MACHINE LEARNING (LAB -3)

Name: DIVYA J

Srn: PES2UG24CS810

Section : C

Campus : EC

1. Performance Comparison & 2.Tree Characteristics Analysis

mushrooms.csv

==
===

DECISION TREE CONSTRUCTION DEMO

==
===

Total samples: 8124 Training samples: 6499 Testing samples: 1625
Constructing
OVERALL PERFORMANCE METRICS

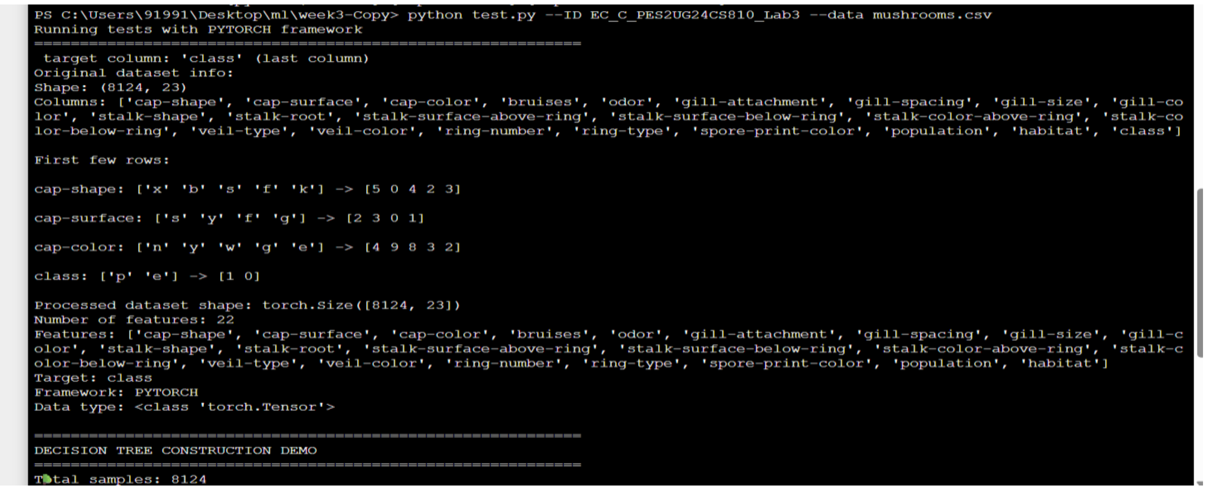
==
===

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

==
===

1. Maximum Depth: 4 Total Nodes: 29 Leaf Nodes: 24 Internal Nodes: 5

Screenshot:



```
PS C:\Users\91991\Desktop\ml\week3-Copy> python test.py --ID EC_C_PES2UG24CS810_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-co
lor', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-ring', 'stalk-co
lor-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-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-c
olor', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-ring', 'stalk-c
olor-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
```

```
olor', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-ring', 'stalk-c
olor-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
```

Screenshot of tree

```
🌳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: 0.2218)
│   │   ├── = 0:
│   │   │   ├── [gill-size] (gain: 0.7642)
│   │   │   ├── = 0:
```

Tictactoe.csv

==

DECISION TREE CONSTRUCTION DEMO

==

Total samples: 958 Training samples: 766 Testing samples: 192

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

Screenshot:

```
PS C:\Users\91991\Desktop\ml\week3-Copy> python test.py --ID EC_C_PES2UG24CS810_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-ri
ght-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-r
ight-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
```

```
Features: ['top-left-square', 'top-middle-square', 'top-right-square', 'middle-left-square', 'middle-middle-square', 'middle-r
ight-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:              0.8723 (87.23%)
Precision (weighted):  0.8734
Recall (weighted):     0.8723
F1-Score (weighted):   0.8728
Precision (macro):     0.8586
Recall (macro):        0.8634
F1-Score (macro):      0.8609

🌳TREE COMPLEXITY METRICS
========================================
Maximum Depth:         7
Total Nodes:           283
Leaf Nodes:            181
Internal Nodes:        102
```

