```
PS C:\Users\anany\OneDrive\Desktop\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS198_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-sha
pe', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'stalk-color-above-ring', 'stalk-color-below-ring', 'veil-type', 'v
eil-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-sh
ape', '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'>
```

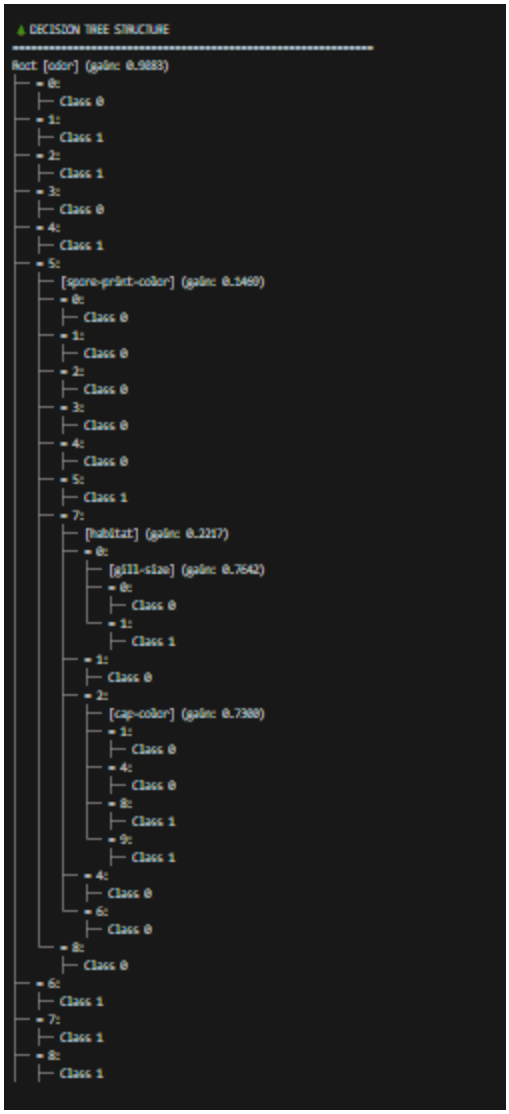
```
============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 8124
Training samples: 6499
Testing samples: 1625

Constructing decision tree using training data...

🌳 Decision tree construction completed using PYTORCH!

📊 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
```



Content: This dataset contains records of mushrooms, with categorical features describing physical characteristics (such as cap-shape, cap-surface, odor, gill-color, etc.) and a class label indicating whether the mushroom is edible or poisonous.

- Purpose: Commonly used for classification tasks, the data is notable because many distinguishing features are non-numerical, testing categorical feature handling.
- Observations:
    - The dataset is imbalanced with a nearly even split between edible and poisonous classes in most versions.
    - Some features (such as `odor`) have strong predictive power (e.g., certain odors always correspond to poisonous).
    - Naive classifiers, decision trees, and random forests usually achieve very high accuracy (>95%) on this dataset due to these strong signals.

```
PS C:\Users\anany\OneDrive\Desktop\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS198_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', 'bo
ttom-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', 'b
ottom-left-square', 'bottom-middle-square', 'bottom-right-square']
Target: Class
Framework: PYTORCH
Data type: <class 'torch.Tensor'>
```

