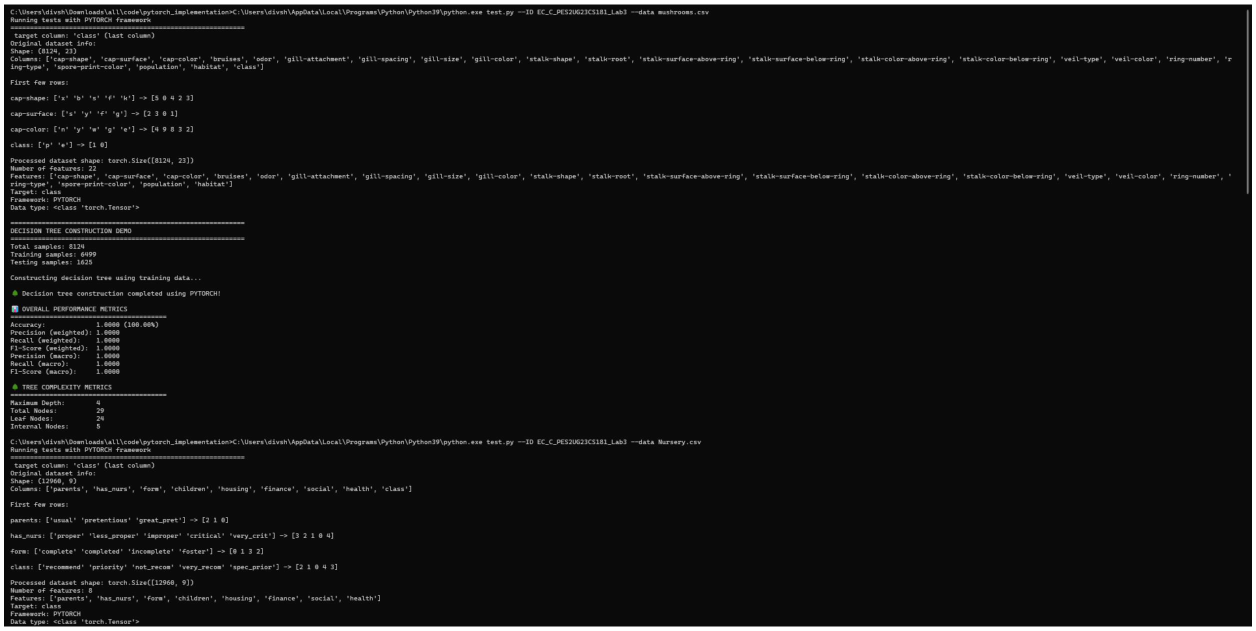
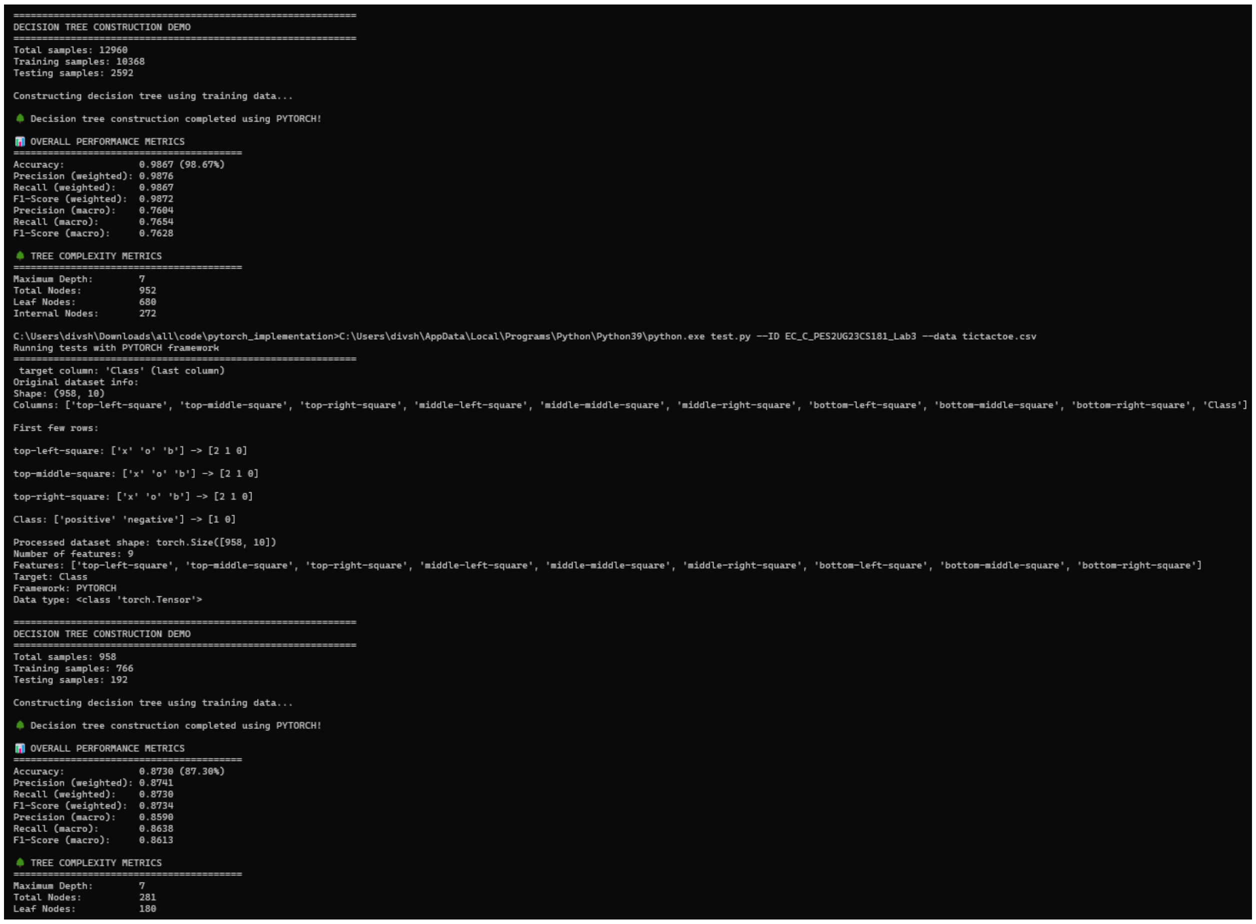
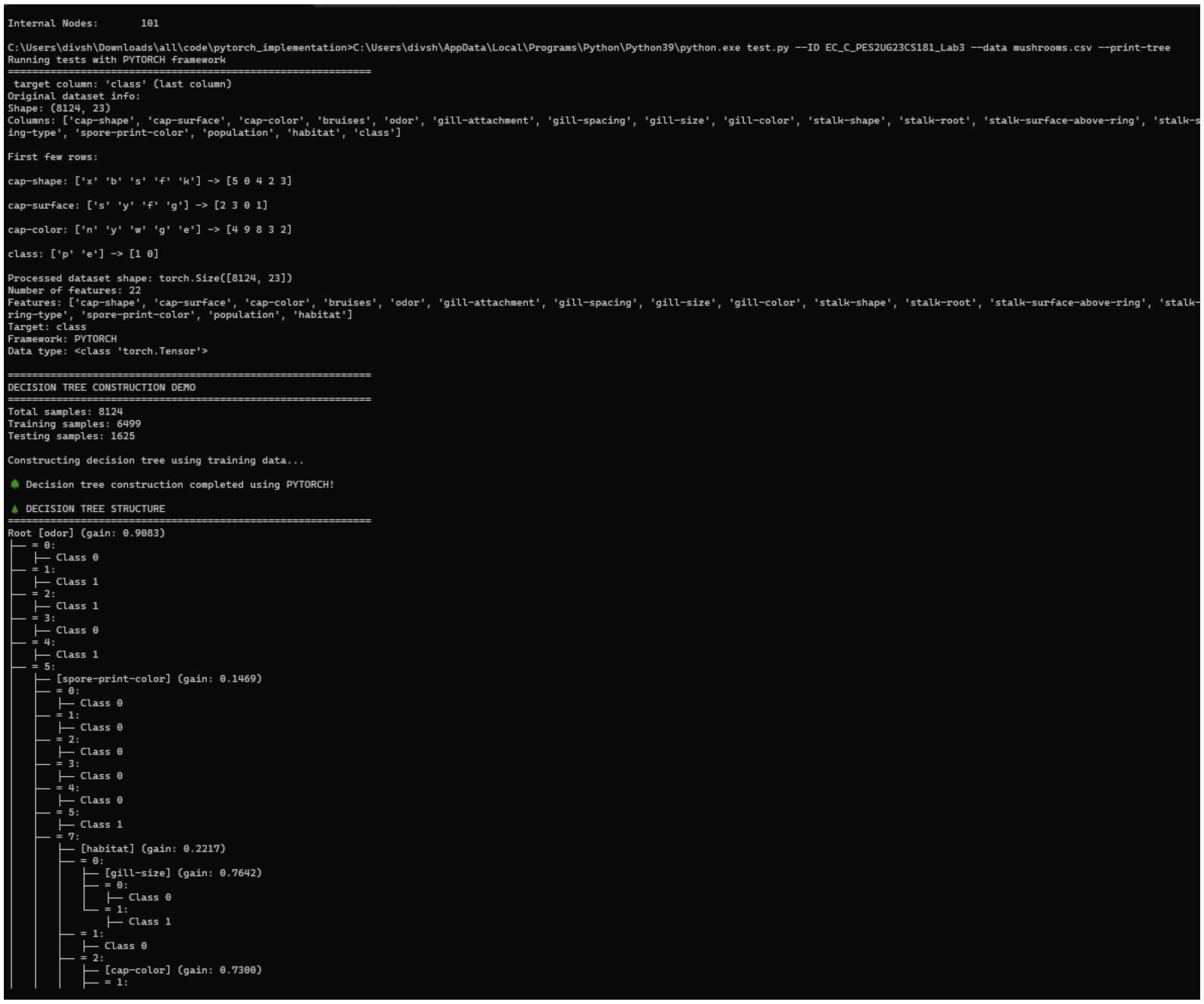


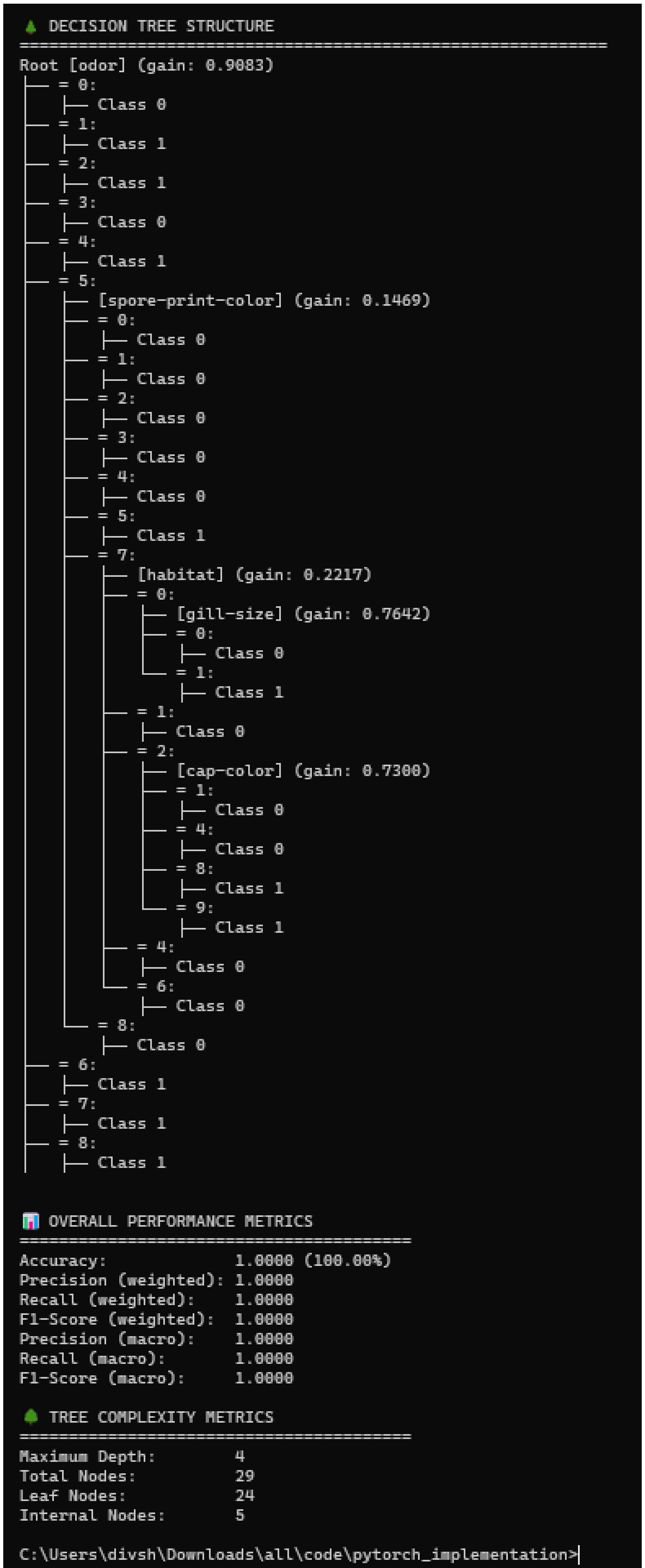
```
C:\Users\divsh\Downloads\all\code\pytorch_implementation>C:\Users\divsh\AppData\Local\Programs\Python\Python39\python.exe test.py --ID EC_C_PES2UG23CS181_Lab3 --data mushrooms.csv
