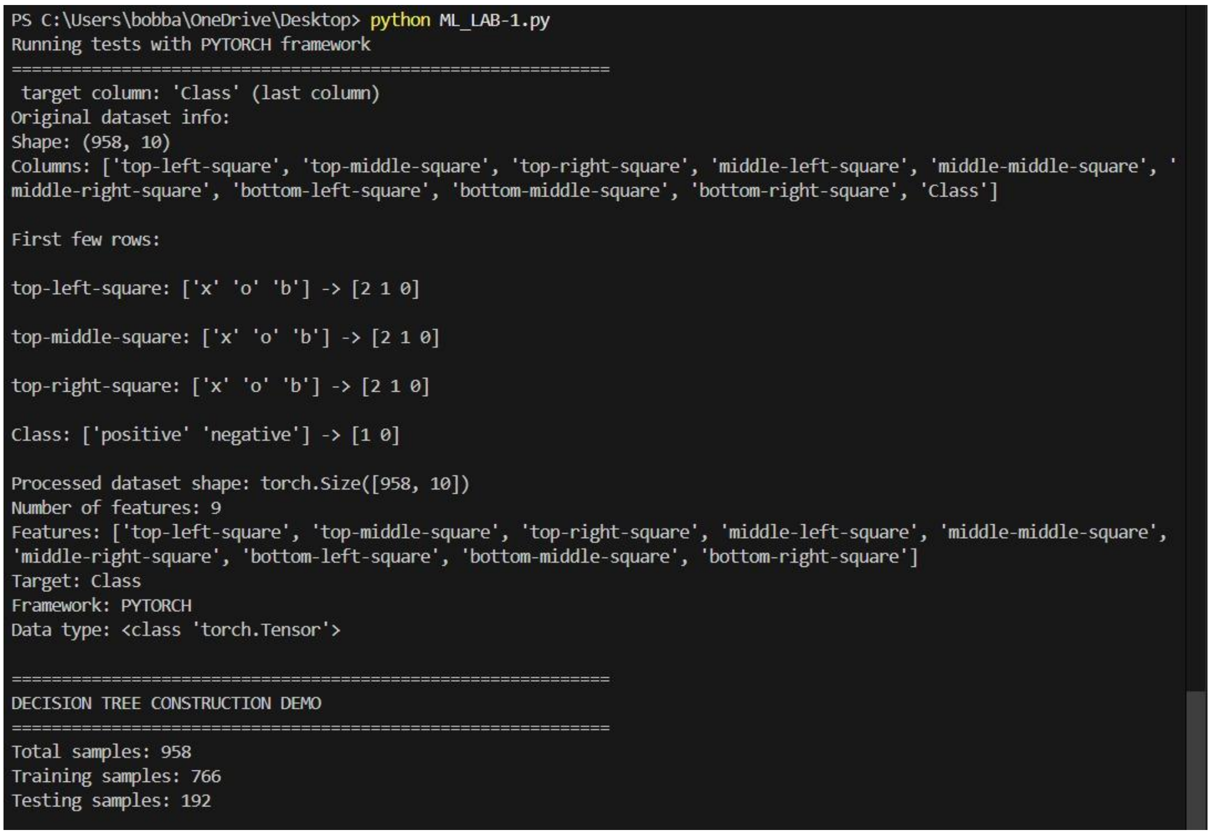
The primary advantage of a decision tree model for all three domains is its **interpretability**. Unlike a "black box" model like a neural network, a decision tree's logic can be easily followed and understood.

- **Mushroom**: The model's extreme simplicity and high accuracy make it a perfect candidate for applications where transparency and trust are paramount. The final tree can be a set of clear, human-readable rules (e.g., "If odor is almond and gill color is pink, classify as edible").
- **Tic-Tac-Toe**: The tree structure can be used to visualize and understand winning strategies for the game. Each split represents a key decision point, and the branches show the consequences of each move.

