UE23CS352A : Machine Learning

Lab 3 – Decision Tree Classifier – Multi – Dataset Analysis

Navya G N	PES2UG23CS372	Section F

Testing and Visualization Basic Testing:

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

python test.py --ID CAMPUS_SECTION_SRN_Lab3 --data tictactoe.csv

python test.py --ID CAMPUS_SECTION_SRN_Lab3 --data nursery.csv

Tree Visualization:

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

Analysis Requirements

1. Performance Comparison

Compare the following metrics across all three datasets:

•Accuracy: Overall classification accuracy

•Precision: True positives / (True positives + False positives)

•Recall: True positives / (True positives + False negatives

•F1-Score: Harmonic mean of precision and recall

	Mushroom.csv	Nursery.csv	Tictactoe.csv
Accuracy	100%	98.67%	87.30%
Precision(weighted)	1.0000	0.9876	0.8741
Recall(weighted)	1.0000	0.9867	0.8730
F1-Score(weighted)	1.0000	0.9872	0.8734

2.Tree Characteristics Analysis: Analyze and compare:

• Tree Depth: Maximum depth of the constructed trees

• Number of Nodes: Total nodes in each tree

	Mushroom.csv	Nursery.csv	Tictactoe.csv
Tree Depth	4	7	7
Number of Nodes	29	952	281

Analysis :

Mushroom.csv

- Very shallow tree compared to others.
- Mushroom dataset is relatively simple: the class (edible/poisonous) can be determined using just a few key features.
- High accuracy (100%) confirms that only a handful of splits are enough to fully separate classes.
- Features selected at the top are very discriminative (like odor, spore-print-color).

Nursery.csv

- This dataset produces a very large and complex tree.
- Nursery dataset has many categorical attributes with multiple values, which leads to many splits and a bushy tree.
- Depth 7 means decisions require more attributes to classify a sample correctly.
- The tree is more prone to overfitting, but accuracy is still very high (98.67%).
- This shows that while complex, the dataset has strong attribute–class relationships that the tree captures.

Tictactoe.csv

- Tree depth is same as Nursery (7), but with fewer nodes (281 vs 952).
- The tic-tac-toe dataset is less complex in terms of branching because attributes have fewer possible values (x, o, b).
- Accuracy is lower (87.3%), showing that even though the tree grows deep, the attribute combinations are not always sufficient for perfect classification (game outcomes can be subtle and overlapping).

• Most Important Features: Attributes selected as root and early splits

Mushroom.csv

- Root Attribute: odor (gain: 0.9083)
  - This is the most important feature: immediately separates mushrooms into edible/poisonous groups.
  - Many branches from odor directly lead to leaf nodes (Class 0 or Class 1).
- Second-Level Attribute: spore-print-color (gain: 0.1469)
  - When odor alone isn't sufficient (e.g., odor = 5), spore-print-color becomes the next best splitter.
- Third-Level Attributes:
  - habitat (gain: 0.2217)
  - gill-size (gain: 0.7642)

- cap-color (gain: 0.7300)

Analysis: The early splits show that odor is dominant, but when odor is less discriminative (certain values), the model relies on a combination of spore-print-color, habitat, gill-size, and cap-color.

Nursery.csv

- The root feature (health) dominates decisions: children with poor health are often rejected early.
- Socio-economic attributes (finance, housing, parents) appear at upper levels, showing their strong influence.
- health (gain: 0.9595)
- has_nurs (gain: 0.3555)
- parents (gain: 0.1673)
- form (gain: 0.0171), children (gain: 0.0662), housing (gain: 0.2401), and finance (gain: 0.9710)

Tictactoe.csv

- Root node :middle-middle-square (gain: 0.0834) → This is the center square of the board, which is critical in TicTacToe strategy.
- First Few Splits:
  - bottom-left-square (gain: 0.1056)
  - top-right-square (gain: 0.9024 in one branch)
  - top-left-square, bottom-right-square, and top-middle-square also appear very early

Analysis:

- The root being the center square matches human intuition: controlling the center is the strongest move in TicTacToe.
- Corners (top-right-square, bottom-left-square, top-left-square) appear next, reflecting their high strategic importance.
- Edge squares (top-middle-square, middle-left-square, etc.) show up deeper, fine-tuning win/loss prediction.

• Tree Complexity: Relationship between tree size and dataset characteristics

Dataset	Tree Complexity	Dataset Characteristics	Relationship
Mushroom.csv	Small depth,compact	Few highly discriminative features (odor, spore-print-color)	Combinatorial nature → very large tree
Nursery.csv	Medium to large depth,wide branching	Multi-class labels, moderately correlated features	Weaker features + multi-class → larger tree
Tictactoe.csv	Very large, deep, repetitive splits	9 attributes, 3 values each → 19,683 states	Combinatorial nature → very large tree

3. Dataset-Specific Insights For each dataset, analyze:

• Feature Importance: Which attributes contribute most to classification

• Class Distribution: How balanced are the target classes

• Decision Patterns: Common decision paths in the tree

• Overfitting Indicators: Signs of overfitting in tree structure

Mushroom.csv

- Feature Importance
  - Attributes like odor, spore-print-color, and gill-size dominate early splits.
  - Especially odor is the most critical root node in most decision trees (a strong indicator of edibility).

