100 XP

Exercise - Create a machine learning model

8 minutes

Sandbox activated! Time remaining: 22 min

You have used 1 of 10 sandboxes for today. More sandboxes will be available tomorrow.

learn-notebooks-e7ce49a0-5e7f-4f9d-b581-73cca09c388a · Kernel idle Python 3.7.9

Exercise: Training and Running Your First Model

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

In the examples below, there is 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 is vitally important that these code blocks are run in the correct order, and nothing is missed.

> **Note**: If you've never used the Jupyter Notebooks environment before, there are a few things you should be aware of:
>
> - Notebooks are made up of *cells*. Some cells (like this one) contain *markdown* text, while others (like the one beneath this one) contain code.
> - You can run each code cell by using the ► **Run** button. the ► **Run** button will show up when you hover over the cell.
> - The output from each code cell will be displayed immediately below the cell.
> - Even though the code cells can be run individually, some variables used in the code are global to the notebook. That means that you should run all of the code cells **in order**. There may be dependencies between code cells, so if you skip a cell, subsequent cells might not run correctly.

Preparing data

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

```
import pandas
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to
!pip install statsmodels


# Make a dictionary of data for boot sizes
# and harness size 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, if we simple write the name
# of the variable and it is printed nicely
dataset
```

[1] ✓ 12 sec

...

18	35	50
19	41	59
20	42	59
21	38	59
22	37	55
23	35	50
24	40	55
25	36	52
26	35	53
27	39	54
28	41	61
29	37	56

30	35	55
31	41	60
32	39	57
33	41	56
34	42	61
35	42	58
36	36	53
37	37	57
38	37	57
39	39	55
40	42	60
41	35	51
42	36	52

+ Code + Markdown

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 the harness size (output).

Selecting a model

The first thing we must do is select a model. We're just getting started, so we will 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 tra
```

[2] ✓ 1 sec

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

Training our model

OLS models have two parameters (a slope and an offset), but these have not 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 code below fits our model to data you have now seen

```
# Load some libraries to do the hard work for us
import graphing

# 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]}")
```

[3] ✓ 3 sec

```
The following model parameters have been found:

Line slope: 0.5859254167382707

Line Intercept: 5.719109812682602
```

```
fitted_model.params
```

[4] ✓ <1 sec

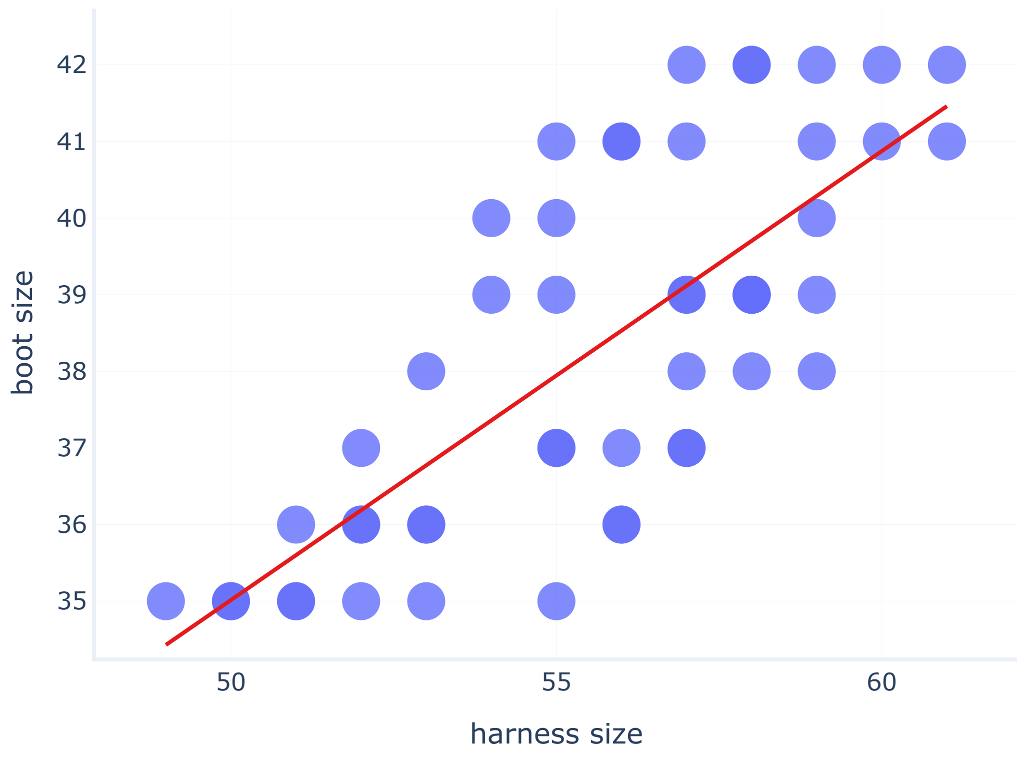
```
Intercept       5.719110
harness_size    0.585925
dtype: float64
```

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

```
import graphing
```

```
# Show a graph of the result
# Don't worry about how this works for now
graphing.scatter_2D(dataset,    label_x="harness_size",
                                label_y="boot_size",
                                trendline=lambda x: fitted_model.params[1] *
................................)
```

[5] ✓ 12 sec



The graph above 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 will the estimated boot size.

Using the model

Now 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 code below to see how we can use our model now it is 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])
```

[ ] Press shift + enter to execute cells

If you would like, change the value of `52.5` in `harness_size` to a new value and run the block above to see the model in action.

Summary

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

Next unit: What are inputs and outputs?

Continue >