Decision Tree Classifier – Custom Implementation

Programming in Python Language
Michał Janik AGH

---

1. Project Overview

The goal of this project is to implement a simple, fully transparent **Decision Tree Classifier** using only Python and NumPy.
Unlike scikit-learn, this implementation is written **entirely from scratch**, focusing on full algorithmic understanding rather than performance.

The tree uses the **CART methodology**, based on binary splits and impurity minimization (Gini or Entropy).
It supports configurable parameters such as maximum depth and minimum samples required to split.

---

2. Project Structure

The project is split into modules for clarity and maintainability:

• decision_tree.py

Contains the full implementation of:

- impurity functions (`gini`, `entropy`),
- the `Node` dataclass used as the internal structure of the tree,
- `DecisionTreeClassifier` containing training, prediction, and scoring logic,
- optional comparison function using scikit-learn (`compare`).

• tests/

Includes unit and integration tests for:

- impurity correctness,
- split logic,
- end-to-end tree training,
- compatibility tests with scikit-learn (if installed).

---

3. Algorithm Description

3.1. Dataset Overview

The implementation is dataset-agnostic.
It accepts any NumPy array X (shape: n_samples × n_features) and integer labels array y.

Typical examples used for development and testing:

- simple binary classification (0/1),
- small synthetic datasets,
- Iris dataset (via scikit-learn, optional).

This keeps the algorithm simple and reproducible.

3.2. Data Preprocessing

The classifier assumes minimal preprocessing:

1. **Ensure that feature values in X are numeric**
   (Decision Trees cannot process strings directly).
2. **Convert labels to integers**
   (fit() handles this internally using np.array(y, dtype=int)).
3. **Optional normalization**
   Not required, because decision trees are scale-invariant.

Unlike deep learning models, decision trees handle raw values without augmentation or normalization.

3.3. Tree Construction Algorithm

The decision tree implements the following components:

• Impurity Metrics

Two impurity functions are available:

- **Gini impurity**
- **Shannon entropy**

Both measure how mixed the labels are in a node.

• Split Selection Strategy

The tree performs **exhaustive search** across:

- every feature,
- every possible split point (midpoint between sorted unique values).

For each candidate split:

1. The dataset is divided into left/right nodes.
2. Both sides' impurity is calculated.
3. A weighted impurity score is computed.
4. Impurity reduction (gain) is measured.
5. The best split is selected.

This mimics the logic used by CART.

• Stopping Conditions

The recursion stops when:

- all labels in the node are identical,
- maximum depth is reached,
- no split yields a positive impurity gain,
- one side of the split would contain fewer samples than min_samples_split.

In these cases the node becomes a **leaf** with a majority label.

• Tree Representation

Each node is represented by:

```
@dataclass
class Node:
    feature
    threshold
    left
    right
    value  # used only in leaf nodes
```

This makes the structure lightweight and easy to print or visualize.

3.4. Training Process

Training is handled by:

```
DecisionTreeClassifier.fit(X, y)
```

Key elements:

- Recursively builds the tree using _build().

- Node purity and depth limits are respected.
- Majority voting is used for leaf assignment.
- The resulting tree is stored as `self.tree_`.

The training process is deterministic (no randomness).

3.5. Evaluation and Inference

Prediction works by traversing the tree:

- starting from the root,
- comparing a feature to its threshold,
- moving left or right until reaching a leaf node.

Accuracy is measured with:

```
score(X, y)
```

which computes standard classification accuracy.

3.6. Optional Comparison with Scikit-learn

If scikit-learn is installed, running:

```
compare(max_depth=3)
```

will:

1. Train this custom classifier on the Iris dataset.
2. Train scikit-learn's `DecisionTreeClassifier`.
3. Print both accuracies.

This allows validating correctness and experiment-level performance.

---

4. Installation

To run the project, set up a Python virtual environment and install dependencies.

```
python -m venv .venv
.\.venv\Scripts\activate   # Windows
# or
source .venv/bin/activate  # Linux/Mac

pip install numpy scikit-learn pytest
```

---

5. How to Run the Code

Train and test the tree manually

```
python decision_tree.py
```

Run scikit-learn comparison

```
python decision_tree.py --demo
```

Run the unit tests

```
pytest tests/
```

The project can be extended by adding plotting, exporting the tree structure, or supporting additional impurity metrics.