• Class Distribution

  - Very balanced dataset (edible vs poisonous mushrooms are nearly equal).
  - Balanced distribution → tree learns meaningful splits without bias.

• Decision Patterns

  - Simple patterns:
    - If odor = foul → poisonous.
    - If odor = none → check other attributes like gill-size, spore-print-color.
  - Many leaves correspond directly to clear edible/poisonous decisions.

• Overfitting Indicators

- Low risk of overfitting because the dataset has strongly predictive attributes (like odor).
- Even shallow trees achieve high accuracy (>99%)

tictactoe.csv

• Feature Importance

- Root splits often start with top-left or center squares.
- Early splits focus on positions critical to game outcome (center & corners).

• Class Distribution

- Slightly imbalanced depending on win/loss distribution.
- More "loss" outcomes than "win" because there are many ways to lose but fewer optimal strategies to win.

• Decision Patterns

- If center = x → likely win.
- If center = o and top-left = x → check next corners.
- Decision paths resemble logical "if-else" game strategies.

• Overfitting Indicators

- High risk of overfitting:
  - Many unique board configurations exist.
  - Tree may grow deep to cover every possible state.
- Generalization limited; pruning helps control complexity.

Nursery.csv

Feature Importance

- Root often splits on parents' occupation, financial standing, or number of children.
- Early splits focus on socio-economic conditions.

• Class Distribution

- Very imbalanced:

- Some classes like *recommend* or *priority* dominate.
- Rare classes (*very_recom*) appear less often, making it harder to classify.

• Decision Patterns

- Example:
  - If parents = great_pret → priority.
  - If parents = pretentious and financial = convenient → recommend.
- Reflects rules based on family/social structure.

• Overfitting Indicators

- Large number of categorical attributes + many levels per attribute → tree grows very large.
- Higher chance of overfitting rare classes (e.g., "not_recom" vs "priority").
- Needs pruning or max-depth tuning for better generalization.

4. Comparative Analysis Report

Write a comprehensive report addressing:

a) Algorithm Performance:

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

Mushroom dataset has the highest accuracy -100% because a few attributes like odor has almost perfectly separate classes. The tree can make correct decisions with 1–2 splits.

b. • How does dataset size affect performance?

- A larger dataset like Nursery helps learn more patterns but also encourages deeper, bushier trees, increasing variance unless you regularize.
- TicTacToe is smaller, so performance depends more on how well the tree captures combinational interactions than on sheer data volume.
- Mushroom shows that feature quality matters more than size: even a moderate dataset attains near-perfect accuracy when one feature is highly discriminative.

c. • What role does the number of features play?

- Mushroom has many attributes, but just one or two (e.g., odor, later spore-print-color/gill-size) dominate, yielding shallow, accurate trees.
- TicTacToe has nine board squares but each is tri-valued and their interactions drive the label, so the tree repeats tests and grows deep.
- Nursery has a moderate number of features but high cardinality per feature and five classes, which together force wide branching and deeper paths.

b)Data Characteristics Impact:

• How does class imbalance affect tree construction?

- Nursery is skewed across five classes, so the tree tends to favor majority classes and creates long, narrow branches to carve out minority classes.
- TicTacToe is closer to balanced but still has asymmetric patterns (more ways to fail than to win), which can bias some leaves.
- Mushroom is roughly balanced, letting the tree form clean high-purity splits without strong majority-class bias.

•Which types of features (binary vs multi-valued) work better?

- Binary or low-cardinality features usually yield simpler, more generalizable trees because each split doesn't fragment the data too much.
- Multi-valued features (common in Nursery and all TicTacToe squares with 3 values) create many child nodes, fragmenting data and increasing overfitting risk unless pruned.
- Mushroom is the exception where a multi-valued but extremely informative feature (odor) behaves almost like a perfect binary test between classes.

c)Practical Applications:

• For which real-world scenarios is each dataset type most relevant?

- Mushroom-like problems suit domains with a dominant biomarker or red-flag signal (e.g., toxicity screening, fraud rules with a single strong indicator, industrial QA with a critical test).
- TicTacToe-like problems represent rule-based, interaction-heavy decision-making, such as move selection in games, troubleshooting flows, or policy engines with conditional logic.

- Nursery-like problems map to eligibility and prioritization tasks (e.g., benefits allocation, admissions, triage) where multiple socio-economic factors must be weighed together.

•What are the interpretability advantages for each domain?

- Mushroom offers very high interpretability, since a short path like “if odor = foul → poisonous” is easy to trust and audit.
- TicTacToe paths resemble human-readable strategies (check center, then corners, then responses), making them understandable albeit longer.
- Nursery reveals policy logic chains (e.g., *health → has_nurs → parents → housing/finance*), which is useful for governance, though the overall tree can be large and cognitively heavy.

