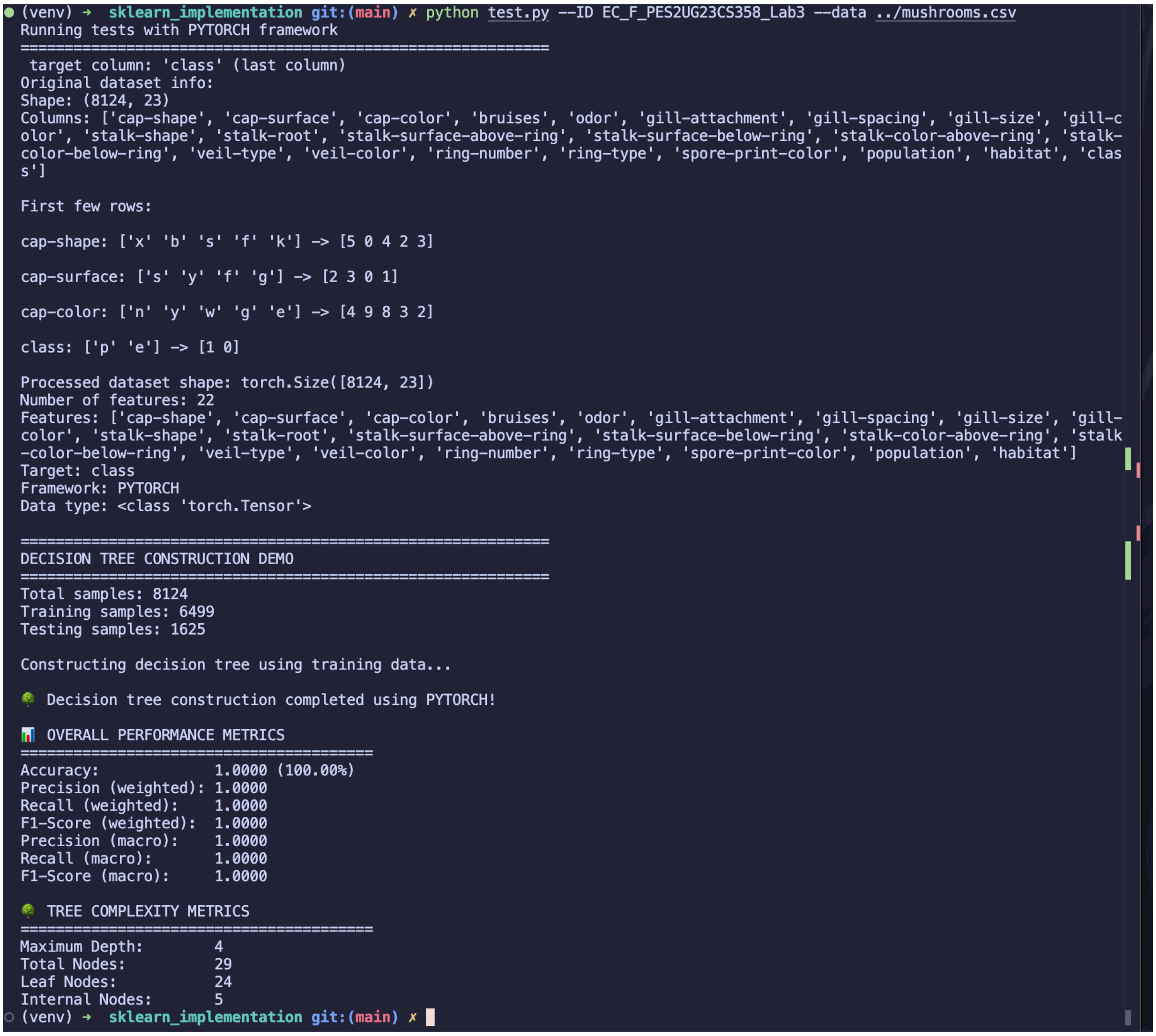
MACHINE LEARNING

DECISION TREE CLASSIFIER MULTI-DATASET ANALYSIS

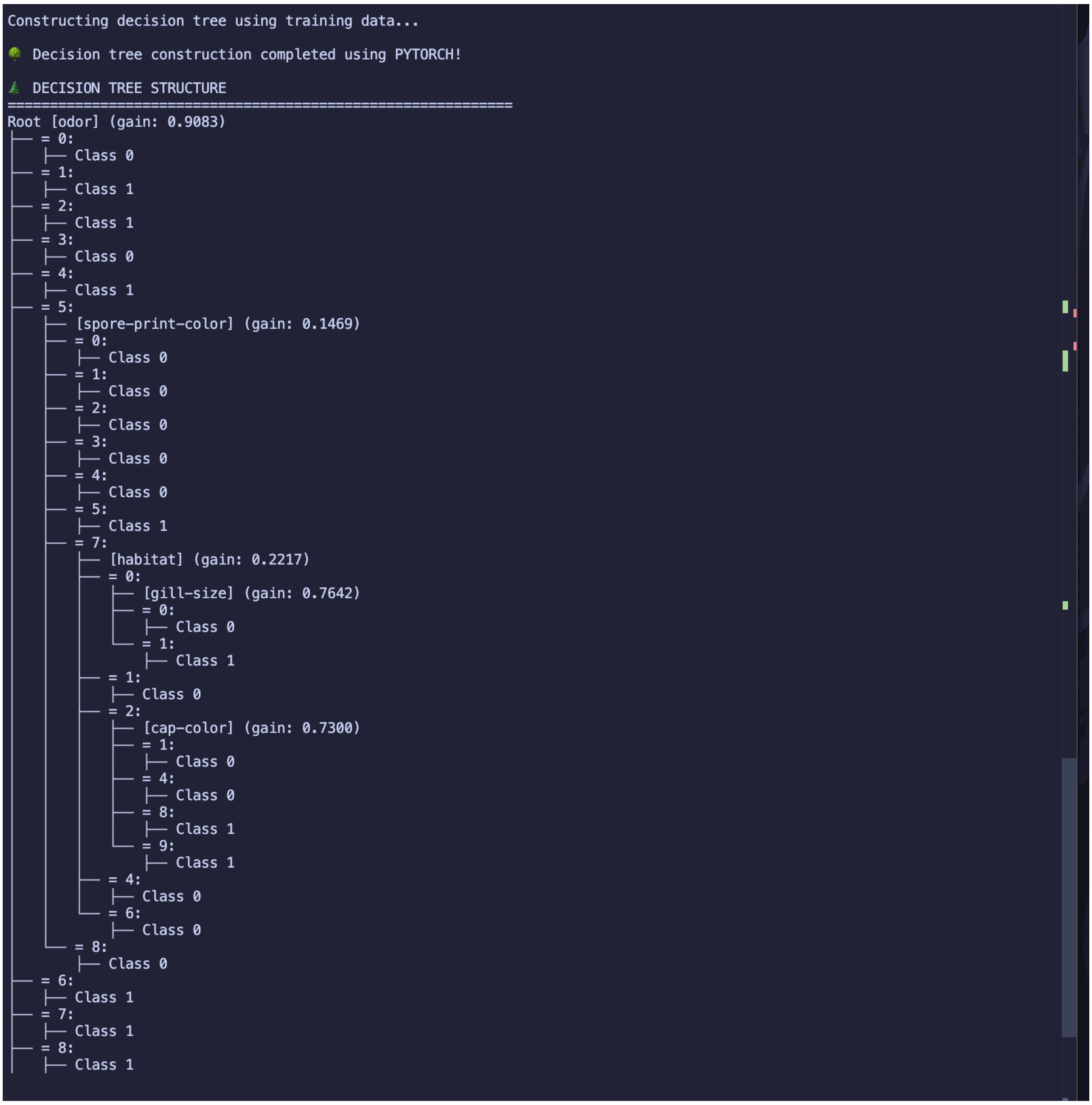
NAME: NAGULA ANISH

SECTION: F
SRN: PES2UG23CS358

1. Mushrooms.csv



```
● (venv) ➜  sklearn_implementation git:(main) ✗ python test.py --ID EC_F_PES2UG23CS358_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-c
  olor', 'stalk-shape', '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', 'clas
  s']

  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', '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
○ (venv) ➜  sklearn_implementation git:(main) ✗ 
```



```
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:
│   │   │   │   ├── 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
```

1. **Performance Comparison**

- Accuracy obtained is := 100%
  - This tells us that the dataset is perfectly-fit and clean.
- Precision (Weighted & Macro) := 1.000
  - This means that 100% of the predictions made by the model were correct.
- Recall (Weighted & Macro) := 1.000

    - The model had made all correct predictions which meant it found all the poisonous mushrooms.

