Introducing Scikit-Learn

There are several Python libraries that provide solid implementations of a range of machine learning algorithms. One of the best known is Scikit-Learn, a package that provides efficient versions of a large number of common algorithms. Scikit-Learn is characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation. A benefit of this uniformity is that once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward.

This section provides an overview of the Scikit-Learn API; a solid understanding of these API elements will form the foundation for understanding the deeper practical discussion of machine learning algorithms and approaches in the following chapters.

We will start by covering *data representation* in Scikit-Learn, followed by covering the *Estimator* API, and finally go through a more interesting example of using these tools for exploring a set of images of handwritten digits.

Data Representation in Scikit-Learn

Machine learning is about creating models from data: for that reason, we'll start by discussing how data can be represented in order to be understood by the computer. The best way to think about data within Scikit-Learn is in terms of tables of data.

Data as table

A basic table is a two-dimensional grid of data, in which the rows represent individual elements of the dataset, and the columns represent quantities related to each of these elements. For example, consider the Iris dataset, famously analyzed by Ronald Fisher in 1936. We can download this dataset in the form of a Pandas `DataFrame` using the Seaborn library:

```
In[1]: import seaborn as sns
       iris = sns.load_dataset('iris')
       iris.head()
```

```
Out[1]:    sepal_length  sepal_width  petal_length  petal_width species
        0           5.1          3.5           1.4          0.2  setosa
        1           4.9          3.0           1.4          0.2  setosa
        2           4.7          3.2           1.3          0.2  setosa
        3           4.6          3.1           1.5          0.2  setosa
        4           5.0          3.6           1.4          0.2  setosa
```

Here each row of the data refers to a single observed flower, and the number of rows is the total number of flowers in the dataset. In general, we will refer to the rows of the matrix as *samples*, and the number of rows as `n_samples`.

Likewise, each column of the data refers to a particular quantitative piece of information that describes each sample. In general, we will refer to the columns of the matrix as *features*, and the number of columns as `n_features`.

Features matrix

This table layout makes clear that the information can be thought of as a two-dimensional numerical array or matrix, which we will call the *features matrix*. By convention, this features matrix is often stored in a variable named `X`. The features matrix is assumed to be two-dimensional, with shape `[n_samples, n_features]`, and is most often contained in a NumPy array or a Pandas `DataFrame`, though some Scikit-Learn models also accept SciPy sparse matrices.

The samples (i.e., rows) always refer to the individual objects described by the dataset. For example, the sample might be a flower, a person, a document, an image, a sound file, a video, an astronomical object, or anything else you can describe with a set of quantitative measurements.

The features (i.e., columns) always refer to the distinct observations that describe each sample in a quantitative manner. Features are generally real-valued, but may be Boolean or discrete-valued in some cases.

Target array

In addition to the feature matrix `X`, we also generally work with a *label* or *target* array, which by convention we will usually call `y`. The target array is usually one dimensional, with length `n_samples`, and is generally contained in a NumPy array or Pandas `Series`. The target array may have continuous numerical values, or discrete classes/labels. While some Scikit-Learn estimators do handle multiple target values in the form of a two-dimensional `[n_samples, n_targets]` target array, we will primarily be working with the common case of a one-dimensional target array.

Often one point of confusion is how the target array differs from the other features columns. The distinguishing feature of the target array is that it is usually the quantity we want to *predict from the data*: in statistical terms, it is the dependent variable. For example, in the preceding data we may wish to construct a model that can predict the species of flower based on the other measurements; in this case, the `species` column would be considered the feature.

