Explore decision trees

Explore decision trees

As you know, tree-based learning is one of the most effective machine learning techniques that are currently used in industry today. Many different algorithms use a tree-based architecture to make their predictions. The decision tree is the basic building block of these algorithms, as well as a powerful predictive algorithm in itself. Data professionals rely on decision trees as powerful tools that enhance modeling decisions. In this reading, you will take a deep dive into decision trees, how they are structured, how they work, and how they are built.

What is a decision tree?

Decision trees are a flowchart-like structure that uses branching paths to predict the outcomes of events, the probability of certain outcomes, or to reach a decision. They can be used for classification problems, where a specific class or outcome is predicted—like whether or not a sports team will win a game. They can also be used for regression problems, where a continuous variable is predicted—like the price of a car. This reading focuses on classification trees, but in both cases, the models depend on the same underlying decision process.

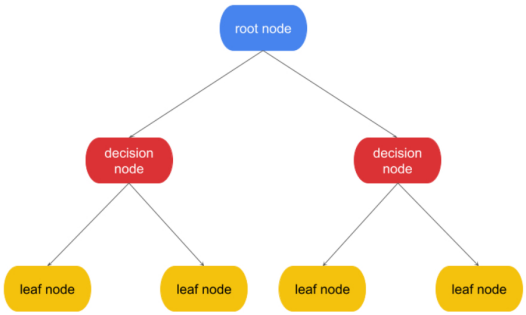
Decision trees have been used for decades for analyzing problems. However, technological advances have made it so decision trees can be created by computers, offering much deeper and more accurate analysis than humans alone could ever achieve.

Note: All examples in this reading depict nodes that split into just **two** child nodes, because this is how they are implemented in modeling libraries, including scikit-learn. While it's theoretically possible to split each node into three, four, or even more new groups (as depicted previously in the "Shall I play soccer?" example), this is an impractical approach because of its computational complexity. A two-level binary split is functionally equivalent to a single-level three-way split, but much simpler in terms of computational demand.

The structure of a classification tree

Decision trees only resemble actual trees if you flip them upside down, because they start with the root at the top and grow downward so the "leaves" are at the bottom. Decision trees are made of nodes. Nodes are groups of samples. There are different types of nodes, depending on how they function in the tree. The first node in a decision tree is called the **root node**. The first split always comes off of the root node, which divides the samples into two new nodes based on the values they contain for a particular feature.

These two new nodes are referred to as **child nodes** of the root. A child node is any node that results from a split. The node that the child splits from is known as the **parent node.** Each of these two new child nodes in turn splits the data again, based on a new criterion. This process continues until the nodes stop splitting. The bottom-level nodes that do not split are called **leaf nodes**. All the nodes above the leaf nodes are called **decision nodes**, because they all make a decision that sorts the data either to the left or to the right.



Decisions and splits

In a decision tree, the data is split and passed down through decision nodes until reaching a leaf node. A decision node is split on the criterion that minimizes the **impurity** of the classes in their resulting children. Impurity refers to the degree of mixture with respect to class. Nodes with low impurity have many more of one class than any other. A perfect split would have no impurity in the resulting child nodes; it would partition the data with each child containing only a single class. The worst possible split would have high impurity in the resulting child nodes; both of the child nodes would have equal numbers of each class.

When building a tree and growing a new node, a set of potential split points is generated for every predictor variable in the dataset. An algorithm is used to calculate the "purity" of the child nodes that would result from each split point of each variable. The feature and split point that generate the purest child nodes are selected to partition the data.

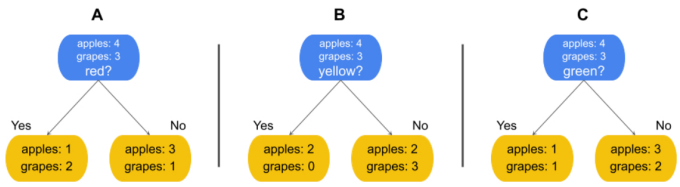
To determine the set of potential split points that will be considered for a variable, the algorithm first identifies what type of variable it is—such as categorical or continuous—and the range of values that exist for that variable.

Categorical variables

If the predictor variable is categorical, the decision tree algorithm will consider splitting based on category. Here's a small dataset of fruits, to illustrate. The data contains samples of fruit that are either apples or grapes. It also has the fruits' color (yellow, red, or green) and diameter in centimeters. Each of these variables affects the decision tree algorithm.