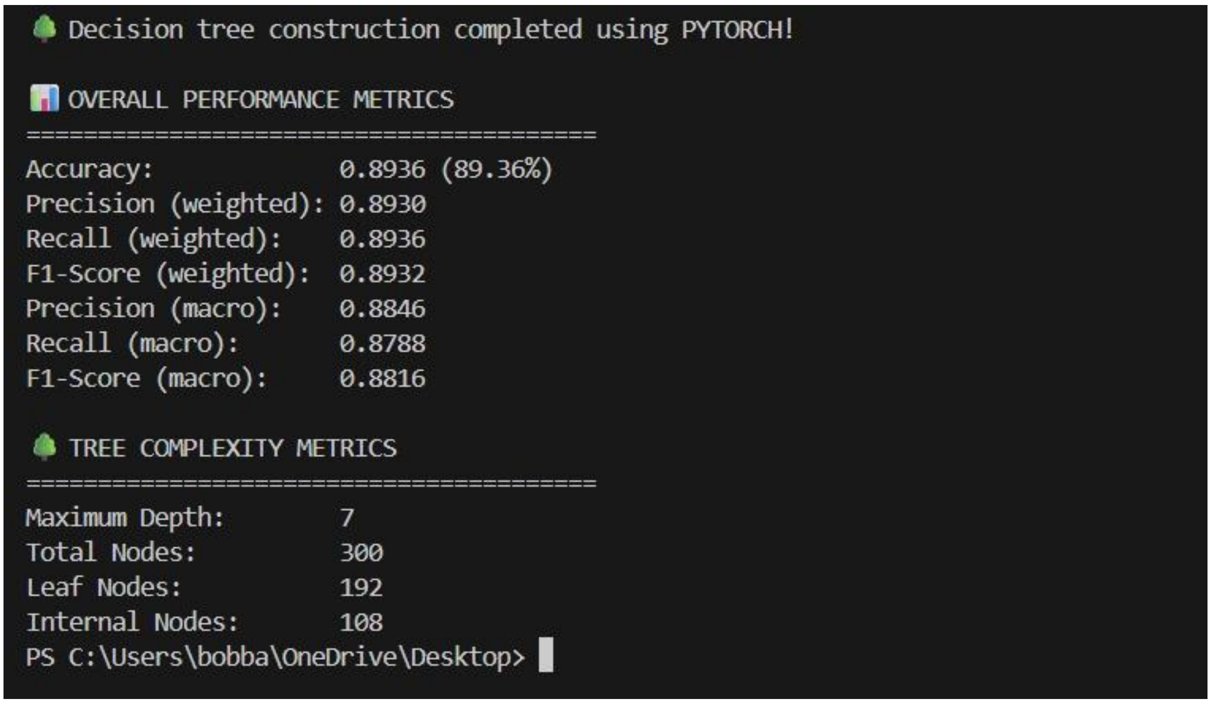
- **Nursery**: For a complex decision like nursery admission, the tree can highlight the most influential factors. For example, it might reveal that "social" and "finance" are the most critical attributes for determining a recommendation, providing clear and actionable insights for decision-makers. The tree helps to explain *why* a certain recommendation was made, which is crucial in sensitive domains.

Output images for accuracy:

Tic-Tac-Toe:

