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 24, 2025

Contents

1 Introduction

2 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.

2.1 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.

2.2 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 determines how candidate thresholds are chosen when splitting based on numerical features. If set to `None` (the default), all midpoints between unique values are considered. Otherwise:

- If `n_quantiles` is an integer, the values are divided into that many quantiles, and the candidate thresholds are the boundaries between these quantiles.

While lower values of `n_quantiles` reduce the number of candidate thresholds, speeding up computation but potentially leading to suboptimal splits, higher values or setting it to `None` (to consider all midpoints) increase the search granularity, increasing the probability of finding an optimal split, but at the cost of additional computation time.

isolate_one Parameter:

The `isolate_one` parameter controls how splits are made when splitting based on categorical features. If set to `False` (the default), all data points with a feature value lower or equal (i.e., lower or equal alphabetically) to the threshold are assigned to the left child, while all other data points are assigned to the right child. Otherwise:

- If `isolate_one` is set to `True`, the algorithm creates a one-vs-rest split, where all data points with a feature value equal to the threshold go to the left child, while all other data points go to the right child.

This parameter affects the granularity of splits for categorical features. Setting `isolate_one` to `True` results in more precise splits, capturing finer patterns in the data but potentially increasing the risk of overfitting. In contrast, setting it to `False` produces broader, more generalized splits, improving computational efficiency and helping reduce overfitting.

Private Methods:

- `_build_tree()`: Recursively builds the tree by splitting the data based on the best feature and threshold. It stops if any stopping condition is met, e.g., `max_depth`.
- `_get_most_common_label()`: This method finds and returns the most common label in a given array.
- `_find_best_split()`: Finds the best feature and threshold for splitting the data.
- `_calculate_information_gain()`: Computes the information gain from a potential split based on a selected criterion.
- `_split()`: Splits the data based on the selected feature and threshold.
- `_gini_impurity()`: Computes the Gini impurity for the given labels.
- `_scaled_entropy()`: Computes the scaled entropy for the given labels.
- `_square_root_impurity()`: Computes the "square root" impurity for the given labels.
- `_traverse_tree()`: Traverses the tree for a single input sample and returns the predicted label.

Public Methods:

- `fit()`: Initializes the root node and builds the tree using the `_build_tree()` method.
- `predict()`: Predicts the labels for the given input samples by traversing the tree for each sample using the `_traverse_tree()` method.

_build_tree() Method:

The `_build_tree()` method constructs a decision tree by starting at the root node and progressing recursively to the leaf nodes. At each node, it selects a random subset of features, as specified by the `n_features` parameter, and employs the `_find_best_split()` method to determine the optimal feature and threshold for splitting the data. Once the best split is found, the `_split()` method is called to partition the data accordingly. The process is then repeated recursively on the resulting subsets to continue building the tree.

The recursion halts when a stopping condition is met, such as reaching the `max_depth`, achieving pure nodes, having fewer than `min_samples_split` samples per node, or when subsequent splits yield information gains lower than `min_information_gain`. Upon termination, the method assigns the most frequent label among the samples at that node as the predicted label for that node.

_find_best_split() Method:

The `_find_best_split()` method identifies the optimal split for a given node in the decision tree. It iterates through all features selected by the `_build_tree()` method and evaluates all candidate thresholds by calling the `_calculate_information_gain()` method to determine the feature-threshold combination that maximizes information gain. The creation of candidate thresholds differs between numerical and categorical features:

- For numerical features, potential thresholds are determined based on the `n_quantiles` parameter (as described above).
- For categorical features, candidate thresholds consist of all unique values in the feature.

Any missing values are excluded when determining the candidate thresholds.

_split() Method:

The `_split()` method partitions the data based on a specified feature and threshold. The partitioning strategy differs for numerical and categorical features:

- For numerical features, data points with a feature value lower than or equal to the threshold are assigned to the left child, while all other data points are assigned to the right child.
- For categorical features, the partitioning depends on the `isolate_one` parameter (as described above).

After the split, any data points with a missing feature value are randomly distributed between the left and right child. The probability of being assigned to each child is proportional to the number of data points assigned to that child during the split.

