Decision Trees from Scratch

Applied Machine Learning in Engineering - Exercise 05

TU Berlin, Summer Term 2025

Prof. Dr.-Ing. Merten Stender – `merten.stender@tu-berlin.de`

Learning Objectives

By the end of this assignment, you will be able to:

- Calculate information entropy and information gain for binary classification problems.
- Implement a split logic and optimal split search for decision trees.
- Apply a recursive logic for binary splitting of data.
- Visualize split boundaries and interpret decision tree behavior.

Task 1: Implementing Core Functions for Decision Trees (45 minutes, at home)

In this task, you will implement core functions required to construct a binary decision tree. Use the provided dataset `decision_tree_dataset.txt`, which includes:

- Two input features (`x1`, `x2`)
- A binary target variable (`0` or `1`)

Each data point is labeled, and your model will aim to split the space to separate the classes best. By convention, the **left child** node contains samples that satisfy $x \leq$ (split value), and the **right child** node includes samples that are $x >$ (split value), where x is a single feature column of the dataset.
Implement the following functions:

(a) `entropy(y: np.ndarray)`: Compute the information entropy $H(y)$ for a vector of labels `y`. Return the entropy as a floating-point number.

(b) `information_gain(y_parent: np.ndarray, index_split: np.ndarray)`: Compute the information gain of a given split. The split is represented by the corresponding binary mask `index_split`, where `1` denotes assignment to the left child and `0` to the right child.

(c) `best_split(x: np.ndarray, y: np.ndarray)`: Loop over all features contained in the dataset `x` (N samples in rows, n features in columns) and over all potential splitting thresholds to find the optimal split with the highest information gain (utilize the entropy and information gain functions implemented before). Return the `split_dim` (feature index) and the `split_val` (threshold) for the best split.

(d) `create_split(x: np.ndarray, split_dim: int, split_val: float)`: Partition the dataset `x`: Return a binary mask indicating which data points go to the left or right child node. Do not return the splitted dataset itself.

Assume `x.shape = [N, n]` and ensure all functions are vectorized as much as possible.

Task 2: Validation and Visualization (45 minutes)

Validate your implementation using the provided dataset `decision_tree_dataset.txt` or generate synthetic binary-classification data on your own.

(a) Load the dataset using:

`data = np.loadtxt("decision_tree_dataset.txt")`

Extract the two feature dimensions `x = data[:, :2]` and the binary labels `y = data[:, 2]`.

(b) Use your implementations from Task 1 to find the best split of the given data set (i.e., the feature dimension along which to split, and the splitting threshold).

(c) Validate results manually by computing entropy and information gain for the chosen split.

(d) Implement a simple plotting function:

- Use color to differentiate classes.
- Indicate the decision boundary defined by `split_dim` and `split_val`.
- Display the information gain in the plot title.

(e) Visualize the decision boundary by sampling many points from the feature space and coloring them according to the predicted class label.

(f) Wrap your testing / validation code in a main block:

`if __name__ == "__main__":`

Task 3: Using `scikit-learn`'s DecisionTreeClassifier (30 minutes)

In this task, you will use the built-in `DecisionTreeClassifier` from `scikit-learn` to perform binary classification on the same dataset.

(a) Import the classifier `from sklearn.tree import DecisionTreeClassifier` and load the dataset from Task 2.

(b) Fit a decision tree model using a maximum depth of 1, i.e., only a single split:

`clf = DecisionTreeClassifier(criterion="entropy", max_depth=1)`

`clf.fit(X, y)`

(c) Predict the labels: `y_pred = clf.predict(X)`

(d) Plot the data and show the decision boundary of the trained model. Use the same visualization style as in Task 2.

(e) Compare the learned split (feature and threshold) to the one you obtained in Task 1. Use:

`clf.tree_.feature` and `clf.tree_.threshold`

Are the results consistent with your manual implementation? Discuss any discrepancies and speculate on their cause (e.g., tie-breaking, floating-point precision)

(f) Now, relax the condition on the maximum depth and let the DT classifier perform the optimal classification. Plot the result, and compare it to the ground truth.

Bonus (optional, content of next exercise): Extend your implementation to recursively apply splits and construct a full decision tree with a maximum depth of 2 or 3.