**Unit-01**

**Introduction**

Machine-learning models are computer algorithms that use data to make estimations (educated guesses) or decisions. Machine-learning models differ from traditional algorithms in how they're designed. When traditional computer software needs to be improved, people edit it. In contrast, a *machine-learning algorithm* uses data to get better at a specific task.

For example, spam filters use machine learning. Twenty years ago, spam filters didn't have many examples from which to learn and weren't good at identifying what is and isn’t spam. As more spam has arrived and been labeled as junk by human users, the machine-learning algorithms have gained more experience and become better at their job.

**Boots that fit**

Throughout this module, we use an example scenario to explain key machine-learning concepts.

In this scenario, you own a shop that sells harnesses for avalanche-rescue dogs, and you’ve recently expanded to also sell doggy boots. Customers all seem to pick the correct harness sizes, but are constantly ordering doggy boots that are the wrong size. You know most customers buy harnesses and boots in the same transaction, which gives you an idea: perhaps you could approximate which doggy boots are the correct size, depending on the harness chosen. Then, you could warn customers if the boots they have selected are likely to be the incorrect size before they make the purchase.

During this module, we create a machine-learning model that implements this idea. Along the way, we use this scenario to introduce you to some basic machine-learning concepts and demonstrate how to use them in a practical setting.

**Learning objectives**

In this module, you'll:

* Explore how machine learning differs from traditional software.
* Create and test a machine-learning model.
* Load a model and use it with new data.

**Unit-02**

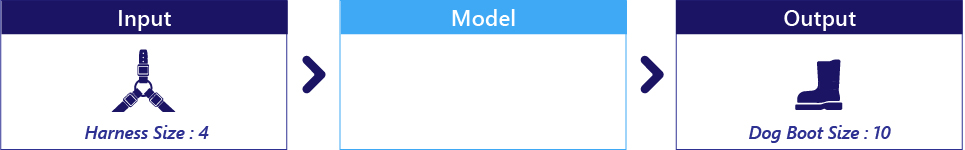
**What are machine learning models?**

The model is the core component of machine learning, and ultimately what we're trying to build. A model might estimate how old a person is from a photo, predict what you might like to see on social media, or decide where a robotic arm should move. In our scenario, we want to build a model that can estimate the best boot size for a dog based on their harness size.

Models can be built in many ways. For example, a traditional model that simulates how an airplane flies is built by people, using knowledge of physics and engineering. Machine-learning models are special; rather than being edited by people so that they work well, machine learning models are shaped by data. They learn from experience.

**How to think about models**

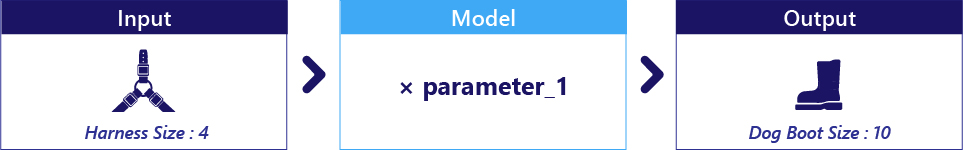
You can think of a model as a function that accepts data as an input and produces an output. More specifically, a model uses input data to estimate something else. For example, in our scenario, we want to build a model that's given a harness size and estimates boot size:



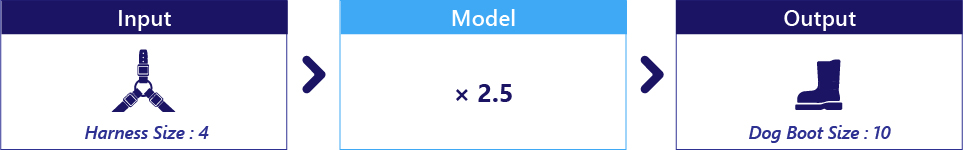
Harness size and dog boot size are data; they aren't part of the model. Harness size is our input, dog boot size is the output.