Running tests with PYTORCH framework
============================================================
 target column: 'class' (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', 'r
ing-type', 'spore-print-color', 'population', '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', 'population', '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

C:\Users\divsh\Downloads\all\code\pytorch_implementation>C:\Users\divsh\AppData\Local\Programs\Python\Python39\python.exe test.py --ID EC_C_PES2UG23CS181_Lab3 --data Nursery.csv
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'>
```



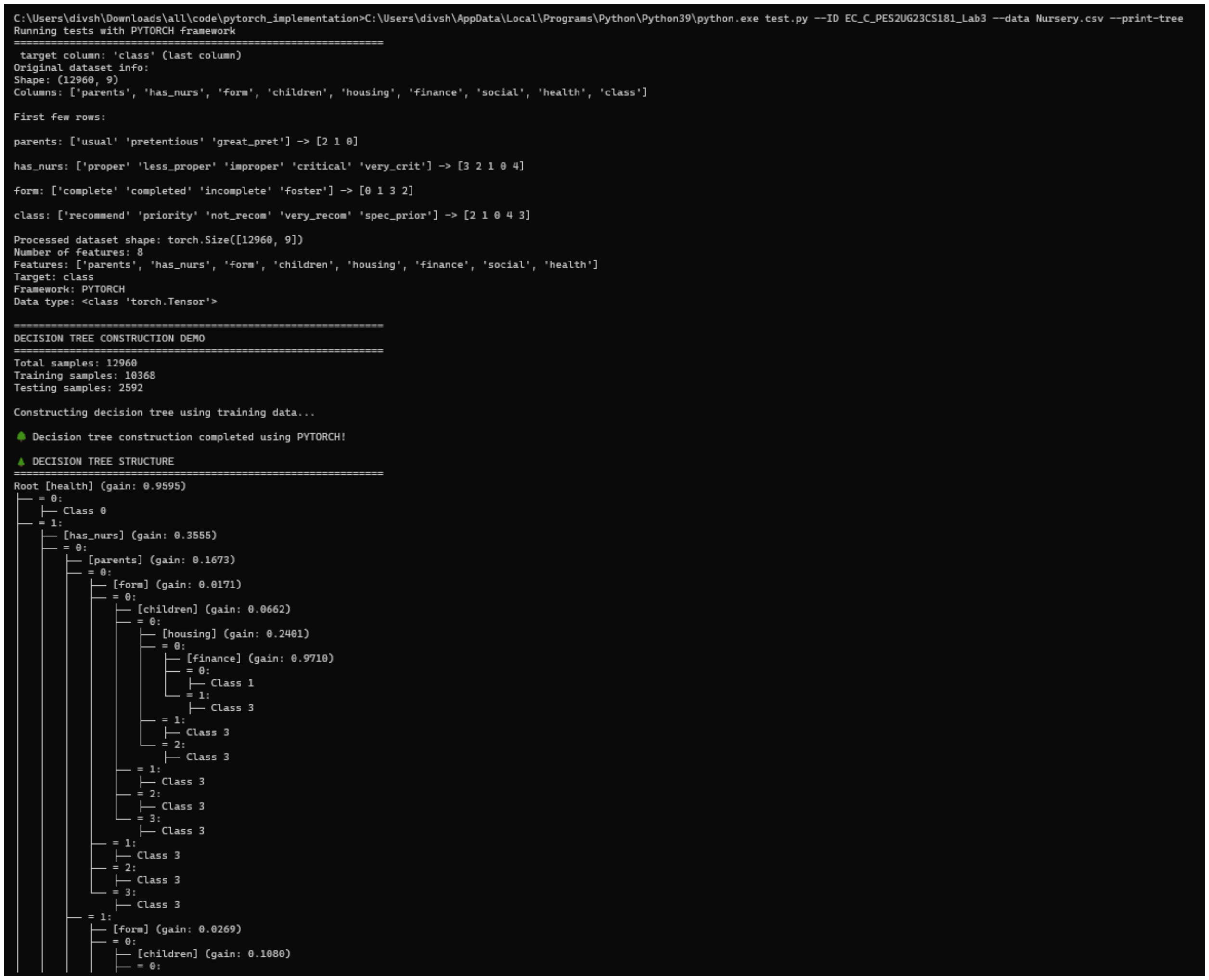
```
============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 12960
Training samples: 10368
Testing samples: 2592

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

C:\Users\divsh\Downloads\all\code\pytorch_implementation>C:\Users\divsh\AppData\Local\Programs\Python\Python39\python.exe test.py --ID EC_C_PES2UG23CS181_Lab3 --data tictactoe.csv
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

Constructing decision tree using training data...

