University of Milan

Faculty of Political, Economic and Social Sciences

Binary Tree Classifier from Scratch for Mushroom Classification

Final Project in the Subject Machine Learning

Julia Maria Wdowinska

Data Science for Economics
I year
Master's Degree
Matriculation Number: 43288A

January 11, 2025

Contents

1 Introduction

2 Dataset Description

The dataset used in this study is a simulated version inspired by the Mushroom Data Set from J. Schlimmer. It contains 61,069 hypothetical mushrooms, each described by 20 features and classified as either definitely edible or definitely poisonous/of unknown edibility. Table 1 presents all 21 variables included in the dataset.

Table 1: Mushroom Dataset Variables

Variable	**Type**	**Possible Values**
class	categorical	e (edible), p (poisonous/of unknown edibility)
cap-diameter	numerical	float number in cm
cap-shape	categorical	b (bell), c (conical), x (convex), f (flat), s (sunken), p (spherical), o (others)
cap-surface	categorical	i (fibrous), g (grooves), y (scaly), s (smooth), h (shiny), l (leathery), k (silky), t (sticky), w (wrinkled), e (fleshy)
cap-color	categorical	n (brown), b (buff), g (gray), r (green), p (pink), u (purple), e (red), w (white), y (yellow), l (blue), o (orange), k (black)
does-bruise-bleed	categorical	t (bruises or bleeding), f (no)
gill-attachment	categorical	a (adnate), x (adnexed), d (decurrent), e (free), s (sinuate), p (pores), f (none), ? (unknown)
gill-spacing	categorical	c (close), d (distant), f (none)
gill-color	categorical	see cap-color, f (none)
stem-height	numerical	float number in cm
stem-width	numerical	float number in mm
stem-root	categorical	b (bulbous), s (swollen), c (club), u (cup), e (equal), z (rhizomorphs), r (rooted)
stem-surface	categorical	see cap-surface, f (none)
stem-color	categorical	see cap-color, f (none)
veil-type	categorical	p (partial), u (universal)
veil-color	categorical	see cap-color, f (none)
has-ring	categorical	t (ring), f (none)
ring-type	categorical	c (cobwebby), e (evanescent), r (flaring), g (grooved), l (large), p (pendant), s (sheathing), z (zone), y (scaly), m (movable), f (none), ? (unknown)
spore-print-color	categorical	see cap-color
habitat	categorical	g (grasses), l (leaves), m (meadows), p (paths), h (heaths), u (urban), w (waste), d (woods)
season	categorical	s (spring), u (summer), a (autumn), w (winter)

The class distribution is balanced, i.e., 27,181 mushrooms are edible, and 33,888 are poisonous or of unknown edibility. However, 9 variables contain missing values (see Table 2).

Table 2: Missing Values Count and Percentage

Variable	**Missing Values**
cap-surface	14,120 (23.12%)
gill-attachment	9,884 (16.18%)
gill-spacing	25,063 (41.04%)
stem-root	51,538 (84.39%)
stem-surface	38,124 (62.43%)
veil-type	57,892 (94.80%)
veil-color	53,656 (87.86%)
ring-type	2,471 (4.05%)
spore-print-color	54,715 (89.60%)

3 Tree Classifier Implementation

Tree predictors are fundamental tools in machine learning, widely applied to classification and regression tasks. They represent a hierarchy of decision rules, where data points are recursively split into subsets based on feature values. The main advantage of tree predictors is their ability to handle both numerical and categorical features. Tree predictors are also straightforward, making them a popular choice when interpretability is a priority.

In this study, a complete binary tree classifier - where each internal node has exactly two children - has been implemented in Python. A detailed description of the classes and methods created is provided below.

3.1 TreeNode Class

The `TreeNode` class represents a single node in a binary tree classifier. Each node can either be an internal node or a leaf node. Internal nodes split the data based on a specific feature and threshold, while leaf nodes store the predicted label. The attributes and methods of this class are designed to support the recursive structure of the tree classifier.