**Models are often simple code**

Models are often not meaningfully different from simple functions with which you're already familiar. Like other code, they contain logic and parameters. For example, the logic might be *multiply the harness size by parameter\_1*:



If parameter\_1 here was 2.5, our model would multiply harness size by 2.5 and return the result:

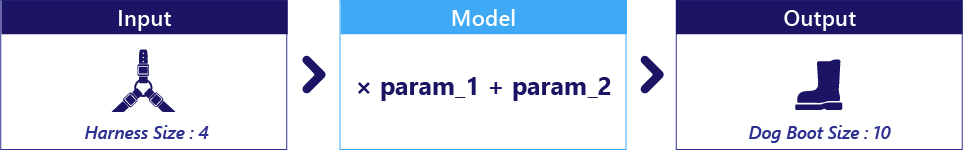


**Select a model**

There are many model types, some simple and some complex.

Like all code, simpler models are often the most reliable and easy to understand, while complex models can potentially perform impressive feats. Which kind of model you should choose depends on your goal. For example, medical scientists often work with models that are relatively simple, because they're reliable and intuitive. By contrast, AI-based robots typically rely on complex models.

The first step in machine learning is selecting the kind of model that you'd like to use. So, we're choosing a model based on its internal logic. For example, we might select a two-parameter model to estimate dog boot size from harness size:

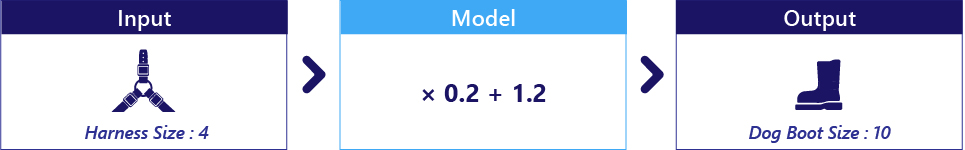


Notice how we selected a model based on how it works logically, but not based on its parameter values. In fact, at this point, the parameters haven't been set to any particular value.

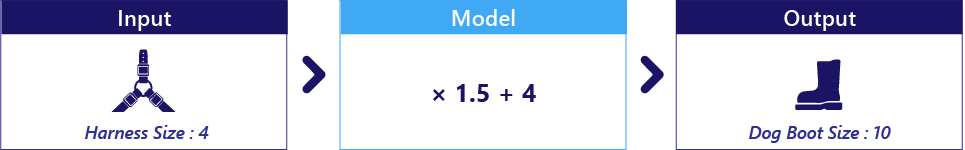
**Parameters are discovered during training**

The human designer doesn't select parameter values. Instead, parameter values are set to an initial guess, then adjusted during an automated learning process called training.

Given our selection of a two-parameter model, we start by providing random guesses for our parameters:



These random parameters mean the model isn’t good at estimating boot size, so we perform training. During training, these parameters are automatically changed to two new values that give better results:



Exactly how this process works is something we progressively explain throughout your learning journey.

# Unit-03

# Exercise: Train and Run Your First Model

We've learned that models are computer code that processes information to make a prediction or a decision. Here, we train a model to guess a comfortable boot size for a dog, based on the size of the harness that fits it.

In the following examples, there's no need to edit any code. Try to read it, understand it, then press the **Run** button to run it. As always with these notebooks, it's vitally important that these code blocks are run in the correct order, and nothing is missed.

## Preparing data

The first thing we do with a model is load data. We cover this in more detail in a later exercise. For now, we just write our data directly in our code. Review and run the following code to get started:

import pandas

!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/graphing.py

!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/Data/doggy-boot-harness.csv

!pip install statsmodels

# Make a dictionary of data for boot sizes

# and harness sizes in cm

data = {

    'boot\_size' : [ 39, 38, 37, 39, 38, 35, 37, 36, 35, 40,

                    40, 36, 38, 39, 42, 42, 36, 36, 35, 41,

                    42, 38, 37, 35, 40, 36, 35, 39, 41, 37,

                    35, 41, 39, 41, 42, 42, 36, 37, 37, 39,

                    42, 35, 36, 41, 41, 41, 39, 39, 35, 39

 ],

    'harness\_size': [ 58, 58, 52, 58, 57, 52, 55, 53, 49, 54,

                59, 56, 53, 58, 57, 58, 56, 51, 50, 59,

                59, 59, 55, 50, 55, 52, 53, 54, 61, 56,

                55, 60, 57, 56, 61, 58, 53, 57, 57, 55,

                60, 51, 52, 56, 55, 57, 58, 57, 51, 59

                ]

}

# Convert it into a table using pandas

dataset = pandas.DataFrame(data)

# Print the data

# In normal python we would write

# print(dataset)

# but in Jupyter notebooks, we simply write the name

# of the variable and it is printed nicely

dataset

As you can see, we have the sizes of boots and harnesses for 50 avalanche dogs.

