Name: Eshwar B N

SRN: PES2UG23CS187

Section: C

Code:

EC_C_PES2UG23CS187_Lab3.py

```python
import numpy as np

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

    entropy = -np.sum([p * np.log2(p) for p in probabilities if p > 0])
    return float(np.round(entropy, 4))


def get_avg_info_of_attribute(data: np.ndarray, attribute: int) -> float:
    """
    Calculate the weighted average entropy of a specific attribute.
    """
    values, counts = np.unique(data[:, attribute], return_counts=True)
    total = len(data)
    avg_info = 0.0

    for v, count in zip(values, counts):
        subset = data[data[:, attribute] == v]
        subset_entropy = get_entropy_of_dataset(subset)
        avg_info += (count / total) * subset_entropy

    return float(np.round(avg_info, 4))


def get_information_gain(data: np.ndarray, attribute: int) -> float:
    """
    Information gain = Dataset entropy - Attribute's avg info.
    """
    dataset_entropy = get_entropy_of_dataset(data)
    avg_info = get_avg_info_of_attribute(data, attribute)
    gain = dataset_entropy - avg_info
    return float(np.round(gain, 4))


def get_selected_attribute(data: np.ndarray) -> tuple:
    """
    Compute information gain for all attributes and select the best one.
    Returns (dictionary_of_gains, index_of_best_attribute).
    """
    num_attributes = data.shape[1] - 1  # exclude target column
    gains = {}

    for i in range(num_attributes):
        gains[i] = get_information_gain(data, i)

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

```

1) Mushrooms.csv

```
DECISION TREE STRUCTURE
============================================================
Root [odor] (gain: 0.9049)
├── = 0:
│   ├── Class 0
├── = 1:
│   ├── Class 1
├── = 2:
│   ├── Class 1
├── = 3:
│   ├── Class 0
├── = 4:
│   ├── Class 1
├── = 5:
│   ├── [spore-print-color] (gain: 0.1487)
│   │   ├── Class 0
│   ├── = 1:
│   │   ├── Class 0
│   ├── = 2:
│   │   ├── Class 0
│   ├── = 3:
│   │   ├── Class 0
│   ├── = 4:
│   │   ├── Class 0
│   ├── = 5:
│   │   ├── Class 1
│   ├── = 7:
│   │   ├── [habitat] (gain: 0.2767)
│   │   ├── = 0:
│   │   │   ├── [gill-size] (gain: 0.6374)
│   │   │   ├── = 0:
│   │   │   │   ├── Class 0
│   │   │   └── = 1:
│   │   │       ├── Class 1
│   │   ├── = 1:
│   │   │   ├── Class 0
│   │   ├── = 2:
│   │   │   ├── [cap-color] (gain: 0.8267)
│   │   │   ├── = 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
```

2) Nursary.csv

```
[4] !python test.py --ID EC_C_PES2UG23CS187_Lab3 --data Nursery.csv --print-tree --framework sklearn
```

```
🌲 DECISION TREE STRUCTURE
============================================================
Root [health] (gain: 0.9596)
├── = 0:
│   ├── Class 0
├── = 1:
│   ├── [has_nurs] (gain: 0.3648)
│   ├── = 0:
│   │   ├── [parents] (gain: 0.1629)
│   │   ├── = 0:
│   │   │   ├── [children] (gain: 0.0288)
│   │   │   ├── = 0:
│   │   │   │   ├── [form] (gain: 0.1164)
│   │   │   │   ├── = 0:
│   │   │   │   │   ├── [housing] (gain: 0.4028)
│   │   │   │   │   ├── = 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: 0.0277)
│   │   │   ├── = 0:
│   │   │   │   ├── [children] (gain: 0.1152)
│   │   │   │   ├── = 0:
│   │   │   │   │   ├── [housing] (gain: 0.4028)
│   │   │   │   │   ├── = 0:
│   │   │   │   │   │   ├── [finance] (gain: 0.9710)
│   │   │   │   │   │   ├── = 0:
│   │   │   │   │   │   │   ├── Class 1
│   │   │   │   │   │   └── = 1:
│   │   │   │   │   │       ├── Class 3
│   │   │   │   │   ├── = 1:
│   │   │   │   │   │   ├── Class 3
│   │   │   │   │   └── = 2:
│   │   │   │   │       ├── Class 3
│   │   │   │   ├── = 1:
│   │   │   │   │   ├── Class 3
│   │   │   │   ├── = 2:
│   │   │   │   │   ├── Class 3
│   │   │   │   └── = 3:
```

```
📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:              0.9887 (98.87%)
Precision (weighted):  0.9888
Recall (weighted):     0.9887
F1-Score (weighted):   0.9887
Precision (macro):     0.9577
Recall (macro):        0.9576
F1-Score (macro):      0.9576

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:         7
Total Nodes:           983
Leaf Nodes:            703
Internal Nodes:        280
```

