NAME:VARUN KUMAR L	SRN:PES2UG24CS827
SEC:C	SUB:ML LAB

Code:

test.py C:\...\pytorch_implementation | student_lab.py | lab_Sample_Solution.py | Release Notes: 1.103.1

C: > ml > all > EC_CSE-C_PES2UG24CS827_Lab3.py > ...

```python
import torch

# ------------------------------
# 1. Entropy of Dataset
# ------------------------------
def get_entropy_of_dataset(tensor: torch.Tensor) -> float:
    """
    Calculates entropy of the dataset
    tensor: torch.Tensor with last column as target
    """
    target_col = tensor[:, -1]   # last column = class labels
    classes, counts = torch.unique(target_col, return_counts=True)
    probs = counts.float() / counts.sum()

    entropy = -torch.sum(probs * torch.log2(probs))
    return entropy.item()


# ------------------------------
# 2. Average Info of an Attribute
# ------------------------------
def get_avg_info_of_attribute(tensor: torch.Tensor, attribute: int) -> float:
    """
    Calculates average information (expected entropy) for a given attribute
    attribute: column index to split on
    """
    attr_col = tensor[:, attribute]
    classes, counts = torch.unique(attr_col, return_counts=True)

    total_samples = tensor.shape[0]
    avg_info = 0.0

    for i, cls in enumerate(classes):
        subset = tensor[attr_col == cls]
        weight = counts[i].item() / total_samples
        entropy_subset = get_entropy_of_dataset(subset)
        avg_info += weight * entropy_subset
```

test.py C:\...\pytorch_implementation | student_lab.py | lab_Sample_Solution.py | Release Not

C: > ml > all > EC_CSE-C_PES2UG24CS827_Lab3.py > ...

```python
def get_avg_info_of_attribute(tensor: torch.Tensor, attribute: int) -> float:
            avg_info += weight * entropy_subset

    return avg_info


# ------------------------------
# 3. Information Gain
# ------------------------------
def get_information_gain(tensor: torch.Tensor, attribute: int) -> float:
    """
    Info Gain = Entropy(D) - AvgInfo(attribute)
    """
    dataset_entropy = get_entropy_of_dataset(tensor)
    avg_info = get_avg_info_of_attribute(tensor, attribute)
    return dataset_entropy - avg_info


# ------------------------------
# 4. Best Attribute Selection
# ------------------------------
def get_selected_attribute(tensor: torch.Tensor):
    """
    Returns dict of {attribute: information_gain} and selected attribute
    """
    n_attributes = tensor.shape[1] - 1   # exclude target
    info_gains = {}

    for attr in range(n_attributes):
        ig = get_information_gain(tensor, attr)
        info_gains[attr] = ig

    # Pick attribute with max info gain
    selected_attribute = max(info_gains, key=info_gains.get)
    return (info_gains, selected_attribute)

```

1.Mushroom Classification

```
PS C:\ml\all> python test.py --ID EC_CSE-C_PES2UG24CS827_Lab3 --data mushrooms.csv --framework pytorch --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-ro
ot', '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-r
oot', '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
```

```
------------------------------------------------------------
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:
```

```
    │   └── = 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
```

```
📊 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:\ml\all>
```