We want to use harness size to estimate boot size. This means harness\_size is our input. We want a model that will process the input and make its own estimations of the boot size (output).

## Select a model

The first thing we need to do is select a model. We're just getting started, so let's start with a very simple model called OLS. This is just a straight line (sometimes called a trendline).

Let's use an existing library to create our model, but we won't train it yet.

# Load a library to do the hard work for us

import statsmodels.formula.api as smf

# First, we define our formula using a special syntax

# This says that boot\_size is explained by harness\_size

formula = "boot\_size ~ harness\_size"

# Create the model, but don't train it yet

model = smf.ols(formula = formula, data = dataset)

# Note that we have created our model but it does not

# have internal parameters set yet

if not hasattr(model, 'params'):

    print("Model selected but it does not have parameters set. We need to train it!")

**Output:**

Model selected but it does not have parameters set. We need to train it!

## Train our model

OLS models have two parameters (a slope and an offset), but these haven't been set in our model yet. We need to train (fit) our model to find these values so that the model can reliably estimate dogs' boot size based on their harness size.

The following code fits our model to data you've now seen:

# Train (fit) the model so that it creates a line that

# fits our data. This method does the hard work for

# us. We will look at how this method works in a later unit.

fitted\_model = model.fit()

# Print information about our model now it has been fit

print("The following model parameters have been found:\n" +

        f"Line slope: {fitted\_model.params[1]}\n"+

        f"Line Intercept: {fitted\_model.params[0]}")

**Output:**

The following model parameters have been found: Line slope: 0.5859254167382711 Line Intercept: 5.719109812682577

Notice how training the model set its parameters. We could interpret these directly, but it's simpler to see it as a graph:

import matplotlib.pyplot as plt

# Show a scatter plot of the data points and add the fitted line

# Don't worry about how this works for now

plt.scatter(dataset["harness\_size"], dataset["boot\_size"])

plt.plot(dataset["harness\_size"], fitted\_model.params[1] \* dataset["harness\_size"] + fitted\_model.params[0], 'r', label='Fitted line')

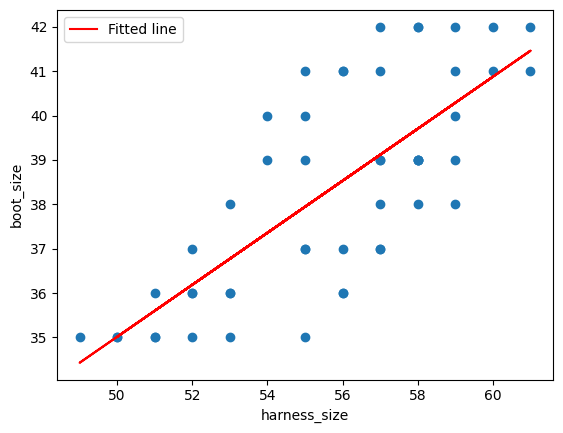
# add labels and legend

plt.xlabel("harness\_size")

plt.ylabel("boot\_size")

plt.legend()

**Output:**

****

The preceding graph shows our original data as circles with a red line through it. The red line shows our model.

We can look at this line to understand our model. For example, we can see that as harness size increases, so does the estimated boot size.

## Use the model

Now that we've finished training, we can use our model to predict a dog's boot size from their harness size.

For example, by looking at the red line, we can see that that a harness size of 52.5 (x axis) corresponds to a boot size of about 36.5 (y axis).