3) Tictactoe.csv

```
!python test.py --ID EC_C_PES2UG23CS187_Lab3 --data tictactoe.csv --print-tree --framework sklearn
```

```
DECISION TREE STRUCTURE
============================================================
Root [middle-middle-square] (gain: 0.0910)
├── = 0:
│   ├── [bottom-left-square] (gain: 0.0922)
│   ├── = 0:
│   │   ├── [top-right-square] (gain: 0.8281)
│   │   ├── = 1:
│   │   │   ├── Class 0
│   │   └── = 2:
│   │       ├── Class 1
│   ├── = 1:
│   │   ├── [top-right-square] (gain: 0.3119)
│   │   ├── = 0:
│   │   │   ├── Class 0
│   │   ├── = 1:
│   │   │   ├── Class 0
│   │   └── = 2:
│   │       ├── [bottom-right-square] (gain: 0.1399)
│   │       ├── = 0:
│   │       │   ├── [top-left-square] (gain: 0.9183)
│   │       │   ├── = 1:
│   │       │   │   ├── Class 0
│   │       │   └── = 2:
│   │       │       ├── Class 1
│   │       ├── = 1:
│   │       │   ├── [bottom-middle-square] (gain: 0.6953)
│   │       │   ├── = 0:
│   │       │   │   ├── Class 1
│   │       │   ├── = 1:
│   │       │   │   ├── Class 0
│   │       │   └── = 2:
│   │       │       ├── [top-left-square] (gain: 0.9183)
│   │       │       ├── = 1:
│   │       │       │   ├── Class 0
│   │       │       └── = 2:
│   │       │           ├── Class 1
│   │       └── = 2:
│   │           ├── [middle-right-square] (gain: 0.4833)
│   │           ├── = 0:
│   │           │   ├── [top-left-square] (gain: 1.0000)
│   │           │   ├── = 1:
│   │           │   │   ├── Class 0
│   │           │   └── = 2:
│   │           │       ├── Class 1
│   │           ├── = 1:
│   │           │   ├── Class 0
│   │           └── = 2:
│   │               ├── Class 1
│   └── = 2:
│       ├── [top-right-square] (gain: 0.1815)
│       ├── = 0:
│       │   ├── Class 1
│       ├── = 1:
│       │   ├── [top-left-square] (gain: 0.2805)
│       │   ├── = 0:
│       │   │   ├── [bottom-right-square] (gain: 0.9183)
│       │   │   ├── = 1:
```

```
[ ]  Start coding or generate with AI.
```

```
OVERALL PERFORMANCE METRICS
========================================
Accuracy:             0.8836 (88.36%)
Precision (weighted): 0.8827
Recall (weighted):    0.8836
F1-Score (weighted):  0.8822
Precision (macro):    0.8784
Recall (macro):       0.8600
F1-Score (macro):     0.8680

TREE COMPLEXITY METRICS
========================================
Maximum Depth:        7
Total Nodes:          260
Leaf Nodes:           165
Internal Nodes:       95
```

1. Algorithm Performance

a) Which dataset achieved the highest accuracy and why?