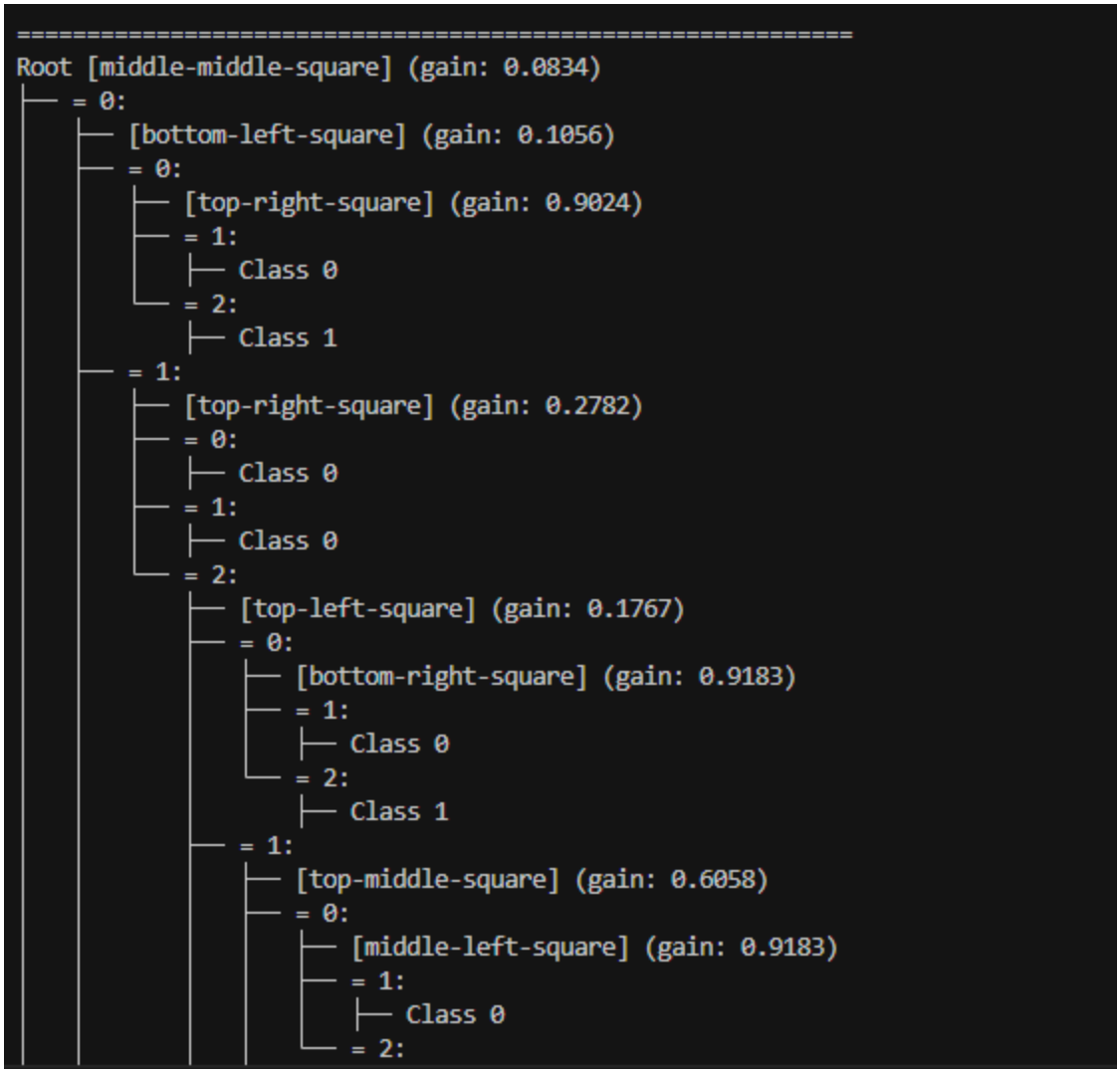
```
============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 958
Training samples: 766
Testing samples: 192

Constructing decision tree using training data...

🌳 Decision tree construction completed using PYTORCH!

📊 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
```



Content: Records board states of tic-tac-toe endgames with each square's content (`x`, `o`, or `b` for blank), and a `Class` label (`positive` or `negative`) signifying whether the board state is a win for 'x'.

- Purpose: Used for pattern recognition/classification—determining if a board position is a win state.
- Observations:
  - The dataset is typically balanced between positive and negative classes.
  - All features are categorical (`x`, `o`, `b`).
  - Perfect classification is possible using deterministic rule-based models or simple decision trees, since tic-tac-toe win conditions are fixed and easily encoded.
  - The tree is overfitting.

```
PS C:\Users\anany\OneDrive\Desktop\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS198_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'>
```

