Standardizing Data

Learn about data standardization and implement it with scikit-learn.

Chapter Goals:

- Learn about data standardization

A. Standard data format

Data can contain all sorts of different values. For example, Olympic 100m sprint times will range from 9.5 to 10.5 seconds, while calorie counts in large pepperoni pizzas can range from 1500 to 3000 calories. Even data measuring the exact same quantities can range in value (e.g. weight in kilograms vs. weight in pounds).

When data can take on any range of values, it makes it difficult to interpret. Therefore, data scientists will convert the data into a standard format to make it easier to understand. The standard format refers to data that has 0 mean and unit variance (i.e. standard deviation = 1), and the process of converting data into this format is called *data standardization*.

Data standardization is a relatively simple process. For each data value, x, we subtract the overall mean of the data, μ, then divide by the overall standard deviation, σ. The new value, z, represents the standardized data value. Thus, the formula for data standardization is:

$$z = \frac{x - \mu}{\sigma}$$

B. NumPy and scikit-learn

For most scikit-learn functions, the input data comes in the form of a NumPy array.

Note: The array's rows represent individual data observations, while each column represents a particular feature of the data, i.e. the same format as a

spreadsheet data table.

The scikit-learn data preprocessing module is called `sklearn.preprocessing`. One of the functions in this module, `scale`, applies data standardization to a given axis of a NumPy array.

```
# predefined pizza data
# Newline to separate print statements
print('{}\n'.format(repr(pizza_data)))

from sklearn.preprocessing import scale
# Standardizing each column of pizza_data
col_standardized = scale(pizza_data)
print('{}\n'.format(repr(col_standardized)))

# Column means (rounded to nearest thousandth)
col_means = col_standardized.mean(axis=0).round(decimals=3)
print('{}\n'.format(repr(col_means)))

# Column standard deviations
col_stds = col_standardized.std(axis=0)
print('{}\n'.format(repr(col_stds)))
```

We normally standardize the data independently across each feature of the data array. This way, we can see how many standard deviations a particular observation's feature value is from the mean.

For example, the second data observation in `pizza_data` has a net weight of 1.6 standard deviations above the mean pizza weight in the dataset.

If for some reason we need to standardize the data across rows, rather than columns, we can set the `axis` keyword argument in the `scale` function to 1. This may be the case when analyzing data within observations, rather than within a feature. An example of this would be analyzing a particular student's test scores in terms of standard deviations from that student's average test score.

Time to Code!

The coding exercise in this chapter is to complete a generic data standardization function, `standardize_data`.

This function will standardize the input NumPy array, `data`, by using the

`scale` function (imported in the backend).

Set `scaled_data` equal to `scale` applied with `data` as the only argument. Then return `scaled_data`.

```
def standardize_data(data):
  # CODE HERE
  pass
```