

In[29]:

```
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
```

Out[29]:

```
Test set predictions:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]
```

In[30]:

```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

Out[30]:

```
Test set score: 0.97
```

We can also use the `score` method of the `knn` object, which will compute the test set accuracy for us:

In[31]:

```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

Out[31]:

```
Test set score: 0.97
```

For this model, the test set accuracy is about 0.97, which means we made the right prediction for 97% of the irises in the test set. Under some mathematical assumptions, this means that we can expect our model to be correct 97% of the time for new irises. For our hobby botanist application, this high level of accuracy means that our model may be trustworthy enough to use. In later chapters we will discuss how we can improve performance, and what caveats there are in tuning a model.

Summary and Outlook

Let's summarize what we learned in this chapter. We started with a brief introduction to machine learning and its applications, then discussed the distinction between supervised and unsupervised learning and gave an overview of the tools we'll be using in this book. Then, we formulated the task of predicting which species of iris a particular flower belongs to by using physical measurements of the flower. We used a dataset of measurements that was annotated by an expert with the correct species to build our model, making this a supervised learning task. There were three possible species, *setosa*, *versicolor*, or *virginica*, which made the task a three-class classification problem. The possible species are called *classes* in the classification problem, and the species of a single iris is called its *label*.

The Iris dataset consists of two NumPy arrays: one containing the data, which is referred to as `X` in `scikit-learn`, and one containing the correct or desired outputs,

which is called `y`. The array `X` is a two-dimensional array of features, with one row per data point and one column per feature. The array `y` is a one-dimensional array, which here contains one class label, an integer ranging from 0 to 2, for each of the samples.

We split our dataset into a *training set*, to build our model, and a *test set*, to evaluate how well our model will generalize to new, previously unseen data.

We chose the *k*-nearest neighbors classification algorithm, which makes predictions for a new data point by considering its closest neighbor(s) in the training set. This is implemented in the `KNeighborsClassifier` class, which contains the algorithm that builds the model as well as the algorithm that makes a prediction using the model. We instantiated the class, setting parameters. Then we built the model by calling the `fit` method, passing the training data (`X_train`) and training outputs (`y_train`) as parameters. We evaluated the model using the `score` method, which computes the accuracy of the model. We applied the `score` method to the test set data and the test set labels and found that our model is about 97% accurate, meaning it is correct 97% of the time on the test set.

This gave us the confidence to apply the model to new data (in our example, new flower measurements) and trust that the model will be correct about 97% of the time.

Here is a summary of the code needed for the whole training and evaluation procedure:

In[32]:

```
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

Out[32]:

```
Test set score: 0.97
```

This snippet contains the core code for applying any machine learning algorithm using `scikit-learn`. The `fit`, `predict`, and `score` methods are the common interface to supervised models in `scikit-learn`, and with the concepts introduced in this chapter, you can apply these models to many machine learning tasks. In the next chapter, we will go into more depth about the different kinds of supervised models in `scikit-learn` and how to apply them successfully.

Supervised Learning

As we mentioned earlier, supervised machine learning is one of the most commonly used and successful types of machine learning. In this chapter, we will describe supervised learning in more detail and explain several popular supervised learning algorithms. We already saw an application of supervised machine learning in [Chapter 1](#): classifying iris flowers into several species using physical measurements of the flowers.

Remember that supervised learning is used whenever we want to predict a certain outcome from a given input, and we have examples of input/output pairs. We build a machine learning model from these input/output pairs, which comprise our training set. Our goal is to make accurate predictions for new, never-before-seen data. Supervised learning often requires human effort to build the training set, but afterward automates and often speeds up an otherwise laborious or infeasible task.

Classification and Regression

There are two major types of supervised machine learning problems, called *classification* and *regression*.

In classification, the goal is to predict a *class label*, which is a choice from a predefined list of possibilities. In [Chapter 1](#) we used the example of classifying irises into one of three possible species. Classification is sometimes separated into *binary classification*, which is the special case of distinguishing between exactly two classes, and *multiclass classification*, which is classification between more than two classes. You can think of binary classification as trying to answer a yes/no question. Classifying emails as either spam or not spam is an example of a binary classification problem. In this binary classification task, the yes/no question being asked would be “Is this email spam?”