- `F1 Score` (Weighted & Macro) := 1.000
    - It is 1.000 as it is the `Harmonic Mean` of the `precision` and `recall`.

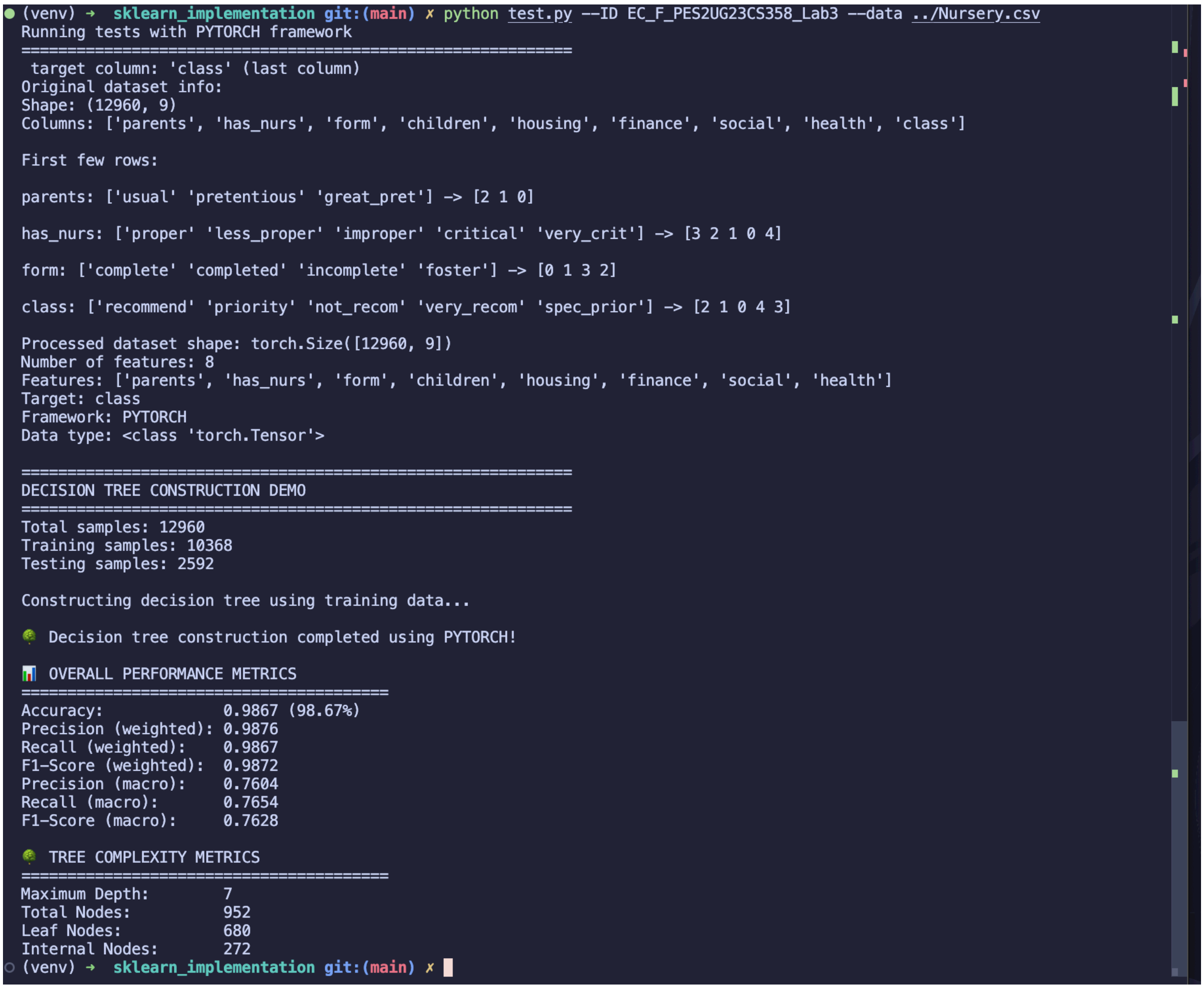
2. Tree Characteristics Analysis

- Tree Depth (Maximum-Depth) := 4
- Number Of Nodes (Total nodes) := 29 (Leaf := 24, Internal := 5).
- The attribute `Odor` is selected as the root and then to `spore-print-color` and then to `habitat`.
- As its depth is 4 its considered as a *shallow-tree* and also its complexity is *Low.*

