New Package Introduction - Decision Trees

**Decision Tree**

A decision tree is a non-parametric supervised learning approach that can be used for classification as well as regression problems. It has a tree structure that is hierarchical and consists of a root node, branches, internal nodes, and leaf nodes.

**Decision Tree Regression**

sklearn.tree.DecisionTreeRegressor(\*, criterion='squared\_error', splitter='best', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, ccp\_alpha=0.0)[source]

**Example**

from sklearn.tree import DecisionTreeRegressor  
dtree\_regressor = **DecisionTreeRegressor()**  
dtree\_regressor.fit(x\_train**,** y\_train)

You can refer to the decision tree regressor sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html).

Similarly, we can implement the decision tree classifier through the sklearn library.

**Decision Tree Classification**

sklearn.tree.DecisionTreeClassifier(\*, criterion='gini', splitter='best', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, class\_weight=None, ccp\_alpha=0.0

**Example**

from sklearn.tree import DecisionTreeClassifier  
dtree\_classifier = **DecisionTreeClassifier()**  
dtree\_classifier.fit(x\_train, y\_train)

You can refer to the decision tree regressor sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html).

**Label Encoding**

Sklearn provides a very efficient tool for encoding the levels of categorical features into numeric values. LabelEncoder encodes labels with a value between 0 and n\_classes-1, where n is the number of distinct labels. If a label repeats, it assigns the same value as assigned earlier.

**Example**

from sklearn.preprocessing import LabelEncoder  
ln = **LabelEncoder()**  
ln.fit\_transform(data[‘column’]) # **column** should be any categorical variable

You can refer to the label encoder sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#:~:text=LabelEncoder%20can%20be%20used%20to%20normalize%20labels.&text=It%20can%20also%20be%20used,and%20comparable)%20to%20numerical%20labels.).

Happy Learning!

Additional Notes from Gemini (10-Jul-2025):

Of course! The DecisionTreeClassifier in scikit-learn uses a criterion to measure the quality of a split. Here are the different criteria and what they mean.

### Criteria for Decision Trees

The criterion parameter determines the function used to evaluate how well a split separates the data into different classes. The goal is to choose splits that result in the purest possible nodes.

#### 1. Gini Impurity (criterion='gini')

* **Meaning**: The **Gini impurity** measures the probability of incorrectly classifying a randomly chosen element from a dataset if it were randomly labeled according to the distribution of labels in the subset. A Gini impurity of **0** is the best possible score, meaning the node is completely pure (i.e., all samples in the node belong to a single class).
* **Formula**: The Gini impurity for a set of items with J classes is calculated as:

Gini=1−i=1∑J​(pi​)2

where pi​ is the fraction of items belonging to class i.

* **Use Case**: This is the default criterion because it's computationally efficient and works very well in practice.

#### 2. Entropy / Information Gain (criterion='entropy')

* **Meaning**: **Entropy** is a concept from information theory that measures the level of disorder or uncertainty in a node. A node with high entropy is very mixed, while a node with low entropy is more pure. The algorithm tries to maximize the **information gain** at each split, which is the reduction in entropy. An entropy of **0** indicates a perfectly pure node.
* **Formula**: The entropy for a set of items with J classes is calculated as:

Entropy=−i=1∑J​pi​log2​(pi​)

where pi​ is the fraction of items belonging to class i.

* **Use Case**: Using entropy can sometimes lead to more balanced trees, but it is computationally more expensive than Gini impurity because it requires calculating a logarithm.

#### 3. Log Loss (criterion='log\_loss')

* **Meaning**: This is another name for **cross-entropy**. When criterion='log\_loss' is used, the model effectively uses the same entropy calculation as above. It's simply an alias for 'entropy'. So, choosing 'log\_loss' is identical to choosing 'entropy'.

### Key Differences

| Feature | **Gini Impurity** | **Entropy** |
| --- | --- | --- |
| **Calculation** | Faster, as it doesn't involve logarithms. | Slower, due to the logarithm calculation. |
| **Tree Structure** | Tends to isolate the most frequent class. | Tends to produce slightly more balanced trees. |
| **Output** | In practice, the final results are often very similar to those from entropy. | Often leads to very similar tree structures and performance as Gini. |