Tree predictors for binary classification

Michele Attilio Iodice
Università degli studi di Milano
Via Celoria 18, I-20133,Milan, Italy
Email: micheleattilio.iodice@studenti.unimi.it

October 19, 2024

Abstract

This paper presents the implementation of decision tree for binary classification applied on the Mushroom Dataset to determine whether mushrooms are poisonous. The tree predictors use single-feature binary tests as the decision criteria at any internal node. It was performed hyperparameter tuning on the tree predictors. By reusing the already implemented tree predictor class/structure a Random Forest classifiers was implemented. The implementation is carried out using a custom 'TreePredictor' class, with different splitting and stopping criteria. We further explore the effect of various hyperparameters, such as 'max depth', on model performance. The paper also discusses techniques for efficient hyperparameter tuning, the results of the training and test of the models and the importance of model evaluation on a validation set.

1 Introduction

Decision trees are widely used machine learning models due to their simplicity, interpretability, and versatility. They are fundamental components of ensemble methods like Random Forests. This paper presents an efficient implementation of decision trees using a 'TreePredictor' class and 'TreeNode' class created from scratch. The tree predictors are trained adopting four criteria for the expansion of the leaves, such as:

- Gini Index
- Scaled Entropy
- Information Gain

- Mean Squared Error

and 3 stopping criteria such as:

- `Max Depth` – the maximum depth of the tree.
- `Min Samples Split` – the minimum number of samples needed to split a node.
- `Min Impurity Threshold` – the minimum impurity decrease needed to split.

The training error of each tree predictor was computed according to the 0-1 loss. The Evaluation was computed using the K-Fold Cross Validation method. After it has been examined the effect of hyperparameter tuning on model performance.
By reusing the tree predictor class/structure, with the best hyperparameter, has been implemented, trained and evaluated a Random Forests classifier. The Experiments were conducted using the Mushroom dataset. Experiments show that the Tree Predictor performances are about 68% of test accuracy and respectively about 30% and 31% of training and evaluation error. In the rest of the report, in section 2 we will provide insights into the implementation of the predictors. Section 3, presents the dataset used, the analysis of the training carried out and are explained the tests and evaluations results obtained. In section 4, some conclusions will be drawn, and some future work will be proposed.

2 Methodology

The main task of this work is the implementation from scratch of tree predictors for binary classification to determine whether mushrooms are poisonous. Below, the methodology applied, the classes and functions implemented and the tuning process of the hyperparameters will be described.

2.1 Decision Tree Implementation

First, was implemented a basic class/structure for the nodes of the tree predictors. The class will represent a decision tree node and will have the following components:

- `Constructor:` – Initializes the node with default or provided values.

- `set_leaf:` – Converts the current node into a leaf, assigning a class label (either p for poisonous or e for edible).
- `predict method:` – This method traverses the tree based on the decision function at each node and returns the predicted class for a given input data point.

After the decision tree is implemented using the 'TreePredictor' class. It recursively splits the dataset based on the chosen splitting criterion (e.g., Gini, entropy, MSE). Until the chosen stopping conditions is fulfilled. The tree predictor will have the following components:

- `Constructor:` Initializes the tree predictor and accepts information about decision criteria for each feature.
- `Splitting Criterion:` Implements the logic for selecting the best split.
- `Stopping Criterion:` Defines the conditions to halt tree growth.
- `Training Procedure:` Trains the tree using the provided training set.
- `Evaluation Procedure:` Evaluates the performance of the trained tree on a test set.

Algorithm 1 summarizes the tree-growing process.

Algorithm 1 Recursive Tree Growth

```
Input: Training data X, Labels y, Depth d
Output: Root node of the decision tree
if Stopping criterion met then
    Create a leaf node with the most common class.
else
    Find the best split criterion using splitting_criterion.
    Split the data into left and right nodes.
    Recursively grow the left and right subtrees.
end if
```

2.1.1 Splitting Criteria

Below are the splitting functions tested in the study

- `Gini Function:` It is a crucial component in decision tree algorithms, particularly for classification tasks. It helps to evaluate the quality of a split based on the Gini impurity measure. Below is a detailed breakdown of the mathematical formulation for the Gini splitting function. For a given node with C unique classes, the Gini impurity G is defined as:

$$G = 1 - \sum_{i=1}^{C} p_i^2 \tag{1}$$

  where:

  - $p_i = \frac{n_i}{N}$ is the proportion of samples belonging to class i,
  - n_i is the number of samples in class i,
  - $N = \sum_{i=1}^{C} n_i$ is the total number of samples in the node.

