<u>MACHINE LEARNING LAB-3</u>

NAME:- Drishti Golchha
SRN:- PES2UG23CS185
SECTION:-5C

OUTPUTS:-

```
PS C:\Users\drish> cd OneDrive
PS C:\Users\drish\OneDrive> cd Desktop
PS C:\Users\drish\OneDrive\Desktop> cd code
PS C:\Users\drish\OneDrive\Desktop\code> cd pytorch_implementation
PS C:\Users\drish\OneDrive\Desktop\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS185_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-surfa
ce-below-ring', 'stalk-color-above-ring', 'stalk-color-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-color', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surf
ace-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
PS C:\Users\drish\OneDrive\Desktop\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS185_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-sq
uare', '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-s
quare']
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
PS C:\Users\drish\OneDrive\Desktop\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS185_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
PS C:\Users\drish\OneDrive\Desktop\code\pytorch_implementation>
```

Mushrooms.csv

```
🌲 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:
```

The tree is overfitting

Tictactoe.csv

```
🌲 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
```

The tree is overfitting

Nursery.csv

```
│   │   │   │   ├─ Class 1
│   │   │   ├─ = 1:
│   │   │   │   ├─ [housing] (gain: 0.1886)
│   │   │   │   ├─ = 0:
│   │   │   │   │   ├─ [finance] (gain: 0.5577)
│   │   │   │   │   ├─ = 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:
│   │   │   │   │   │   ├─ [children] (gain: 0.7219)
│   │   │   │   │   │   ├─ = 0:
│   │   │   │   │   │   │   ├─ Class 1
│   │   │   │   │   │   ├─ = 1:
│   │   │   │   │   │   │   ├─ Class 3
│   │   │   │   │   │   ├─ = 2:
│   │   │   │   │   │   │   ├─ Class 3
│   │   │   │   │   │   └─ = 3:
│   │   │   │   │   │       ├─ Class 3
```

The tree is overfitting

a) Algorithm Performance

1. Which dataset achieved the highest accuracy and why?

- Datasets with well-separated class distributions (e.g., binary classification with clear attribute splits) typically achieve higher accuracy.
- For example, a dataset like Mushrooms (edible vs poisonous) tends to yield very high accuracy because features such as color, odor, and gill shape are strongly correlated with the target variable.
- The accuracy is highest when features are highly informative (strong predictors) and redundancy/noise is minimal.

2. How does dataset size affect performance?

- Small datasets may lead to overfitting, as the tree might memorize training examples instead of generalizing.
- Larger datasets improve generalization by covering more variation in data, leading to more robust splits.
- However, very large datasets may increase training time and require pruning/regularization to avoid overly deep trees.

3. What role does the number of features play?

- More features give the algorithm more potential splits, which can improve performance if features are relevant.
- However, too many irrelevant or redundant features can lead to overfitting and decreased interpretability.
- Decision trees perform best when provided with a moderate number of highly informative features.

b) Data Characteristics Impact

1. How does class imbalance affect tree construction?

- In imbalanced datasets (e.g., 90% class A, 10% class B), the tree tends to favor the majority class, leading to poor recall for the minority class.
- Splitting criteria like information gain may become biased toward majority class features.
- Techniques such as SMOTE, class weighting, or balanced splitting criteria can help mitigate this issue.

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

- Binary features (Yes/No, True/False) often lead to simpler, shallower trees and are easy to interpret.
- Multi-valued categorical features provide richer splits but may cause over-branching, leading to deep, complex trees.
- In practice, binary features often yield more robust trees, while multi-valued features need careful handling (e.g., grouping categories or using feature selection).

c) Practical Applications

1. For which real-world scenarios is each dataset type most relevant?

- Binary feature datasets: Medical diagnosis (disease vs no disease), fraud detection, spam filtering.
- Multi-valued categorical datasets: Market segmentation, recommendation systems, text classification.
- Continuous + categorical datasets: Credit scoring, risk assessment, customer churn prediction.

2. What are the interpretability advantages for each domain?

- Medical & healthcare (binary features): Trees are highly interpretable for doctors to understand "if-then" decision rules.
- Retail/Marketing (multi-valued): Helps businesses see which product attributes or demographics drive sales.
- Finance (mixed features): Provides transparency in risk models, ensuring fairness and regulatory compliance.

3. How would you improve performance for each dataset?

- For small datasets: Use pruning or cross-validation to avoid overfitting.
- For large datasets: Apply tree depth limits or switch to Random Forests/Gradient Boosted Trees for better scalability.
- For imbalanced datasets: Apply resampling, cost-sensitive learning, or ensemble methods.
- For high-dimensional datasets: Perform feature selection or dimensionality reduction before training.