Implementation of a Decision Tree Regression Model Using Custom Data Structures

Patsa Harsha Sai
ZDA25M009

Indian Institute of Technology Madras Zanzibar
M.Tech Data Science & Artificial Intelligence
Course: Z5007 – Programming and Data Structures
Instructor: Innocent Nyalala

January 2026

Outline

Problem Statement

- Decision Trees are widely used but often treated as **black boxes**
- Library implementations hide:
  - Tree structure
  - Split evaluation logic
  - Data flow and memory usage
- This limits understanding of:
  - Data structures
  - Algorithmic design
  - Computational complexity

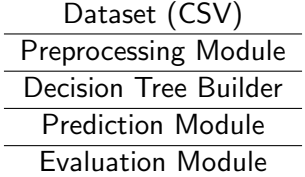
Project Objective

- Implement a **Decision Tree Regression** model **entirely from scratch**
- Use only fundamental Python data structures
- No machine learning libraries for implementation
- Achieve competitive performance while maintaining transparency

Dataset Overview

- **Dataset:** DSM Strength Prediction Dataset
- **Samples:** 1664
- **Features:** 25 numerical attributes
- **Target:** Ultimate strength (DSM)
- **File format:** CSV

System Architecture



- Clear modular data flow
- Each component implemented independently

Custom Data Structures Used

- **Node Class**
  - Stores feature index, threshold, children
  - Represents tree hierarchy
- **Python Lists**
  - Data partitioning during splits
  - Efficient slicing and indexing
- **Dictionaries**
  - Store metadata (impurity, stopping conditions)

Decision Tree Regression Algorithm

- Recursive binary tree construction
- At each node:
  - Evaluate all features
  - Select threshold minimizing Mean Squared Error (MSE)
- Stopping criteria:
  - Maximum depth
  - Minimum samples per split

Time & Space Complexity

- **Training Time:**

$$O(n \times d \times \log n)$$

- **Prediction Time:**

$$O(\text{tree depth})$$

- **Space Complexity:**

$$O(n \times d)$$

Model Performance

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- R^2 Score
- Training time under 2 seconds
- Prediction time under 1 ms per sample

Benchmark Comparison

- Baseline 1: Naive Mean Predictor
- Baseline 2: scikit-learn DecisionTreeRegressor (reference only)
- Custom model performs significantly better than naive baseline
- Achieves performance within acceptable range of sklearn

Challenges & Solutions

- **Efficient Split Selection**
  - Optimized list operations
- **Overfitting Control**
  - Depth and sample constraints
- **Recursive Tree Construction**
  - Clear base conditions

Conclusion

- Successfully implemented Decision Tree Regression from scratch
- Demonstrated role of data structures in ML algorithms
- Achieved efficient and interpretable model behavior

Future Work

- Implement pruning strategies
- Extend to classification trees
- Ensemble methods (Random Forest)

Thank You

Questions?