• How would you improve performance for each dataset?

- Mushroom: Limit tree depth and use pruning to avoid overfitting; accuracy is already very high.
- TicTacToe: Add strategy-based features, prune the tree, or try Random Forest/Boosting for better capture of patterns.
- Nursery: Handle class imbalance with weights or resampling, reduce rare categories, and use pruning or boosting methods for better performance.

<u>Testing and Visualization Basic Testing:</u>

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

python test.py --ID CAMPUS_SECTION_SRN_Lab3 --data tictactoe.csv

python test.py --ID CAMPUS_SECTION_SRN_Lab3 --data nursery.csv

```
PS C:\Users\Lenovo\OneDrive\Desktop\Sem V\ML\LAB\Week 3 - Decision Trees\pytorch_implementation> python test.py --ID EC_F_PES2UG23
CS372_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-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
PS C:\Users\Lenovo\OneDrive\Desktop\Sem V\ML\LAB\Week 3 - Decis
```

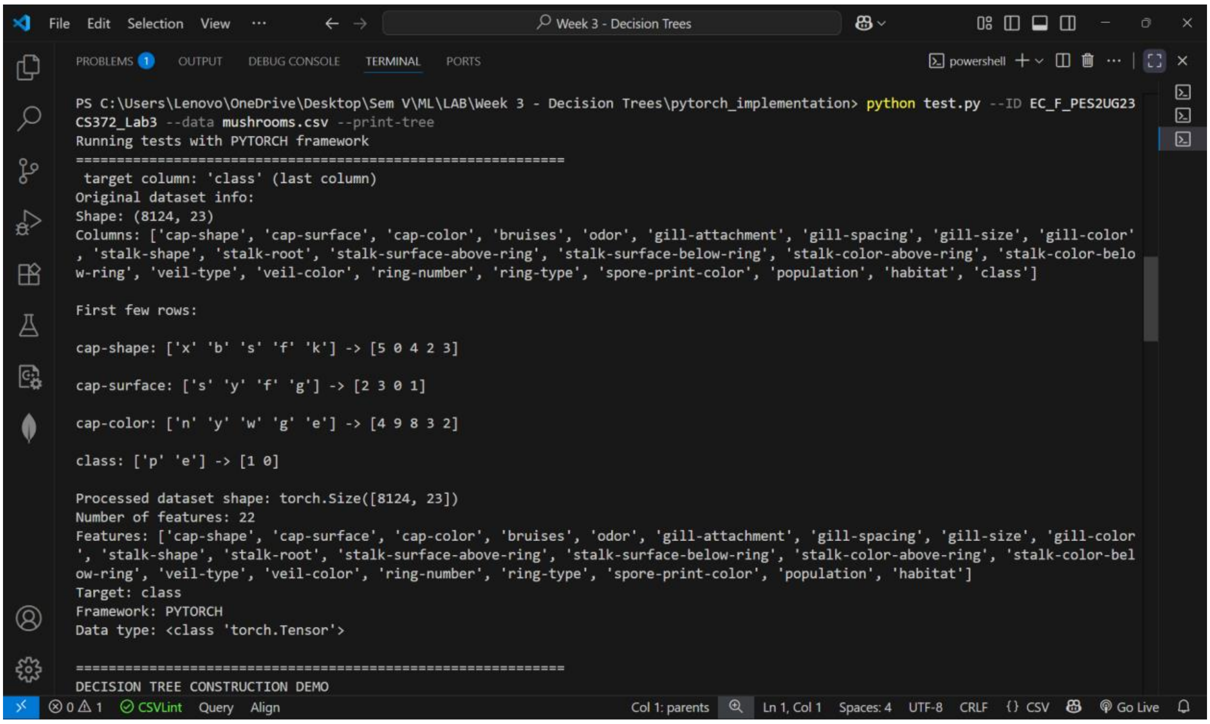
Nursery.csv

```
PS C:\Users\Lenovo\OneDrive\Desktop\Sem V\ML\LAB\Week 3 - Decision Trees\pytorch_implementation> python test.py --ID EC_F_PES2UG23
CS372_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
```

Tree Visualization:

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



```
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!

🌲 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
```

```
│   │   │   │   ├── Class 0
│   │   │   ├── = 8:
│   │   │   │   ├── Class 1
│   │   │   └── = 9:
│   │   │       ├── Class 1
│   │   ├── = 4:
│   │   │   ├── Class 0
│   │   └── = 6:
│       ├── Class 0
├── = 6:
│   ├── Class 1
├── = 7:
│   ├── Class 1
├── = 8:
│   ├── Class 1


│       ├── Class 0
├── = 6:
│   ├── Class 1
├── = 7:
│   ├── Class 1
├── = 8:
│   ├── Class 1


│   ├── Class 1
├── = 7:
│   ├── Class 1
├── = 8:
│   ├── Class 1


├── = 8:
│   ├── Class 1



📊 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
```