  The splitting function evaluates the weighted impurity of the child nodes resulting from a split. It is given by the formula:

$$G_S = \frac{N_{\text{left}}}{N} G_{\text{left}} + \frac{N_{\text{right}}}{N} G_{\text{right}} \tag{2}$$

- `Entropy:` It is a key measure to assess the quality of a split, based on the concept of entropy from information theory. It is used to calculate how much "information" is gained when a dataset is split into two child nodes. Here's the mathematical formulation for the entropy splitting function, including a detailed explanation.
  For a node with C unique classes, the entropy H is calculated as:

$$H = -\sum_{i=1}^{C} p_i \log_2(p_i) \tag{3}$$

  where:

  - $p_i = \frac{n_i}{N}$ is the proportion of samples belonging to class i,
  - n_i is the number of samples in class i,
  - $N = \sum_{i=1}^{C} n_i$ is the total number of samples in the node.

The entropy splitting function evaluates the weighted entropy of the child nodes after a split. It is given by the formula:

$$H_S = \frac{N_{\text{left}}}{N} H_{\text{left}} + \frac{N_{\text{right}}}{N} H_{\text{right}} \tag{4}$$

- `Information Gain:` This function is used to measure how much "information" or "purity" is gained after a split. It is based on the concept of entropy and is used to select the feature that maximizes the reduction in uncertainty or disorder (entropy).
  The Information Gain IG is calculated as:

$$IG = H_{\text{parent}} - H_S \tag{5}$$

  Where:

  - H_{parent} is the entropy of the parent node.
  - H_S is the weighted average entropy of the child nodes after the split.

- `Mean Squared Error (for Regression):` This function commonly used in regression trees to evaluate the quality of a split based on the squared differences between the predicted values and the actual values. The goal is to minimize the MSE when determining the best split for the data
  The Mean Squared Error for a set of values y is defined as:

$$\text{MSE}(y) = \frac{1}{N} \sum_{j=1}^{N} (y_j - \bar{y})^2 \tag{6}$$

  Where:

  - N is the number of samples,
  - y_j is the j-th value in the dataset,
  - $\bar{y}$ is the mean of the values in the dataset, given by:

$$\bar{y} = \frac{1}{N} \sum_{j=1}^{N} y_j \tag{7}$$

The splitting function evaluates the quality of a split by calculating the weighted average of the Mean Squared Errors of the child nodes created by the split as expressed bellow:

$$\text{MSE}_S = \frac{N_{\text{left}}}{N}\text{MSE}(y_{\text{left}}) + \frac{N_{\text{right}}}{N}\text{MSE}(y_{\text{right}}) \tag{8}$$

2.1.2 Stopping Criteria

The stopping criteria in decision trees are conditions that determine when to halt the growth of the tree during the training phase. These criteria help prevent overfitting and ensure that the model generalizes well to unseen data. Here's a breakdown of the used stopping criteria:

- **Maximum Depth Reached**:limits how deep the tree can grow. Each level of depth represents a split based on some feature, and limiting the depth helps control the complexity of the model. It's a function that checks whether the current depth of the tree has reached the maximum allowable depth:

$$f_{\text{max\_depth}}(d, D) = \begin{cases} 1 & \text{if } d \geq D \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

  Where:

  - d is the current depth of the tree,
  - D is the maximum allowable depth.

- **Minimum Samples Per Leaf**:This criterion ensures that a leaf node (a terminal node) contains a minimum number of training samples. If a node has fewer samples than this threshold, it will not be split further. The function is defined as bellow:

$$f_{\text{min\_samples}}(n, N) = \begin{cases} 1 & \text{if } n \leq N \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

  Where:

  - n is the number of samples in the current node,
  - N is the minimum number of samples required per leaf.