_traverse_tree() Method:

The `_traverse_tree()` method traverses the decision tree to predict the label for a single instance. Starting at the root node, it follows the tree's decision rules until it reaches a leaf node. If the feature value at the current node is missing, the method randomly decides whether to move to the left or right child, based on the ratio of data points assigned to each child.

3 Training and Validation Procedures

Training a model on the training set and testing it on the test set is a fundamental practice in machine learning. The goal is to develop a model that generalizes well to new, unseen data, and not just memorizes the data seen during training (a phenomenon known as overfitting).

In this study, a set of custom functions has been implemented to facilitate robust model training, validation, and hyperparameter tuning.

Functions:

- `k_fold_partition()`: This function divides the data into k folds for k-fold cross-validation. The number of folds is specified by the user (the default is 5).
- `k_fold_cv_estimate()`: This function uses the `k_fold_partition()` function to create k folds and trains the model k times, each time using $k-1$ folds for training and the remaining fold for testing. The average test error across all iterations (i.e., the cross-validation estimate) is returned.
- `hyperparameter_tuning()`: This function iterates through all parameter combinations generated by the `_parameter_combinations()` function, or a random subset if specified, and uses the `k_fold_cv_estimate()` function to compute the cross-validation estimate for a model with these parameters. The lowest cross-validation estimate and the corresponding parameters are then returned.
- `k_fold_nested_cv()`: This function implements nested cross-validation. The `k_fold_partition()` function is used to create k folds and the model is trained k times. Each time, the `hyperparameter_tuning()` function is invoked on $k-1$ folds and the model with the best parameters is tested on the remaining fold. Then the average test error across all iterations is returned.
- `_parameter_combinations()`: This helper function generates all possible combinations of hyperparameters from a specified grid.
- `accuracy_metric()`: This function computes the accuracy of the model's predictions by calculating the proportion of correct predictions out of all predictions.
- `precision_metric()`: This function computes precision, defined as the ratio of true positive predictions to the total predicted positives.
- `recall_metric()`: This function computes recall, defined as the ratio of true positive predictions to the total actual positives.
- `f1_metric()`: This function computes the F1 score, the harmonic mean of precision and recall.

4 Dataset Description

The dataset used in this study is a simulated version inspired by the Mushroom Data Set. It contains 61,069 hypothetical mushrooms, each described by 20 features and classified as either definitely edible or definitely poisonous/of unknown edibility. The class distribution is balanced, with 27,181 mushrooms (44.51%) classified as edible and 33,888 (55.49%) as poisonous, ensuring no class is overrepresented in subsequent analyses.

During preprocessing, 146 duplicate rows (0.24%) were identified and removed to avoid undue weighting of observations. This step reduced the dataset to 60,923 mushrooms, consisting of 27,181 (44.62%) edible and 33,742 (55.38%) poisonous instances. Additionally, nine variables were found to contain missing values, as summarized in Table 1.

Upon examining the possible values for each variable, some unexpected values not defined by the dataset's author were observed, such as "d" for cap-surface and "f" for stem-root. A comprehensive overview of all 21 variables, their types, and possible values is provided in Table 2.

Notably, several gill- and stem-related variables include "f" as a possible value, signifying "none". This raised a question about whether "f" represents a valid value or missing data. Analysis revealed that if any gill-related

Table 1: Missing Values Count and Percentage

	Missing Values	
Variable	**Count**	**Percentage**
cap-surface	14 120	23.12
gill-attachment	9855	16.14
gill-spacing	25 062	41.04
stem-root	51 536	84.39
stem-surface	38 122	62.42
veil-type	57 746	94.56
veil-color	53 510	87.62
ring-type	2471	4.05
spore-print-color	54 597	89.40

