Z5007: Programming & Data Structures
Milestone 2 Progress Report

Implementation of a Decision Tree Regression Model Using Custom Data Structures

PATSA HARSHA SAI (ZDA25M009)
Instructor: Innocent Nyalala

December 2025

1. Progress Summary

Significant progress has been made in implementing a complete Decision Tree Regression model from scratch, aligned with the project proposal timeline (Weeks 6–10). A functional end-to-end pipeline now exists, including dataset preprocessing, custom data structure design, utility function development, recursive tree construction, and prediction mechanisms.

During final verification, an issue causing NaN predictions during standalone execution (via main.py) was identified. This was traced to missing preprocessing steps and internal tree nodes lacking fallback prediction values. The issue has been fully resolved by integrating consistent preprocessing and ensuring that all tree nodes store valid mean target values. The model now produces stable numeric predictions without errors.

To ensure clarity and maintainability, all implemented modules (`node.py`, `utils.py`, `tree.py`) include appropriate comments and docstrings explaining complex logic, design choices, and function responsibilities. This aligns with the rubric requirement for clean code with documentation.

Note: Pandas is used strictly for dataset loading and preprocessing; all algorithmic components such as splitting, impurity computation, and tree construction are implemented using pure Python and NumPy, as required.

1.1 Project Structure & Environment Setup

A modular project structure has been created in Google Drive for Colab-based development:

```
DecisionTreeProject/
    data/
    src/
        node.py
        utils.py
        tree.py
        __init__.py
    main.py
    README.md
    milestone2_run_log.txt
```

This structure promotes maintainability and aligns with rubric expectations for organized code design.

1.2 Dataset Loading & Preprocessing

The DSM Strength Prediction dataset (1664 rows, 25 features) was successfully loaded both in the notebook environment and in the standalone execution script (main.py). Identical preprocessing logic is applied in all execution contexts to ensure consistency.

Preprocessing steps completed:

- Standardization of column names
- Conversion of object columns to numeric types
- Removal of duplicate rows
- Handling of missing values via `dropna()`
- Train-test split (70% train, 30% test)

1.3 Core Data Structure: Node Class

A `Node` class was implemented with:

- `feature_index`
- `threshold`
- `left` and `right` child references
- `value` for leaf nodes
- `is_leaf()` method

This serves as the foundation of the decision tree hierarchy.

Default hyperparameters currently used: `max_depth = 10`, `min_samples_split = 2`, `min_samples_leaf = 1`.

1.4 Utility Functions

Implemented in `utils.py`:

- **MSE** impurity function
- **Dataset splitting** function
- **Best split** search algorithm
- Basic stopping criteria

Candidate thresholds for continuous features are selected as midpoints between sorted unique feature values. This reduces redundant split evaluations and improves computational efficiency while preserving optimal split quality.

A node automatically becomes a leaf when all target values within the node are identical, since no further reduction in impurity is possible.

1.5 Decision Tree Regressor Implementation

A fully functional `DecisionTreeRegressor` class was implemented with:

- `fit()` for model training
- recursive `_build_tree()` method
- `predict_one()` and `predict()` functions

To ensure robustness during prediction, each internal tree node stores the mean target value of its corresponding data subset. This guarantees valid predictions even in edge cases such as early stopping or degenerate splits.

1.6 Functional Verification

A synthetic test ($X = [1, 2, 3, 4, 5]$, $y = x^2$) produced predictions:

$$[9,\ 25]$$

demonstrating correct recursion, leaf creation, and tree traversal. The model was further validated on the full DSM dataset using standalone execution via python main.py.

After preprocessing, the model generated stable numeric predictions with no NaN values. A sample execution produced 498 test predictions with values ranging approximately from 4.19 to 254.13, confirming correct tree traversal and leaf value handling.

1.7 Overall Progress

Approximately 75–80% of the overall project is complete, with the core algorithm fully functional. Recent improvements include optimized threshold selection, pure-node stopping conditions, and safeguards against invalid recursive splits.

2. Technical Details

2.1 Data Structures Implemented

1. Node Class: Represents decision rules or leaf predictions. **2. NumPy-based array slicing:** Enables efficient split operations and feature-by-feature threshold evaluation.

2.2 Algorithms Implemented

1. MSE Impurity:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

2. Dataset Splitting: Samples are separated into left ($\leq$ threshold) and right ($>$ threshold) subsets.

Variance Reduction Formula (Gain):

$$Gain = MSE_{parent} - \left(\frac{n_L}{n}MSE_L + \frac{n_R}{n}MSE_R\right)$$

3. Best Split Selection: Evaluates candidate split thresholds chosen as midpoints between sorted unique feature values. Weighted child MSE is computed:

$$MSE_{weighted} = \frac{n_L}{n}MSE_L + \frac{n_R}{n}MSE_R$$

4. Recursive Tree Construction: Stops based on maximum depth, insufficient samples, identical target values, or inability to reduce impurity.

5. Prediction Traversal: Each sample navigates down the tree based on decision rules.

2.3 Complexity Analysis

- **Best Split Search:** $O(n \times d \times u)$
- **Tree Construction:** Average $O(n \log n)$, worst-case $O(n^2)$
- **Prediction:** $O(\text{depth})$

Where n = number of samples, d = number of features, and u = candidate split thresholds (midpoints).

Memory usage during training will be measured using the Python `memory_profiler` package to quantify memory growth during recursion.

3. Challenges & Solutions

Challenge 1: Recursion Depth Management

Issue: Risk of deep, unbalanced recursion. **Solution:** Implemented `max_depth` and `min_samples_split` to prevent unnecessary branching.

Challenge 2: Computational Cost of Split Evaluation

Issue: Evaluating many thresholds is expensive. **Solution:** Optimized with NumPy and skipped invalid splits. Further optimization may include pruning strategies, threshold caching, and early stopping to reduce unnecessary split evaluations.

Challenge 3: Dataset Formatting Issues

Issue: Mixed formatting and object-type numeric columns. **Solution:** Standardized column names and enforced strict numeric conversion.

Challenge 4: NaN Predictions During Standalone Execution

Issue: While the model worked correctly in the notebook environment, initial standalone execution using main.py produced NaN predictions despite clean data. **Solution:** The issue was traced to missing preprocessing steps and the absence of fallback prediction values in internal tree nodes. This was resolved by integrating full preprocessing logic into main.py and ensuring that each node stores the mean target value of its data subset. **Outcome:** The corrected implementation now produces consistent, valid predictions across all execution environments, with no NaN values observed.

Current Blocker (Required by Rubric)

An initial version of the split selection algorithm evaluated all unique feature values, which increased training time. This issue has now been resolved by implementing a midpoint-based threshold selection strategy, along with additional stopping conditions and defensive checks. As a result, training stability and performance have improved, and the algorithm now scales more efficiently on the full dataset.

4. Remaining Work

- Implement additional pruning strategies and minimum leaf size constraints
- Add tree statistics (depth, node count, leaf count)
- Optimize split evaluation
- Evaluate performance (MAE, RMSE, R^2)
- Benchmark against baselines, including sklearn's DecisionTreeRegressor (reference only, not used for implementation)
- Prepare final report and presentation slides

Updated Timeline

Week	Remaining Tasks
11	Optimization, advanced stopping criteria
12	Benchmarking & metrics computation
13	Documentation & diagrams
14	Final testing, packaging, presentation

Confidence of Completion

The project is on schedule, and completion confidence is **high**.