Hyperparameter tuning

In this reading, you will learn about hyperparameters, and how tuning them can affect model performance. You will develop a deeper understanding of some of the most commonly tuned hyperparameters for decision trees. Additionally, you will learn one process that is used to find optimal sets of hyperparameters.

Hyperparameter tuning

Throughout this program, you've learned about modeling techniques to make predictions or better understand data. Sometimes, the models perform well right out of the box, using the basic or default application of their underlying theory. Most often, however, this is not the case. Each dataset presents its own set of characteristics for which the data professional must tailor the model.

Depending on the characteristics of both the data and the algorithm used to model it, a model might overfit or underfit the data. Remember, the aim of a predictive model is to identify underlying, intrinsic patterns and characteristics in data that are representative of *all* such distributions, and use these characteristics to make predictions on new data.

Overfitting is when the model learns the training data so closely that it captures more than the intrinsic patterns of *all* such data distributions, and ends up learning noise or idiosyncrasies particular to *just* the training data. This results in a model that scores very well on the training data but considerably worse on unseen data because it cannot generalize well.

On the other hand, underfitting is when the model does not learn the patterns and characteristics of the training data well, and consequently fails to make accurate predictions on new data. It's typically easier to identify underfitting, because the model performs poorly on both training and test data. The best models neither underfit nor overfit the data. They identify intrinsic patterns within it, but do not capture randomness or noise.

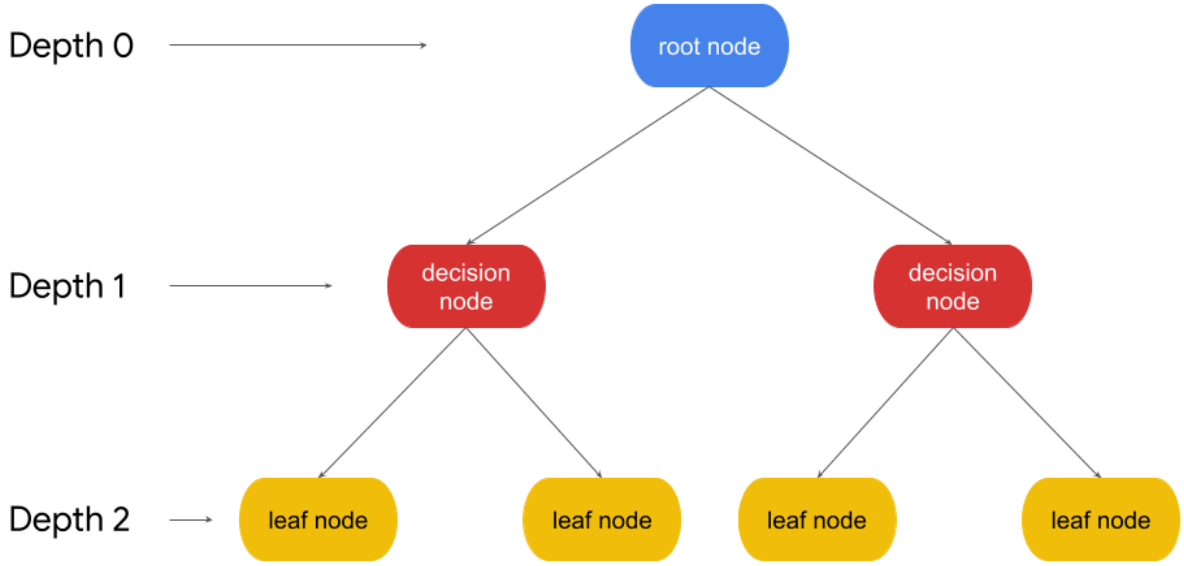
One way of helping to achieve this balance is through the use of **hyperparameters**. Hyperparameters are aspects of a model *that you set* before the model is trained, and that affect how the model fits the data. They are not derived from the data itself. Hyperparameter tuning is the process of adjusting the hyperparameters to build a model that best fits the data. (Note that you'll often hear them referred to as "parameters," much like you might hear the word "theory" used to mean "idea," when it actually has a very specific scientific meaning. It's okay, as long as you understand the difference.)

Hyperparameters for decision trees

There are many different hyperparameters available to control how a decision tree grows. Each hyperparameter affects something very specific related to the growth conditions. One might affect what causes a node to split, while another might limit how deep the tree is allowed to grow, and yet another might change the way node purity is calculated. This reading will introduce you to three hyperparameters: max_depth, min_samples_split, and min_samples_leaf.