With this target array in mind, we can use Seaborn (discussed earlier in "Visualization with Seaborn" on page 311) to conveniently visualize the data (see Figure 5-12):

```
In[2]: %matplotlib inline
       import seaborn as sns; sns.set()
       sns.pairplot(iris, hue='species', size=1.5);
```

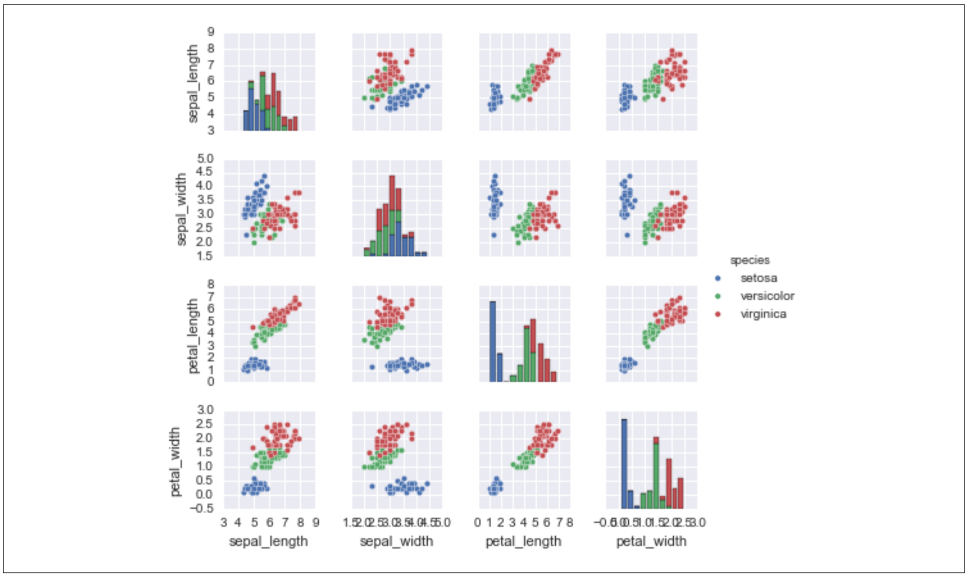

Figure 5-12. A visualization of the Iris dataset

For use in Scikit-Learn, we will extract the features matrix and target array from the `DataFrame`, which we can do using some of the Pandas `DataFrame` operations discussed in Chapter 3:

```
In[3]: X_iris = iris.drop('species', axis=1)
       X_iris.shape

Out[3]: (150, 4)

In[4]: y_iris = iris['species']
       y_iris.shape

Out[4]: (150,)
```

To summarize, the expected layout of features and target values is visualized in Figure 5-13.

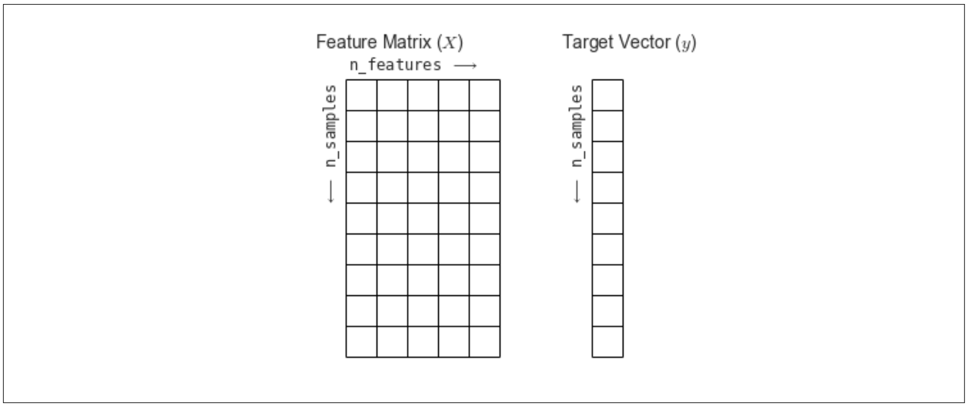

Figure 5-13. Scikit-Learn's data layout

With this data properly formatted, we can move on to consider the *estimator* API of Scikit-Learn.

Scikit-Learn's Estimator API

The Scikit-Learn API is designed with the following guiding principles in mind, as outlined in the Scikit-Learn API paper:

Consistency
: All objects share a common interface drawn from a limited set of methods, with consistent documentation.

Inspection
: All specified parameter values are exposed as public attributes.

Limited object hierarchy
: Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas `DataFrame`s, SciPy sparse matrices) and parameter names use standard Python strings.

Composition
: Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.

Sensible defaults
: When models require user-specified parameters, the library defines an appropriate default value.