Decision Tree

Decision Tree

Decision Trees are a fundamental machine learning algorithm used for both classification and regression tasks. They work by creating a tree like model of decisions that mimics human decision-making process.

The algorithm builds the tree by recursively splitting the data based on feature values that best seperate the target classes. At each node, it asks yes/no question about a feature and splits the data accordingly.

There are two types of decision Trees.

Classification Trees : Predict categorical outcomes

Regression : Predict Continuous values.

So, This is a non parametric model → works of Non linear data

- It is a white box model (explains reasons behind prediction

There are few different types of algorithms internally implemented.

ID3
C4.5
C5.0
CART (Classification and Regression Trees)

We will study cart algo, since sklearn have internally implemented CART Algorithm.

CART – Classification and Regression Trees.

Given training vectors $x_i \in R^n$, i=1,..., l and a label vector $y \in R^l$, a decision tree recursively partitions the feature space such that the samples with the same labels or similar target values are grouped together.

Let the data at node m be represented by Q_m with n_m samples. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ subsets

$$Q_m^{left}(\theta) = \{(x, y) | x_j \leq t_m\}$$
$$Q_m^{right}(\theta) = Q_m \setminus Q_m^{left}(\theta)$$

The quality of a candidate split of node m is then computed using an impurity function or loss function $H()$, the choice of which depends on the task being solved (classification or regression)

$$G(Q_m, \theta) = \frac{n_m^{left}}{n_m} H(Q_m^{left}(\theta)) + \frac{n_m^{right}}{n_m} H(Q_m^{right}(\theta))$$

Select the parameters that minimises the impurity

$$\theta^* = \operatorname{argmin}_\theta G(Q_m, \theta)$$

Recurse for subsets $Q_m^{left}(\theta^*)$ and $Q_m^{right}(\theta^*)$ until the maximum allowable depth is reached, $n_m < \min_{samples}$ or $n_m = 1$.

$G(Q_m, \theta)$ → Ginimpurity function → $1 - \sum_{i=1}^{k} P_k^2$

Where P^k → Probability of each class.

Complete Gini Impurity Decision Tree Guide

Sample Dataset: Tennis Playing Decision

ID	Weather	Humidity	Wind	Play Tennis
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	Normal	Weak	Yes
5	Rain	Normal	Strong	No
6	Overcast	Normal	Strong	Yes
7	Sunny	Normal	Weak	Yes
8	Rain	High	Weak	Yes

Total Distribution: 8 samples → 5 Yes (62.5%), 3 No (37.5%)

Step 1: Understanding Gini Impurity Formula

```
Gini Impurity = 1 - Σ(pi)²
```

Where:

- **pi** = Probability of class i
- **Σ** = Sum over all classes
- **Range:** 0 (perfectly pure) to 0.5 (maximum impurity for binary classification)

Intuition:

Gini measures "How mixed up are the classes in this node?"

- **Gini = 0:** All samples belong to one class (pure)
- **Gini = 0.5:** Equal mix of classes (most impure)
- **Lower Gini = Better separation**

Step 2: Calculate Root Node Gini

Root Node Distribution: 5 Yes, 3 No out of 8 total

Calculate probabilities:

```
P(Yes) = 5/8 = 0.625
```

```
P(No) = 3/8 = 0.375
```

Apply Gini formula:

```
Gini = 1 - (P(Yes)² + P(No)²)
```

```
Gini = 1 - (0.625² + 0.375²)
```

```
Gini = 1 - (0.390625 + 0.140625)
```

```
Gini = 1 - 0.53125 = 0.46875
```

Root Gini Impurity = 0.469

Step 3: Test All Possible Splits

Algorithm Process: Test each feature and find the split that gives maximum Information Gain

A) Split on Weather Feature

Sunny Branch

Samples: 1,2,7

Distribution: 1 Yes, 2 No

```
P(Yes) = 1/3 = 0.333
```

```
P(No) = 2/3 = 0.667
```

```
Gini = 1 - (0.333² + 0.667²)
```

```
Gini = 1 - (0.111 + 0.444) = 0.445
```