🌲 Decision tree construction completed using PYTORCH!

📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:             0.8730 (87.30%)
Precision (weighted): 0.8741
Recall (weighted):    0.8730
F1-Score (weighted):  0.8734
Precision (macro):    0.8590
Recall (macro):       0.8638
F1-Score (macro):     0.8613

🌲 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        7
Total Nodes:          281
Leaf Nodes:           180
```



```
Internal Nodes:       101

C:\Users\divsh\Downloads\all\code\pytorch_implementation>C:\Users\divsh\AppData\Local\Programs\Python\Python39\python.exe test.py --ID EC_C_PES2UG23CS181_Lab3 --data mushrooms.csv --print-tree
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-s
ing-type', 'spore-print-color', 'population', '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-
ring-type', 'spore-print-color', 'population', '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!

🌲 DECISION TREE STRUCTURE
============================================================
Root [odor] (gain: 0.9083)
├── = 0:
│   ├── Class 0
├── = 1:
│   ├── Class 1
├── = 2:
│   ├── Class 1
├── = 3:
│   ├── Class 0
├── = 4:
│   ├── Class 1
├── = 5:
│   ├── [spore-print-color] (gain: 0.1469)
│   ├── = 0:
│   │   ├── Class 0
│   ├── = 1:
│   │   ├── Class 0
│   ├── = 2:
│   │   ├── Class 0
│   ├── = 3:
│   │   ├── Class 0
│   ├── = 4:
│   │   ├── Class 0
│   ├── = 5:
│   │   ├── Class 1
│   ├── = 7:
│   │   ├── [habitat] (gain: 0.2217)
│   │   ├── = 0:
│   │   │   ├── [gill-size] (gain: 0.7642)
│   │   │   ├── = 0:
│   │   │   │   ├── Class 0
│   │   │   └── = 1:
│   │   │       ├── Class 1
│   │   ├── = 1:
│   │   │   ├── Class 0
│   │   ├── = 2:
│   │   │   ├── [cap-color] (gain: 0.7300)
│   │   │   ├── = 1:
```

