uday sai

4 Followers About

Decision Trees : A Non- Parametric Machine Learning Model

uday sai May 12, 2020 · 6 min read ★

> Decision Trees are non-parametric, versatile ML algorithms that can perform both classification and regression and even multi-output tasks.

A non-parametric model is a model in which the shape of the decision boundary is not predefined unlike parametric models (can be linear or non-linear models). Consequently, these models (Decision Tree inn this case) can adapt to data efficiently (downside is overfitting we will discuss later in this article). Besides Decision Tree are white box models meaning which not only predict the target values efficiently but also project the predictive patterns which is similar to Rule-based system (To some extent).

Training and Visualizing Trees

The Decision Trees in sklearn use CART training algorithms. The CART algorithms results in Binary trees meaning any particular node can split into two children. The other training algorithms are ID3 and can be implemented from scratch as there is no inbuilt procedure available.

training models.

```
In [1]: from sklearn.datasets import load_iris
        from sklearn.tree import DecisionTreeClassifier
        iris=load_iris()
        X=iris.data[:,2:] #petal length and width
        y=iris.target
        tree_clf=DecisionTreeClassifier(max_depth=2)
        tree_clf.fit(X,y)

Out[1]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                               max_depth=2, max_features=None, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, presort='deprecated',
                               random_state=None, splitter='best')
```

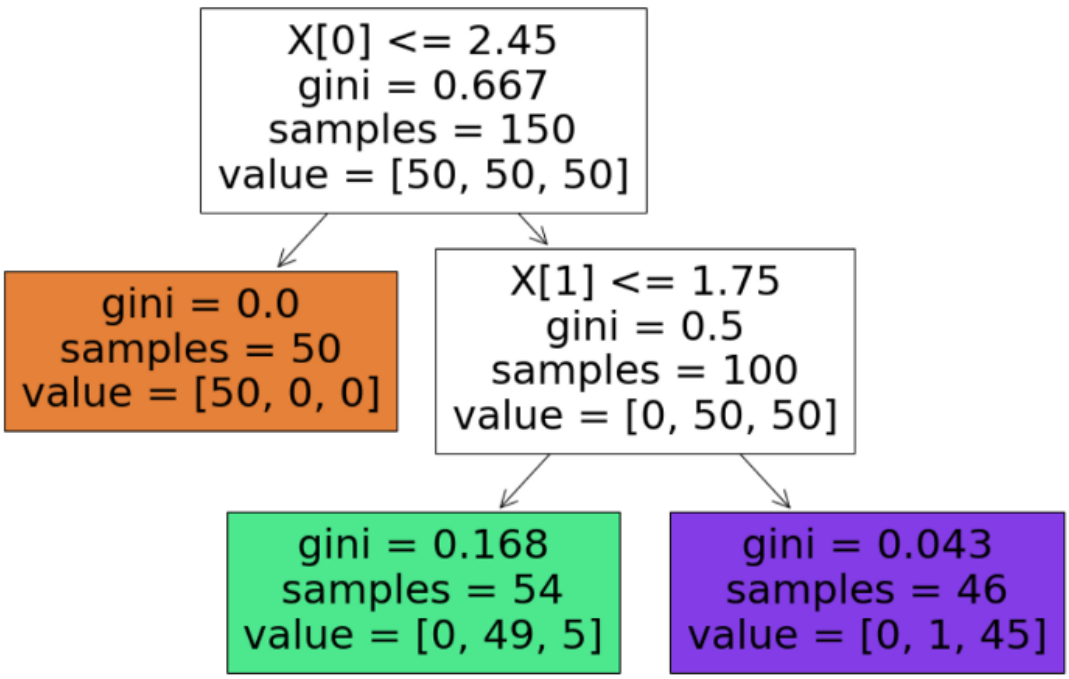
Image : Decision Trees in sklearn using CART training models

Visualization of Decision Tree

The Decision Tree can be visualized using matplotlib and tree.plot_tree in sklearn. The following code snippet shows how to visualize decision trees.

```
In [2]: from sklearn import tree
        import matplotlib.pyplot as plt
        %matplotlib inline
        plt.figure(figsize=(15,10))
        tree.plot_tree(tree_clf,filled=True)

Out[2]: [Text(334.8, 453.0, 'X[0] <= 2.45\ngini = 0.667\nsamples = 150\nvalue = [50, 50, 50]'),
         Text(167.4, 271.8, 'gini = 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
         Text(502.20000000000005, 271.8, 'X[1] <= 1.75\ngini = 0.5\nsamples = 100\nvalue = [0, 50, 50]'),
         Text(334.8, 90.59999999999997, 'gini = 0.168\nsamples = 54\nvalue = [0, 49, 5]'),
         Text(669.6, 90.59999999999997, 'gini = 0.043\nsamples = 46\nvalue = [0, 1, 45]')]
```



The decision trees are very intuitive and are based on rules. Each split of the decision tree scans all the features and select the threshold of split by greedy approach. This guarantees that an optimal solution is resulted by CART algorithms. However the best solution of Decision Tree is a NP- Complete Problem.