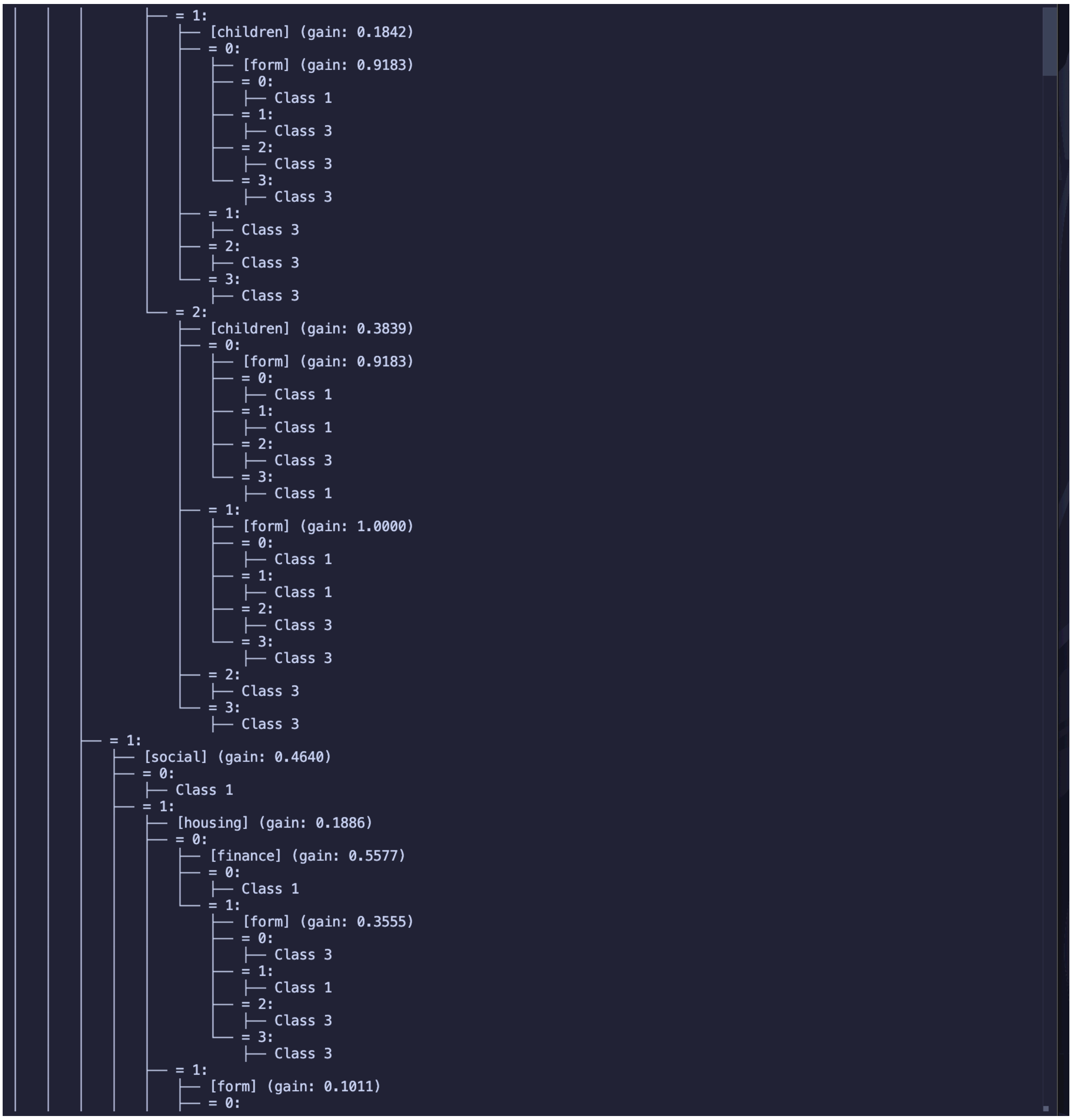
3. Dataset-Specific Insights

- `Odor` attribute contributes the most to classification.
- Balanced.
- No over-fitting.