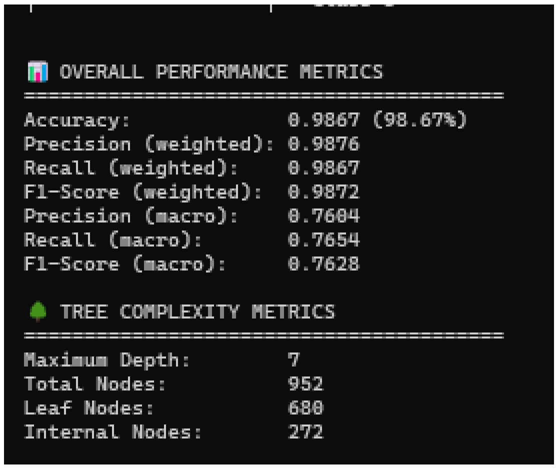


```
🌲 DECISION TREE STRUCTURE
============================================================
Root [odor] (gain: 0.9083)
├── = 0:
│   ├── Class 0
├── = 1:
│   ├── Class 1
├── = 2:
│   ├── Class 1
├── = 3:
│   ├── Class 0
├── = 4:
│   ├── Class 1
├── = 5:
│   ├── [spore-print-color] (gain: 0.1469)
│   ├── = 0:
│   │   ├── Class 0
│   ├── = 1:
│   │   ├── Class 0
│   ├── = 2:
│   │   ├── Class 0
│   ├── = 3:
│   │   ├── Class 0
│   ├── = 4:
│   │   ├── Class 0
│   ├── = 5:
│   │   ├── Class 1
│   ├── = 7:
│   │   ├── [habitat] (gain: 0.2217)
│   │   ├── = 0:
│   │   │   ├── [gill-size] (gain: 0.7642)
│   │   │   ├── = 0:
│   │   │   │   ├── Class 0
│   │   │   └── = 1:
│   │   │       ├── Class 1
│   │   ├── = 1:
│   │   │   ├── Class 0
│   │   ├── = 2:
│   │   │   ├── [cap-color] (gain: 0.7300)
│   │   │   ├── = 1:
│   │   │   │   ├── Class 0
│   │   │   ├── = 4:
│   │   │   │   ├── Class 0
│   │   │   ├── = 8:
│   │   │   │   ├── Class 1
│   │   │   └── = 9:
│   │   │       ├── Class 1
│   │   ├── = 4:
│   │   │   ├── Class 0
│   │   └── = 6:
│   │       ├── Class 0
│   └── = 8:
│       ├── Class 0
├── = 6:
│   ├── Class 1
├── = 7:
│   ├── Class 1
├── = 8:
│   ├── Class 1


📊 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

C:\Users\divsh\Downloads\all\code\pytorch_implementation>
```

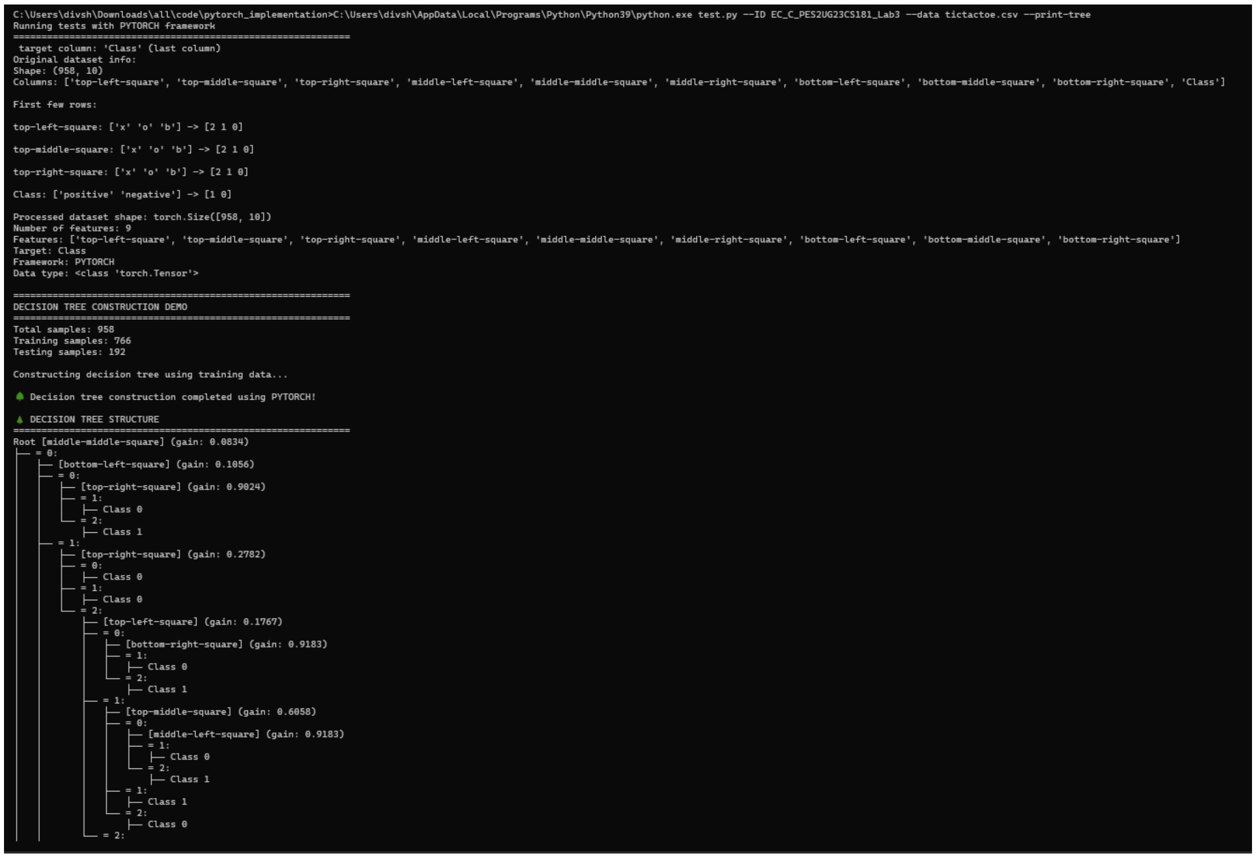


```
C:\Users\divsh\Downloads\all\code\pytorch_implementation>C:\Users\divsh\AppData\Local\Programs\Python\Python39\python.exe test.py --ID EC_C_PES2UG23CS181_Lab3 --data Nursery.csv --print-tree
Running tests with PYTORCH framework
============================================================
 target 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!

