**Hyperparameter Tuning**

In this lesson, we are going to take a look at how we can improve our models using one of SageMaker's features. In particular, we are going to explore how we can use SageMaker to perform hyperparameter tuning.

In many machine learning models there are some parameters that need to be specified by the model creator and which can't be determined directly from the data itself. Generally the approach to finding the best parameters is to train a bunch of models with different parameters and then choose the model that works best.

SageMaker provides an automated way of doing this. In fact, SageMaker also does this in an intelligent way using Bayesian optimization. What we will do is specify ranges for our hyperparameters. Then, SageMaker will explore different choices within those ranges, increasing the performance of our model over time.

In addition to learning how to use hyperparameter tuning, we will look at Amazon's CloudWatch service. For our purposes, CloudWatch provides a user interface through which we can examine various logs generated during training. This can be especially useful when diagnosing errors.

NEXT

**Introduction to Hyperparameter Tuning**

Let's take a look at how we can use SageMaker to improve our Boston housing model. To begin with, we will remind ourselves how we train a model using SageMaker.

Essentially, tuning a model means training a bunch of models, each with different hyperparameters, and then choosing the best performing model. Of course, we still need to describe two different aspects of hyperparameter tuning:

1) What is a *bunch* of models? In other words, how many different models should we train?

2) Which model is the *best* model? In other words, what sort of metric should we use in order to distinguish how well one model performs relative to another.

NEXT

# Boston Housing Example

In the video below we will look at how we can set up a hyperparameter tuning job using the high level approach in SageMaker. To following along, open up the Boston Housing - XGBoost (Hyperparameter Tuning) - High Level.ipynb notebook in the Tutorials directory.

Generally speaking, the way to think about hyperparameter tuning inside of SageMaker is that we start with a **base** collection of hyperparameters which describe a default model. We then give some additional set of hyperparameters ranges. These ranges tell SageMaker which hyperparameters can be varied, with the goal being to improve the default model.

We then describe how to compare models, which in our instance is just by way of specifying a metric. Then we describe how many total models we want SageMaker to train.

**Note:** In addition to creating a tuned model in this notebook, we also saw how the attach method can be used to create an Estimator object which is attached to an already completed training job. This method is useful in other situations as well.

You will notice that throughout this module we train the same model multiple times. In most of the Boston Housing notebooks, for example, we train an XGBoost model with the same hyperparameters. The reason for this is so that each notebook is self contained and can be run even if you haven't run the other notebooks.

In your case however, you've probably already created an XGBoost model on the Boston Housing training set with the standard hyperparameters. If you wanted to, you could use the attach method to avoid having to re-train the model.

# Mini-Project: Tuning the Sentiment Analysis Model

Now that you've seen an example of how to use SageMaker to tune a model, it's your turn to try it out!

Inside of the Mini-Projects folder is a notebook called IMDB Sentiment Analysis - XGBoost (Hyperparameter Tuning).ipynb. Inside of the notebook are some tasks for you to complete.

**Note:** To make things a little more interesting, there is a small error in the notebook. Try not to anticipate where the error is. Instead, just continue through the notebook as if nothing is wrong and when an error occurs, try to use CloudWatch to diagnose and fix it.

# Boston Housing In-Depth

Now we will look at creating a hyperparameter tuning job using the low level approach. Just like in the other low level approaches, this method requires us to describe the various properties that our hyperparameter tuning job should have.

To follow along, open up the Boston Housing - XGBoost (Hyperparameter Tuning) - Low Level.ipynb notebook in the Tutorials folder.

Creating a hyperparameter tuning job using the low level approach requires us to describe two different things.

The first, is a training job that will be used as the **base** job for the hyperparameter tuning task. This training job description is almost exactly the same as the standard training job description except that instead of specifying **HyperParameters** we specify **StaticHyperParameters**. That is, the hyperparameters that we do **not** want to change in the various iterations.

The second thing we need to describe is the tuning job itself. This description includes the different ranges of hyperparameters that we **do** want SageMaker to vary, in addition to the total number of jobs we want to have created and how to determine which model is best.

# What have we learned so far?

In this lesson we took a look at how we can use SageMaker to tune a model. This can be helpful once we've found a good base model and we want to try and iterate a bit to refine our model and get a little more out of it.

We also looked at using CloudWatch to monitor our training jobs so that we can better diagnose errors when they arise. This can be especially helpful when training more complicated models such as those in which you can incorporate your own code.

# What's next?

In the next lesson we will take a look at updating a deployed model. Once you've developed a model and deployed it the story is rarely over. What happens if some of the built in assumptions about your model change over time?

We will look at how you can create a new model which more accurately reflects the current state of your problem and then update an existing endpoint so that it uses your new model rather than the original one. In addition, using SageMaker, we can do this without needing to shut down the endpoint. This means that whatever application is using your deployed model won't notice any sort of disruption in service.