dimensionality reduction algorithms becomes clearer in higher-dimensional cases. For example, we might wish to visualize important relationships within a dataset that has 100 or 1,000 features. Visualizing 1,000-dimensional data is a challenge, and one way we can make this more manageable is to use a dimensionality reduction technique to reduce the data to two or three dimensions.

Some important dimensionality reduction algorithms that we will discuss are principal component analysis (see "In Depth: Principal Component Analysis" on page 433) and various manifold learning algorithms, including Isomap and locally linear embedding (see "In-Depth: Manifold Learning" on page 445).

Summary

Here we have seen a few simple examples of some of the basic types of machine learning approaches. Needless to say, there are a number of important practical details that we have glossed over, but I hope this section was enough to give you a basic idea of what types of problems machine learning approaches can solve.

In short, we saw the following:

Supervised learning
: Models that can predict labels based on labeled training data

Classification
: Models that predict labels as two or more discrete categories

Regression
: Models that predict continuous labels

Unsupervised learning
: Models that identify structure in unlabeled data

Clustering
: Models that detect and identify distinct groups in the data

Dimensionality reduction
: Models that detect and identify lower-dimensional structure in higher-dimensional data

In the following sections we will go into much greater depth within these categories, and see some more interesting examples of where these concepts can be useful.

All of the figures in the preceding discussion are generated based on actual machine learning computations; the code behind them can be found in the online appendix.

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`.