view of how each algorithm builds a model. We will examine the strengths and weaknesses of each algorithm, and what kind of data they can best be applied to. We will also explain the meaning of the most important parameters and options.[4] Many algorithms have a classification and a regression variant, and we will describe both.

It is not necessary to read through the descriptions of each algorithm in detail, but understanding the models will give you a better feeling for the different ways machine learning algorithms can work. This chapter can also be used as a reference guide, and you can come back to it when you are unsure about the workings of any of the algorithms.

Some Sample Datasets

We will use several datasets to illustrate the different algorithms. Some of the datasets will be small and synthetic (meaning made-up), designed to highlight particular aspects of the algorithms. Other datasets will be large, real-world examples.

An example of a synthetic two-class classification dataset is the `forge` dataset, which has two features. The following code creates a scatter plot (Figure 2-2) visualizing all of the data points in this dataset. The plot has the first feature on the x-axis and the second feature on the y-axis. As is always the case in scatter plots, each data point is represented as one dot. The color and shape of the dot indicates its class:

In[2]:

```
# generate dataset
X, y = mglearn.datasets.make_forge()
# plot dataset
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Class 0", "Class 1"], loc=4)
plt.xlabel("First feature")
plt.ylabel("Second feature")
print("X.shape: {}".format(X.shape))
```

Out[2]:

```
X.shape: (26, 2)
```

4 Discussing all of them is beyond the scope of the book, and we refer you to the `scikit-learn` documentation for more details.

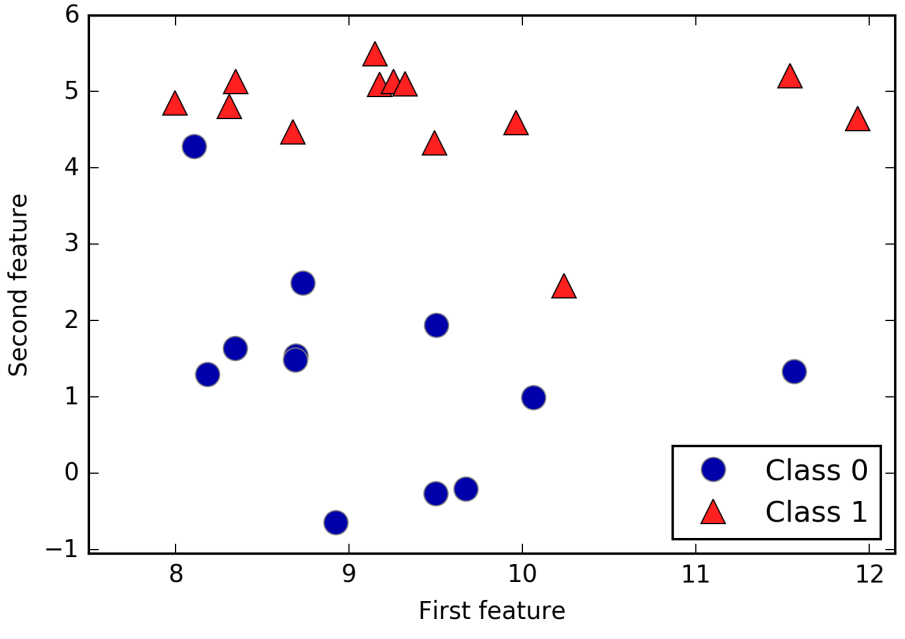

Figure 2-2. Scatter plot of the forge dataset

As you can see from `X.shape`, this dataset consists of 26 data points, with 2 features.

To illustrate regression algorithms, we will use the synthetic `wave` dataset. The `wave` dataset has a single input feature and a continuous target variable (or *response*) that we want to model. The plot created here (Figure 2-3) shows the single feature on the x-axis and the regression target (the output) on the y-axis:

In[3]:

```
X, y = mglearn.datasets.make_wave(n_samples=40)
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("Feature")
plt.ylabel("Target")
```