2. Nursery.csv



```
● (venv) ➜  sklearn_implementation git:(main) ✗ python test.py --ID EC_F_PES2UG23CS358_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
○ (venv) ➜  sklearn_implementation git:(main) ✗ 
```



```
│   │   │       ├── = 1:
│   │   │       │   ├── [children] (gain: 0.1842)
│   │   │       │   ├── = 0:
│   │   │       │   │   ├── [form] (gain: 0.9183)
│   │   │       │   │   ├── = 0:
│   │   │       │   │   │   ├── Class 1
│   │   │       │   │   ├── = 1:
│   │   │       │   │   │   ├── Class 3
│   │   │       │   │   ├── = 2:
│   │   │       │   │   │   ├── Class 3
│   │   │       │   │   └── = 3:
│   │   │       │   │       ├── Class 3
│   │   │       │   ├── = 1:
│   │   │       │   │   ├── Class 3
│   │   │       │   ├── = 2:
│   │   │       │   │   ├── Class 3
│   │   │       │   └── = 3:
│   │   │       │       ├── Class 3
│   │   │       └── = 2:
│   │   │           ├── [children] (gain: 0.3839)
│   │   │           ├── = 0:
│   │   │           │   ├── [form] (gain: 0.9183)
│   │   │           │   ├── = 0:
│   │   │           │   │   ├── Class 1
│   │   │           │   ├── = 1:
│   │   │           │   │   ├── Class 1
│   │   │           │   ├── = 2:
│   │   │           │   │   ├── Class 3
│   │   │           │   └── = 3:
│   │   │           │       ├── Class 1
│   │   │           ├── = 1:
│   │   │           │   ├── [form] (gain: 1.0000)
│   │   │           │   ├── = 0:
│   │   │           │   │   ├── Class 1
│   │   │           │   ├── = 1:
│   │   │           │   │   ├── Class 1
│   │   │           │   ├── = 2:
│   │   │           │   │   ├── Class 3
│   │   │           │   └── = 3:
│   │   │           │       ├── Class 3
│   │   │           ├── = 2:
│   │   │           │   ├── Class 3
│   │   │           └── = 3:
│   │   │               ├── Class 3
│   │   ├── = 1:
│   │   │   ├── [social] (gain: 0.4640)
│   │   │   ├── = 0:
│   │   │   │   ├── Class 1
│   │   │   ├── = 1:
│   │   │   │   ├── [housing] (gain: 0.1886)
│   │   │   │   ├── = 0:
│   │   │   │   │   ├── [finance] (gain: 0.5577)
│   │   │   │   │   ├── = 0:
│   │   │   │   │   │   ├── Class 1
│   │   │   │   │   └── = 1:
│   │   │   │   │       ├── [form] (gain: 0.3555)
│   │   │   │   │       ├── = 0:
│   │   │   │   │       │   ├── Class 3
│   │   │   │   │       ├── = 1:
│   │   │   │   │       │   ├── Class 1
│   │   │   │   │       ├── = 2:
│   │   │   │   │       │   ├── Class 3
│   │   │   │   │       └── = 3:
│   │   │   │   │           ├── Class 3
│   │   │   │   ├── = 1:
│   │   │   │   │   ├── [form] (gain: 0.1011)
│   │   │   │   │   ├── = 0:
```

1. **Performance Comparison**

- Accuracy obtained is := 98.67%
    - This tells us that the dataset is almost perfectly-fit and clean and has a little bit of noise present in it.
- Precision (Weighted & Macro) := 0.9876 & 0.7604
    - 0.9876 tells us that almost all of the predictions made by the model were correct but has made a few miss-predictions.

    - 0.7604 tells us that the minority classes have lower precision.
- `Recall` (Weighted & Macro) := 0.9867 & 0.7654
    - 0.9867 tells us that almost all of the positive values were found by the model.
    - 0.7654 tells us that the model finds it difficult to detect the minority classes.
- `F1 Score` (Weighted & Macro) := 0.9872 & 0.7628
    - 0.9872 is a relatively high score so has balanced performance
    - 0.7628 is comparatively lower score so lesser balanced performance.