We don't have to do this by eye, though. We can use the model in our program to predict any boot size we like. Run the following code to see how we can use our model now that it's trained:

# harness\_size states the size of the harness we are interested in

harness\_size = { 'harness\_size' : [52.5] }

# Use the model to predict what size of boots the dog will fit

approximate\_boot\_size = fitted\_model.predict(harness\_size)

# Print the result

print("Estimated approximate\_boot\_size:")

print(approximate\_boot\_size[0])

**Output:**

Estimated approximate\_boot\_size: 36.48019419144181

If you'd like, change the value of 52.5 in harness\_size to a new value and run the preceding block to see the model in action.

## Summary

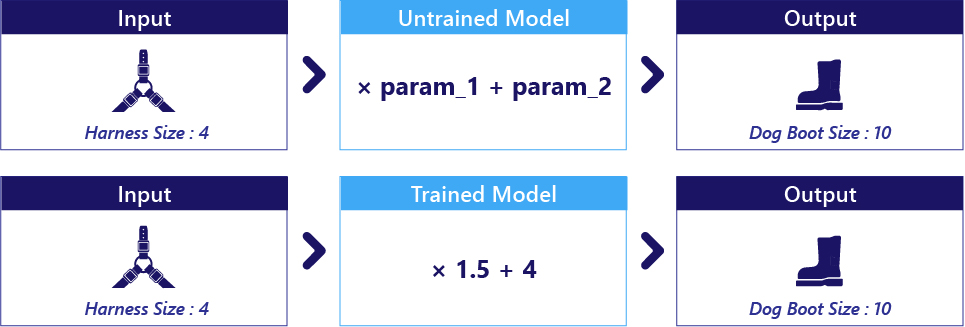
Well done! You've trained your first model. We've demonstrated some topics here without detailed explanation in order to just get your feet wet. In later units, we explain many of these topics in more detail.

# Unit-04

# What are inputs and outputs?

The goal of training is to improve a model so that it can make high-quality estimations or predictions. Once trained, you can use a model in the real world like normal software.

Models don't train themselves. They're trained using data plus two pieces of code, the objective function and the optimizer. Let's explore how these components work together to train a model to work well.



## The objective

The objective is what we want the model to be able to do. For example, our scenario's objective is to be able to estimate a dog's boot size based on their harness size.

For a computer to understand our objective, we need to provide our goal as a code snippet called an objective function (also known as a cost function). Objective functions judge whether the model is doing a good job (estimating boot size well) or bad job (estimating boot size badly). We cover objective functions in more depth in later learning material.

## The data

Data refers to the information that we provide to the model (also known as inputs). In our scenario, the input is harness size.

Data also refers to information that the objective function might need. For example, if our objective function reports whether the model guessed the boot size correctly, it needs to know the correct boot size! For this reason, in our previous exercise, we provided both harness sizes and the correct answers to the training code.

We'll practice working with data in the next exercise.

## The optimizer

During training, the model makes a prediction, and the objective function calculates how well it performed. The optimizer is code that then changes the model's parameters so the model will do a better job next time.

How an optimizer adjusts the parameters is complex, and something we cover in later material. Don't be intimidated, though; we don't normally write our own optimizers, we use open-source frameworks where the hard work has been done for us.

It's important to keep in mind that the objective, data, and optimizer are simply a means to train the model. They aren't needed once training is complete. It's also important to remember that training only changes the parameter values inside of a model; it doesn't change what kind of model is used.

# Unit-05

# Exercise: Datasets in Python

In the previous exercise, we loaded some data and fit a model to it. Several aspects of this were simplified, particularly that the data was hard-coded into our python script, and we didn't spend any time really looking at the data itself.

Here, we load data from a file, filter it, and graph it. Doing so is a very important first step in order to build proper models or to understand their limitations.

As before, there's no need to edit any code in this unit's examples. Try to read it, understand it, then press the **Run** button to run it. As always, it's vitally important that these code blocks are run in the correct order, and nothing is missed.

## Load data with Pandas

There are various libraries that help you work with data. In Python, one of the most common libraries is Pandas. We used pandas briefly in the previous exercise. Pandas can open data saved as text files and store it in an organized table called a DataFrame.