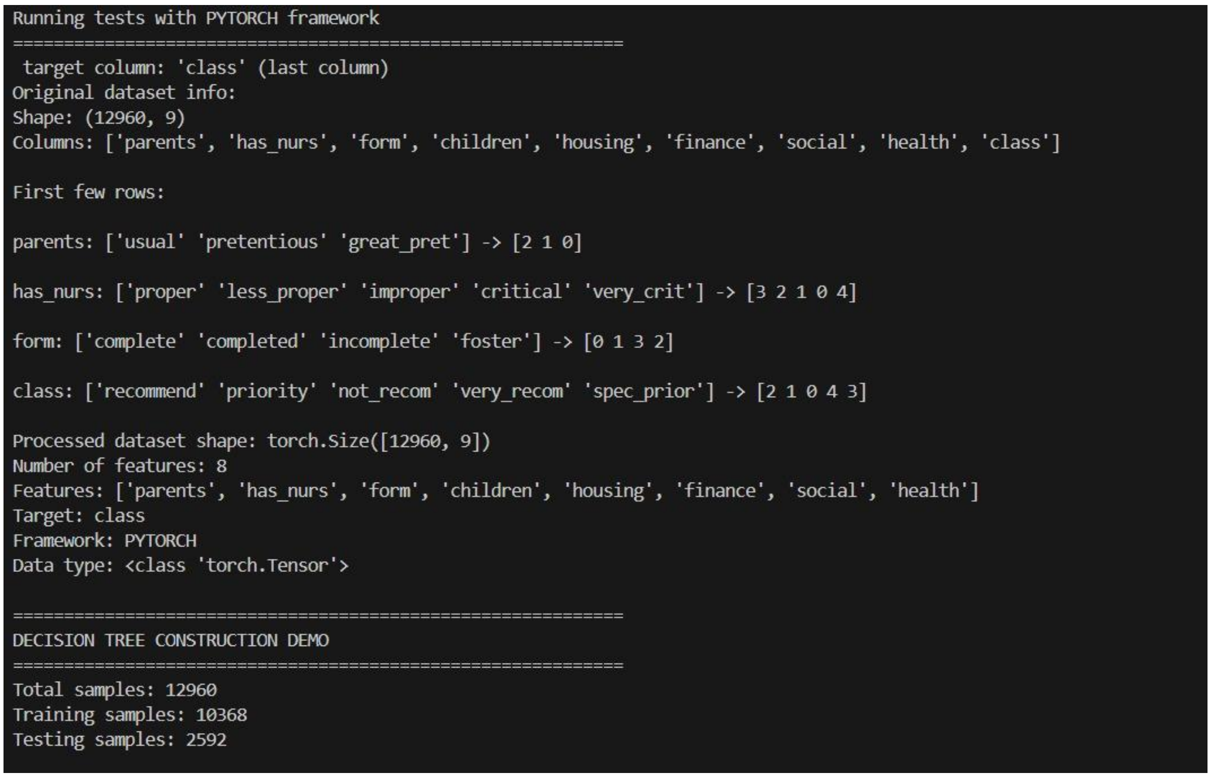


```
PS C:\Users\bobba\OneDrive\Desktop> python ML_LAB-1.py
Running tests with PYTORCH framework
============================================================
 target column: 'Class' (last column)
Original dataset info:
Shape: (958, 10)
Columns: ['top-left-square', 'top-middle-square', 'top-right-square', 'middle-left-square', 'middle-middle-square', '
middle-right-square', 'bottom-left-square', 'bottom-middle-square', 'bottom-right-square', 'Class']

First few rows:

top-left-square: ['x' 'o' 'b'] -> [2 1 0]

top-middle-square: ['x' 'o' 'b'] -> [2 1 0]

top-right-square: ['x' 'o' 'b'] -> [2 1 0]

Class: ['positive' 'negative'] -> [1 0]

Processed dataset shape: torch.Size([958, 10])
Number of features: 9
Features: ['top-left-square', 'top-middle-square', 'top-right-square', 'middle-left-square', 'middle-middle-square',
'middle-right-square', 'bottom-left-square', 'bottom-middle-square', 'bottom-right-square']
Target: Class
Framework: PYTORCH
Data type: <class 'torch.Tensor'>

============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 958
Training samples: 766
Testing samples: 192
```

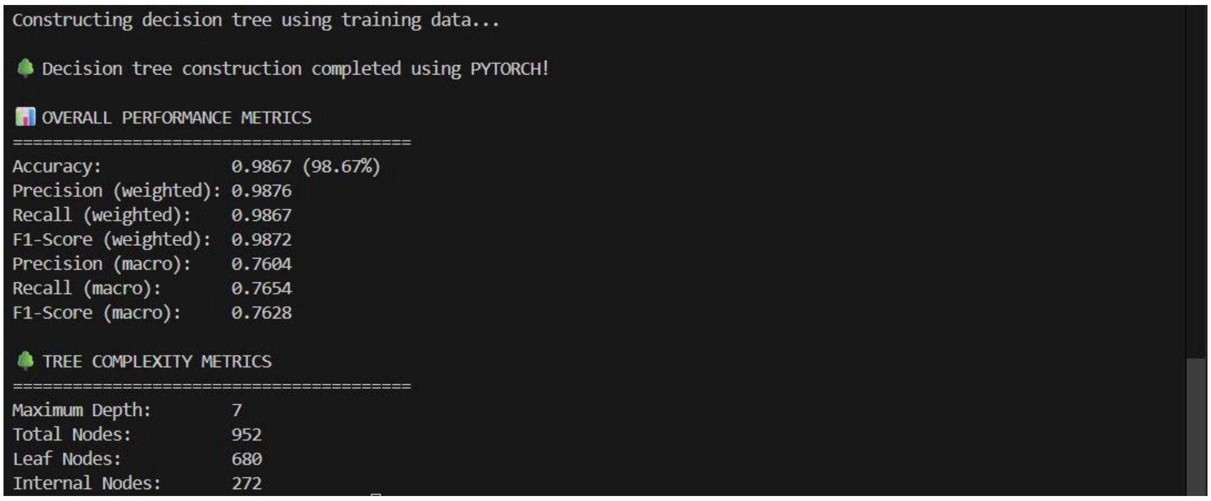


```
🌳 Decision tree construction completed using PYTORCH!

📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:              0.8936 (89.36%)
Precision (weighted): 0.8930
Recall (weighted):    0.8936
F1-Score (weighted):  0.8932
Precision (macro):    0.8846
Recall (macro):       0.8788
F1-Score (macro):     0.8816

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        7
Total Nodes:          300
Leaf Nodes:           192
Internal Nodes:       108
PS C:\Users\bobba\OneDrive\Desktop>
```