Attributes:

- `feature_index` (int or None): The feature index used for splitting the data at this node.
- `threshold_value` (float or None): The threshold value that determines how the data is split at this node.
- `left_child` (TreeNode or None): The left child node.
- `right_child` (TreeNode or None): The right child node.
- `left_ratio` (float or None): The ratio of samples that go to the left child. This value is particularly useful for handling missing values and calculating probabilities.
- `leaf_value` (int or None): The predicted label associated with the leaf node.

If the node is a leaf node, then `feature_index`, `threshold_value`, `left_child`, `right_child`, and `left_ratio` are `None`, while `leaf_value` is an integer. If the node is not a leaf node, then only `leaf_value` is `None`.

Methods:

- `is_leaf()`: This method checks whether the current node is a leaf node. It returns `True` if the node has a `leaf_value`, and `False` otherwise.

3.2 DecisionTreeClassifier Class

The `DecisionTreeClassifier` class implements a decision tree for binary classification. It recursively splits the data based on specific features and thresholds, creating a tree structure that can be used for predicting labels. The attributes and methods of this class are designed to support model training, hyperparameter tuning, and prediction.

Attributes:

- `min_samples_split` (int): The minimum number of samples required to split a node.
- `max_depth` (int or None): The maximum depth of the tree. If set to `None`, the tree expands until all nodes are pure, contain fewer than `min_samples_split` samples, or further splitting results in an information gain below `min_information_gain`.
- `n_features` (int, float, or str): The number of features to consider when identifying the best split. This can be specified as an integer, a float, or one of the following strings: `‘sqrt’` or `‘log2’`.
- `criterion` (str): The function to measure the quality of a split. Options are: `‘gini’`, `‘scaled_entropy’`, and `‘square_root’`.
- `min_information_gain` (float): The minimum information gain required to perform a split.
- `n_quantiles` (int or None): The number of quantiles to consider when determining the best threshold for continuous features. If set to `None`, the algorithm uses midpoints of unique values.
- `isolate_one` (bool): Whether to isolate a single value for categorical features, creating a one-vs-rest split.

- `root` (TreeNode or None): The root node of the decision tree.
- `depth` (int): The final depth of the tree after it has been built.

min_samples_split Parameter:

The `min_samples_split` parameter controls the minimum number of samples a node must contain to be eligible for splitting. If a node has fewer than `min_samples_split` samples, it becomes a leaf node, and no further splits are attempted.

A higher value for `min_samples_split` reduces the depth of the tree, making it less prone to capturing noise in the data. Conversely, a lower value allows the tree to grow deeper and potentially capture finer details, which can be beneficial for highly complex datasets but may increase the risk of overfitting. The default value is 2.

n_features Parameter:

The `n_features` parameter specifies the number of features to consider when identifying the best split. The default value is `None`, which results in all features in the dataset being considered. Otherwise:

- If `n_features` is an integer, this specifies the exact number of features to consider. If the value exceeds the total number of features in the dataset, all features are considered instead.
- If `n_features` is a float, it represents a fraction of the total number of features. The number of features to consider is calculated by multiplying this fraction by the total number of features and truncating the decimal part to obtain an integer. At least one feature is considered.
- If `n_features` is a string, it can be either `'sqrt'` or `'log2'`, and the number of features is calculated as follows:
  - `'sqrt'`: Sets the number of features to the square root of the total number of features, truncating the decimal part to obtain an integer. At least one feature is considered.
  - `'log2'`: Sets the number of features to the base-2 logarithm of the total number of features, truncating the decimal part to obtain an integer. At least one feature is considered.

This parameter allows the decision tree model to use a subset of features, which can help improve the model's efficiency and performance, particularly when working with high-dimensional datasets.

criterion Parameter:

The `criterion` parameter specifies the function used to measure the quality of a split in the decision tree. The available options are:

