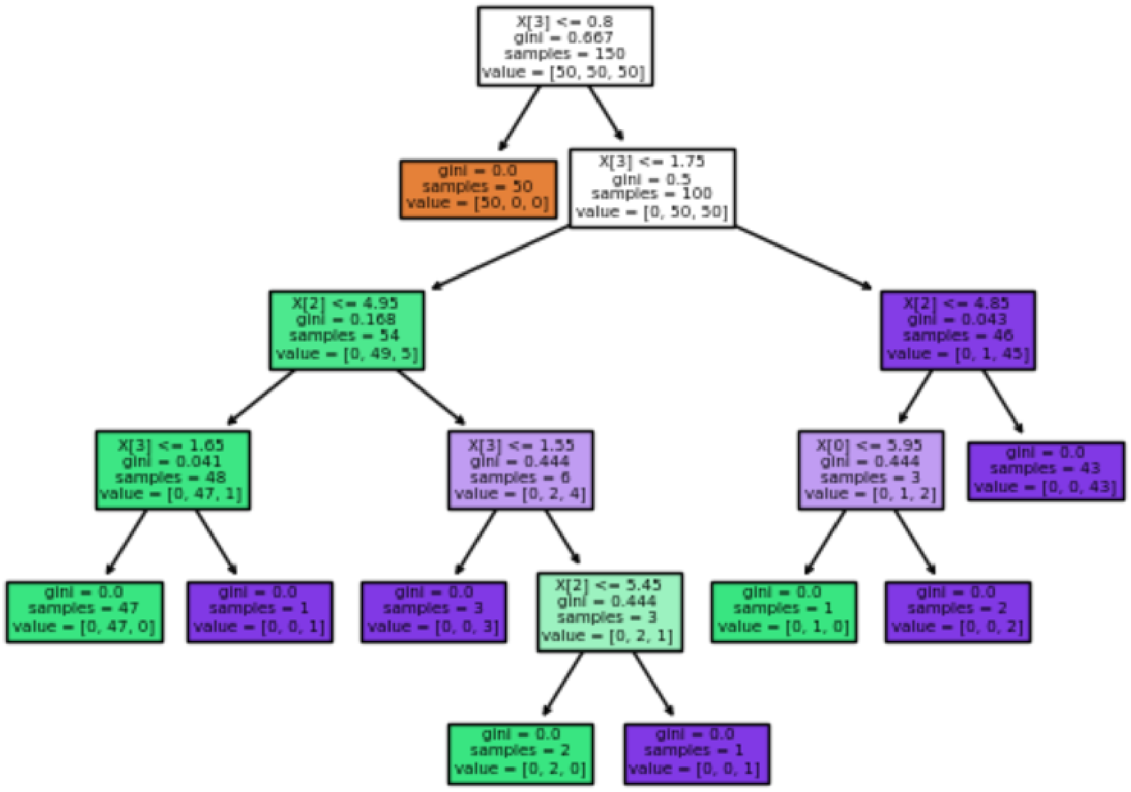
Comparison between the sklearn decision trees and the theory

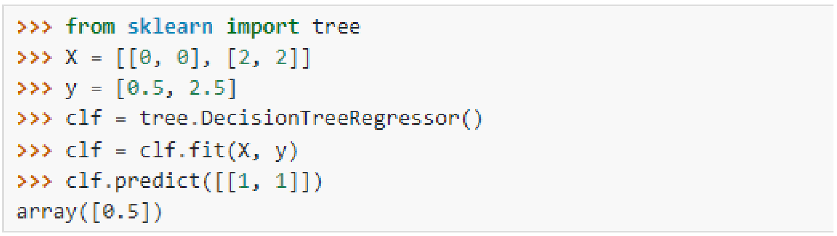
Interpretation

We have discussed that the possibility to interpret the way that DT makes the decision is one of the greatest advantages of this method. This allows us, not only to get the final result, but to understand why that result is chosen. This ease of interpretation makes the visualization of the algorithm possible. Sklearn software has multiple ways of visualizing the tree. We could use the **plot_tree** function or export the tree in **Graphviz** format. In the picture below, we can see the visualization example from the sklearn documentation by using the plot_tree function.



Usages

In class, we have focused on using the tree for binary or multiclassification. Sklearn supports both of these usages. However, the implementation of sklearn can also be used for regression, using the **DecisionTreeRegressor** class. This is an important difference from what we have discussed, and it is certainly worth mentioning. The main difference between using the sklearn decision tree for classification and regression is in the second parameter. The second parameter is the vector of labels used for training. In the case of classifications, it will hold the integer values, and in case of the regression float point values. In the picture below, there is a simple example from the documentation of using the decision tree for the regression problem.



```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([0.5])
```

Overfitting

Sklearn deals with overfitting by using pruning, setting the minimum number of samples at a leaf node or setting the maximum depth of the tree. We have talked about all these methods in our lectures. The specific algorithm that sklearn uses for pruning is called **Minimal Cost-Complexity Pruning** (details in the documentation). Also, using the parameters such as **min_samples_split** or **min_samples_leaf** can help in solving this problem.

Data

We have seen that DTs can handle both categorical and numerical data. In fact, we have discussed some ways on how to deal with numerical data and transform them to the categorical ones. However, the sklearn implementation of DTs doesn't support the categorical variables as the input, so only the numerical variables can be used.

Tree algorithm

In class, we have been presented with the **ID3** algorithm, which creates a tree in a greedy manner, choosing the attribute that will yield the largest Information Gain. The problem is that the ID3 finds the best (by the aforementioned criteria) categorical attribute, and the sklearn implementation works with numerical data. **C4.5** algorithm solves this problem by dynamically defining a discrete attribute that partitions the continuous attribute value into the set of intervals. We have discussed how to do this in class, we can set the threshold for splitting the intervals in such a way that it maximizes the Information Gain. **CART** algorithm is very similar to C4.5, but it supports the numerical target variables, so it can be used for regression.The sklearn implementation uses an optimized version of this CART algorithm.

Impurity measures

Impurity measure is really important because the Information Gain, which the algorithm uses to choose the next attribute, depends on it. The impurity measures displayed in the sklearn documentation are **Gini** and Entropy (**Log Loss**). These are exactly the measures that we have been taught. Also, we have mentioned that we could use the **Misclassification** impurity measure. Below, I have put the formulas from the sklearn documentation to show that they are the as the ones that we have learned in our lectures.

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k) \qquad H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \qquad H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$$

Multi-output problems

The sklearn implementation of the decision tree is also capable of solving another type of problem that we have not discussed in class, the Multi-output problems. In this case, the parameter that represents the labels is a 2d array of shape **(n_samples, n_outputs)**. Two changes need to be made in order to adjust to this problem. Firstly, n values should be stored in leaves, instead of 1. Also, the splitting criteria should take into account the average reduction across all n outputs. Sklearn has the Multi-output problem support for both the **DecisionTreeClassifier** and **DecisionTreeRegressor**.