🌲 DECISION TREE STRUCTURE
============================================================
Root [health] (gain: 0.9595)
├── = 0:
│   ├── Class 0
├── = 1:
│   ├── [has_nurs] (gain: 0.3555)
│   ├── = 0:
│   │   ├── [parents] (gain: 0.1673)
│   │   ├── = 0:
│   │   │   ├── [form] (gain: 0.0171)
│   │   │   ├── = 0:
│   │   │   │   ├── [children] (gain: 0.0662)
│   │   │   │   ├── = 0:
│   │   │   │   │   ├── [housing] (gain: 0.2401)
│   │   │   │   │   ├── = 0:
│   │   │   │   │   │   ├── [finance] (gain: 0.9710)
│   │   │   │   │   │   ├── = 0:
│   │   │   │   │   │   │   ├── Class 1
│   │   │   │   │   │   └── = 1:
│   │   │   │   │   │       ├── Class 3
│   │   │   │   │   ├── = 1:
│   │   │   │   │   │   ├── Class 3
│   │   │   │   │   └── = 2:
│   │   │   │   │       ├── Class 3
│   │   │   │   ├── = 1:
│   │   │   │   │   ├── Class 3
│   │   │   │   ├── = 2:
│   │   │   │   │   ├── Class 3
│   │   │   │   └── = 3:
│   │   │   │       ├── Class 3
│   │   │   ├── = 1:
│   │   │   │   ├── Class 3
│   │   │   ├── = 2:
│   │   │   │   ├── Class 3
│   │   │   └── = 3:
│   │   │       ├── Class 3
│   │   ├── = 1:
│   │   │   ├── [form] (gain: 0.0269)
│   │   │   ├── = 0:
│   │   │   │   ├── [children] (gain: 0.1080)
│   │   │   │   ├── = 0:
```

……….. And goes on and on and on

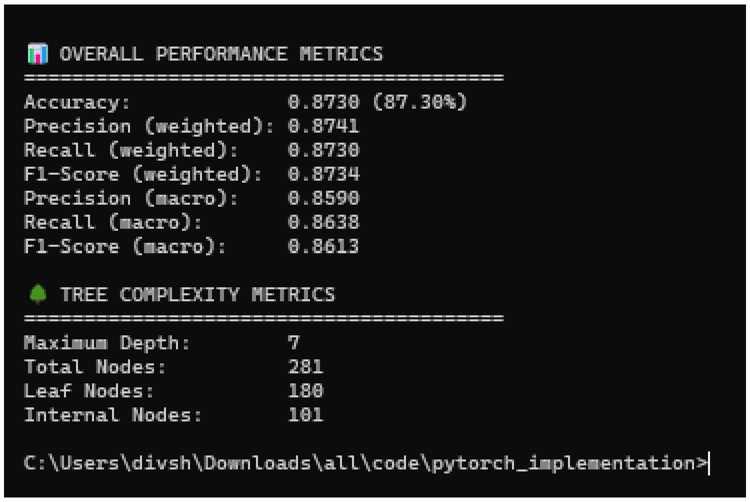


```
📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:              0.9867 (98.67%)
Precision (weighted):  0.9876
Recall (weighted):     0.9867
F1-Score (weighted):   0.9872
Precision (macro):     0.7604
Recall (macro):        0.7654
F1-Score (macro):      0.7628

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



```
C:\Users\divsh\Downloads\all\code\pytorch_implementation>C:\Users\divsh\AppData\Local\Programs\Python\Python39\python.exe test.py --ID EC_C_PES2UG23CS181_Lab3 --data tictactoe.csv --print-tree
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

Constructing decision tree using training data...

🌳 Decision tree construction completed using PYTORCH!