max_depth

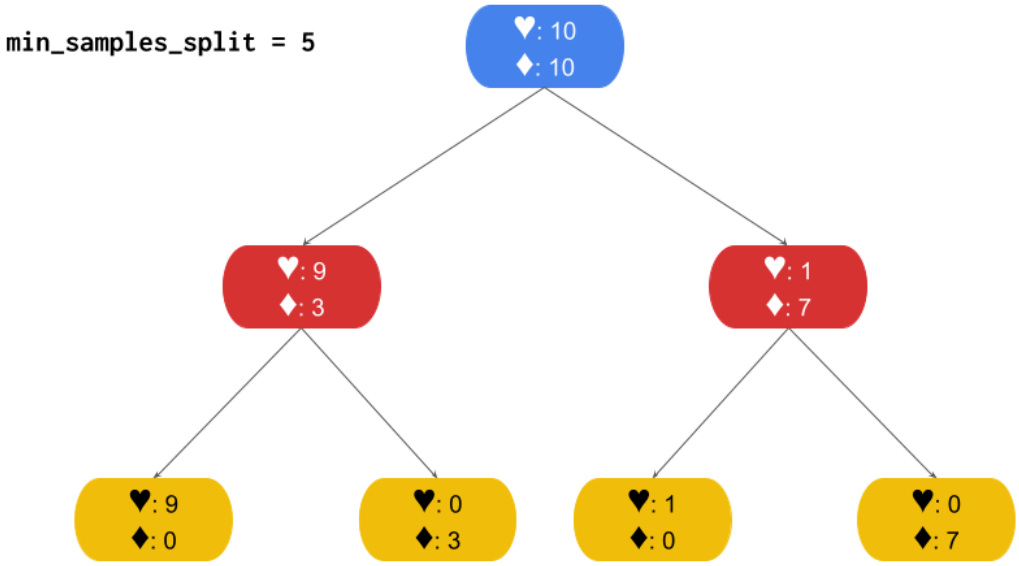
max_depth defines how deep the tree is allowed to grow. The depth of the tree is the distance, measured in number of levels, from the root node to the furthest leaf node. The root node would have a depth of zero, the child of the root node would have a depth of one, and so on.

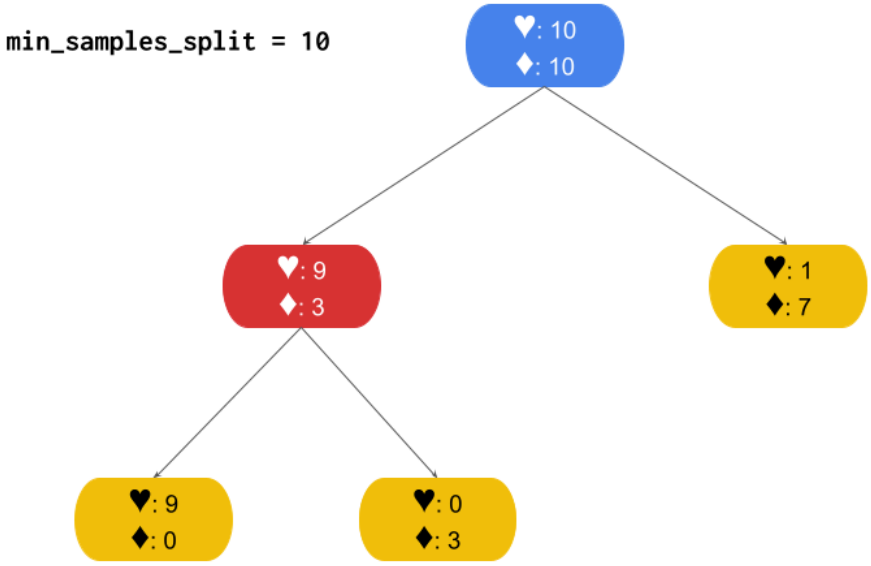


An unrestricted decision tree will continue splitting until every leaf node contains only a single class. As you increase the max depth parameter, the performance of the model on the training set will continue to increase. It's possible for a tree to grow so deep that leaves contain just a single sample. However, this overfits the model to the training data, and the performance on the testing data would probably be much worse. A quick intuition of why this happens: if you let a tree have so many nodes representing so many specific decision rules that perfectly align with the details of the training data, how likely do you think it is for those exact decision nodes to match new data in the real world? On the other hand, a tree that is not allowed to grow deeply enough will have high bias and fail to make accurate predictions. The best decision tree models are neither too shallow nor too deep, but just right.

min_samples_split

min_samples_split is the minimum number of samples that a node must have for it to split into more nodes. For example, if you set this to 10, then any node that contains nine or fewer samples will automatically become a leaf node. It will not continue splitting. However, if the node contains 10+ samples, it may continue to split into child nodes. The greater the value you use for min_samples_split, the sooner the tree will stop growing. The minimum possible value is two, because two is the smallest number that can be divided into two separate child nodes.