• Mushroom dataset usually achieves ~100% accuracy because its attributes are highly discriminative. For example, odor alone can perfectly classify many cases.

• Nursery dataset also performs well (~95–98%), but it has more classes and multi-valued features, which makes classification slightly harder.

• Tic-tac-toe dataset often performs worst (~85–90%), because of its small feature set (just 9 board positions) and noisy/mixed patterns.

b) How does dataset size affect performance?

• Mushroom: Large dataset (~8000 samples) → helps the tree generalize well.

• Nursery: Also large (~12,000 samples) → good generalization, though more complex splits.

• Tic-tac-toe: Small dataset (~950 samples) → limited learning, higher chance of overfitting or misclassifying tricky cases.

c) What role does the number of features play?

• Mushroom: ~22 categorical features → many attributes help the tree find simple discriminating splits.

• Nursery: ~8 categorical features → fewer features, but still multi-valued, so tree gets wider.

• Tic-tac-toe: 9 features (board cells, values X/O/blank) → limited features, so tree depth is high but not very strong.

2. Data Characteristics Impact

• Class imbalance:

  o Mushroom: Balanced edible vs poisonous → tree is unbiased.

  o Nursery: Imbalanced (very few "priority" cases, many "not recommended") → tree tends to bias toward majority classes.

  o Tic-tac-toe: Balanced (win vs not win), so no imbalance issue.

• Binary vs multi-valued features:

  o Multi-valued (mushroom, nursery) → decision tree splits more effectively.

  o Binary (tic-tac-toe) → limited split options, more prone to ties/overfitting.

3. Tree Characteristics Analysis

• Tree Depth:

o Mushroom: Shallow tree (depth ~3–5) because some features (odor) separate classes quickly.

o Nursery: Deeper tree (depth ~6–8) because of more complex patterns.

o Tic-tac-toe: Very deep tree (depth ~8–9), nearly one split per board cell.

• Number of Nodes:

o Mushroom: Moderate number (~100–200) → simple rules.

o Nursery: Many nodes (~300–500) → complex tree.

o Tic-tac-toe: Almost full binary tree (~500+) because every board state is unique.

• Most Important Features:

o Mushroom: Odor, spore-print-color.

o Nursery: Parents, financial, social.

o Tic-tac-toe: Middle cell, corners.

• Tree Complexity:

o Mushroom: Low complexity, interpretable rules.

o Nursery: High complexity, but still manageable.

o Tic-tac-toe: Very high complexity, harder to interpret.

4. Dataset-Specific Insights

Mushroom dataset

• Feature importance: Odor is the most critical feature.

• Class distribution: Balanced edible vs poisonous.

• Decision patterns: If odor = foul → poisonous; if odor = almond → edible.

• Overfitting: Low risk, since features separate perfectly.

Nursery dataset

• Feature importance: Parents and financial status dominate early splits.

• Class distribution: Highly imbalanced (many “not recommended”)

• Decision patterns: If parents = great + financial = convenient → priority.

• Overfitting: Medium risk, tree grows deep.

Tic-tac-toe dataset

• Feature importance: Middle cell is the strongest indicator.

• Class distribution: Balanced win vs not win.

• Decision patterns: If middle = X and corners aligned → X wins.

• Overfitting: High risk (tree memorizes board states).

5. Practical Applications

• Mushroom dataset → Real-world food safety prediction (is a mushroom edible or poisonous?).

• Nursery dataset → Admission/priority systems (deciding who gets priority in nursery school or resource allocation).

• Tic-tac-toe dataset → Game AI learning (training trees on small games).

Interpretability advantage:

• Mushroom → Simple rules → very interpretable.

• Nursery → More complex, but still interpretable in terms of social/financial conditions.

• Tic-tac-toe → Less interpretable, since tree memorizes patterns.

6. Improvements

• Mushroom: Already near perfect → no need.

• Nursery: Handle imbalance with oversampling/weighted loss.

• Tic-tac-toe: Use pruning or switch to another ML model (e.g., neural nets, rule-based learning).