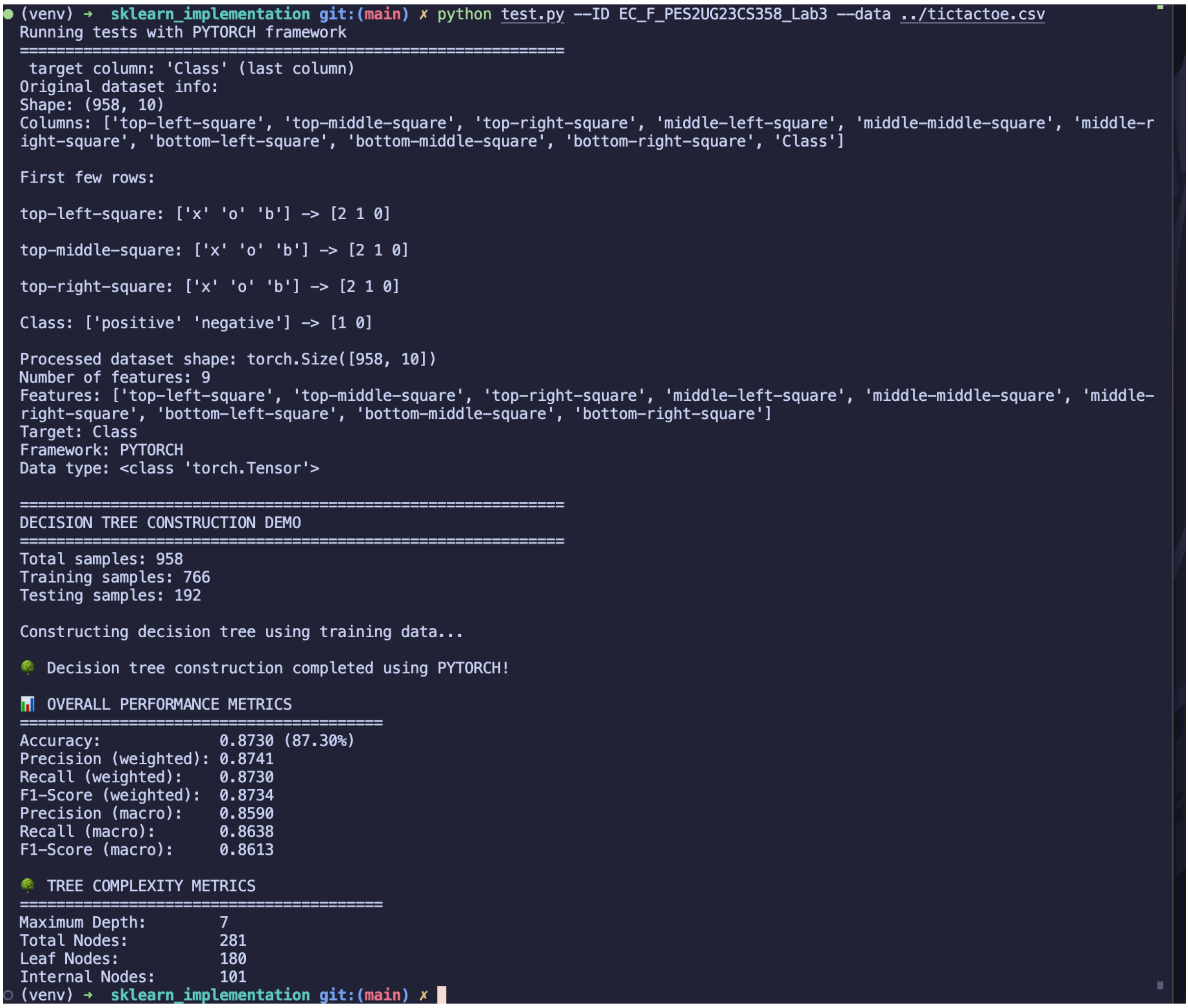
2. **Tree Characteristics Analysis**

- Tree Depth (Maximum-Depth) := 7
- Number Of Nodes (Total nodes) := 952 (Leaf := 680, Internal := 272).
- Early splits on `finance` / `social` / `health`.
- As its depth is 7 and has a huge no.of leaf and internal nodes, its complexity is *High.*