Nursery:

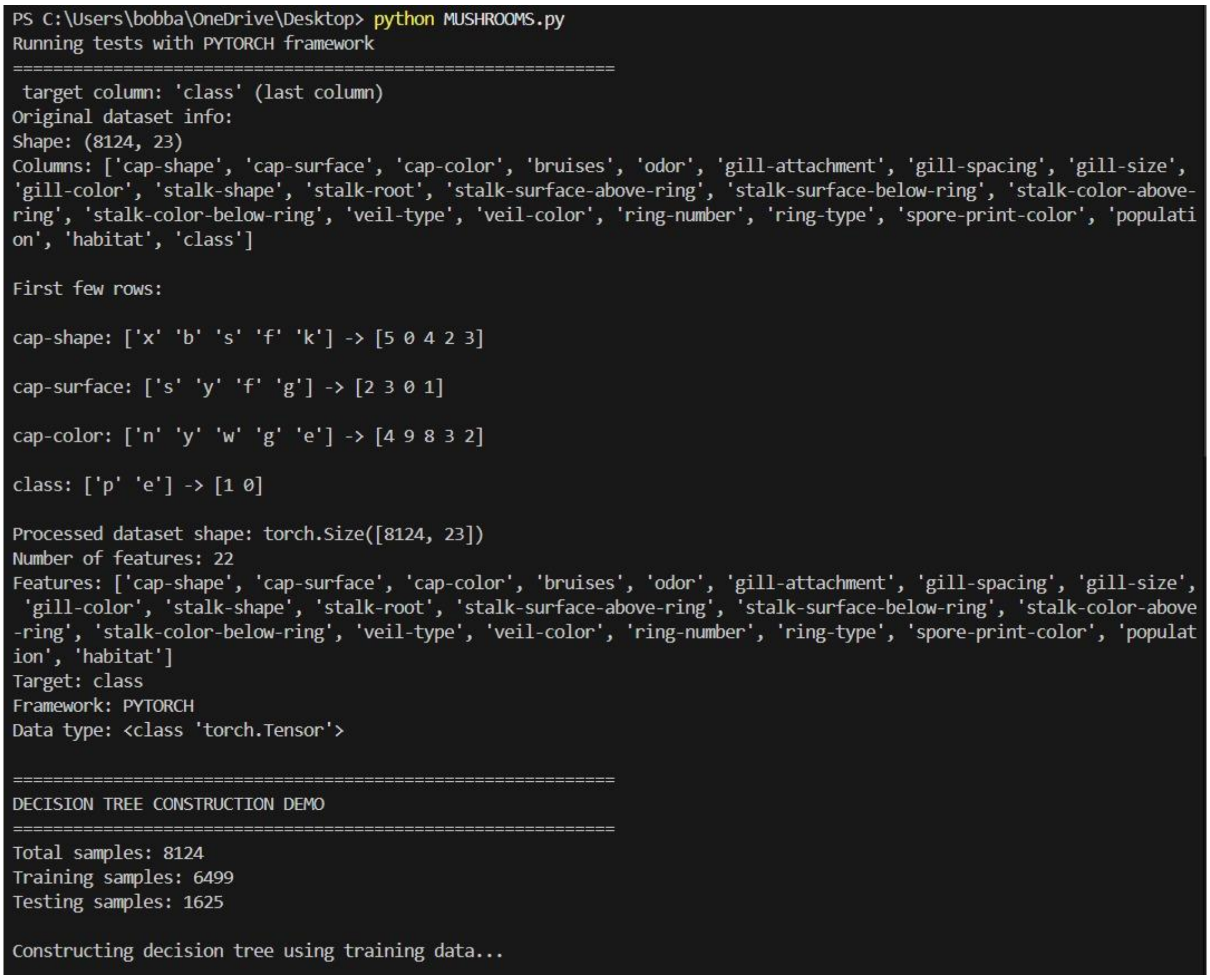


```
Running tests with PYTORCH framework
============================================================
 target column: 'class' (last column)
Original dataset info:
Shape: (12960, 9)
Columns: ['parents', 'has_nurs', 'form', 'children', 'housing', 'finance', 'social', 'health', 'class']

First few rows:

parents: ['usual' 'pretentious' 'great_pret'] -> [2 1 0]

has_nurs: ['proper' 'less_proper' 'improper' 'critical' 'very_crit'] -> [3 2 1 0 4]

form: ['complete' 'completed' 'incomplete' 'foster'] -> [0 1 3 2]

class: ['recommend' 'priority' 'not_recom' 'very_recom' 'spec_prior'] -> [2 1 0 4 3]

Processed dataset shape: torch.Size([12960, 9])
Number of features: 8
Features: ['parents', 'has_nurs', 'form', 'children', 'housing', 'finance', 'social', 'health']
Target: class
Framework: PYTORCH
Data type: <class 'torch.Tensor'>

============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 12960
Training samples: 10368
Testing samples: 2592
```

