Machine Learning

Name: Tanishk Maheshwari

SRN: PES1UG23AM334

LAB2

Mushroom.csv Dataset

Student_Lab.py code-

```
import torch

def get_entropy_of_dataset(tensor: torch.Tensor) -> float:
    """
    Calculate entropy of the dataset using the target column (last column).
    """
    target_col = tensor[:, -1]
    values, counts = torch.unique(target_col, return_counts=True)
    probabilities = counts.float() / counts.sum()

    entropy = -torch.sum(probabilities * torch.log2(probabilities + 1e-9))  #
avoid log2(0)
    return round(float(entropy.item()), 4)


def get_avg_info_of_attribute(tensor: torch.Tensor, attribute: int) -> float:
    """
    Calculate average information (weighted entropy) for a given attribute.
    """
    total_samples = tensor.size(0)
    attribute_values = tensor[:, attribute]
    unique_vals, counts = torch.unique(attribute_values, return_counts=True)

    avg_info = 0.0
    for val, count in zip(unique_vals, counts):
        mask = (attribute_values == val)
        subset = tensor[mask]
        subset_entropy = get_entropy_of_dataset(subset)
        weight = count.item() / total_samples
        avg_info += weight * subset_entropy

    return round(float(avg_info), 4)


def get_information_gain(tensor: torch.Tensor, attribute: int) -> float:
    """
```

```
    Information Gain = Entropy(dataset) - Avg_Info(attribute)
    """
    total_entropy = get_entropy_of_dataset(tensor)
    avg_info = get_avg_info_of_attribute(tensor, attribute)
    info_gain = total_entropy - avg_info
    return round(float(info_gain), 4)


def get_selected_attribute(tensor: torch.Tensor) -> tuple:
    """
    Select attribute with highest information gain.
    Returns (gain_dict, best_attribute_index).
    """
    n_features = tensor.size(1) - 1  # exclude target
    gains = {}

    for attr in range(n_features):
        gains[attr] = get_information_gain(tensor, attr)

    best_attr = max(gains, key=gains.get)
    return gains, best_attr
```

Overall Performance Analysis-

```
📊 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
```

Tree Characteristics Analysis-

Mushroom Dataset:

- Shallow tree
- Strong root feature (odour)
- Few nodes required.
- Tree is simple but powerful due to highly discriminative features.

Dataset Specific Insights-

Mushroom:

- Features are very clear and separable (e.g., odor, spore-print-colour).
- Results in near perfect classification accuracy.

Output 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:
│    │              ├── [top-middle-square] (gain: 0.3392)
│    │              ├── = 0:
│    │              │    ├── [middle-left-square] (gain: 0.9183)
│    │              │    ├── = 0:
│    │              │    │    ├── Class 0
│    │              │    ├── = 1:
│    │              │    │    ├── Class 1
│    │              │    └── = 2:
│    │              │         ├── Class 0
│    │              ├── = 1:
│    │              │    ├── [middle-left-square] (gain: 0.9183)
│    │              │    ├── = 0:
│    │              │    │    ├── Class 1
│    │              │    ├── = 1:
│    │              │    │    ├── Class 1
│    │              │    └── = 2:
│    │              │         ├── Class 0
│    │              └── = 2:
│    │                   ├── Class 1
│    └── = 2:
│         ├── [
│         │         ├── = 1:
│         │         │    ├── [top-right-square] (gain: 0.9183)
│         │         │    ├── = 0:
│         │         │    │    ├── Class 0
│         │         │    ├── = 1:
│         │         │    │    ├── Class 0
│         │         │    └── = 2:
│         │         │         ├── Class 1
│         │         └── = 2:
│         │              ├── Class 1
│         └── = 2:
│              ├── Class 1
```

2)TicTacToe.csv

Overall Performance Analysis-

```
📊 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
```

Tree Characteristics Analysis-

Tic-Tac-Toe Dataset:

- Moderate tree depth
- Binary features
- Paths resemble game logic.
- Tree remains interpretable.

Dataset Specific Analysis-

Tic-Tac-Toe:

- All 9 features are binary and directly linked to outcomes.
- Leads to good accuracy with interpretable rules.

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-surface-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-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!

🌲 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:
│   │   │   │   ├── Class 0
│   │   │   └── = 1:
│   │   │       ├── Class 1
│   └── = 8:
│       ├── Class 0
├── = 6:
│   ├── Class 1
├── = 7:
│   ├── Class 1
├── = 8:
│   ├── Class 1
```

3)Nursery.csv

Performance Analysis Report-

```
📊 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 Characteristics Analysis-

Nursery Dataset:

- Deep tree with many nodes, root at 'health'.
- Multi-valued features cause high complexity, making interpretation harder.

Dataset Specific Insights-

Nursery:

- Features are multi-valued and target classes imbalanced.
- This increases tree depth and lowers accuracy, also showing risk of overfitting.

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-surface-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-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!

🌲 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
   └── = 8:
│       ├── Class 0
├── = 6:
│   ├── Class 1
├── = 7:
│   ├── Class 1
├── = 8:
│   ├── Class 1



🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        4
Total Nodes:          29
Leaf Nodes:           24
Internal Nodes:       5
```

A) Algorithm Performance

a. Which Dataset Achieved the Highest Accuracy and Why?

The Mushroom dataset gave the best accuracy.

• It has 22 features, and many are very useful (like odor, color) to tell if a mushroom is edible

or poisonous.

• The features are clear and separate, so the decision tree can easily split and classify.

• There's very little noise, so the model rarely gets confused.

b. How Does Dataset Size Affect Performance?

Bigger datasets usually give better results because:

• The model sees more examples, learns better, and avoids overfitting.

• But after a certain size, adding more data doesn't improve much.

• Small datasets can cause overfitting since the model may learn patterns that don't represent

the real-world well.

c. What Role Does the Number of Features Play?

Features matter a lot:

• More useful features → higher accuracy (but can also increase complexity).

• Mushroom (22 features): Lots of helpful attributes → high accuracy.

• Tic-Tac-Toe (9 features): All features are directly useful → good accuracy with small tree.

• Nursery (8 features): Features are multi-valued and not always strong → harder to classify

correctly.

B) Data Characteristics Impact

a. Class Imbalance Impact

If one class has many more examples than others, the model may favour the majority class and ignore

smaller ones.

• Example: Nursery dataset struggles with rare classes like "special priority".

• This causes shallow splits for common classes, and deeper, less accurate splits for rare ones.

b. Binary vs Multi-Valued Features

Binary features (yes/no) → simple, stable, less overfitting.

Multi-valued features → complex, can cause deep trees and overfitting, especially if values don't

relate well to the target.

• Binary = easier to interpret.

• Multi-valued = needs careful handling.

C) Practical Applications & Interpretability

a. Real-World Relevance of Each Dataset Type

• Mushroom: Food safety, farming, toxicology studies.

• Tic-Tac-Toe: Game AI, reinforcement learning, strategy analysis.

• Nursery: School admissions, social services, welfare systems.

b. Interpretability Advantages for Each Domain

• Mushroom: Easy rules like odor = foul → poisonous are clear and useful.

• Tic-Tac-Toe: Decision paths match game logic, easy to visualize strategies.

• Nursery: Harder to interpret due to many values, but important in policy areas for fairness

and accountability.

c. How to Improve Performance for Each Dataset?

• Mushroom: Remove extra/unnecessary features, fix class imbalance if needed.

• Tic-Tac-Toe: Balance training data, maybe use ensembles for tricky cases (like draws).

• Nursery: Handle multi-valued features better (group/bin), fix class imbalance, prune trees to

reduce overfitting, try hybrid models if needed.