Table 2: 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), d (not specified by the author)
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-or-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)
gill-spacing	categorical	c (close), d (distant), f (none)
gill-color	categorical	n (brown), b (buff), g (gray), r (green), p (pink), u (purple), e (red), w (white), y (yellow), o (orange), k (black), 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), r (rooted), f (not specified by the author)
stem-surface	categorical	i (fibrous), y (scaly), s (smooth), h (shiny), k (silky), t (sticky), f (none), g (not specified by the author)
stem-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), f (none)
veil-type	categorical	u (universal)
veil-color	categorical	n (brown), u (purple), e (red), w (white), y (yellow), k (black)
has-ring	categorical	t (ring), f (none)
ring-type	categorical	e (evanescent), r (flaring), g (grooved), l (large), p (pendant), z (zone), m (movable), f (none)
spore-print-color	categorical	n (brown), g (gray), r (green), p (pink), u (purple), w (white), k (black)
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)

variable has a value of "f", all other gill-related variables for that mushroom also have the value "f". A similar pattern was found for stem-related variables. Specifically, 3,414 mushrooms lacked gill-related information, and 915 lacked stem-related information. These 915 mushrooms also had stem-height and stem-width equal to 0, confirming the consistency of this pattern.

Further investigation, informed by domain knowledge, clarified that some mushrooms naturally lack gills or stems. This observation suggests that the "f" values reflect genuine biological characteristics rather than missing data. Consequently, these values were treated as valid in subsequent analyses.

Lastly, the veil-type variable was examined for its potential utility in predictive modeling. This variable only takes the value "u", but it was missing for the majority (94.56%) of observations. Additionally, when veil-type was missing, both classes (edible and poisonous) were represented, indicating that the missingness did not provide meaningful predictive information for classification. Given its lack of variability and predictive utility, the veil-type variable was excluded from further analysis.

5 Model Training and Evaluation

The implemented `DecisionTreeClassifier` class is capable of handling missing values, so no columns or rows were removed from the dataset in the initial analysis. To identify the best hyperparameters, nested cross-validation with 5 folds for both the outer and inner loops was performed using the following parameter grid:

```
parameter_grid = { 'min_samples_split': [2, 5, 10, 20], 'max_depth': [5, 10, 15, 20, None], 'n_feature
```

Evaluating all possible parameter combinations would have required testing 4,320 combinations. To reduce

computational cost, 50 parameter combinations were randomly selected for evaluation. The results of the nested cross-validation were as follows:

- **Mean Test Error:** 0.0094
- **Mean Accuracy:** 0.9906
- **Mean Precision:** 0.9929
- **Mean Recall:** 0.9901
- **Mean F1 Score:** 0.9915

The hyperparameters corresponding to the minimum test error were:

- `min_samples_split`: 10
- `max_depth`: 20
- `n_features`: None
- `criterion`: square_root
- `min_information_gain`: 0.0
- `n_quantiles`: 10
- `isolate_one`: False

The best model achieved the following metrics:

- **Accuracy:** 0.9916
- **Precision:** 0.9925
- **Recall:** 0.9922
- **F1 Score:** 0.9923

Subsequently, an alternative approach was tested by removing features with more than 40% missing values (`veil-color`, `spore-print-color`, `stem-root`, `stem-surface`, and `gill-spacing`) and dropping rows with any remaining missing values. This preprocessing step resulted in a dataset containing 36,948 observations, 14 features, and the following class distribution:

- **Edible:** 16,944 (45.86%)
- **Poisonous:** 20,004 (54.14%)

The same nested cross-validation procedure (5 folds for both loops) and parameter grid were applied, with 50 randomly selected parameter combinations evaluated. The results on the cleaned dataset were as follows:

- **Mean Test Error:** 0.0032
- **Mean Accuracy:** 0.9968
- **Mean Precision:** 0.9972
- **Mean Recall:** 0.9970
- **Mean F1 Score:** 0.9971

The best hyperparameters for the cleaned dataset matched those found previously:

- `min_samples_split`: 10
- `max_depth`: 20
- `n_features`: None
- `criterion`: square_root
- `min_information_gain`: 0.0

- `n_quantiles`: 10
- `isolate_one`: False

The performance metrics for the best model on the cleaned dataset were:

- **Accuracy:** 0.9970
- **Precision:** 0.9975
- **Recall:** 0.9970
- **F1 Score:** 0.9973