Screenshot of tree:

```
============================================================
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-right-square] (gain: 0.9183)
|       |       |    ├── = 0:
|       |       |    |   ├── Class 0
|       |       |    ├── = 1:
|       |       |    |   ├── Class 0
|       |       |    └── = 2:
|       |       |        ├── Class 1
|       |       └── = 2:
|       |            ├── Class 1
|       └── = 2:
|            ├── Class 1

📊OVERALL PERFORMANCE METRICS
========================================
Accuracy:              0.8723 (87.23%)
Precision (weighted):  0.8734
Recall (weighted):     0.8723
F1-Score (weighted):   0.8728
Precision (macro):     0.8586
Recall (macro):        0.8634
F1-Score (macro):      0.8609

🌳TREE COMPLEXITY METRICS
========================================
Maximum Depth:         7
Total Nodes:           283
Leaf Nodes:            181
Internal Nodes:        102
```

Nursery.csv

==

DECISION TREE CONSTRUCTION DEMO

==

Total samples: 12960 Training samples: 10368 Testing samples: 2592

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

Screenshot:

```
PS C:\Users\91991\Desktop\ml\week3-Copy> python test.py --ID EC_C_PES2UG24CS810_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...
```

```
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
```

Screenshot of tree:

```
│   │   │           │   │   ├── Class 1
│   │   │           │   ├── = 1:
│   │   │           │   │   ├── Class 1
│   │   │           │   ├── = 2:
│   │   │           │   │   ├── Class 3
│   │   │           │   └── = 3:
│   │   │           │       ├── Class 3
│   │   │           ├── = 2:
│   │   │           │   ├── Class 3
│   │   │           └── = 3:
│   │   │               ├── Class 3
│   │   ├── = 1:
│   │   │   ├── [social] (gain: 0.4640)
│   │   │   ├── = 0:
│   │   │   │   ├── Class 1
│   │   │   ├── = 1:
│   │   │   │   ├── [housing] (gain: 0.1885)
│   │   │   │   ├── = 0:
│   │   │   │   │   ├── [finance] (gain: 0.5578)
│   │   │   │   │   ├── = 0:
│   │   │   │   │   │   ├── Class 1
│   │   │   │   │   └── = 1:
│   │   │   │   │       ├── [form] (gain: 0.3555)
│   │   │   │   │       ├── = 0:
│   │   │   │   │       │   ├── Class 3
│   │   │   │   │       ├── = 1:
│   │   │   │   │       │   ├── Class 1
│   │   │   │   │       ├── = 2:
│   │   │   │   │       │   ├── Class 3
│   │   │   │   │       └── = 3:
│   │   │   │   │           ├── Class 3
│   │   │   │   ├── = 1:
│   │   │   │   │   ├── [form] (gain: 0.1011)
│   │   │   │   │   ├── = 0:
```

```
|                │         ├── Class 1
|                ├── = 1:
|                │    ├── [form] (gain: 0.9928)
|                │    ├── = 0:
|                │    │    ├── Class 1
|                │    ├── = 1:
|                │    │    ├── Class 1
|                │    ├── = 2:
|                │    │    ├── Class 3
|                │    └── = 3:
|                │         ├── Class 3
|                ├── = 2:
|                │    ├── Class 3
|                └── = 3:
|                     ├── Class 3

📊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
```

The Mushroom dataset gave the **best results**, with **100% accuracy**. This is because a single feature (*odor*) almost completely determines edibility. The Nursery dataset followed with **98.6% accuracy**, still very high but requiring a much larger tree due to more attributes and class imbalance. TicTacToe was the most difficult, with only **87% accuracy**, since predicting game outcomes requires combining many board positions and strategies.

