AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Decision Tree Classifier

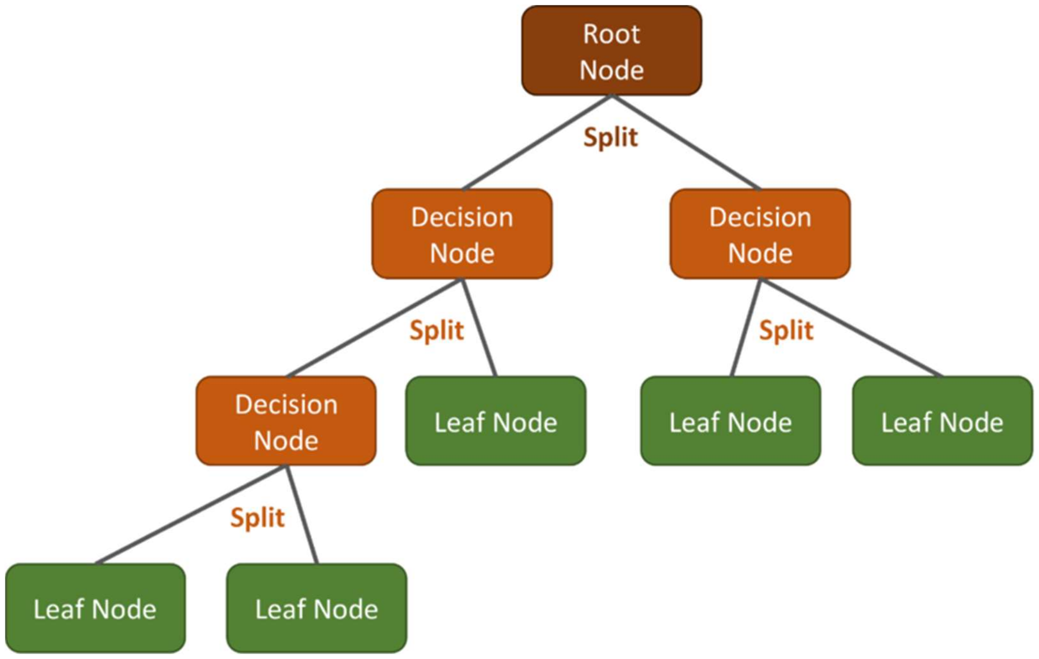
Programmer: Michał Janik

Tester: Maciej Skrobisz

1. Project Overview

The goal of this project is to implement a simple, fully transparent **Decision Tree Classifier** using only Python and NumPy.

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.



A Decision Tree classifier creates an upside-down tree to make predictions, starting at the top with a question about an important feature in your data, then branches out based on the answers. As you follow these branches down, each stop asks another question, narrowing down the possibilities. This question-and-answer game continues until you reach the bottom — a leaf node — where you get your final prediction or classification.

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,

• tests/

Includes unit and integration tests for:

- impurity correctness,
- split logic,
- end-to-end tree training
- comparison function using scikit-learn (compare).

3. Algorithm description

The implemented classifier is a dataset-agnostic Decision Tree based on the CART methodology.
It operates on numeric feature data provided as a NumPy array $X \in R^{n \times m}$ and integer class labels y.

The algorithm recursively constructs a binary tree by selecting, at each node, the feature and threshold that maximize impurity reduction, using either Gini impurity or Shannon entropy.
All possible split points are evaluated as midpoints between sorted feature values.

Tree growth stops when a node becomes pure, a predefined maximum depth is reached, no split yields positive impurity gain, or the minimum number of samples required for a split is not satisfied.
In such cases, the node is converted into a leaf and assigned the majority class.

Model training is deterministic and performed using recursive tree construction.
Prediction is achieved by traversing the tree from the root to a leaf node based on simple threshold comparisons, and classification accuracy is used as the evaluation metric.

---

4. Installation

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

```
python -m venv .venv

.\.venv\Scripts\activate  # Windows
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 the unit tests