- **Minimum Impurity Threshold**:This criterion checks whether the impurity (uncertainty) of a node is below a specified threshold. Impurity is measured using entropy metrics. As defined bellow:

$$f_{\text{min\_impurity}}(I, T) = \begin{cases} 1 & \text{if } I \leq T \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

  Where:

  - I is the impurity of the current node (calculated using a method such as entropy),
  - T is the impurity threshold for stopping.

2.2 Hyperparameter Tuning

Hyperparameter tuning is essential for maximizing the model's performance. The hyperparameters to tune in the tree predictor is chosen based by the stopping criteria used. So we performed the hyperparameter for each splitting criteria on a fixed stopping criteria chose apriori.

2.2.1 Using Maximum Depth Reached

The tuning is based on the max_depth parameter (the maximum depth of the decision tree), the values vary depending on the complexity of the dataset and the objective of finding a good compromise between overfitting and underfitting.

Range of Values to Test:

- `Low values (1-5):` Testing with very low values of max_depth will help you understand whether a very simple decision tree can capture the structure of the problem. These values may cause underfitting, but are important to see the performance of a very generalised model.
- `Intermediate values (6-15):` In this range, the tree will be deep enough to capture more complex relationships in the data. This is where the right balance between underfitting and overfitting can be found.
- `High values (>15):` Very deep trees can lead to overfitting, i.e. the model learns the training data too well, but does not generalise well to the test data. However, it is useful to include these values to see where the tree becomes too complex.

2.2.2 Using Minimum Samples Per Leaf

The tuning is based on the min_samples_split parameter determines the minimum number of samples required to split a node. Higher values will lead to larger nodes and greater generalisation, while lower values may lead to more complex patterns.

Range of Values to Test:

- `Low values (2-10):` These values represent the minimum number of samples required to perform a split. 2 is the default value and means that each node can be split if it has at least two samples.
- `Intermediate values (10-50):` These values help control the complexity of the tree and can prevent overfitting.
- `High values (50-100):` Higher values can prevent the tree from overfitting the training data and can be useful for generalisation.

2.2.3 Using Minimum Impurity Threshold

The tuning is based on the impurity_threshold parameter determines the threshold below which a node will not be further subdivided. You can think of this as a way to limit the growth of the tree based on how 'pure' a node must be.

Range of Values to Test:

- `Low values (0.0 - 0.5):` Start with low values to see how purity affects tree height.
- `Intermediate Values (0.5 - 0.9):` Tests values that begin to be more restrictive, leading to fewer subdivisions.
- `High values (0.9 - 1.0):` These values can lead to a simpler shaft and may prevent overfitting.

For each hyperparameter setting and for each splitting criteria, we measured the training and validation error, plotting the errors for comparison. Tuning is essential to obtain a good compromise between model complexity and generalisation capability. Analysing the tuning results allows the performance of the model to be optimised and its reliability to be improved on unseen data

2.3 Random Forest Implementation

Random forests combine multiple decision trees to reduce overfitting. Using the 'TreePredictor' as the base learner, a random forest was implemented by training multiple decision trees on different bootstrap samples of the training data, only a random subset of features is considered. The final prediction is made by aggregating the predictions of individual trees using majority voting.

Algorithm 2 Random Forest Construction

1: **Input:** Training data X, Labels y, Number of trees n
2: **for** each tree i from 1 to n **do**
3: Create a bootstrap sample X_i, y_i from X, y
4: Train a TreePredictor on X_i, y_i
5: **end for**
6: Return the ensemble of trained trees

Each tree votes, and the majority class prediction is returned.
We tuned the following hyperparameters for the random forest:

- `n_trees` - number of trees in the forest.
- `max_features` - number of features to consider for each split.
- `max_Depths` - the max Deph of the tree.

3 Experiments and Results

In this Section are reports the results obtained during the training and testing phase of the tree predictors and random forest models used, and which is the best hyperparameter obtained by the tuning.