Let's open some text data that's stored on disk. Our data is saved in a file called doggy-boot-harness.csv.

import pandas

!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/graphing.py

!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/Data/doggy-boot-harness.csv

# Read the text file containing data using pandas

dataset = pandas.read\_csv('doggy-boot-harness.csv')

# Print the data

# Because there are a lot of data, use head() to only print the first few rows

dataset.head()

As you can see, this dataset contains information about dogs, including their doggy boot size, harness size, sex, and age in years.

Data is stored as columns and rows, similar to a table you might see in Excel.

## Filter data by Columns

Data is easy to filter by columns. We can either type this directly, like dataset.my\_column\_name, or like so: dataset["my\_column\_name"].

We can use this to either extract data, or to delete data.

Let's take a look at the harness sizes, and delete the sex and age\_years columns.

# Look at the harness sizes

print("Harness sizes")

print(dataset.harness\_size)

# Remove the sex and age-in-years columns.

del dataset["sex"]

del dataset["age\_years"]

# Print the column names

print("\nAvailable columns after deleting sex and age information:")

print(dataset.columns.values)

## Filter data by Rows

We can get data from the top of the table by using the head() function, or from the bottom of the table by using the tail() function.

Both functions make a shallow copy of a section of our dataframe. Here, we're sending these copies to the print() function. We can also use the head and tail views for other purposes, such as for use in analyses or graphs.

# Print the data at the top of the table

print("TOP OF TABLE")

print(dataset.head())

# print the data at the bottom of the table

print("\nBOTTOM OF TABLE")

print(dataset.tail())

We can also filter logically. For example, we can look at data for dogs who have a harness smaller than a size 55.

This works by calculating a True or False value for each row, then keeping only those rows where the value is True.

# Print how many rows of data we have

print(f"We have {len(dataset)} rows of data")

# Determine whether each avalanche dog's harness size is < 55

# This creates a True or False value for each row where True means

# they are smaller than 55

is\_small = dataset.harness\_size < 55

print("\nWhether the dog's harness was smaller than size 55:")

print(is\_small)

# Now apply this 'mask' to our data to keep the smaller dogs

data\_from\_small\_dogs = dataset[is\_small]

print("\nData for dogs with harness smaller than size 55:")

print(data\_from\_small\_dogs)

# Print the number of small dogs

print(f"\nNumber of dogs with harness size less than 55: {len(data\_from\_small\_dogs)}")

This looks like a lot of code, but we can compress the important parts into a single line.

Let's do something similar: restrict our data to only those with boot sizes smaller than 40.

# Make a copy of the dataset that only contains dogs with

# a boot size below size 40

# The call to copy() is optional but can help avoid unexpected

# behaviour in more complex scenarios

data\_smaller\_paws = dataset[dataset.boot\_size < 40].copy()

# Print information about this

print(f"We now have {len(data\_smaller\_paws)} rows in our dataset. The last few rows are:")

data\_smaller\_paws.tail()

## Graph Data

Graphing your data is often the easiest way to understand it.

Lets make a simple graph of harness size versus boot size for our avalanche dogs with smaller feet.

# Load and prepare matplotlib to use for plotting graphs

import matplotlib.pyplot as plt

# Show a graph of harness size by boot size:

plt.scatter(data\_smaller\_paws["harness\_size"], data\_smaller\_paws["boot\_size"])

# add labels

plt.xlabel("harness\_size")

plt.ylabel("boot\_size")

## Create New Columns

The preceding graph shows the relationship we want to investigate for our store, but some customers might want harness-size lists in inches, not centimeters. How can we view these harness sizes in imperial units?

To do this, we need to create a new column called harness\_size\_imperial and put that on the X axis instead.

Creating new columns uses very similar syntax to what we've seen before.

# Convert harness sizes from metric to imperial units

# and save the result to a new column

data\_smaller\_paws['harness\_size\_imperial'] = data\_smaller\_paws.harness\_size / 2.54

# Show a graph of harness size in imperial units

plt.scatter(data\_smaller\_paws["harness\_size\_imperial"], data\_smaller\_paws["boot\_size"])

plt.xlabel("harness\_size\_imperial")

plt.ylabel("boot\_size")

We've now graphed our new column of data (harness\_size\_imperial) against boot size for dogs with small paws.