```
pytest tests/
```

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

Tests

Maciej Skrobisz

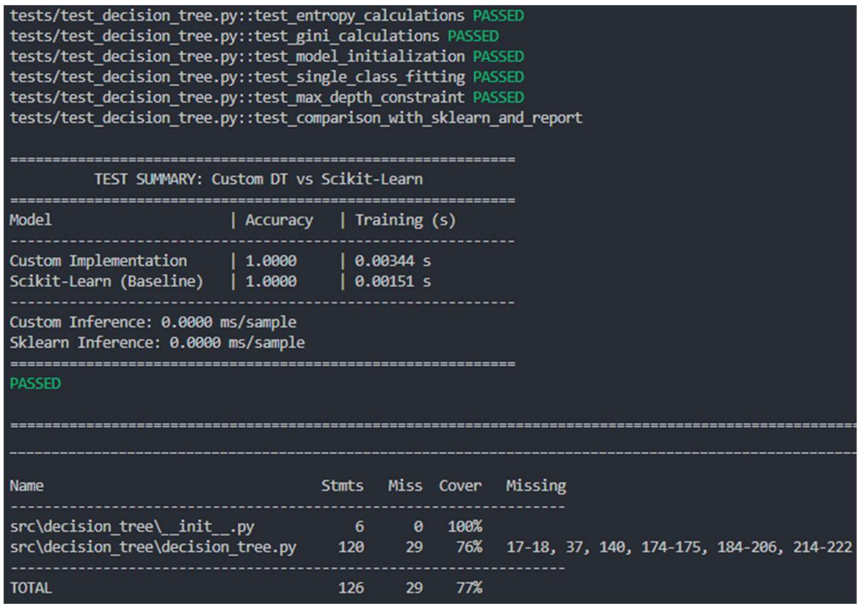
1. Unit Tests

The pytest framework was used to check the correctness of the mathematical formulas and the tree structure.

The tests covered:

- **Entropy and Gini Impurity:** Verifying if calculations for split quality are correct (e.g., entropy is 0 for a pure node).
- **Model Initialization:** Checking if parameters like max_depth are stored correctly.
- **Constraints:** Ensuring the tree does not grow deeper than the specified max_depth.
- **Edge Cases:** Testing how the model handles datasets with only one class.

Results:

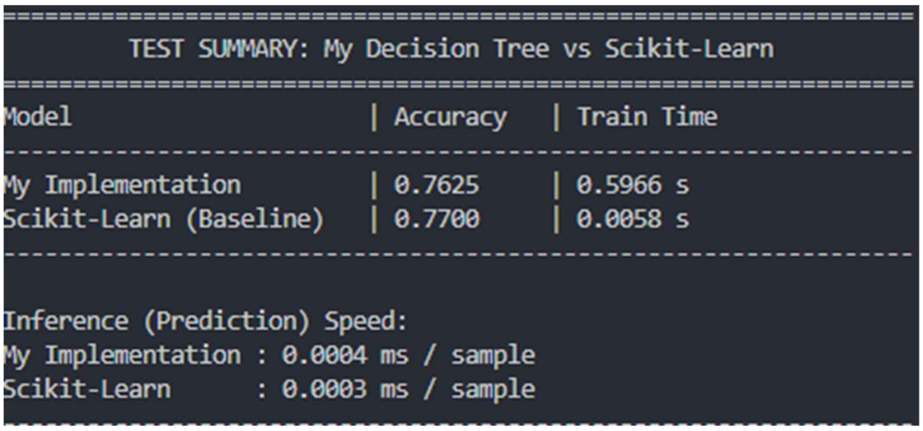


```
tests/test_decision_tree.py::test_entropy_calculations PASSED
tests/test_decision_tree.py::test_gini_calculations PASSED
tests/test_decision_tree.py::test_model_initialization PASSED
tests/test_decision_tree.py::test_single_class_fitting PASSED
tests/test_decision_tree.py::test_max_depth_constraint PASSED
tests/test_decision_tree.py::test_comparison_with_sklearn_and_report

============================================================
          TEST SUMMARY: Custom DT vs Scikit-Learn
============================================================
Model                     | Accuracy    | Training (s)
------------------------------------------------------------
Custom Implementation     | 1.0000      | 0.00344 s
Scikit-Learn (Baseline)   | 1.0000      | 0.00151 s
------------------------------------------------------------
Custom Inference: 0.0000 ms/sample
Sklearn Inference: 0.0000 ms/sample
============================================================
PASSED

====================================================================================================

----------------------------------------------------------------------------------------------------

Name                                 Stmts   Miss  Cover   Missing
------------------------------------------------------------------
src\decision_tree\__init__.py            6      0   100%
src\decision_tree\decision_tree.py     120     29    76%   17-18, 37, 140, 174-175, 184-206, 214-222
------------------------------------------------------------------
TOTAL                                  126     29    77%
```

All 6 tests passed successfully. The code coverage is 77%, covering all core logic.

2. Performance Comparison (Benchmark)

A comparison was run between the tested implementation and DecisionTreeClassifier from scikit-learn.
The two experiments were: one on the small Iris dataset (verification) and one on a larger synthetic dataset (stress test).



```
=============================================================
        TEST SUMMARY: My Decision Tree vs Scikit-Learn
=============================================================
Model                    | Accuracy    | Train Time
-------------------------------------------------------------
My Implementation        | 0.7625      | 0.5966 s
Scikit-Learn (Baseline)  | 0.7700      | 0.0058 s
-------------------------------------------------------------

Inference (Prediction) Speed:
My Implementation : 0.0004 ms / sample
Scikit-Learn      : 0.0003 ms / sample
-------------------------------------------------------------
```

Experiment A: Iris Dataset (Verification)

- **Goal:** Check if the model learns simple rules correctly.
- **Result:** Both models achieved **100% accuracy**.

Experiment B: Synthetic Dataset (Stress Test)

- **Data:** 2000 samples, 10 features.
- **Goal:** Compare accuracy and training speed on larger data.

3. Observations and Conclusions

Based on the test results, the following conclusions can be drawn:

1. **High Accuracy:** The tested implementation is mathematically correct. On the large dataset, the difference in accuracy compared to scikit-learn is only **0.75%**. This proves that the split logic (Gini/Entropy) works as expected.

2. **Performance Gap:** Scikit-learn is approximately **100x faster** in training. This is expected because scikit-learn is highly optimized and written in C/Cython, while the tested implementation runs in pure Python. However, a training time of ~0.6 seconds for 2000 samples is still acceptable for educational purposes.

3. **Fast Prediction:** The inference speed (making predictions) is very fast in both models. Once the tree is built, traversing it is instant, regardless of the language used.

Summary: The project successfully implements a working Decision Tree classifier that rivals the standard library in accuracy, although it is naturally slower due to the lack of low-level optimizations.