3.1 Dataset

The dataset used in this work is the Mushroom dataset. Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981).[1]
This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined

with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like 'leaflets three, let it be' for Poisonous Oak and Ivy. The total number of Instances is 8124, the class is poisonous=p, edibile=e and the number of Attributes is 22 (all nominally valued). There are missing attribute values denoted by '?'. The Class Distribution is of edible: 4208 (51.8%), poisonous: 3916 (48.2%)

3.2 Evaluation Metrics

The models were evaluated using different metrics. For the evaluation of the training of tree predictor and random forest has been used the 0-1 loss evaluation metrics. It is a simple evaluation metric used in classification tasks to measure how often a classifier makes an incorrect prediction. The mathematical formula for a single prediction is:

For a dataset with multiple samples (n samples), the average 0-1 loss is computed as:

$$L_{\text{avg}} = \frac{1}{n}\sum_{i=1}^{n} L_i(y, \hat{y}) \quad with \quad L_i(y, \hat{y}) = \begin{cases} 0, & \text{if } y = \hat{y} \\ 1, & \text{if } y \neq \hat{y} \end{cases} \tag{12}$$

Where:

- n is the total number of samples.
- y is the true label (ground truth),
- $\hat{y}$ is the predicted label by the classifier.

For the testing evaluation has been used the accurancy metrics. The mathematical formula for accuracy in a classification problem is the ratio of correctly predicted labels to the total number of predictions made. It is given by:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{13}$$

3.3 Training Analysis

First, split the dataset into three parts as follows:

- `Training and Validation set:` Used to train and tune the model.
- `Test set:` Used to evaluate the final performance of the model after training.

Data was splitted into a training + validation set (80%) and a test set (20%). On the training and validation set using a k-fold kross validation the data was splitted into k=5 subsets (folds). The tree predictor was trained on k-1 folds and evaluate on the remaining fold. Repeat this process k times, each time with a different validation fold. At the end average the results to get a more reliable estimate of model performance.
Bellow an example of the tree predictor training results using max depth reached as stopping criteria and Gini index as splitting criteria.

Figure 1: Training an Validation error report

During the training phase, several predictor trees were trained, using the 4 splitting criteria and 3 stopping criteria defined in the previous sections, based on the this has been builted twelve tree predictors which obtained a training and validation error value of between 31% and 32% for training and between 30% and 31% for validation respectively.
The hyperparameter tuning was performed on the predictor trained with Max Depths Reached as stopping criteria and Gini Index as splitting criteria, tune on the parameter max-Depths in the range value as described in the previous section 'Hyperparameter Tuning'
Below, the plot showing the training and validation error for each value of the max depths parameter; The best param value is 6 based that obtain

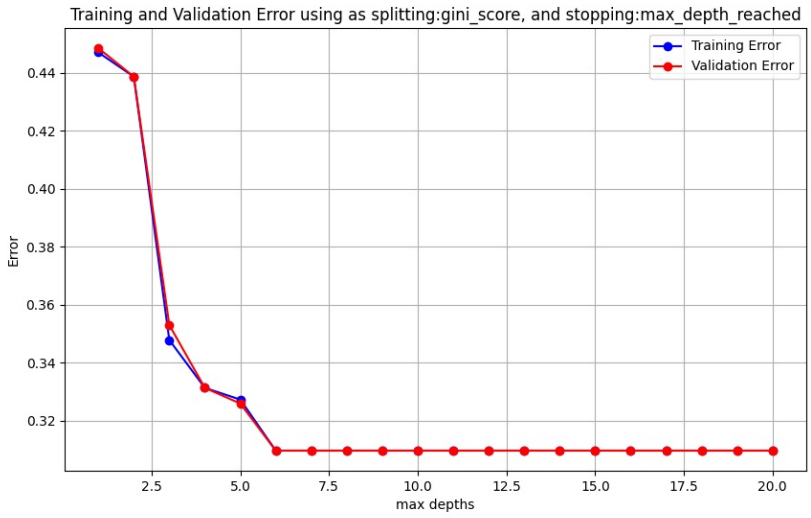

Figure 2: Training an Validation error report

validation error of 31% the predictor then was tested on this parameter on the test set obtain a Test Accuracy of 68%