# Unit-06

# How to use a model

Completed 100 XP

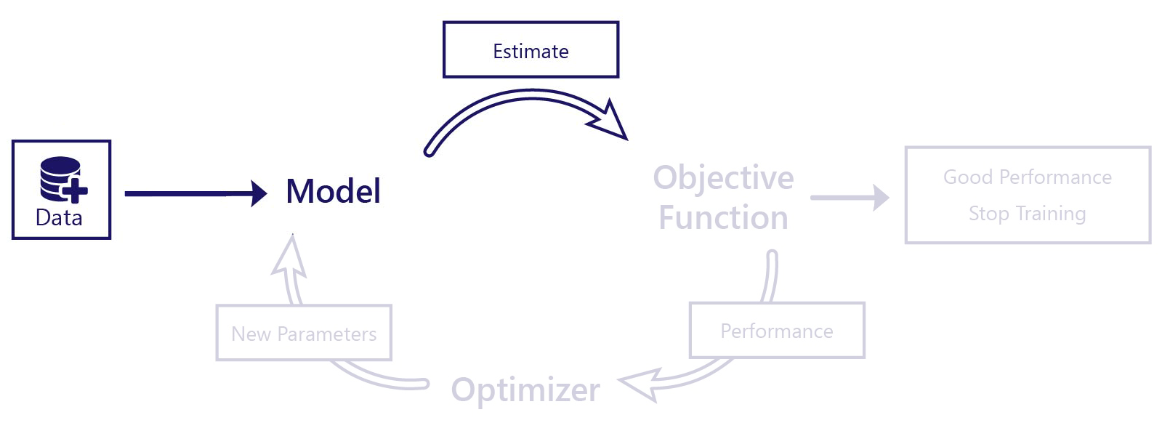
* 6 minutes

Let's revise how these parts fit together to train a model.

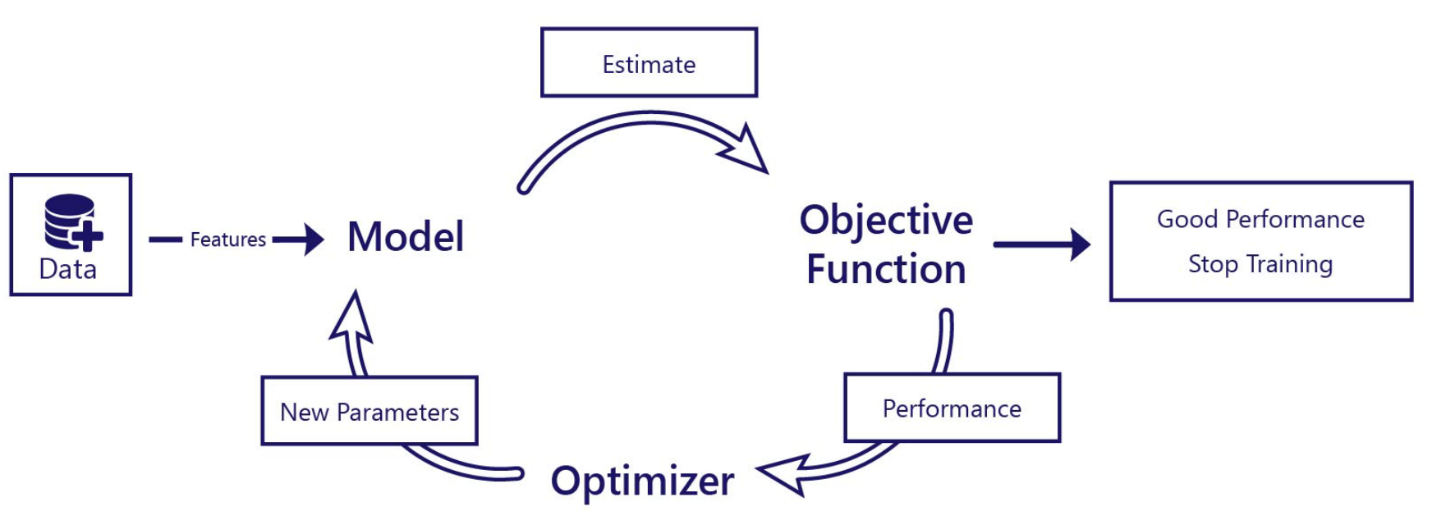
## Training versus using a model

It's important to make a distinction between training and using a model.