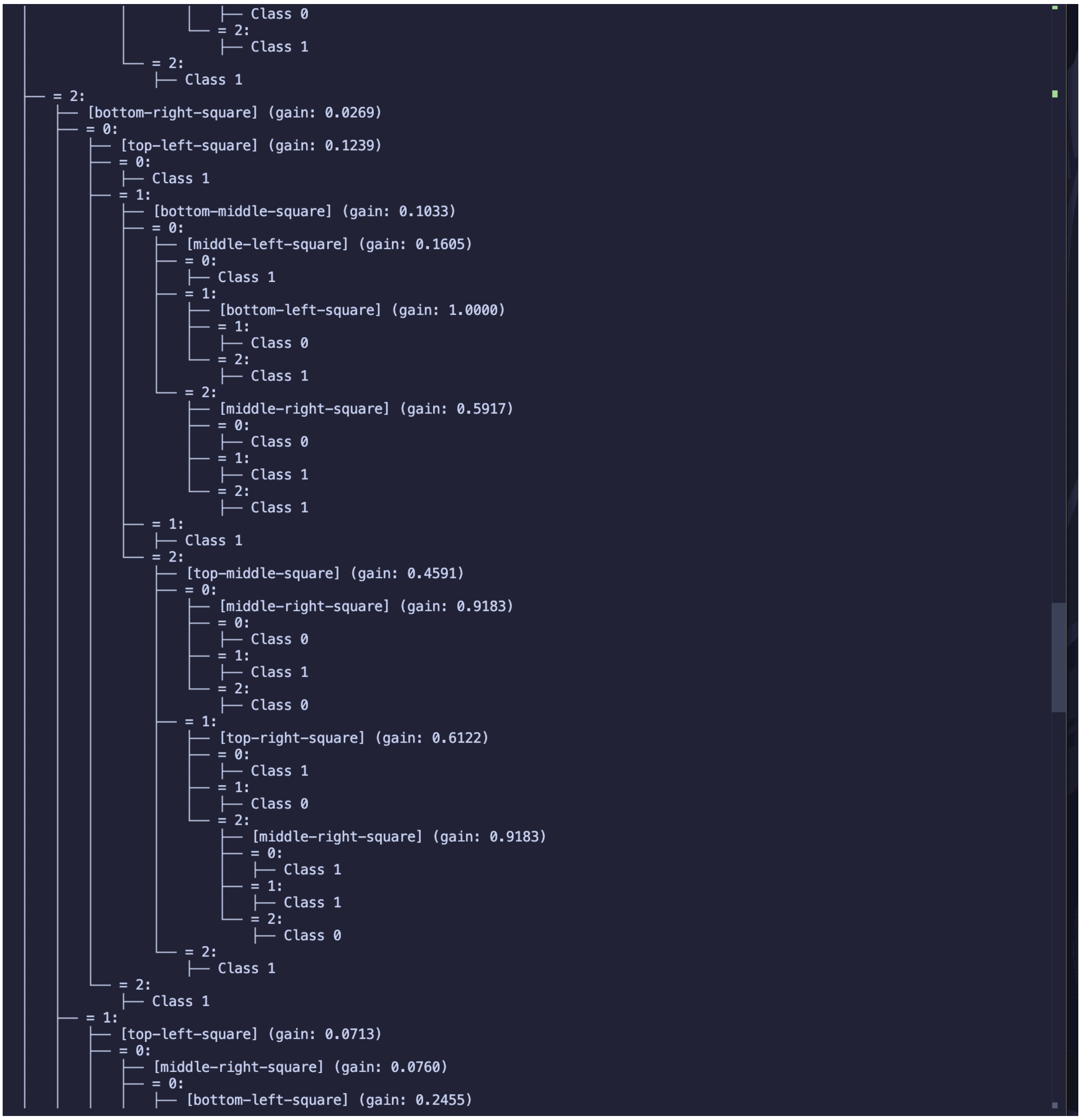
3. **Dataset-Specific Insights**

- `Finance`, `Social`, `Health` attribute contributes the most to classification.
- Unbalanced.
- Over-fitting.