In the training of Random Forest model, the choice of hyperparameters can significantly influence its performance, so for tuning the hyperparameters n_trees, max_features, and max_Depths a specific range of values is chosen for each parameter:

- `Number of Trees (n_trees):` controls how many decision trees are built in the forest. More trees tend to improve performance, but also increase computational cost,range to test: [100, 200, 500].
- `Maximum Features (max_features):`This hyperparameter controls how many features are considered when looking for the best split at each node. The smaller the number of features considered, the more diverse the trees in the forest will be, range to test: [sqrt, log2, 0.5 x total_features].
- `Max Depths (max_Depths):` This parameter controls the max depth of the tree. It affects how deep the trees grow, range to test:[2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20].

The best validation error obtained by the random forest is about 0.06%

3.4 Test and Evaluation of the Proposed Algorithms

3.4.1 Tree Predictor

After tuning, the tree predicotrs were evaluated on the test set. The best decision tree achieved an accuracy of `68`%, Following the result of evaluation on test set;

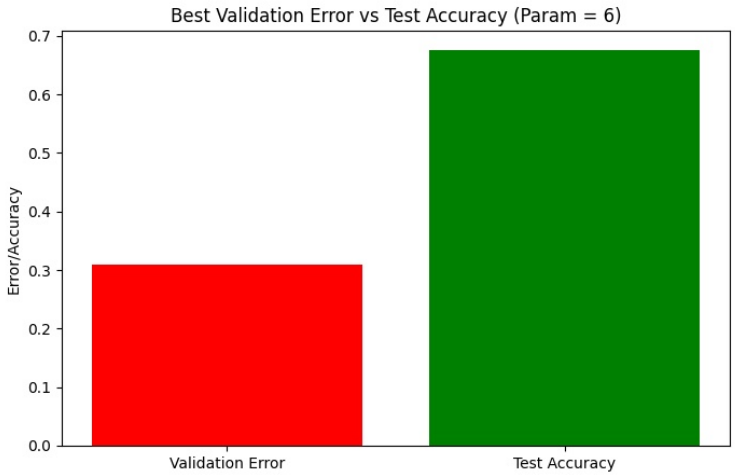

Figure 3: Test Accuracy report

However, all predictors achieved an accuracy of between 60% and 68%.

Tree Predictor	**Test Accuracy (%)**	**Training Time (s)**
Max Depths and Gini Index	68	0.87
Max Depths and Entropy	68	0.87
Max Depths and Information Gain	68	0.87
Max Depths and MSE	67	0.87
Min Samples and Gini Index	68	3.73
Min Samples and Entropy	66	3.73
Min Samples and Information Gain	66	3.73
Min Samples and MSE	66	3.73
Min Impurity and Gini Index	60	3.77
Min Impurity and Entropy	61	3.77
Min Impurity and Information Gain	62	3.77
Min Impurity and MSE	61	3.77

Table 1: Test accuracy and training time comparison between the Tree Predictors.

3.4.2 Random Forest

The Random Forest model achieved a higher test accuracy of 96.5%, outperforming the single Tree Predictor. The model also demonstrated improved generalization, with the gap between training and validation performance significantly reduced.

Model	**Test Accuracy (%)**	**Training Time (s)**
Tree Predictor	68	0.87
Random Forest	99.94	1.71

Table 2: Test accuracy and training time comparison between Tree Predictor and Random Forest.

4 Conclusion and Future Works

This paper presents a tree predictor and random forest implementation using a custom 'TreePredictor' class and 'RandomForest' class. We explored the effect of hyperparameter tuning and identified the best set of parameters through a systematic validation process based on the k-fold cross validation and on the 0-1 loss error as metrics.It revealed the impact of 'min_samples_split' and 'max_depth' on performances. Random forests, as expected, provided more robust performance compared to a single tree predictor.
Future work may focus on extending this framework to multi-class classification and comparing with other ensemble methods such as gradient boosting. Furthermore, one could train this framework with other type of data and datasets known in literature.

Acknowledgment

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

References

[1] Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred, A. Knopf, Donor: Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu) Date: 27 April 1987.