Decision Trees

Atta

March 4, 2025

Introduction

- Versatile machine learning models used for both classification and regression tasks.
- Intuitive, easy to interpret.
- Can handle both numerical and categorical data.

Structure of a Decision Tree

A decision tree is a hierarchical model that makes decisions based on a series of questions. It consists of the following elements:

- **Root Node**: Represents the entire dataset.
- **Internal Nodes**: Test a particular attribute and split the data.
- **Branches**: Outcomes of tests, connecting nodes.
- **Leaf Nodes**: Terminal nodes providing the final decision or prediction.

Growing a Decision Tree

The process of growing a decision tree involves recursively splitting the data based on the most informative features. The key steps are:

- **Step 1:** Select the best attribute to split the data.
- **Step 2:** Split the data into subsets.
- **Step 3:** Recursively repeat Steps 1 and 2 on each subset until stopping criteria are met.

Splitting Criteria

The choice of splitting criteria depends on the type of task.

For Classification Tasks

- **Gini Impurity:** Measures the probability of incorrect classification.
- **Entropy:** Measures the amount of information or uncertainty.

For Regression Tasks

- **Mean Squared Error (MSE):** Measures the average squared difference between predicted and actual values.

Advantages and Disadvantages

Advantages

- Easy to understand and interpret.
- Can handle both numerical and categorical data.
- Requires minimal data preparation.
- Can model non-linear relationships.

Disadvantages

- May result in overly complex trees that do not generalize well (overfitting).
- Can be unstable with small variations in the data.
- Biased towards attributes with more levels in classification tasks.

Mathematical Formulations

Gini Index

The Gini index measures the impurity of a node:

$$\text{Gini}(t) = 1 - \sum_{i=1}^{c} p(i|t)^2$$

$$\Delta\text{Gini}(s,t) = \text{Gini}(t) - p_L \cdot \text{Gini}(t_L) - p_R \cdot \text{Gini}(t_R)$$

Where:

- t is the current node.
- c is the number of classes.
- $p(i|t)$ is the proportion of samples belonging to class i at node t.
- s is a potential split.
- t_L and t_R are the left and right child nodes after the split.
- p_L and p_R are the proportions of samples going to the left and right nodes.

Mean Squared Error (MSE)

The MSE measures the error for regression tasks:

$$\text{MSE}(t) = \frac{1}{N_t} \sum_{i \in D_t} (y_i - \bar{y}_t)^2$$

$$\Delta\text{MSE}(s,t) = \text{MSE}(t) - \frac{N_{t_L}}{N_t} \cdot \text{MSE}(t_L) - \frac{N_{t_R}}{N_t} \cdot \text{MSE}(t_R)$$

Where:

- t is the current node.
- N_t is the number of samples at node t.
- D_t is the set of samples at node t.
- y_i is the target value of sample i.
- $\bar{y}_t$ is the mean target value at node t.
- s is a potential split.
- t_L and t_R are the left and right child nodes after the split.
- N_{t_L} and N_{t_R} are the number of samples in the left and right nodes.

CART Objective

The CART algorithm aims to find the best split s^*:

Where $\Delta I(s,t)$ is the improvement in the splitting criterion (e.g., Gini or MSE).

CART Algorithm

The Classification and Regression Trees (CART) algorithm is a popular method for constructing decision trees. Below is the pseudocode for the CART algorithm:

Algorithm 1 CART Algorithm

```
1:  Input: Dataset $D$, Features $F$, Target variable $y$
2:  Output: Decision Tree $T$
3:  procedure BuildTree($D$, $F$, $y$)
4:      if Stopping condition met then
5:          Create a leaf node with the predicted value
6:          return leaf node
7:      end if
8:      Find the best split $(f^*, t^*)$ where $f^* \in F$ and $t^*$ is the threshold
9:      Split the dataset $D$ into $D_{\text{left}}$ and $D_{\text{right}}$ based on $(f^*, t^*)$
10:     Create a decision node with the split $(f^*, t^*)$
11:     $T_{\text{left}} \leftarrow$ BuildTree($D_{\text{left}}, F, y$)
12:     $T_{\text{right}} \leftarrow$ BuildTree($D_{\text{right}}, F, y$)
13:     return decision node with $T_{\text{left}}$ and $T_{\text{right}}$ as children
14: end procedure
15: Initialize: $T \leftarrow$ BuildTree($D, F, y$)
16: Return: $T$
```

Computational Complexity

The computational complexity of decision trees can be expressed as:

$$O(m \cdot n \log n)$$

Where:

- m is the number of features.
- n is the number of examples in the training set.

This complexity arises from:

- Sorting the features: $O(m \cdot n \log n)$.
- Finding the best split at each node: $O(n \cdot m)$.

In the worst case, when the tree becomes unbalanced, the complexity can increase to:

$$O(m \cdot n^2)$$

Gini vs. Entropy

Both Gini index and entropy are valid impurity measures for classification tasks in decision trees, but they have some differences:

- **Computational Efficiency:** Gini index is faster to compute than entropy, as it does not involve logarithms.
- **Sensitivity to Class Imbalance:** Entropy is more sensitive to class imbalance and may perform better for skewed datasets.
- **Tree Structure:** Gini tends to isolate the most frequent class, while entropy produces more balanced trees.
- **Performance:** The choice between Gini and entropy often has little impact on overall performance.
- **Use Cases:** Gini is preferred for binary classification, while entropy is suitable for multi-class problems.

Regularization Hyperparameters

Decision Trees make very few assumptions about the training data, and when left unconstrained, the tree structure will adapt itself to the data, potentially overfitting it. This is often called a nonparametric model. To avoid overfitting, we use regularization techniques, which constrain the tree's freedom during training. Regularization can be controlled by several hyperparameters, which are specific to the algorithm used. Here are some key regularization hyperparameters in Decision Trees:

- **max_depth**: Restricts the maximum depth of the tree. The default value is None, meaning no restriction. Reducing max_depth helps prevent overfitting.
- **min_samples_split**: Specifies the minimum number of samples a node must have before it can be split. Increasing this value helps regularize the model.
- **min_samples_leaf**: Specifies the minimum number of samples that a leaf node must have. This can help avoid overfitting, especially in deep trees.
- **min_weight_fraction_leaf**: Similar to min_samples_leaf, but expressed as a fraction of the total number of weighted instances.
- **max_leaf_nodes**: Restricts the maximum number of leaf nodes in the tree. Reducing this can regularize the model by limiting the tree's complexity.
- **max_features**: Specifies the maximum number of features evaluated for splitting at each node. Reducing this value can also help to regularize the model.

Increasing min_* hyperparameters or reducing max_* hyperparameters can help prevent overfitting and improve generalization.

Instability

Decision Trees some limitations:

- **Sensitivity**: Orthogonal decision boundaries are making them sensitive to the rotation of the training set.

  When the data is rotated, the decision boundary can become unnecessarily complex, potentially affecting generalization.
- **Instability to Small Variations in Data**: Decision Trees can be very sensitive to small changes in the training data. For example, removing a single data point can lead to a significantly different tree structure.
- **Stochastic Nature**: Since the training algorithm used by Scikit-Learn is stochastic, even the same training data can produce different models unless the `random_state` hyperparameter is set to a fixed value.
- **Random Forests** help mitigate this instability by averaging predictions over multiple trees, improving stability and generalization.