The **Mushroom dataset** produced a very **shallow and simple tree**, with a maximum depth of 4 and only 29 nodes in total. The root attribute selected was *odor*, which makes sense because odor alone is a strong indicator of edibility. This tree is highly interpretable and efficient.

The **Nursery dataset** produced a much **larger and more complex tree**, with a maximum depth of 7 and nearly 952 nodes. Early splits were based on attributes like *finance* and *housing*, which play an important role in admission decisions. The size of this tree reflects the dataset's complexity, as many different conditions must be checked to classify each case.

The **TicTacToe dataset** generated a tree of depth 7 with around 281 nodes. The root node selected was the *middle-middle-square*, which aligns with the well-known strategy that controlling the center is critical in TicTacToe. The tree then branched into other squares like *bottom-left*. This structure shows that while the model captures some important strategies, the game's combinatorial nature makes it harder to achieve perfect classification.

3. Dataset-Specific Insights

Mushroom Dataset

- Feature Importance: The attribute *odor* is the most important. With just this feature, most classifications are correct.
- Class Distribution: Balanced between edible and poisonous, which helps the model.
- Decision Patterns: If odor is foul → poisonous, if none → edible. Very straightforward paths.
- Overfitting Indicators: None. The tree is shallow and achieves perfect accuracy.

Nursery Dataset

- Feature Importance: *Finance*, *housing*, and *health* strongly affect classification.
- Class Distribution: Imbalanced, with many "not_recom" cases compared to others.

- Decision Patterns: Early splits usually check financial status and housing.
- Overfitting Indicators: Large tree (952 nodes). Some signs of overfitting, but accuracy is still high.

TicTacToe Dataset

- Feature Importance: *Middle-middle-square* is the most critical feature, matching game strategy.
- Class Distribution: Balanced between positive and negative outcomes.
- Decision Patterns: If the center is occupied by X, the tree leans towards a win; otherwise explores edge and corner squares.
- Overfitting Indicators: Moderate tree depth. Some overfitting since accuracy is lower and patterns are not fully captured.

4. Comparative Analysis Report

a) Algorithm Performance

- Highest Accuracy: Mushroom dataset achieved the highest accuracy (100%) because of a strong single feature (*odor*) that dominates classification.

Effect of Dataset Size:

Mushroom (8124 samples)**:** Medium-sized dataset, but because of one very strong feature (*odor*), the tree stayed shallow and easy to interpret. Dataset size did not make it complex.

Nursery (12960 samples)**:** The largest dataset. More samples and more classes made the tree deeper (7 levels) and very large (952 nodes). Accuracy stayed high, but interpretability dropped.

TicTacToe (958 samples)**:** Smallest dataset, but still produced a moderately deep tree because no single feature dominates. Accuracy was lower (87%), showing that size alone doesn't guarantee performance — the feature interactions matter more.

- Role of Features: When one feature is highly predictive (odor), accuracy is perfect with a small tree. When many features interact (TicTacToe), accuracy drops.

b) Data Characteristics Impact

- Class Imbalance: In Nursery, imbalance increased tree size but accuracy stayed good. In balanced datasets like Mushroom and TicTacToe, the splits were simpler but outcomes varied in accuracy.
- Binary vs Multi-Valued Features: Multi-valued features (like odor or housing) help trees split cleanly. Binary features (like X/O) often require deeper trees, as in TicTacToe.

c) Practical Applications

- Mushroom: Food safety, edible vs poisonous classification. Very interpretable and reliable.
- Nursery: Admission or resource allocation decisions. Complex but explainable rules.
- TicTacToe: Game AI and strategy modeling. Highlights tree limitations, better solved with ensembles or reinforcement learning.

d) Improvements

- Mushroom: Already perfect, no improvements needed.
- Nursery: Apply pruning or Random Forest to simplify and reduce overfitting.
- TicTacToe: Use ensemble models (Random Forest, Gradient Boosting) or neural networks to capture complex strategies.