Decision Tree from Scratch

Bhanu Prasanna Koppolu

Table of contents

> **Note**
>
> Full implementation available at GitHub - ML from Scratch

Decision Tree

The main concept for decision tree is to split the tree into binary cases. Each node has 2 branches. There is only 1 root node. The final nodes are called the **leaf nodes** which is our classification/regression output.

Key Concepts

The main concepts here are **Information Gain**, **Entropy**, and **Gini Index**.

Gini Index, Entropy, and Information Gain are all measures of impurity or disorder in decision tree algorithms to decide how to split a node.

Entropy

Entropy is a measure of uncertainty or randomness in a node. A perfectly pure node (all data points belong to one class) has an entropy of 0, while a node with an even distribution of all classes has the highest entropy.

$$\text{Entropy}(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

where c is the number of classes, and p_i is the proportion of examples in class i.

Gini Index

Gini Index is another measure of impurity or disorder. It calculates the probability of incorrectly picking a random element if it were randomly labeled according to the distribution of labels in the subset. A Gini index of 0 means perfect purity, while Gini index of 1 means all elements are impure.

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c} p_i^2$$

where c is the number of classes, and p_i is the proportion of examples in class i.

Information Gain

Information Gain is the reduction in impurity achieved by splitting a dataset on a particular attribute. It is the difference between the impurity of the parent node and the weighted average of the impurities of the child nodes.

$$\text{Information Gain}(S) = \text{Entropy}(S) - \sum_{j=1}^{v} \frac{|S_j|}{|S|} \cdot \text{Entropy}(S_j)$$

$$\text{Information Gain}(S) = \text{Gini}(S) - \sum_{j=1}^{v} \frac{|S_j|}{|S|} \cdot \text{Gini}(S_j)$$

Decision tree algorithms choose the attribute that yields the highest information gain for the next split, as this is the most effective at separating the data into pure subsets.

Understanding v

v is specific to a feature, not all features.

Let's take an example: I have a dataset for house prices with 5 features: Area, Location, number of bedrooms, washrooms, and living room area. The goal is to predict the price. Now, v is not 5 because v is specific to a feature. For example, the Area (SQFT), so v can be > 2000 and ≤ 2000 (representing $v = 2$).

The decision tree splits it binary. This is the **CART** (Classification and Regression Trees). But there are others like C4.5, ID3, which allow multi-way splits. sklearn only implements CART with different criterions: entropy, gini, and log-loss. **Gini is faster because no computation of log**.

Cardinality

- $|S|$ (the cardinality of S): Total count of all data points in the current parent node before the split.
- $|S_j|$ (the cardinality of S sub j): Count of data points that go down the j-th branch after the split.
- The ratio $\frac{|S_j|}{|S|}$ is a simple fraction or probability, always between 0 and 1.

Decision Tree Nodes

The nodes are just simple logical gates. They contain only the information necessary to sort a data point into the correct child node and to record the statistics about the data subset that ended up there.

Node Structure

```
class Node:
    """Represents a node in the decision tree."""

    def __init__(self, feature_index=None, threshold=None, left=None,
                 right=None, info_gain=None, value=None):
        self.feature_index = feature_index  # Index of feature to split on
        self.threshold = threshold          # Threshold value for the split
        self.left = left                    # Left child node (<=threshold)
        self.right = right                  # Right child node (>threshold)
        self.info_gain = info_gain          # Information gain from this split
        self.value = value                  # Leaf value (class label or mean)
```

Decision Tree Classification

```
class DecisionTreeClassification:

    def __init__(self, max_depth=50, min_samples_split=5, impurity_type="gini"):
        self.max_depth = max_depth           # Maximum depth of tree
        self.min_samples_split = min_samples_split  # Min samples to split
        self.impurity_type = impurity_type   # "gini" or "entropy"

    def _entropy(self, p_k):
        """Calculate entropy: H(S) = -sum(p_k * log2(p_k))"""
        return -1 * np.sum(p_k * np.log2(p_k))

    def _gini(self, p_k):
        """Calculate gini index: G(S) = 1 - sum(p_k^2)"""
        return 1 - np.sum(p_k ** 2)

    def _build_tree(self, S, depth):
```

```
        """Recursively build tree by finding best splits."""
        # Returns leaf node if: pure node, max depth reached, or min samples
        # Otherwise finds best feature/threshold and splits recursively
        pass

    def fit(self, X_train, y_train):
        """Build decision tree from training data."""
        self.root_node = self._build_tree(S=np.arange(X_train.shape[0]), depth=0)

    def predict(self, X):
        """Traverse tree for each sample to get predictions."""
        # For each sample, start at root and follow left/right based on threshold
        pass
```

Decision Tree Regression

There is not a lot of difference between the classification and regression version:

- Instead of class labels we have **values**
- Instead of gini/entropy we have **Variance** or **Sum of Squared Errors (SSE)** - both use Mean
- At leaf node you store the **mean of** y_i, not class

```
class DecisionTreeRegressor:

    def __init__(self, max_depth=50, min_samples_split=5, impurity_type="variance"):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.impurity_type = impurity_type   # "variance" or "sse"

    def _build_tree(self, S, depth):
        """Same as classification but uses variance/SSE instead of gini/entropy."""
        # Leaf value = mean of y values in the node
        pass

    def fit(self, X_train, y_train):
        """Build regression tree from training data."""
        pass

    def predict(self, X):
```

```
        """Traverse tree to get predicted values (means)."""
        pass
```

For variance:

$$I = \text{Var}(y_S)$$

For SSE:

$$I = \text{Var}(y_S) \times n$$

where n is the number of samples.