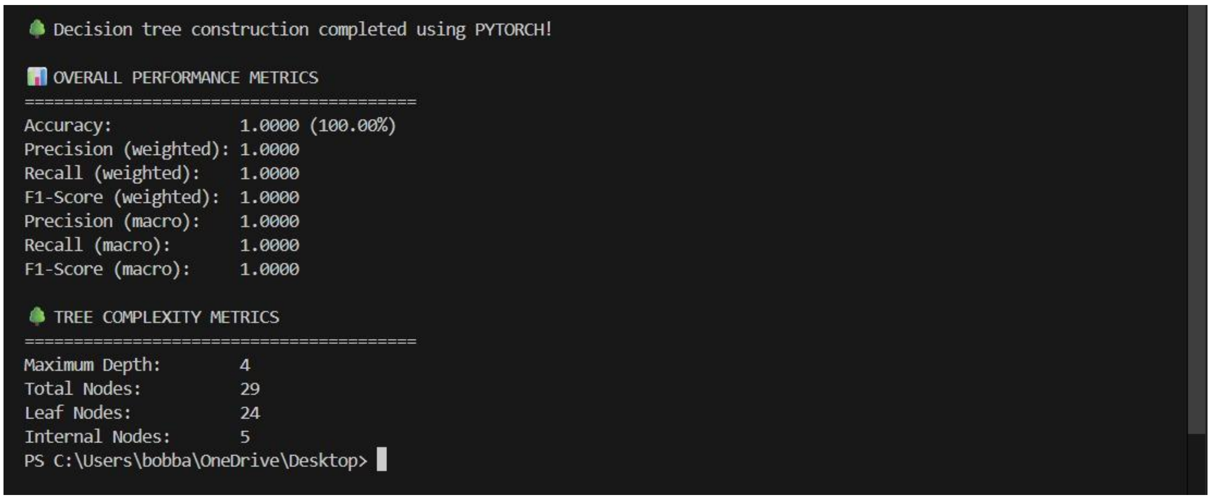


```
Constructing decision tree using training data...

🌳 Decision tree construction completed using PYTORCH!

📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:             0.9867 (98.67%)
Precision (weighted): 0.9876
Recall (weighted):    0.9867
F1-Score (weighted):  0.9872
Precision (macro):    0.7604
Recall (macro):       0.7654
F1-Score (macro):     0.7628

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        7
Total Nodes:          952
Leaf Nodes:           680
Internal Nodes:       272
```

Mushrooms:



```
PS C:\Users\bobba\OneDrive\Desktop> python MUSHROOMS.py
Running tests with PYTORCH framework
============================================================
 target column: 'class' (last column)
Original dataset info:
Shape: (8124, 23)
Columns: ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor', 'gill-attachment', 'gill-spacing', 'gill-size',
'gill-color', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-
ring', 'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-type', 'spore-print-color', 'populati
on', 'habitat', 'class']

First few rows:

cap-shape: ['x' 'b' 's' 'f' 'k'] -> [5 0 4 2 3]

cap-surface: ['s' 'y' 'f' 'g'] -> [2 3 0 1]

cap-color: ['n' 'y' 'w' 'g' 'e'] -> [4 9 8 3 2]

class: ['p' 'e'] -> [1 0]

Processed dataset shape: torch.Size([8124, 23])
Number of features: 22
Features: ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor', 'gill-attachment', 'gill-spacing', 'gill-size',
 'gill-color', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above
-ring', 'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-type', 'spore-print-color', 'populat
ion', 'habitat']
Target: class
Framework: PYTORCH
Data type: <class 'torch.Tensor'>

============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 8124
Training samples: 6499
Testing samples: 1625

Constructing decision tree using training data...
```



```
🌳 Decision tree construction completed using PYTORCH!

📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:             1.0000 (100.00%)
Precision (weighted): 1.0000
Recall (weighted):    1.0000
F1-Score (weighted):  1.0000
Precision (macro):    1.0000
Recall (macro):       1.0000
F1-Score (macro):     1.0000

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        4
Total Nodes:          29
Leaf Nodes:           24
Internal Nodes:       5
PS C:\Users\bobba\OneDrive\Desktop>
```

Name: Bobba Koushik

SRN: PES2UG23CS133

Section: 5C