Using a model means providing inputs and receiving an estimation or prediction. We do this process both when we're training our model and when we or our customers use it in the real world. Using a model normally takes less than a few seconds.



In contrast, training a model is the process of improving how well a model works. Training requires that we use the model, the objective function, and the optimizer in a special loop. Training can take minutes or days to complete. Usually, we only train a model once. Once it's trained, we can use it as many times as we like without making further changes.



For example, in our avalanche-rescue dog store scenario, we want to train a model using a public dataset. The dataset changes the model so that it can predict a dog's boot size based on their harness size. Once our model is trained, we use the model as part of our online store to make sure customers are buying doggy boots that fit their dogs.

## Data for use, data for training

Recall that a dataset is a collection of information about objects or things. For example, a dataset might contain information about dogs:

| **Dog ID** | **Boot Size** | **Harness Size** | **Dog Color** | **Breed** |
| --- | --- | --- | --- | --- |
| 0 | 27 | 12 | Brown | St Bernard |
| 1 | 26 | 11 | Black | Labrador |
| 2 | 25 | 10 | White | Labrador |
| 3 | 29 | 14 | Black | Black Shepherd |

When we use our model, we only need the column(s) of data that the model accepts as input. These columns are called features. In our scenario, if our model accepts harness size and estimates boot size, then our feature is harness size.

During training, the objective function usually needs to know both the model's output and what the correct answer is. These values are called labels. In our scenario, if our model predicts boot size, then boot size is our label.

So, to use a model, we only ever need features, while during training we usually need both features and labels. During training in our scenario, we need both our harness-size feature and our boot-size label. When we use our model in our website, we only need to know the harness-size feature; our model then estimates the boot size for us to use.

## I've finished training. What now?

Once a model has finished training, you can save it to a file by itself. **We no longer need the original data, the objective function, or the model optimizer**. When we want to use the model, we can load it from disk, provide it with new data, and get back a prediction.

In our next exercise, we practice saving a model, loading it from disk, and using it as we would in the real world. To complete our online store scenario, we also practice using the model's outputs to warn our customers if they seem to be buying the wrong size doggy boots.

# Unit-07

# Exercise: Using a Trained Model on New Data

In Unit 3, we created a basic model that let us find the relationship between a dog's harness size and their boot size. We showed we could then use this model to make a prediction about a new, previously unseen dog.

It's common to build, train, then use a model while we're just learning about machine learning; but in the real world, we don't want to train the model every time we want to make a prediction.

Consider our avalanche-dog equipment store scenario:

* We want to train the model just once, then load that model onto the server that runs our online store.
* Although the model is trained on a dataset we downloaded from the internet, we actually want to use it to estimate the boot size of our customers' dogs who aren't in this dataset!

How can we do this?

Here, we'll:

1. Create a basic model.
2. Save it to disk.
3. Load it from disk.
4. Use it to make predictions about a dog who was not in the training dataset.

## Load the dataset

Let's begin by opening the dataset from file.

import pandas

!pip install statsmodels

!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/graphing.py

!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/Data/doggy-boot-harness.csv

# Load a file containing dog's boot and harness sizes

data = pandas.read\_csv('doggy-boot-harness.csv')

# Print the first few rows

data.head()

## Create and train a model

As we've done before, we create a simple Linear Regression model and train it on our dataset.

import statsmodels.formula.api as smf

# Fit a simple model that finds a linear relationship

# between boot size and harness size, which we can use later

# to predict a dog's boot size, given their harness size

model = smf.ols(formula = "boot\_size ~ harness\_size", data = data).fit()

print("Model trained!")

## Save and load a model

Our model is ready to use, but we don't need it yet. Let's save it to disk.

import joblib

model\_filename = './avalanche\_dog\_boot\_model.pkl'

joblib.dump(model, model\_filename)

print("Model saved!")