Note : NP- Complete Problem is both NP and NP-Hard problem. NP is the set of problems whose solution can be verified in a polynomial time. An NP-Hard problem is a problem to which any NP problem can be reduced in a polynomial time.

Computational Complexity

The training complexity of decision trees is O(n*m*(log(m))) where n is the number of instances and m is the number of leaf nodes. The prediction complexity of decision tress is O(log(m)) as we need to traverse a binary tree in which we have m leaf nodes. For small training sets (less than a few thousand instances), Scikit-Learn can speed up training by presorting the data (set presort=True), but this slows down training considerably for larger training sets.

Making Predictions and Estimating class Probabilities

To make predictions for a new instance we traverse through the tree which is constructed and based on condition in each node. For example, let us consider an instance [5,1.5] (Petal length and Petal width respectively) and based on the tree structure we generated (in the above picture) we check for Petal Length value and as the value (5>2.45) we reach to the right node and now we check for Petal Width value (1.5≤1.75) as the condition is true we reach to the left child. As we have no other conditions we don't need to evaluate any other conditions and output the probabilities of each class.

The Decision Tree Classifiers output the probability of each class that the instance can belong to. For example in the below code snippet we can see that for an instance (5,1.5) the probabilities of setosa, versicolor and virginica are (0,90.7,9) percentages and when we try to predict the result it predicts that the instance belong to versicolor.

```
In [3]: print(tree_clf.predict_proba([[5,1.5]]))
        tree_clf.predict([[5,1.5]])

        [[0.         0.90740741 0.09259259]]
Out[3]: array([1])
```

Gini Impurity and Entropy

As we discussed earlier each split in Decision Tree is based on the lowest impurity. The impurity of a node can be measured in two ways like Gini impurity and entropy.
The gini impurity is calculated for a node as follows:
For example let us consider the left node at depth=2. The values are (0,49,5) and samples are 54, so now gini=$1-((0/54)^2+(49/54)^2+(5/54)^2)$ i.e., now gini= 0.168
The entropy is calculated for a node as follows:
The entropy for same node is calculated as entropy=$-(0/54)\log(0/54)-(49/54)\log(49/54)-(5/54)\log(5/54)=0.31$
In practice either of the values does not make much difference on the performance of the decision tress. Gini impurity is slightly faster to compute, so it is a good default. However, when they differ, Gini impurity tends to isolate the most frequent class in its own branch of the tree, while entropy tends to produce slightly more balanced trees.

Regularization and Hyperparameters

Like any other ML model decision trees can overfit the data too. So we can prevent overfitting by reducing the depth of Decision trees like max_depth or other parameters like max_features (maximum number of features considered for split at each node), max_leaf_nodes (the maximum number of leaf nodes). Conversely we can prevent overfitting by increasing the other parameters like min_samples_leaf (the minimum number of samples a leaf node must have), min_samples_split(samples a node must have before it can split).
The overfitting can be prevented by reducing max_hyperparameters and increasing min_hyperparameters. This will regularize the model
Pruning
Besides in Decision trees using ID3 training models or other training models we perform a statistical Chi-2 test for independence. A node whose children are all leaf nodes is considered unnecessary if the purity improvement it provides is not statistically significant. The null hypothesis is the improvement is purely the result of chance. If the probability called (p-value) is higher than a given threshold (typically 5%, controlled by a hyperparameter), then the node is considered unnecessary and its children are deleted. The pruning continues until all unnecessary nodes have been pruned.

(Mean squared Error) unlike the impurity score. To predict the continuous values we will traverse through the tree and the predicted value is value in the node which is simply average of all the samples prediction values present in the particular node.

This tree looks very similar to the classification tree you built earlier. The main difference is that instead of predicting a class in each node, it predicts a value. For example, suppose you want to make a prediction for a new instance with x3 = 6.0. You traverse the tree starting at the root, and you eventually reach the leaf node that predicts value=32.113. This prediction is simply the average target value of the 46 training instances associated to this leaf node. This prediction results in a Mean Squared Error (MSE) equal to 41.296 over these 46 instances.

Image : Decision Tree Regressor

Instability

Some of the limitations with Decision Tree are as follows:

First, as you may have noticed, Decision Trees love orthogonal decision boundaries (all splits are perpendicular to an axis),which makes them sensitive to training set rotation. For example, the following shows a simple linearly separable dataset: on the left, a Decision Tree can split it easily, while on the right, after the dataset is rotated by 45°, the decision boundary looks unnecessarily convoluted. Although both Decision Trees fit the training set perfectly, it is very likely that the model on the right will not generalize well. One way to limit this problem is to use PCA , which often results in a better orientation of the training data.

Image : Sensitivity to Training set rotation

More generally, the main issue with Decision Trees is that they are very sensitive to small variations in the training data. For example, if you just remove the widest Iris-Versicolor from the iris training set (the one with petals 4.8 cm long and 1.8 cm wide) and train a new Decision Tree we may get a different tree. Actually, since the training algorithm used by Scikit-Learn is stochastic you may get very different models even on the same training data (unless you set the random_state hyperparameter).

Data Science Machine Learning Decision Tree Decision Tree Classifier Decision Tree Regressor