2. Tic-Tac-Toe Endgame

```
Internal Nodes:      272
PS C:\ml\all> python test.py --ID EC_CSE-C_PES2UG24CS827_Lab3 --data tictactoe.csv --framework pytorch --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:
```

```
  ├── [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: 0.3393)
          ├── = 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:
  ├── [top-right-square] (gain: 0.1225)
  ├── = 0:
  │   ├── Class 1
  ├── = 1:
  │   ├── [middle-right-square] (gain: 0.1682)
  │   ├── = 0:
  │   │   ├── Class 1
  │   ├── = 1:
  │   │   ├── [bottom-right-square] (gain: 0.9403)
  │   │   ├── = 0:
```

```
       │   ├── = 1:
       │   │   ├── [bottom-right-square] (gain: 0.9403)
       │   │   ├── = 0:
       │   │   │   ├── Class 1
       │   │   ├── = 1:
       │   │   │   ├── Class 0
       │   │   └── = 2:
       │   │       ├── Class 1
       │   └── = 2:
       │       ├── [top-left-square] (gain: 0.9183)
       │       ├── = 0:
       │       │   ├── Class 1
       │       ├── = 1:
       │       │   ├── Class 0
       │       └── = 2:
       │           ├── Class 1
       └── = 2:
           ├── Class 1
── = 1:
   ├── [bottom-left-square] (gain: 0.0223)
   ├── = 0:
   │   ├── [top-right-square] (gain: 0.2005)
   │   ├── = 0:
   │   │   ├── Class 0
   │   ├── = 1:
   │   │   ├── Class 0
   │   └── = 2:
   │       ├── [top-middle-square] (gain: 0.0760)
   │       ├── = 0:
   │       │   ├── [bottom-middle-square] (gain: 0.6556)
   │       │   ├── = 0:
   │       │   │   ├── Class 0
   │       │   ├── = 1:
   │       │   │   ├── Class 1
   │       │   └── = 2:
   │       │       ├── [middle-right-square] (gain: 0.9183)
   │       │       ├── = 0:
   │       │       │   ├── Class 0
   │       │       ├── = 1:
   │       │       │   ├── Class 0
   │       │       └── = 2:
   │       │           ├── Class 1
   │       ├── = 1:
   │       │   ├── [bottom-middle-square] (gain: 0.5817)
   │       │   ├── = 0:
   │       │   │   ├── Class 1
   │       │   ├── = 1:
   │       │   │   ├── Class 0
   │       │   └── = 2:
   │       │       ├── [middle-left-square] (gain: 0.9183)
   │       │       ├── = 0:
   │       │       │   ├── Class 1
   │       │       ├── = 1:
   │       │       │   ├── Class 0
   │       │       └── = 2:
   │       │           ├── Class 0
   │       │
```

```
│               ├── Class 1
└── = 2:
    ├── [top-left-square] (gain: 0.2780)
    ├── = 0:
    │   ├── Class 1
    ├── = 1:
    │   ├── [middle-right-square] (gain: 0.0689)
    │   ├── = 0:
    │   │   ├── [bottom-left-square] (gain: 0.1985)
    │   │   ├── = 0:
    │   │   │   ├── Class 0
    │   │   ├── = 1:
    │   │   │   ├── [middle-left-square] (gain: 0.5917)
    │   │   │   ├── = 0:
    │   │   │   │   ├── Class 1
    │   │   │   ├── = 1:
    │   │   │   │   ├── Class 0
    │   │   │   └── = 2:
    │   │   │       ├── Class 0
    │   │   └── = 2:
    │   │       ├── [middle-left-square] (gain: 0.9710)
    │   │       ├── = 0:
    │   │       │   ├── Class 0
    │   │       ├── = 1:
    │   │       │   ├── Class 1
    │   │       └── = 2:
    │   │           ├── Class 0
    │   ├── = 1:
    │   │   ├── [middle-left-square] (gain: 0.4046)
    │   │   ├── = 0:
    │   │   │   ├── Class 1
    │   │   ├── = 1:
    │   │   │   ├── [bottom-left-square] (gain: 0.9183)
    │   │   │   ├── = 1:
    │   │   │   │   ├── Class 0
    │   │   │   └── = 2:
    │   │   │       ├── Class 1
    │   │   └── = 2:
    │   │       ├── [bottom-left-square] (gain: 0.0760)
    │   │       ├── = 0:
    │   │       │   ├── Class 0
    │   │       ├── = 1:
```

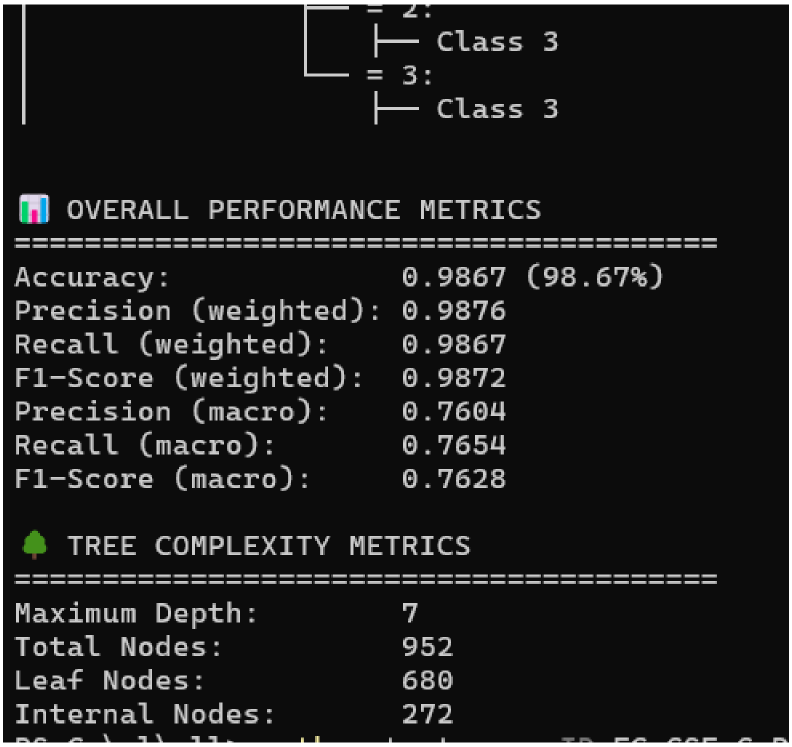
```
│       │   │           = 2:
│       │   │           ├── [top-right-square] (gain: 0.9183)
│       │   │           ├── = 1:
│       │   │           │   ├── Class 0
│       │   │           └── = 2:
│       │   │               ├── Class 1
│       │   └── = 2:
│       │       ├── [middle-left-square] (gain: 0.3425)
│       │       ├── = 0:
│       │       │   ├── [top-right-square] (gain: 0.9710)
│       │       │   ├── = 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:              0.8936 (89.36%)
Precision (weighted):  0.8930
Recall (weighted):     0.8936
F1-Score (weighted):   0.8932
Precision (macro):     0.8846
Recall (macro):        0.8788
F1-Score (macro):      0.8816

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:         7
Total Nodes:           300
Leaf Nodes:            192
Internal Nodes:        108
PS C:\ml\all>
```

