**Step Three: Model Training**

You're ready to start training your first model.

**Splitting your Dataset**

The first step in model training is to randomly split the dataset. This allows you to keep some data hidden during training, so that data can be used to evaluate your model before you put it into production. Specifically, you do this to test against the bias-variance trade-off. If you're interested in learning more, see the **Further learning and reading** section.

Splitting your dataset gives you two sets of data:

* *Training dataset*: The data on which the model will be trained. Most of your data will be here. Many developers estimate about 80%.
* *Test dataset*: The data withheld from the model during training, which is used to test how well your model will generalize to new data.

**Model Training Terminology**

The model training algorithm iteratively updates a model's parameters to minimize some loss function.

Let's define those two terms:

* *Model parameters*: Model parameters are settings or configurations the training algorithm can update to change how the model behaves. Depending on the context, you’ll also hear other more specific terms used to describe model parameters such as *weights* and *biases*. Weights, which are values that change as the model learns, are more specific to neural networks.
* *Loss function:* A loss function is used to codify the model’s distance from this goal. For example, if you were trying to predict a number of snow cone sales based on the day’s weather, you would care about making predictions that are as accurate as possible. So you might define a loss function to be “the average distance between your model’s predicted number of snow cone sales and the correct number.” You can see in the snow cone example this is the difference between the two purple dots.

**Putting it All Together**

The end-to-end training process is

* Feed the training data into the model.
* Compute the loss function on the results.
* Update the model parameters in a direction that reduces loss.

You continue to cycle through these steps until you reach a predefined stop condition. This might be based on a training time, the number of training cycles, or an even more intelligent or application-aware mechanism.

**Advice From the Experts**

Remember the following advice when training your model.

1. Practitioners often use machine learning frameworks that already have working implementations of models and model training algorithms. You could implement these from scratch, but you probably won't need to do so unless you’re developing new models or algorithms.
2. Practitioners use a process called *model selection* to determine which model or models to use. The list of established models is constantly growing, and even seasoned machine learning practitioners may try many different types of models while solving a problem with machine learning.
3. *Hyperparameters* are settings on the model which are not changed during training but can affect how quickly or how reliably the model trains, such as the number of clusters the model should identify.
4. Be prepared to iterate.

*Pragmatic problem solving with machine learning is rarely an exact science, and you might have assumptions about your data or problem which turn out to be false. Don’t get discouraged. Instead, foster a habit of trying new things, measuring success, and comparing results across iterations.*

**Extended Learning**

This information hasn't been covered in the above video but is provided for the advanced reader.

**Linear models**

One of the most common models covered in introductory coursework, linear models simply describe the relationship between a set of input numbers and a set of output numbers through a linear function (think of *y = mx + b* or a line on a *x* vs ychart).

Classification tasks often use a strongly related logistic model, which adds an additional transformation mapping the output of the linear function to the range [0, 1], interpreted as “probability of being in the target class.” Linear models are fast to train and give you a great baseline against which to compare more complex models. A lot of media buzz is given to more complex models, but for most new problems, consider starting with a simple model.

**Tree-based models**

Tree-based models are probably the second most common model type covered in introductory coursework. They learn to categorize or regress by building an extremely large structure of nested *if/else blocks*, splitting the world into different regions at each if/else block. Training determines exactly where these splits happen and what value is assigned at each leaf region.

For example, if you’re trying to determine if a light sensor is in sunlight or shadow, you might train tree of depth 1 with the final learned configuration being something like *if (sensor\_value > 0.698), then return 1; else return 0*;. The tree-based model XGBoost is commonly used as an off-the-shelf implementation for this kind of model and includes enhancements beyond what is discussed here. Try tree-based models to quickly get a baseline before moving on to more complex models.

**Deep learning models**

Extremely popular and powerful, deep learning is a modern approach based around a conceptual model of how the human brain functions. The model (also called a *neural network*) is composed of collections of *neurons* (very simple computational units) connected together by *weights* (mathematical representations of how much information to allow to flow from one neuron to the next). The process of training involves finding values for each weight.

Various neural network structures have been determined for modeling different kinds of problems or processing different kinds of data.

A short (but not complete!) list of noteworthy examples includes:

* **FFNN**: The most straightforward way of structuring a neural network, the Feed Forward Neural Network (FFNN) structures neurons in a series of layers, with each neuron in a layer containing weights to all neurons in the previous layer.
* **CNN**: Convolutional Neural Networks (CNN) represent nested filters over grid-organized data. They are by far the most commonly used type of model when processing images.
* **RNN**/**LSTM**: Recurrent Neural Networks (RNN) and the related Long Short-Term Memory (LSTM) model types are structured to effectively represent *for loops* in traditional computing, collecting state while iterating over some object. They can be used for processing sequences of data.
* **Transformer**: A more modern replacement for RNN/LSTMs, the transformer architecture enables training over larger datasets involving sequences of data.

**Machine Learning Using Python Libraries**

* For more classical models (linear, tree-based) as well as a set of common ML-related tools, take a look at scikit-learn. The web documentation for this library is also organized for those getting familiar with space and can be a great place to get familiar with some extremely useful tools and techniques.
* For deep learning, mxnet, tensorflow, andpytorch are the three most common libraries. For the purposes of the majority of machine learning needs, each of these is feature-paired and equivalent.

**Terminology**

***Hyperparameters*** are settings on the model which are not changed during training but can affect how quickly or how reliably the model trains, such as the number of clusters the model should identify.

A **loss function** is used to codify the model’s distance from this goal

**Training dataset**: The data on which the model will be trained. Most of your data will be here.

**Test dataset**: The data withheld from the model during training, which is used to test how well your model will generalize to new data.

**Model parameters** are settings or configurations the training algorithm can update to change how the model behaves.

**Additional reading**

* The Wikipedia entry on the [bias-variance](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff) trade-off can help you understand more about this common machine learning concept.
* In this [AWS Machine Learning blog post](https://aws.amazon.com/blogs/machine-learning/build-a-model-to-predict-the-impact-of-weather-on-urban-air-quality-using-amazon-sagemaker/), you can see how to train a machine-learning algorithm to predict the impact of weather on air quality using Amazon SageMaker.