ML LAB 1

(ID3 Algorithm)

Comparative Analysis Report

a) Algorithm Performance

a. Which dataset achieved the highest accuracy and why?

- Typically, datasets with **balanced classes, sufficient size, and meaningful features** achieve higher accuracy.
- For example, a dataset with **low noise, clear feature-label correlation, and no severe imbalance** will outperform sparse or noisy datasets.
- High accuracy is usually observed in datasets where the decision boundaries between classes are well-separated.

b. How does dataset size affect performance?

- **Small datasets**: prone to overfitting; the algorithm memorizes patterns instead of generalizing.
- **Large datasets**: improve generalization and allow models to capture complex patterns, but require more computation.
- In decision trees, larger datasets usually reduce variance and yield more stable splits.

c. What role does the number of features play?

- **Few features**: may lead to underfitting if they don't capture enough patterns.
- **Many features**: risk of overfitting if irrelevant/noisy features dominate.
- Feature selection or dimensionality reduction improves interpretability and performance.

b) Data Characteristics Impact

How does class imbalance affect tree construction?

- Decision trees may become **biased toward majority classes**, creating shallow splits for minority classes.
- This leads to poor recall for minority labels.
- Handling imbalance requires **resampling (oversampling/undersampling), cost-sensitive learning, or balanced splitting criteria (like Gini with class weights).**

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

- **Binary features**: Easy to split, less prone to overfitting, simpler tree structure.
- **Multi-valued categorical features**: Can create highly branched trees, risk overfitting if many categories.
- **Numeric features**: Provide flexibility for threshold-based splits.

c) Practical Applications

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

- **Balanced binary datasets**: Fraud detection (fraud vs non-fraud), medical diagnosis (disease vs healthy).
- **Imbalanced datasets**: Rare event detection (cybersecurity attacks, rare diseases).
- **Multi-class datasets**: Image classification, text categorization, speech recognition.
- **High-dimensional datasets**: Genomics, NLP (where feature reduction is critical).

What are the interpretability advantages for each domain?

- **Binary datasets**: Easy to interpret, simple rules.
- **Multi-class datasets**: More complex, but decision trees still provide understandable rules.
- **High-dimensional datasets**: Trees highlight the most important features, aiding domain experts (e.g., gene markers in bioinformatics).

d) How would you improve performance for each dataset?

- **Small datasets**: Data augmentation, cross-validation, pruning to avoid overfitting.
- **Imbalanced datasets**: Oversampling (SMOTE), undersampling, class weights.

- **High-dimensional datasets**: Feature selection (PCA, mutual information), regularization.
- **Large noisy datasets**: Feature engineering, noise filtering, pruning.

Implementation Guidelines (Interpretation)

1. **No Hardcoding** → Write functions that adapt to dataset shape, e.g., `X[:, :-1]` for features and `X[:, -1]` for target.
2. **Framework Options** → Choose either:
   a. **PyTorch**: Use tensors for all operations (matrix multiplication, splits).
   b. **NumPy**: Use arrays; no sklearn shortcuts allowed.
3. **Target Variable** → Always assume the **last column is the label.**
4. **Function Signatures** → Keep the given structure intact (don't rename or reorder parameters).
5. **Additional Functions** → Create helper methods (e.g., for entropy, Gini index, splitting, recursive tree building).

OUTPUT SCREENSHORT
1 MUSHROOM

```
PS C:\Users\Admin\Downloads\ml> python test.py --ID ml_lab --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', 'stalk-surface-ab
ove-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', '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', 'stalk-surface-a
bove-ring', 'stalk-surface-below-ring', 'stalk-color-above-ring', 'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-type', 'spore-print-color', 'populatio
n', '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\Admin\Downloads\ml>
```

2 NURSERY

```
PS C:\Users\Admin\Downloads\ml> python test.py --ID ml_lab --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
```

```
============================================================
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
```

3 TICTACTOE

```
PS C:\Users\Admin\Downloads\ml> python test.py --ID ml_lab --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-sq
uare', '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-s
quare', '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
```

```
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
```

NAME : CHETHANA KR
SRN: PES2UG23CS151
SEC: 5C