Color	Diameter (cm)	Fruit(target)
Yellow	3.5	Apple
Yellow	7	Apple
Red	2	Grape
Red	2.5	Grape
Green	4	Grape
Green	3	Apple
Red	6	Apple

First, the algorithm will consider splitting based on the categorical variable, color. Since there are three categories, three options are considered. Note that "yes" always goes to the left and "no" to the right.

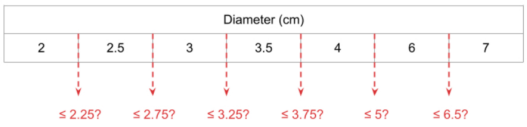


Continuous variables

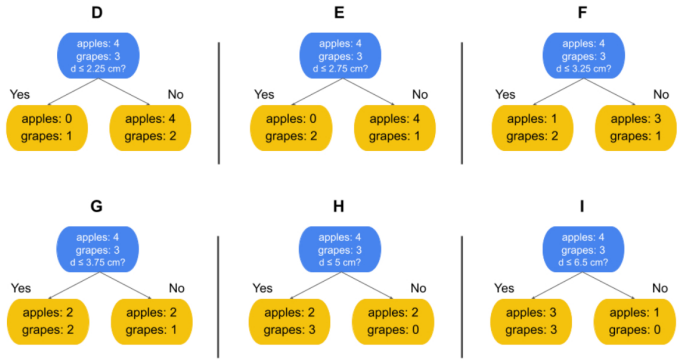
If the predictor variable is continuous, splits can be made anywhere along the range of numbers that exist in the data. Often the potential split points are determined by sorting the values for the feature and taking the mean of each consecutive pair of values. However, there can be any number of split points, and fewer split points can be considered to save computational resources and time. It is very common, especially when dealing with very large ranges of numbers, to consider split points along percentiles of the distribution.

In the case of the fruit example, "Diameter" is a continuous variable. One way a decision tree could handle this is to:

1. **Sort the values, identify average of consecutive values:**



1. **Examine splitting based on these identified means:**



These are the six potential split points for the Diameter feature that were identified by the algorithm. Each option includes the children of that split, but note that at this point none of them has been evaluated yet. That's the next step.

Choosing splits: Gini impurity

Now you know how to determine the *potential* split points. In the fruit example, there are nine options to choose from: A–I. But how do you decide which split to use? This is where the "purity" of the child nodes becomes relevant. Generally, splits are better when each resulting child node contains many more samples of one class than any other, like in example E above, because this means the split is effectively separating the classes—the primary job of the decision tree! In such cases, the child nodes are said to have low impurity (or high purity). The decision tree algorithm determines the split that will result in the lowest impurity among the child nodes by performing a calculation.

There are several possible metrics to use to determine the purity of a node and to decide how to split, including **Gini impurity, entropy, information gain,** and **log loss**. The most straightforward is Gini impurity, and it's also the default for the decision tree classifier in scikit-learn, so this reading will focus on that method. The Gini impurity of a node is defined as:

$$Gini\ impurity = 1 - \sum_{i=1}^{N} P(i)^2$$

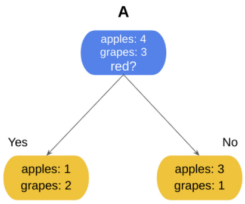
where i= class,

$P(i)$ = the probability of samples belonging to class i in a given node.

In the case of the fruits example, this becomes:

$$Gini\ impurity = 1 - P(apple)^2 - P(grape)^2$$

The Gini impurity is calculated for each child node of each potential split point. For example, there are nine split point options in the fruit example (A–I). The first potential split point is Color=red:



Calculate Gini impurity of each child node:

$$Gini\ impurity = 1 - P(apple)^2 - P(grape)^2$$

$$= 1 - (\frac{number\ of\ apples\ in\ node}{total\ number\ of\ samples\ in\ node})^2 - (\frac{number\ of\ grapes\ in\ node}{total\ number\ of\ samples\ in\ node})^2$$

For the "**red=yes**" child node:

Gini impurity = 1 - $(1/3)^2$ - $(2/3)^2$

= 1 - 0.111 - 0.444

= 0.445

And for the "**red=no**" child node:

Gini impurity = 1 - $(3/4)^2$ - $(1/4)^2$

= 1 - 0.5625 - 0.0625

= 0.375

Now there are two Gini impurity scores for split option A (whether or not the fruit is red)—one for each child node. The final step is to combine these scores by taking their weighted average.