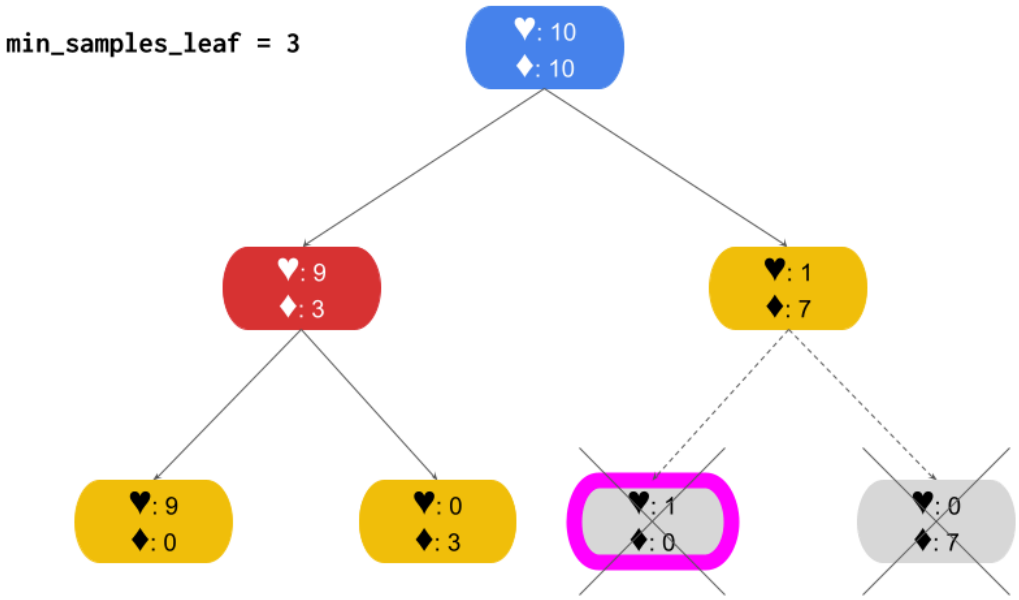




Notice how the tree in the example with min_samples_split=5 continues splitting until all leaves are pure, while the tree with min_samples_split=10 stops splitting on one side even though this node still contains a mix of classes. It stops because it contains only eight samples—below the threshold of 10 indicated by min_samples_split.

min_samples_leaf

min_samples_leaf is similar to min_samples_split, but with an important difference. Instead of defining how many samples the *parent* node must have *before* splitting, min_samples_leaf defines the minimum number of samples that must be in each *child* node *after* the parent splits. Consider the following example, where min_samples_leaf is set to three.



Notice that the right branch of the tree becomes a leaf at depth=1, while the left branch continues splitting until reaching class purity in the leaf nodes at depth=2. The right branch stops because splitting it would result in one of the child nodes containing just a single sample, which is below the threshold of three indicated by min_samples_leaf.

Finding the optimal set of hyperparameters

The values that these hyperparameters can take are limited only by the number of samples in your dataset. That leaves open the possibility for millions of combinations! How do you know what the values should be? The answer is to train a lot of different models to find out. There are a number of ways to do this. Performing a grid search is one of the more popular methods.

Grid search

A grid search is a technique that will train a model for every combination of preset ranges of hyperparameter values. The aim is to find the combination of values that results in a model that both fits the training data well and generalizes well enough to predict accurately on unseen data. After all these models have been trained, you then compare them to find this ideal model—if it exists.

Here is a very basic example of how a grid search might be used to find the best combination of two hyperparameters: max_depth and min_samples_leaf. You can define the set of max_depth values as [6, 8, None], and the set of min_samples_leaf values as [1, 3, 5].

		max_depth		
		6	8	None
min_samples_leaf	1	(1, 6)	(1, 8)	(1, None)
	3	(3, 6)	(3, 8)	(3, None)
	5	(5, 6)	(5, 8)	(5, None)

A tree is grown for each combination of values for each hyperparameter. Note that this grid search results in nine models—the product of the number of values being tried for each hyperparameter (3 x 3). If you wanted to include min_samples_split values of [2, 4, 6, 8] in your search as well, then you'd be growing 3 x 3 x 4 trees, or 36 different models. Note also that, of these combinations of hyperparameters, the (5, 6) combination restricts growth regularizes the most, while the (1, None) combination allows the model to grow unrestricted.

Note: For decision trees (and all tree-based models), restricting growth is a form of **regularization**. Recall from the regression course that regularization refers to the process of reducing model complexity to prevent overfitting. The greater the complexity of a model, the more susceptible it is to overfit the training data. Regularization helps to make the model more generalizable to new data.

With more hyperparameters and a more expansive array of values to search over, grid searches can quickly become computationally expensive. One helpful search strategy is to try a wider array of values for each hyperparameter—say, [3, 6, 9], instead of [3, 4, 5]. If the best model has 6 as the value for this hyperparameter, perhaps try another grid search using [5, 6, 7] as potential values. This technique uses multiple search iterations to progressively home in on an optimal set of values.

Another technique is to define a more comprehensive set of search values from the beginning—say, [3, 4, 5, 6, 7, 8, 9]—and let the model train for what could be a very long time. Which approach you take will depend on your computing environment, your computational resources, and how much time you have.

Key takeaways

- Hyperparameters are aspects of a model that are set before the model trains, affecting the training itself.
- Different model types have different sets of hyperparameters that can be changed
- For decision trees, a few but some of the most important are:
    - **max_depth:** The maximum depth the tree will construct to before stopping
    - **min_samples_split:** The minimum number of samples that a node must have to split into more nodes.
    - **min_samples_leaf:** The minimum number of samples that must be in each child node for the split to complete.

- You can find the optimal set of hyperparameters using different types of algorithms. GridSearch is one of the more popular techniques, and involves specifying in advance all the values you want to try for each hyperparameter, and then training the model for every combination of those values.