MACHINE LEARNING

DECISION TREE CLASSIFIER- MULTI-DATASET ANALYSIS

Chethan S [PES2UG23CS150] Section: C

1. Mushroom.csv

```
PS C:\Users\cheta\OneDrive\Desktop\Code files\ML\all\Lab3> python test.py --ID EC_C_PES2UG23CS150_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',
-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',
r-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\cheta\OneDrive\Desktop\Code files\ML\all\Lab3>
```

1. Accuracy (100.00%):
    - The model achieves perfect accuracy, correctly predicting the target class for all instances. This is an ideal result but may indicate overfitting.
2. Precision, Recall, and F1-Score (Weighted and Macro: 1.0000):
    - All metrics are perfect, meaning the model performs flawlessly across all classes, including minority classes. This suggests the model has perfectly learned the training data.

Tree Complexity Metrics

1. Maximum Depth (4):
    - The decision tree is relatively shallow, indicating a simple and interpretable model.
2. Total Nodes (29):
    - The tree has a small number of nodes, suggesting a concise decision-making process.
3. Leaf Nodes (24):
    - A high proportion of leaf nodes relative to total nodes indicates the tree makes many terminal decisions.
4. Internal Nodes (5):
    - Few internal nodes suggest the tree has minimal splits, further emphasizing simplicity.

2.tictactoe.csv

```
PS C:\Users\cheta\OneDrive\Desktop\Code files\ML\all\Lab3> python test.py --ID EC_C_PES2UG23CS150_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-squar

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-squa
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
```

Overall Performance Metrics

1. Accuracy (87.30%):
    - The model performs reasonably well, correctly predicting the target class for most instances, though there is room for improvement compared to the previous example.
2. Precision (Weighted: 0.8741, Macro: 0.8590):
    - Weighted Precision: Indicates good precision across all classes, weighted by the number of samples in each class.
    - Macro Precision: Slightly lower than weighted precision, suggesting that minority classes may have slightly lower precision.
3. Recall (Weighted: 0.8730, Macro: 0.8638):
    - Weighted Recall: High recall across all classes, showing the model captures most true positives.
    - Macro Recall: Slightly higher than macro precision, indicating the model is better at identifying true positives across all classes.
4. F1-Score (Weighted: 0.8734, Macro: 0.8613):
    - Weighted F1-Score: Balanced performance across all classes.
    - Macro F1-Score: Slightly lower, reflecting imbalances in class performance.

Tree Complexity Metrics

1. Maximum Depth (7):
    - The decision tree is moderately deep, balancing complexity and interpretability.
2. Total Nodes (281):
    - The tree has fewer nodes compared to the previous example, suggesting a simpler decision-making process.
3. Leaf Nodes (180):
    - A moderate number of leaf nodes indicates the tree has a reasonable number of terminal decisions.
4. Internal Nodes (101):
    - Internal nodes represent splits, showing the tree's complexity in dividing the dataset

3. Nursery.csv

```
PS C:\Users\cheta\OneDrive\Desktop\Code files\ML\all\Lab3> python test.py --ID EC_C_PES2UG23CS150_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
```

Overall Performance Metrics

1. Accuracy (98.67%):
    - The model performs exceptionally well, correctly predicting the target class for most instances.
2. Precision (Weighted: 0.9876, Macro: 0.7604):
    - Weighted Precision: Indicates high precision across all classes, weighted by the number of samples in each class.
    - Macro Precision: Lower than weighted precision, suggesting that minority classes may have lower precision.
3. Recall (Weighted: 0.9867, Macro: 0.7654):
    - Weighted Recall: High recall across all classes, showing the model captures most true positives.
    - Macro Recall: Slightly lower, indicating potential challenges in detecting minority classes.
4. F1-Score (Weighted: 0.9872, Macro: 0.7628):
    - Weighted F1-Score: Balanced performance across all classes.

- Macro F1-Score: Lower, reflecting imbalances in class performance.

Tree Complexity Metrics

1. Maximum Depth (7):
    - The decision tree is moderately deep, balancing complexity and interpretability.
2. Total Nodes (952):
    - The tree has a large number of nodes, indicating a detailed decision-making process.
3. Leaf Nodes (680):
    - A high number of leaf nodes suggests the tree has many terminal decisions.
4. Internal Nodes (272):
    - Internal nodes represent splits, showing the tree's complexity in dividing the dataset.

Algorithm Performance:

a.1 Which dataset achieved the highest accuracy and why?

- The dataset achieving 100% accuracy indicates that the decision tree perfectly classified all instances. This is likely due to:
    - Low complexity of the dataset: The dataset may have clear and distinct patterns, making it easy for the tree to separate classes.
    - Balanced classes: If the dataset has evenly distributed classes, the tree can learn without bias toward any particular class.
    - Optimal feature selection: The features in the dataset may be highly relevant and informative, reducing ambiguity in decision-making.

a.2 How does dataset size affect performance?

- Small datasets:
    - Tend to result in higher accuracy during training due to overfitting, as the tree can memorize patterns.
    - May not generalize well to unseen data, leading to poor performance on test datasets.
- Large datasets:
    - Provide more diverse examples, improving generalization and robustness.
    - May require deeper trees and more computational resources, potentially increasing complexity and risk of overfitting if not pruned properly.

a.3 What role does the number of features play?

- Few features:
    - Simplify tree construction and improve interpretability.
    - May limit the model's ability to capture complex relationships, reducing accuracy for datasets with intricate patterns.

b) Data Characteristics Impact

1. Class Imbalance: Imbalance biases the tree toward majority classes; mitigation techniques like oversampling help.
2. Feature Types: Binary features simplify splits; multi-valued features capture complexity but increase tree depth.

c) Practical Applications

1. Real-World Scenarios: Binary features suit fraud detection; multi-valued features work for customer segmentation.
2. Interpretability: Binary features are easier to interpret; multi-valued features provide richer insights but are complex.

Tree Visualization:

python test.py --ID CAMPUS_SECTION_SRN_Lab3 --data mushroom.csv --print-tree