🌲 DECISION TREE STRUCTURE
============================================================
Root [middle-middle-square] (gain: 0.0834)
├── = 0:
│   ├── [bottom-left-square] (gain: 0.1056)
│   ├── = 0:
│   │   ├── [top-right-square] (gain: 0.9024)
│   │   ├── = 1:
│   │   │   ├── Class 0
│   │   └── = 2:
│   │       ├── Class 1
│   ├── = 1:
│   │   ├── [top-right-square] (gain: 0.2782)
│   │   ├── = 0:
│   │   │   ├── Class 0
│   │   ├── = 1:
│   │   │   ├── Class 0
│   │   └── = 2:
│   │       ├── [top-left-square] (gain: 0.1767)
│   │       ├── = 0:
│   │       │   ├── [bottom-right-square] (gain: 0.9183)
│   │       │   ├── = 1:
│   │       │   │   ├── Class 0
│   │       │   └── = 2:
│   │       │       ├── Class 1
│   │       ├── = 1:
│   │       │   ├── [top-middle-square] (gain: 0.6058)
│   │       │   ├── = 0:
│   │       │   │   ├── [middle-left-square] (gain: 0.9183)
│   │       │   │   ├── = 1:
│   │       │   │   │   ├── Class 0
│   │       │   │   └── = 2:
│   │       │   │       ├── Class 1
│   │       │   ├── = 1:
│   │       │   │   ├── Class 1
│   │       │   └── = 2:
│   │       │       ├── Class 0
│   │       └── = 2:
```

And goes on and on



```
📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:              0.8730 (87.30%)
Precision (weighted):  0.8741
Recall (weighted):     0.8730
F1-Score (weighted):   0.8734
Precision (macro):     0.8590
Recall (macro):        0.8638
F1-Score (macro):      0.8613

🌲 TREE COMPLEXITY METRICS
========================================
Maximum Depth:         7
Total Nodes:           281
Leaf Nodes:            180
Internal Nodes:        101

C:\Users\divsh\Downloads\all\code\pytorch_implementation>
```

1. Algorithm Performance

Q1. Which dataset achieved the highest accuracy and why?
A1. The mushroom dataset hit a perfect 100%. That's because some features, like odor, are so strong that they almost single-handedly separate edible from poisonous mushrooms.

Q2. How does dataset size affect performance?
A2. Bigger doesn't always mean better. The nursery dataset is huge (12k+ samples), but its accuracy (98.6%) was slightly lower than mushroom's. Larger datasets give stability, but not guaranteed perfection.

Q3. What role does the number of features play?
A3. More features make the tree grow bigger and deeper. Nursery, with many categorical features, ended up with a massive tree (952 nodes). Mushrooms had fewer, clearer features—so the tree was smaller but spot-on.

---

2. Tree Characteristics Analysis

Q4. Tree Depth comparison?

- Tictactoe: 7
- Nursery: 7
- Mushroom: 4

A4. The mushroom tree is short and sweet, while nursery and tictactoe need deeper layers to handle their complexity.

Q5. Number of nodes?

- Tictactoe: 281
- Nursery: 952
- Mushroom: 29

A5. Nursery’s tree is huge because of multi-valued attributes. Mushrooms, again, are nice and simple.

Q6. Most important features?
A6. Mushroom → odor takes the crown. Nursery → features like parents/finance. Tictactoe → key board positions drive decisions.

Q7. Tree Complexity vs Dataset?
A7. The more variety and attributes a dataset has, the larger the tree. That’s why nursery blows up in size, while mushroom stays compact.

---

3. Dataset-Specific Insights

Q8. Feature Importance?

- Mushroom: odor dominates.
- Nursery: several features matter, no single winner.
- Tictactoe: critical board spots define outcomes.

Q9. Class Distribution?
A9. Mushroom is fairly balanced. Nursery and Tictactoe aren’t, and that imbalance shows in weaker macro scores.

Q10. Decision Patterns?
A10. Mushrooms follow clear, short rules. Nursery requires long paths. Tictactoe has repetitive board-state logic.

Q11. Overfitting Indicators?
A11. Nursery and Tictactoe trees are big and deep—classic overfitting signs. Mushrooms? No such issue, perfectly generalizable.

---

4. Comparative Analysis Report

Q12. Which dataset is most accurate?
A12. Mushroom, hands down (100%).

Q13. How does class imbalance affect tree construction?
A13. In Nursery and Tictactoe, some classes are underrepresented. The tree struggles with them, which drags down the macro precision and recall.

Q14. Binary vs Multi-valued features?
A14. Binary features (like in mushrooms and tictactoe) make life easier for the tree. Multi-valued ones (nursery) make it explode in complexity.

Q15. Practical applications?

- Mushroom → Food safety (edible vs poisonous).
- Nursery → Resource allocation or childcare planning.
- Tictactoe → Strategy and AI in simple games.

Q16. Interpretability advantages?
A16. Mushroom rules are crystal clear. Nursery's tree is too big to easily explain. Tictactoe is understandable but repetitive.

Q17. How would you improve performance?
A17. Pruning would help trim down nursery and tictactoe trees, making them simpler. Handling class imbalance with sampling or weighted splits would also improve fairness.