Overcast Branch

Samples: 3,6

Distribution: 2 Yes, 0 No

```
P(Yes) = 2/2 = 1.0
```

```
P(No) = 0/2 = 0.0
```

```
Gini = 1 - (1.0² + 0.0²)
```

```
Gini = 1 - 1.0 = 0.0
```

✅ **PURE NODE!**

Rain Branch

Samples: 4,5,8

Distribution: 2 Yes, 1 No

```
P(Yes) = 2/3 = 0.667
```

```
P(No) = 1/3 = 0.333
```

```
Gini = 1 - (0.667² + 0.333²)
```

```
Gini = 1 - (0.444 + 0.111) = 0.445
```

Calculate Weighted Gini for Weather Split:

```
Weighted Gini = (n₁/n × Gini₁) + (n₂/n × Gini₂) + (n₃/n × Gini₃)
```

```
Weighted Gini = (3/8 × 0.445) + (2/8 × 0.0) + (3/8 × 0.445)
```

```
Weighted Gini = 0.167 + 0.0 + 0.167 = 0.334
```

Information Gain:

```
Information Gain = Root Gini - Weighted Gini
```

```
Information Gain = 0.469 - 0.334 = 0.135
```

Weather Split → Information Gain = 0.135

Step 4: Test Other Features

B) Split on Humidity Feature

High Humidity

Samples: 1,2,3,8

Distribution: 2 Yes, 2 No

$$Gini = 1 - (0.5^2 + 0.5^2) = 0.5$$

Normal Humidity

Samples: 4,5,6,7

Distribution: 3 Yes, 1 No

$$Gini = 1 - (0.75^2 + 0.25^2) = 0.375$$

$$Weighted\ Gini = (4/8 \times 0.5) + (4/8 \times 0.375) = 0.4375$$

$$Information\ Gain = 0.469 - 0.4375 = 0.0315$$

Humidity Split → Information Gain = 0.032

C) Split on Wind Feature

Weak Wind

Samples: 1,3,4,7,8

Distribution: 4 Yes, 1 No

$$Gini = 1 - (0.8^2 + 0.2^2) = 0.32$$

Strong Wind

Samples: 2,5,6

Distribution: 1 Yes, 2 No

$$Gini = 1 - (0.333^2 + 0.667^2) = 0.445$$

$$Weighted\ Gini = (5/8 \times 0.32) + (3/8 \times 0.445) = 0.367$$

$$Information\ Gain = 0.469 - 0.367 = 0.102$$

Wind Split → Information Gain = 0.102

Feature	Information Gain	Weighted Gini	Decision
Weather	**0.135**	**0.334**	✅ **BEST SPLIT**
Wind	0.102	0.367	Second best
Humidity	0.032	0.4375	Worst split

🎯 **Algorithm chooses WEATHER as root split**
(Highest Information Gain = 0.135)

⚙️ Complete Algorithm Workflow

1. Initialize: Calculate root node Gini impurity

↓

2. For each feature: Test all possible splits

↓

3. For each split: Calculate weighted Gini of resulting branches

↓

4. Calculate Information Gain: Root Gini - Weighted Gini

↓

5. Choose best split: Maximum Information Gain

↓

6. Repeat recursively: For each impure branch, repeat steps 2-5

↓

7. Stop when: All branches are pure OR stopping criteria met

🏠 MSE Regression Tree: House Price Prediction

📊 Dataset

Size (sqft)	Age (years)	Price ($K)
1200	5	250
1500	10	280
800	15	180
2000	2	400
1800	8	350
1000	20	200

Target: Predict house price (continuous values)

📐 Step 1: Calculate Root MSE

$$MSE = \Sigma(yi - \bar{y})^2 / n$$

Mean price ($\bar{y}$): (250+280+180+400+350+200)/6 = 276.67

Calculate deviations:

- $(250-276.67)^2 = 711.11$
- $(280-276.67)^2 = 11.11$
- $(180-276.67)^2 = 9344.44$
- $(400-276.67)^2 = 15211.11$
- $(350-276.67)^2 = 5377.78$
- $(200-276.67)^2 = 5877.78$

$$MSE = (711.11 + 11.11 + 9344.44 + 15211.11 + 5377.78 + 5877.78) / 6 = 6088.89$$

Root MSE = 6088.89

🔍 Step 2: Test Split on Size ≤ 1400

Size ≤ 1400	**Size > 1400**
Houses: 1200, 800, 1000	Houses: 1500, 2000, 1800
Prices: 250, 180, 200	Prices: 280, 400, 350
Mean: 210	Mean: 343.33
MSE = 822.22	**MSE = 3822.22**

Weighted MSE:

```
(3/6 × 822.22) + (3/6 × 3822.22) = 2322.22
```

MSE Reduction: 6088.89 - 2322.22 = 3766.67

Size split → MSE Reduction = 3766.67

🔄 Step 3: Test Split on Age ≤ 10

Age ≤ 10	**Age > 10**
Ages: 5, 10, 2, 8	Ages: 15, 20
Prices: 250, 280, 400, 350	Prices: 180, 200
Mean: 320	Mean: 190
MSE = 3900	**MSE = 200**

Weighted MSE:

```
(4/6 × 3900) + (2/6 × 200) = 2666.67
```

MSE Reduction: 6088.89 - 2666.67 = 3422.22

Age split → MSE Reduction = 3422.22

🏆 Best Split Decision

Split	MSE Reduction	Decision
Size ≤ 1400	**3766.67**	✅ **BEST**
Age ≤ 10	3422.22	Second

Algorithm chooses Size split (highest MSE reduction)

🎯 Key Differences from Classification

- **Target:** Continuous values (prices) vs. classes
- **Prediction:** Mean of leaf samples vs. majority class
- **Impurity:** MSE vs. Gini/Entropy
- **Split criterion:** Minimize MSE vs. maximize Information Gain

Advantages

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created, and blank values to be removed. Note, however, that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data.
- Can work on non-linear datasets.
- Can give you feature importance.

Disadvantages

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node, or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations. Therefore, they are not good at extrapolation.
- This limitation is inherent to the structure of decision tree models. They are very useful for interpretability and for handling non-linear relationships within the range of the training data, but they aren't designed for extrapolation. If extrapolation is important for your task, you might need to consider other types of models.

Dealing with Overfitting

Pruning is a technique used in machine learning to reduce the size of decision trees and to avoid overfitting. Overfitting happens when a model learns the training data too well, including its noise and outliers, which results in poor performance on unseen or test data. Decision trees are susceptible to overfitting because they can potentially create very complex trees that perfectly classify the training data but fail to generalize to new data. Pruning helps to solve this issue by reducing the complexity of the decision tree, thereby improving its predictive power on unseen data.

There are two main types of pruning: pre-pruning and post-pruning.

1. **Pre-pruning (Early stopping):** This method halts the tree construction early. It can be done in various ways: by setting a limit on the maximum depth of the tree, setting a limit on the minimum number of instances that must be in a node to allow a split, or stopping when a split results in the improvement of the model's accuracy below a certain threshold.
2. **Post-pruning (Cost Complexity Pruning):** This method allows the tree to grow to its full size, then prunes it. Nodes are removed from the tree based on the error complexity trade-off. The basic idea is to replace a whole subtree by a leaf node, and assign the most common class in that subtree to the leaf node.

