

Making Predictions

We can now make predictions using this model on new data for which we might not know the correct labels. Imagine we found an iris in the wild with a sepal length of 5 cm, a sepal width of 2.9 cm, a petal length of 1 cm, and a petal width of 0.2 cm. What species of iris would this be? We can put this data into a NumPy array, again by calculating the shape—that is, the number of samples (1) multiplied by the number of features (4):

In[27]:

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
```

Out[27]:

```
X_new.shape: (1, 4)
```

Note that we made the measurements of this single flower into a row in a two-dimensional NumPy array, as `scikit-learn` always expects two-dimensional arrays for the data.

To make a prediction, we call the `predict` method of the `knn` object:

In[28]:

```
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(
    iris_dataset['target_names'][prediction]))
```

Out[28]:

```
Prediction: [0]
Predicted target name: ['setosa']
```

Our model predicts that this new iris belongs to the class 0, meaning its species is *setosa*. But how do we know whether we can trust our model? We don't know the correct species of this sample, which is the whole point of building the model!

Evaluating the Model

This is where the test set that we created earlier comes in. This data was not used to build the model, but we do know what the correct species is for each iris in the test set.

Therefore, we can make a prediction for each iris in the test data and compare it against its label (the known species). We can measure how well the model works by computing the *accuracy*, which is the fraction of flowers for which the right species was predicted:

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,