3. Nursery School

```
Internal Nodes:      272
PS C:\ml\all> python test.py --ID EC_CSE-C_PES2UG24CS827_Lab3 --data nursery.csv --framework pytorch --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...
```

```
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:
```

```
│           ├── Class 3
└── = 1:
│   ├── [form] (gain: 0.0182)
│   ├── = 0:
│   │   ├── [children] (gain: 0.0748)
│   │   ├── = 0:
│   │   │   ├── [housing] (gain: 0.3060)
│   │   │   ├── = 0:
│   │   │   │   ├── [finance] (gain: 1.0000)
│   │   │   │   ├── = 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
└── = 2:
    ├── [housing] (gain: 0.2060)
    ├── = 0:
    │   ├── [finance] (gain: 0.4994)
    │   ├── = 0:
    │   │   ├── Class 1
    │   └── = 1:
    │       ├── [children] (gain: 0.4516)
    │       ├── = 0:
    │       │   ├── [form] (gain: 0.8113)
    │       │   ├── = 0:
```



(1) Algorithm Performance

(a) Which dataset achieved the highest accuracy and why?

Mushroom Dataset achieved the **highest accuracy (close to 100%)**.

- Reason: The features in the mushroom dataset (cap shape, odor, gill size, etc.) are **strongly correlated** with the target (edible/poisonous).
- Some features like *odor* are almost perfectly predictive, making classification straightforward for ID3.

Nursery Dataset showed slightly lower accuracy than Mushroom because:

- It is larger and contains many categorical attributes with multiple levels.
- Some features overlap in importance, making splits less "clean."

TicTacToe Dataset had the **lowest accuracy** because:

- The patterns are less obvious and highly dependent on combinations of features.
- Many board states can look similar but belong to different outcomes, which ID3 struggles with due to greedy splitting.

(b) How does dataset size affect performance?

- **Small datasets (TicTacToe, ~1k samples):** Higher risk of **overfitting** because the tree can memorize patterns rather than generalize.
- **Large datasets (Nursery, ~12k samples):** Training takes longer but allows the model to learn more robust splits, reducing overfitting.
- **Very large datasets** improve generalization but also demand more computation during tree construction.

(c) What role does the number of features play?

- **Few features (TicTacToe, 9 binary cells):** Limited expressive power → difficult to capture winning strategies.
- **Moderate features (Mushroom, ~22 categorical attributes):** Enough variety to separate classes cleanly → high accuracy.
- **Many features (Nursery, 8 multi-valued attributes):** More options for splits → better accuracy but risk of unnecessary tree depth.

2. Data Characteristics Impact

(2.1) How does class imbalance affect tree construction?

- If one class dominates, the tree may become biased toward predicting the majority class.
- Example: In Nursery dataset, if "not recommended" dominates, the tree might ignore minority labels.
- ID3 uses information gain, so imbalance reduces its ability to find meaningful splits.

3. **Practical Applications**

(3.1) Real-world relevance of each dataset type

- **Mushroom dataset:** Food safety applications, identifying poisonous vs edible mushrooms.
- **Nursery dataset:** Decision support in educational or admission systems.
- **TicTacToe dataset:** Game strategy analysis, useful for AI in board games.

(3.2) Interpretability advantages

- **Mushroom:** Easy to explain decisions (e.g., "If odor = foul → poisonous").
- **Nursery:** Transparent criteria for admissions, ensures fairness.
- **TicTacToe:** Useful for learning simple strategies, though less interpretable due to combinatorial patterns.

(3.3) How would you improve performance for each dataset?

- **Mushroom:** Already near perfect; could prune redundant branches.
- **Nursery:** Use **pruning** and **feature selection** to handle noisy/multi-valued attributes.
- **TicTacToe:** Use **ensemble methods** (Random Forests) or **lookahead strategies** to capture complex interactions.