3. TicTacToe.csv



```
● (venv) ➜  sklearn_implementation git:(main) ✗ python test.py --ID EC_F_PES2UG23CS358_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-r
  ight-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!

  📊 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
○ (venv) ➜  sklearn_implementation git:(main) ✗ 
```



```
│           │       │   ├── Class 0
│           │       └── = 2:
│           │           ├── Class 1
│           └── = 2:
│               ├── Class 1
├── = 2:
│   ├── [bottom-right-square] (gain: 0.0269)
│   ├── = 0:
│   │   ├── [top-left-square] (gain: 0.1239)
│   │   ├── = 0:
│   │   │   ├── Class 1
│   │   ├── = 1:
│   │   │   ├── [bottom-middle-square] (gain: 0.1033)
│   │   │   ├── = 0:
│   │   │   │   ├── [middle-left-square] (gain: 0.1605)
│   │   │   │   ├── = 0:
│   │   │   │   │   ├── Class 1
│   │   │   │   ├── = 1:
│   │   │   │   │   ├── [bottom-left-square] (gain: 1.0000)
│   │   │   │   │   ├── = 1:
│   │   │   │   │   │   ├── Class 0
│   │   │   │   │   └── = 2:
│   │   │   │   │       ├── Class 1
│   │   │   │   └── = 2:
│   │   │   │       ├── [middle-right-square] (gain: 0.5917)
│   │   │   │       ├── = 0:
│   │   │   │       │   ├── Class 0
│   │   │   │       ├── = 1:
│   │   │   │       │   ├── Class 1
│   │   │   │       └── = 2:
│   │   │   │           ├── Class 1
│   │   │   ├── = 1:
│   │   │   │   ├── Class 1
│   │   │   └── = 2:
│   │   │       ├── [top-middle-square] (gain: 0.4591)
│   │   │       ├── = 0:
│   │   │       │   ├── [middle-right-square] (gain: 0.9183)
│   │   │       │   ├── = 0:
│   │   │       │   │   ├── Class 0
│   │   │       │   ├── = 1:
│   │   │       │   │   ├── Class 1
│   │   │       │   └── = 2:
│   │   │       │       ├── Class 0
│   │   │       ├── = 1:
│   │   │       │   ├── [top-right-square] (gain: 0.6122)
│   │   │       │   ├── = 0:
│   │   │       │   │   ├── Class 1
│   │   │       │   ├── = 1:
│   │   │       │   │   ├── Class 0
│   │   │       │   └── = 2:
│   │   │       │       ├── [middle-right-square] (gain: 0.9183)
│   │   │       │       ├── = 0:
│   │   │       │       │   ├── Class 1
│   │   │       │       ├── = 1:
│   │   │       │       │   ├── Class 1
│   │   │       │       └── = 2:
│   │   │       │           ├── Class 0
│   │   │       └── = 2:
│   │   │           ├── Class 1
│   │   └── = 2:
│   │       ├── Class 1
│   ├── = 1:
│   │   ├── [top-left-square] (gain: 0.0713)
│   │   ├── = 0:
│   │   │   ├── [middle-right-square] (gain: 0.0760)
│   │   │   ├── = 0:
│   │   │   │   ├── [bottom-left-square] (gain: 0.2455)
```

1. Performance Comparison

- Accuracy obtained is := 87.30%
  - This tells us that the dataset is not clean as the other two and that its accuracy can be improved.
- Precision (Weighted & Macro) := 0.8741 & 0.8590
  - 0.8741 tells us that most of the predictions made by the model were correct but lesser than the other datasets.

  - 0.8590 tells us that the minority classes have lower precision.
- `Recall` (Weighted & Macro) := 0.8730 & 0.8638
  - 0.8730 tells us that most of the positive values were found by the model.
  - 0.8638 tells us that the model finds true positives across all the minority classes.
- `F1 Score` (Weighted & Macro) := 0.8734 & 0.8613
  - 0.8734 is a relatively high score so has balanced performance.
  - 0.8613 is almost same so balanced.

2. **Tree Characteristics Analysis**

- Tree Depth (Maximum-Depth) := 7
- Number Of Nodes (Total nodes) := 281 (Leaf := 180, Internal := 101).
- Early splits on `midlle-middle-square` .
- As its depth is 7 and has a huge no.of leaf and internal nodes but lesser than previous one, so its complexity is *Medium.*

3. **Dataset-Specific Insights**

- `Center-Square` attribute contributes the most to classification.
- Slightly Unbalanced.
- Might be over-fitting.

4. Comparative Analysis Report

1. **Algorithm Performance :=**

    a. The **Mushroom dataset** had the highest accuracy. The `odor` feature made it easy to classify.

    b. Larger dataset increases the number of test cases to be run and hence the accuracy improves, where as the smaller datasets may result in lower accuracy.

    c. Feature quality is more important than quantity. Good features help distinguish between the classes.

2. **Data Characteristics Impact :=**

    - Class imbalance decreases the overall F1 score, for example in nursery weighted values
      show good scores whereas macro scores are poor.
    - The algorithm works well with multi-valued features. It performs best when feature values predict the class.

3. Real World Scenarios :=

- Mushrooms → Good for clear yes/no problems (food safety, spam).
- Nursery → Models complex decisions (university admissions).
- Tic-Tac-Toe → Represents rule-based systems (game AI, network security).
- How to Improve :=
    - **Mushroom:** No changes needed. It works perfectly.
    - **Nursery:** Fix the class imbalance and then, use a stronger algorithm.
    - **Tic-Tac-Toe:** Use a different algorithm for a slightly better score.