Loading our model is just as easy:

model\_loaded = joblib.load(model\_filename)

print("We have loaded a model with the following parameters:")

print(model\_loaded.params)

On our website, we want to take the harness of our customer's dog, then calculate their dog's boot size using the model that we've already trained.

Let's put everything here together to make a function that loads the model from disk, then uses it to predict our customer's dog's boot size.

# Let's write a function that loads and uses our model

def load\_model\_and\_predict(harness\_size):

    '''

    This function loads a pretrained model. It uses the model

    with the customer's dog's harness size to predict the size of

    boots that will fit that dog.

    harness\_size: The dog harness size, in cm

    '''

    # Load the model from file and print basic information about it

    loaded\_model = joblib.load(model\_filename)

    print("We've loaded a model with the following parameters:")

    print(loaded\_model.params)

    # Prepare data for the model

    inputs = {"harness\_size":[harness\_size]}

    # Use the model to make a prediction

    predicted\_boot\_size = loaded\_model.predict(inputs)[0]

    return predicted\_boot\_size

# Practice using our model

predicted\_boot\_size = load\_model\_and\_predict(45)

print("Predicted dog boot size:", predicted\_boot\_size)

## Real-world use

We've done it; we can predict an avalanche dog's boot size based on the size of their harness. Our last step is to use this to warn people if they might be buying the wrong size doggy boots.

As an example, we make a function that accepts the harness size and the size of the boots selected, then returns a message for the customer. We'd integrate this function into our online store.

def check\_size\_of\_boots(selected\_harness\_size, selected\_boot\_size):

    '''

    Calculates whether the customer has chosen a pair of doggy boots that

    are a sensible size. This works by estimating the dog's actual boot

    size from their harness size.

    This returns a message for the customer that should be shown before

    they complete their payment

    selected\_harness\_size: The size of the harness the customer wants to buy

    selected\_boot\_size: The size of the doggy boots the customer wants to buy

    '''

    # Estimate the customer's dog's boot size

    estimated\_boot\_size = load\_model\_and\_predict(selected\_harness\_size)

    # Round to the nearest whole number because we don't sell partial sizes

    estimated\_boot\_size = int(round(estimated\_boot\_size))

    # Check if the boot size selected is appropriate

    if selected\_boot\_size == estimated\_boot\_size:

        # The selected boots are probably OK

        return f"Great choice! We think these boots will fit your avalanche dog well."

    if selected\_boot\_size < estimated\_boot\_size:

        # Selected boots might be too small

        return "The boots you have selected might be TOO SMALL for a dog as "\

               f"big as yours. We recommend a doggy boots size of {estimated\_boot\_size}."

    if selected\_boot\_size > estimated\_boot\_size:

        # Selected boots might be too big

        return "The boots you have selected might be TOO BIG for a dog as "\

               f"small as yours. We recommend a doggy boots size of {estimated\_boot\_size}."

# Practice using our new warning system

check\_size\_of\_boots(selected\_harness\_size=55, selected\_boot\_size=39)

Change selected\_harness\_size and selected\_boot\_size in the preceding example and re-run the cell to see this in action.

## Summary

Well done! We've put together a system that can predict if customers are buying doggy boots that might not fit their avalanche dog, based solely on the size of harness they're purchasing.

In this exercise, we practiced:

1. Creating basic models.
2. Training, then saving them to disk.
3. Loading them from disk.
4. Making predictions with them using new data sets.

# Unit-08

* Module Assesment(Done with 100% Accuracy)

# Unit-09

# Summary

We covered some significant new jargon in this module. Let's recap what we've learned:

* The goal of machine learning is to find patterns in data and use these patterns to make estimates.
* Machine learning differs from normal software development in that we use special code, rather than our own intuition, to improve how well the software works.
* The learning process conceptually uses four components:
  + **Data**, which is information we want to learn from.
  + A **model**, which makes estimates about the data.
  + An **objective** the model is trying to achieve.
  + An **optimizer**, extra code that changes the model depending on its performance.
* You can think of data as features and labels. Features correspond to potential model inputs, while labels correspond to model outputs, or desired model outputs.
* Pandas and Plotly are powerful tools to explore datasets in Python.
* Once we have a trained model, we can save it to disk for later use.