- `'gini'` (the default): The Gini impurity is used, which is computed as:

  $$\text{Gini} = 2 \cdot p_0 \cdot (1 - p_0)$$

  where p_0 is the probability of class '0' within the node.
- `'scaled_entropy'`: The scaled entropy is used. The entropy is scaled by halving the probabilities before applying the standard entropy formula:

  $$\text{Scaled Entropy} = -\sum_i \frac{p_i}{2} \cdot \log_2(p_i + \epsilon)$$

  where p_i is the probability of class i, and ϵ is a small constant to avoid taking the logarithm of zero.
- `'square_root'`: The "square root" impurity is used, which is calculated as:

  $$\text{Square Root Impurity} = \sqrt{p_0 \cdot (1 - p_0)}$$

  where p_0 is the probability of class '0' within the node.

min_information_gain Parameter:

The `min_information_gain` parameter specifies the minimum amount of information gain required to perform a split. Information gain measures the reduction in impurity after a split. It is computed as follows:

$$\text{Information Gain} = \text{Impurity Before Split} - \text{Weighted Impurity After Split}$$

where the impurity is calculated using the selected `criterion`, such as Gini impurity, scaled entropy, or square root impurity. The weighted impurity after the split is calculated as:

$$\text{Weighted Impurity After Split} = \frac{L}{n} \cdot \text{Impurity of Left Child} + \frac{R}{n} \cdot \text{Impurity of Right Child}$$

where:

- L and R are the number of samples in the left and right child nodes, respectively.
- n is the total number of samples in the parent node.

The `min_information_gain` parameter accepts a float value that sets the threshold for the minimum information gain. If the calculated information gain from a potential split is less than this threshold, the split is not performed, and the node becomes a leaf node. The default value is 0.0.

n_quantiles Parameter:

The `n_quantiles` parameter specifies the number of quantiles to consider when determining the best threshold for splitting continuous features.

- If `n_quantiles` is set to `None` (the default), the algorithm considers all midpoints between unique values of the feature as candidate thresholds.
- If `n_quantiles` is an integer, the continuous feature values are divided into that many quantiles, and the candidate thresholds are chosen as the boundaries between these quantiles.

By adjusting the `n_quantiles` parameter, you can control the granularity of threshold selection:

- Lower values of `n_quantiles` reduce the number of candidate thresholds, which can speed up computation but may lead to less optimal splits.
- Higher values of `n_quantiles`, or using `None` (to consider all midpoints), increase the granularity of the search, improving the likelihood of finding an optimal split at the expense of additional computation time.

Methods:

- `fit(X, y)`: Fits the decision tree model to the training data `X` and labels `y`.
- `_build_tree(X, y, depth=0)`: Recursively builds the decision tree by splitting the data based on the best feature and threshold. It stops if the minimum number of samples per split is reached, the maximum depth is reached, or all samples belong to the same class.
- `_get_most_common_label(y)`: Finds the most common label in the array `y`.
- `_find_best_split(X, y, feature_indices)`: Finds the best feature and threshold for splitting the data.
- `_calculate_information_gain(y, feature_column, threshold_value)`: Calculates the information gain from a split based on a selected criterion.
- `_split(feature_column, threshold_value)`: Splits the feature data based on the threshold value, considering missing values and categorical data.
- `_gini_impurity(y)`: Calculates the Gini impurity of the labels.
- `_scaled_entropy(y)`: Calculates the scaled entropy of the labels.
- `_square_root_impurity(y)`: Calculates the square root impurity of the labels.
- `predict(X)`: Predicts the labels for the input data `X` using the trained decision tree model.
- `_traverse_tree(x)`: Traverses the tree for a single input sample `x` and returns the predicted label.

Description:

The `DecisionTreeClassifier` is a flexible and efficient implementation of a decision tree for binary classification. It uses a recursive approach to build the tree, splitting the data at each node based on the best feature and threshold that maximizes information gain. The tree grows until it reaches the stopping criteria, including a maximum depth or a minimum number of samples for a split.

The model supports various splitting criteria, including Gini impurity, scaled entropy, and square root impurity. Additionally, it allows handling of missing values and categorical features, providing flexibility in dealing with real-world data. The tree can be pruned or grown further based on hyperparameters such as maximum depth, minimum samples per split, and the minimum information gain required to make a split.

After training, the model can predict labels for new samples by traversing the tree from the root to a leaf node, where the label is assigned. The model also supports cross-validation for hyperparameter tuning, helping find the best set of parameters for the decision tree.