Pre-Pruning

Pre-pruning, also known as early stopping, is a technique where the decision tree is pruned during the learning process as soon as it's clear that further splits will not add significant value. There are several strategies for pre-pruning:

1. **Maximum Depth:** One of the simplest forms of pre-pruning is to set a limit on the maximum depth of the tree. Once the tree reaches the specified depth during training, no new nodes are created. This strategy is simple to implement and can effectively prevent overfitting, but if the maximum depth is set too low, the tree might be overly simplified and underfit the data.
2. **Minimum Samples Split:** This is a condition where a node will only be split if the number of samples in that node is above a certain threshold. If the number of samples is too small, then the node is not split and becomes a leaf node instead. This can prevent overfitting by not allowing the model to learn noise in the data.
3. **Minimum Samples Leaf:** This condition requires that a split at a node must leave at least a minimum number of training examples in each of the leaf nodes. Like the minimum samples split, this strategy can prevent overfitting by not allowing the model to learn from noise in the data.
4. **Maximum Leaf Nodes:** This strategy limits the total number of leaf nodes in the tree. The tree stops growing when the number of leaf nodes equals the maximum number.
5. **Minimum Impurity Decrease:** This strategy allows a node to be split if the impurity decrease of the split is above a certain threshold. Impurity measures how mixed the classes within a node are. If the decrease is too small, the node becomes a leaf node.
6. **Maximum Features:** This strategy considers only a subset of features for deciding a split at each node. The number of features to consider can be defined and this helps in reducing overfitting.

Advantages of Pre-Pruning:

- **Simplicity:** Pre-pruning criteria such as maximum depth or minimum number of samples per leaf are easy to understand and implement.
- **Computational Efficiency:** By limiting the size of the tree, pre-pruning can substantially reduce the computational cost of training and prediction.
- **Reduced Overfitting:** By preventing the tree from becoming overly complex, pre-pruning can help avoid overfitting the training data and thereby improve the model's generalization performance.
- **Improved Interpretability:** Simpler trees (with fewer nodes) are often easier for humans to interpret.

Disadvantages of Pre-Pruning:

- **Risk of Underfitting:** If the stopping criteria are too strict, pre-pruning can halt the growth of the tree too early, leading to underfitting. The model may become overly simplified and fail to capture important patterns in the data.
- **Requires Fine-Tuning:** The pre-pruning parameters (like maximum depth or minimum samples per leaf) often require careful tuning to find the right balance between underfitting and overfitting.
- **Short Sightedness:** Can prune good nodes if they come after a bad node.

Post-Pruning

Post-pruning, also known as backward pruning, is a technique used to prune the decision tree after it has been built. There are several strategies for post-pruning:

1. Cost Complexity Pruning (CCP): Also known as Weakness Pruning, this technique introduces a tuning parameter (α) that trades off between tree complexity and its fit to the training data. For each value of α, there is an optimal subtree that minimizes the cost complexity criterion. The subtree that minimizes the cost complexity criterion over all values of α is chosen as the pruned tree.
2. Reduced Error Pruning: In this method, starting at the leaves, each node is replaced with its most popular class. If the accuracy is not affected in the validation set, the change is kept.

Advantages of Post-Pruning:

1. Reduced Overfitting: Post-pruning methods can help to avoid overfitting the training data, which can lead to better model generalization and thus better performance on unseen data.
2. Preserving Complexity: Unlike pre-pruning, post-pruning allows the tree to grow to its full complexity first, which means it can capture complex patterns in the data before any pruning is done.
3. Better Performance: Post-pruned trees often outperform pre-pruned trees, as they are able to better balance the bias-variance trade-off.

Disadvantages of Post-Pruning:

1. Increased Computational Cost: Post-pruning can be more computationally intensive than pre-pruning, as the full tree must be grown first before it can be pruned.
2. Requires Validation Set: Many post-pruning methods require a validation set to assess the impact of pruning. This reduces the amount of data available for training the model.
3. Complexity of Implementation: Post-pruning methods, especially those involving tuning parameters (like cost complexity pruning), can be more complex to implement and understand than pre-pruning methods.
4. Risk of Underfitting: Similar to pre-pruning, if too much pruning is done, it can lead to underfitting where the model becomes overly simplified and fails to capture important patterns in the data.