Calculate weighted average of Gini impurities

The weighted average accounts for the different number of samples represented in each Gini impurity score. You cannot simply add them together and divide by two, because the first child node contained three samples and the second child node contained four. The weighted average of the Gini impurities (*Gi)* is calculated as:

Total Gi:

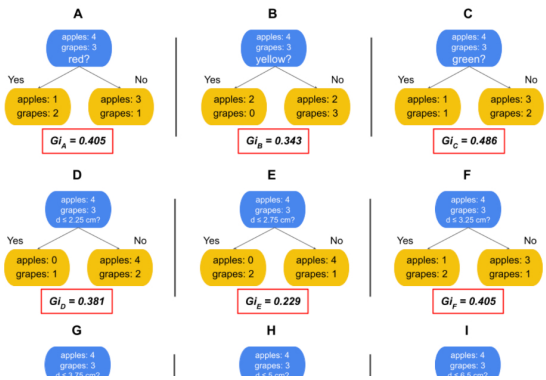
$$= (\frac{number\ of\ samples\ in\ LEFT\ child}{number\ of\ samples\ in\ BOTH\ child\ nodes}) * Gi_{left\ child} + (\frac{number\ of\ samples\ in\ RIGHT\ child}{number\ of\ samples\ in\ BOTH\ child\ nodes}) * Gi_{right\ child}$$

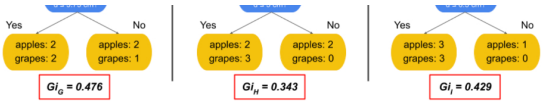
= (3/7 * 0.445) + (4/7 * 0.375)

Gi_{total} = 0.405

Repeat this process for every split option

This same process is repeated for every split option. The fruit example has nine options (A–I):



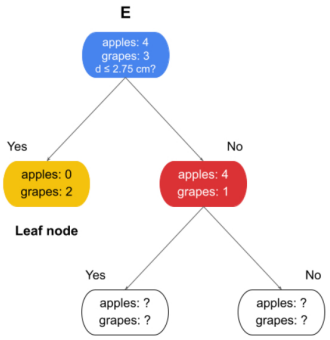


Now there are nine Gini impurity scores ranging from 0.229 to 0.486. Since it's a measure of impurity, the best scores are those closest to zero. In this case, that's option E. The worst of the nine options is option C, because it doesn't separate classes well. The worst possible Gini impurity score is 0.5, which would occur when each child node contains an equal number of each class.

Now that the algorithm has identified the potential split points and calculated the Gini impurity of the child nodes that result from them, it will grow the tree by selecting the split point with the lowest Gini impurity.

Grow the tree

In the case of the given example, the root node would use split option E to split the data. The left child becomes a leaf node, because it contains just one class. However, the right child still does not have class purity, so it becomes a new decision node (in the absence of some imposed stopping condition). The steps outlined above will repeat on the samples in this node to identify the feature and split value that would yield the best result.



Splitting would continue until all the leaves are pure or some imposed condition stops the splitting.

You may have noticed that this process involves *a lot* of computation—and this was only for a dataset with two features and seven observations. As with many machine learning algorithms, the theory and methodology behind decision trees are fairly straightforward and have been around for many years, but it wasn't until the advent of powerful computing capabilities that these solutions were able to be put into practice.

Advantages and disadvantages of classification trees

Advantages:

- Require relatively few pre-processing steps
- Can work easily with all types of variables (continuous, categorical, discrete)
- Do not require normalization or scaling
- Decisions are transparent
- Not affected by extreme univariate values

Disadvantages:

- Can be computationally expensive relative to other algorithms
- Small changes in data can result in significant changes in predictions

Key takeaways

Decision trees are powerful predictive tools that can identify patterns in data that other algorithms might not be able to. They're user-friendly because they require relatively few preprocessing steps compared to other models. They consist of a series of decision nodes, beginning at a root node and ending at leaf nodes. They operate by splitting data at particular feature values that are identified as thresholds. These split thresholds are determined by calculating the impurity of the child nodes that result from them, and selecting the split that yields the least impurity.

Mark as completed