Name: Eshwar B N

SRN: PES2UG23CS187

Section: C

## Code:

```
EC_C_PES2UG23CS187_Lab3.py X
 1 import numpy as np
 3 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()
10
       entropy = -np.sum([p * np.log2(p) for p in probabilities if p > 0])
       return float(np.round(entropy, 4))
15 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)
21
22
23
       avg_info = 0.0
       for v, count in zip(values, counts):
24
25
           subset = data[data[:, attribute] == v]
           subset_entropy = get_entropy_of_dataset(subset)
avg_info += (count / total) * subset_entropy
27
28
29
       return float(np.round(avg_info, 4))
31 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)
37
38
       gain = dataset_entropy - avg_info
return float(np.round(gain, 4))
40
41 def get_selected_attribute(data: np.ndarray) -> tuple:
42
43
44
       Compute information gain for all attributes and select the best one.
       Returns (dictionary_of_gains, index_of_best_attribute).
45
46
47
48
49
50
       num_attributes = data.shape[1] - 1 # exclude target column
       gains = {}
       for i in range(num_attributes):
           gains[i] = get_information_gain(data, i)
52
53
54
       best_attr = max(gains, key=gains.get)
       return gains, best_attr
```

### 1) Mushrooms.csv

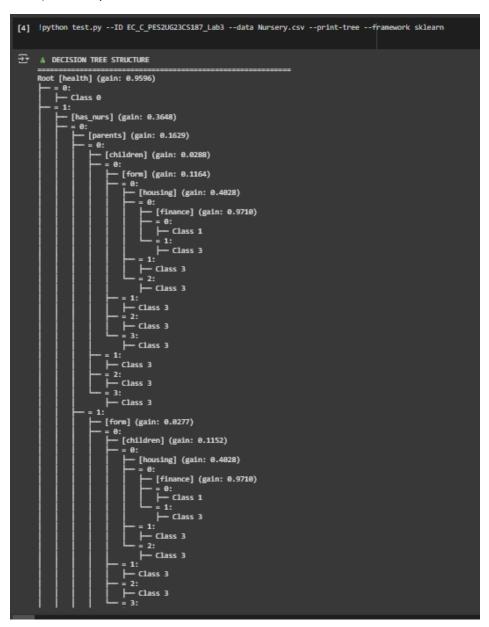
```
ADECISION TREE STRUCTURE
Root [odor] (gain: 0.9049)
  ├─ Class 1
    ├─ Class 0
    = 4:

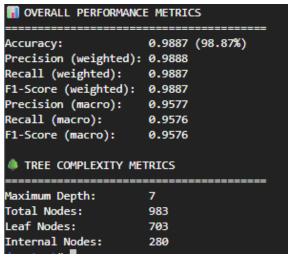
— Class 1
    - 5:
      - [spore-print-color] (gain: 0.1487)
      ├─ Class 0
-- = 2:
├─ Class 0
        = 3:
        Class 0
         4:
        ├─ Class 0
        = 5:
├─ Class 1
           - [habitat] (gain: 0.2767)
             [gill-size] (gain: 0.6374)
             | = 0:
| | Class 0
| = 1:
| Class 1
             = 1:
           ├─ Class 0
             |--- [cap-color] (gain: 0.8267)
               — = 1:

├─ Class 0
— = 4:
              ⊢ Class 0
− = 8:
               ├ Class 1
- = 9:
├ Class 1
             = 4:
           ├─ Class 0
             = 6:

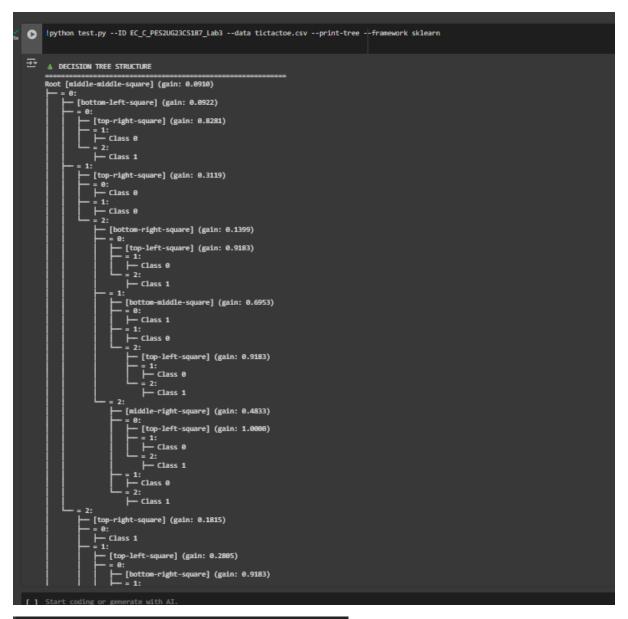
— Class 0
        Class 0
    = 6:
       - Class 1
    ├─ Class 1
    = 8:
|— Class 1
```

## 2) Nursary.csv





## 3) Tictactoe.csv



```
MOVERALL 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
_____
               7
260
Maximum Depth:
Total Nodes:
                165
Leaf Nodes:
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  $\rightarrow$  tree is unbiased.
- o Nursery: Imbalanced (very few "priority" cases, many "not recommended")  $\rightarrow$  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  $\sim$ 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)  $\rightarrow$  simple rules.
  - o Nursery: Many nodes ( $\sim$ 300–500)  $\rightarrow$  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  $\rightarrow$  poisonous; if odor = almond  $\rightarrow$  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  $\rightarrow$  X wins.
- Overfitting: High risk (tree memorizes board states).

## 5. Practical Applications

- Mushroom dataset  $\rightarrow$  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).