✓ 100 XP

Hyperparameters in classification

6 minutes

Hyperparameters can be thought of settings used for training. For example, we might choose to train slowly or quickly. Hyperparameters affect training, and so affect the final model performance. Exactly which hyperparameters are available depends on the kind of model we're training. We usually experiment with hyperparameters to optimize our model's performance.

Random forests as an example

Random forests have different kinds of hyperparameters available. With random forests specifically, the line between architectural decisions hyperparameters can be blurry. This is because hyperparameters don't only affect the parameters inside the model, but also how the trees and forest are structured.

Recall that at the beginning of training each decision tree is provided numerous samples, such as 100 gymnasts, some of whom won medals. A tree must be built that progressively divides these samples into smaller subgroups of athletes. The goal is that these subgroups contain athletes that are alike, such as within each subgroup all athletes won medals, or all didn't. Let's explore some hyperparameters that can affect this training process.

Criteria to split on

During training, the optimizer must decide when to split a node. There are different ways decisions like this can be made, and which method is chosen is called a hyperparameter. In essence, different methods refer to different ways to assess how similar a sample is.

Common methods split nodes are based on **information theory.** This can roughly be thought of as splitting a sample so that the two resulting subsamples are 'purer' than the original. The methods available differ slightly and can result in slight differences in the final resulting tree, very similarly to how cost functions used for gradient descent can give different final models. We'll experiment with two criteria in the next set of exercises.

Minimum impurity decrease

The criterion used to split nodes can be further customized. For example, setting the *minimum purity decrease* means that a node can only be split if it will improve the model by a certain amount or more. There are several related hyperparameters that can block new nodes being created, such as the maximum depth or the minimum number of samples at a node.

The reason we restrict a tree growing too far is to avoid overfitting. Bigger trees are better at matching the training dataset, but they can become so tuned to this training set that they stop working for other data. In other words, restricting how complex a tree becomes can reduce its tendency to overfit.

Maximum number of features

When trees in a random forest are created, they're provided with a subset of training data to fit, and a list of features to use. Importantly, each tree can receive different collections of features. For example, one tree may use Weight and Height, while another uses Height and Age.

Increasing the maximum number of features each tree may receive is likely to improve how well each tree can fit the training set, as more information is provided. Whether this aids or impairs its abilities on the test set can require experimentation. This is because always providing many features can mean that trees in the forest end up more similar to one-other, reducing the advantage of a random forest over a simple decision tree. Finding the balance between these extremes usually requires some experimentation.

Seeding

Model fitting usually relies, at some point, on random numbers. Computers don't produce truly random numbers, but rather contain rules that state how to produce a list of random numbers, given an initial number, called the **random seed.**

For example, if our seed value was 10, and takes the first three ‘random’ numbers, the computer might produce 0.75, 0.13, 0.68. While these numbers appear random, every time we seed from 10, we'll get the same set of numbers.

In machine learning, we use random numbers to initialize the model’s parameters and/or to split datasets into training and test sets. If the random seed is set, the random values used during the training process will be the same every time we rerun our code. Meaning that each time we rerun our code, we assign the same data to test or training sets and train models that have the same **initialization state (initial parameters).**

By contrast, if we don't set the seed, the computer will select one for us-for example, based on the time-which means that running our training twice can give slightly different results.

The random seed isn't strictly a hyperparameter, but we introduce it here to highlight that this external parameter can play a role in the effectiveness of training. While this is usually minor, if the model is very complex, and/or the amount of data available is small, the test-set performance of the model can be markedly different if two different seeds are used. In such situations, often it pays to run training with multiple different seeds to assess to what degree your model design is adequate, and to what degree your performance is simply ‘blind luck’.

Next unit: Exercise - Hyperparameter tuning with random forests

Continue >