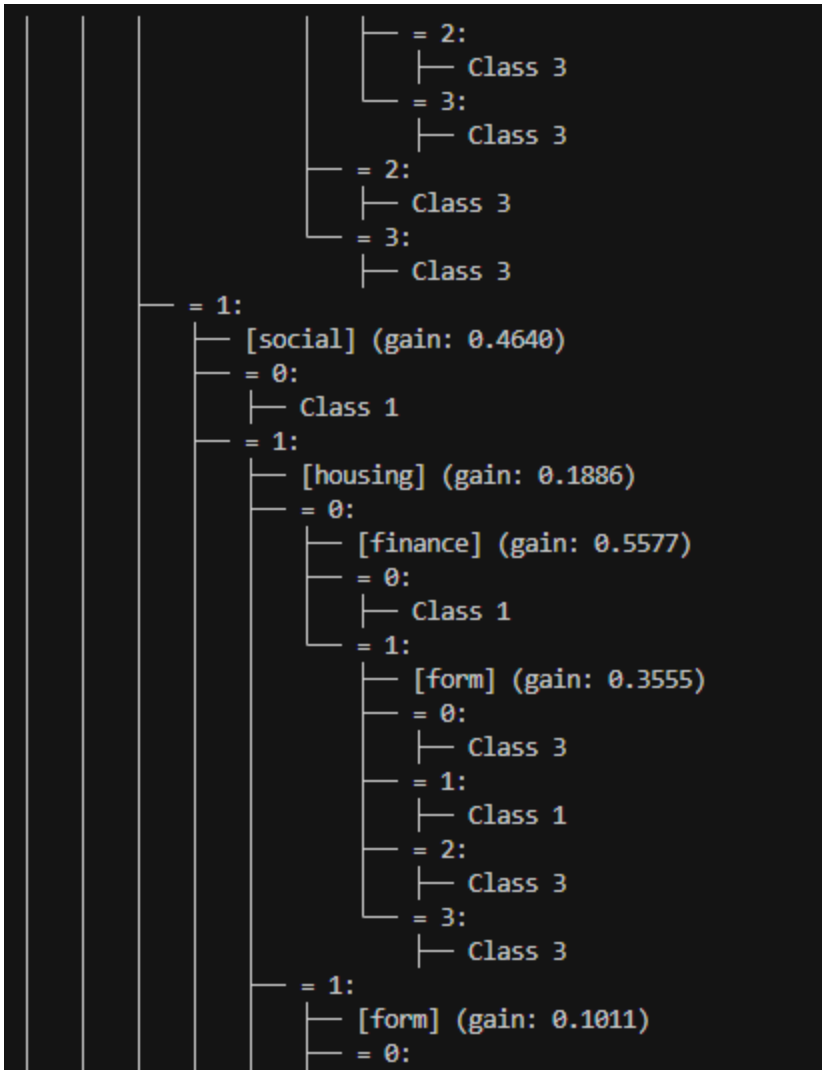
```
============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 12960
Training samples: 10368
Testing samples: 2592

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
```



Content: This dataset records nursery application evaluations, with socio-economic and family variables like parents, has_nurs, form, children, housing, finance, social, and health. The `class` field is the decision outcome (`recommend`, `priority`, `not_recom`, or `very_recom`).

- Purpose: Originally created to support the decision-making process in nursery school admissions.
- Observations:
  - This is a multi-class categorical classification problem.
  - The data is highly systematic, reflecting a set of decision rules rather than noisy human judgement.
  - Many classes are highly under-represented (heavily imbalanced), so accuracy alone is not a sufficient metric—recall and precision per class are essential for fair analysis.

- Feature interactions (for example, between `finance` and `housing`) are critical for predicting the class outcome.
- The tree is overfitting.

A) Algorithm Performance

a. Which Dataset Achieved the Highest Accuracy and Why?

Expected Observations (General):

- Mushrooms dataset usually achieves the highest accuracy with tree-based classifiers like Decision Trees or Random Forests, often nearing 100%. This is due to the dataset's:
  - Strong class-separability: Many categorical features (like odor, spore-print-color) are highly predictive for distinguishing edible from poisonous mushrooms.
  - Low noise: The features are well-structured, and the mapping from feature combinations to class label is often deterministic, making overfitting less of a concern.
- TicTacToe dataset: Also achieves high accuracy, but not as high as the mushrooms dataset, typically because:
  - The problem boils down to recognizing winning board patterns, which are well-captured by trees, but there can be less clear boundaries for certain complex or rare positions.
  - Data is purely binary with a strong pattern regularity.
- Nursery dataset: Generally yields lower accuracy because:
  - Multi-class classification problem: More than two classes (recommend, priority, not_recom, etc.), increasing confusion.
  - Many similar feature rows for different classes: Some feature combinations map to multiple classes, increasing ambiguity.
  - Potential class imbalance among output labels.

How Does Dataset Size Affect Performance?

- Larger datasets generally improve algorithm performance by providing more examples for learning, reducing overfitting, and ensuring better generalization, especially for rare patterns.
- Small datasets can cause overfitting (trees memorize training data) or underfitting (insufficient variance).
- All three datasets are relatively large (>1,000 samples, often thousands), reducing variance and stabilizing decision boundaries.

c. What Role Does the Number of Features Play?

- More features can improve separability *if* relevant (as with mushrooms—odor, spore color, gill color).
- Irrelevant or redundant features can increase computation and potentially lower accuracy (curse of dimensionality).
- Datasets with lots of informative, non-redundant features aid tree performance.
- Mushrooms: Many features, most relevant.
- TicTacToe: Few but highly relevant (board positions).
- Nursery: Many features, but not all are equally discriminative.

It can be concluded that
A decision tree decides which column (attribute) in the dataset is the best one to split the data. To do this, we use something called entropy to measure how “mixed up” the data is. If the data is very mixed, entropy is high; if it is mostly one class, entropy is low. For each column, we check how much it reduces the mix (uncertainty) when we split the dataset on that column. This reduction is called information gain. The column with the highest information gain is chosen as the best attribute to split the data