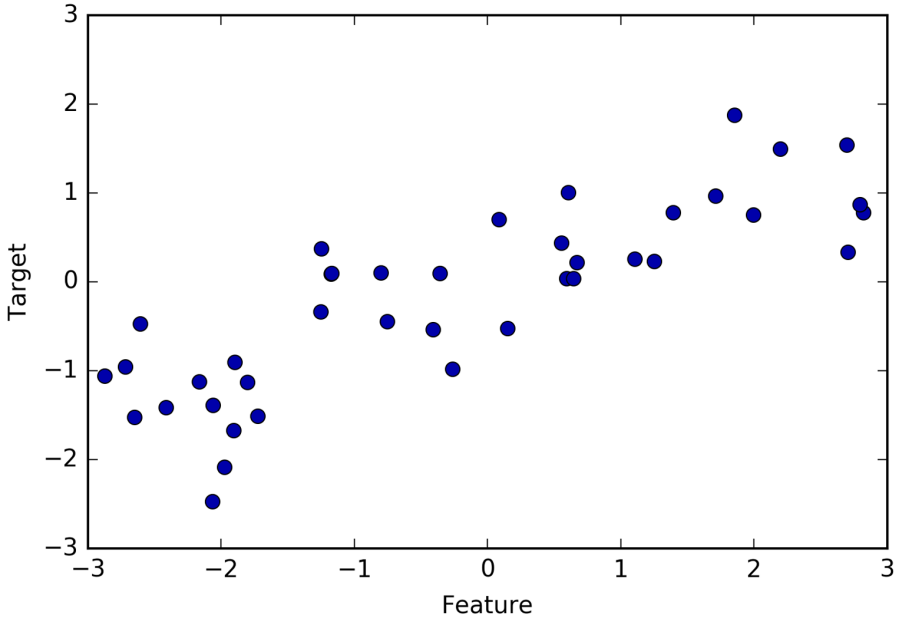

Figure 2-3. Plot of the wave dataset, with the x-axis showing the feature and the y-axis showing the regression target

We are using these very simple, low-dimensional datasets because we can easily visualize them—a printed page has two dimensions, so data with more than two features is hard to show. Any intuition derived from datasets with few features (also called *low-dimensional* datasets) might not hold in datasets with many features (*high-dimensional* datasets). As long as you keep that in mind, inspecting algorithms on low-dimensional datasets can be very instructive.

We will complement these small synthetic datasets with two real-world datasets that are included in `scikit-learn`. One is the Wisconsin Breast Cancer dataset (`cancer`, for short), which records clinical measurements of breast cancer tumors. Each tumor is labeled as "benign" (for harmless tumors) or "malignant" (for cancerous tumors), and the task is to learn to predict whether a tumor is malignant based on the measurements of the tissue.

The data can be loaded using the `load_breast_cancer` function from `scikit-learn`:

`In[4]:`

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): \n{}".format(cancer.keys()))
```

Out[4]:

```
cancer.keys():
dict_keys(['feature_names', 'data', 'DESCR', 'target', 'target_names'])
```

Datasets that are included in `scikit-learn` are usually stored as `Bunch` objects, which contain some information about the dataset as well as the actual data. All you need to know about `Bunch` objects is that they behave like dictionaries, with the added benefit that you can access values using a dot (as in `bunch.key` instead of `bunch['key']`).

The dataset consists of 569 data points, with 30 features each:

In[5]:

```
print("Shape of cancer data: {}".format(cancer.data.shape))
```

Out[5]:

```
Shape of cancer data: (569, 30)
```

Of these 569 data points, 212 are labeled as malignant and 357 as benign:

In[6]:

```
print("Sample counts per class:\n{}".format(
      {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))
```

Out[6]:

```
Sample counts per class:
{'benign': 357, 'malignant': 212}
```

To get a description of the semantic meaning of each feature, we can have a look at the `feature_names` attribute:

In[7]:

```
print("Feature names:\n{}".format(cancer.feature_names))
```

Out[7]:

```
Feature names:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

You can find out more about the data by reading `cancer.DESCR` if you are interested.

We will also be using a real-world regression dataset, the Boston Housing dataset. The task associated with this dataset is to predict the median value of homes in several Boston neighborhoods in the 1970s, using information such as crime rate, proximity to the Charles River, highway accessibility, and so on. The dataset contains 506 data points, described by 13 features:

In[8]:

```
from sklearn.datasets import load_boston
boston = load_boston()
print("Data shape: {}".format(boston.data.shape))
```

Out[8]:

```
Data shape: (506, 13)
```

Again, you can get more information about the dataset by reading the `DESCR` attribute of `boston`. For our purposes here, we will actually expand this dataset by not only considering these 13 measurements as input features, but also looking at all products (also called *interactions*) between features. In other words, we will not only consider crime rate and highway accessibility as features, but also the product of crime rate and highway accessibility. Including derived feature like these is called *feature engineering*, which we will discuss in more detail in Chapter 4. This derived dataset can be loaded using the `load_extended_boston` function:

In[9]:

```
X, y = mglearn.datasets.load_extended_boston()
print("X.shape: {}".format(X.shape))
```

Out[9]:

```
X.shape: (506, 104)
```

The resulting 104 features are the 13 original features together with the 91 possible combinations of two features within those 13.[5]

We will use these datasets to explain and illustrate the properties of the different machine learning algorithms. But for now, let's get to the algorithms themselves. First, we will revisit the *k*-nearest neighbors (*k*-NN) algorithm that we saw in the previous chapter.

5 This is called the binomial coefficient, which is the number of combinations of k elements that can be selected from a set of n elements. Often this is written as $\binom{n}{k}$ and spoken as "n choose k"—in this case, "13 choose 2."

k-Nearest Neighbors

The *k*-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its "nearest neighbors."

k-Neighbors classification

In its simplest version, the *k*-NN algorithm only considers exactly one nearest neighbor, which is the closest training data point to the point we want to make a prediction for. The prediction is then simply the known output for this training point. Figure 2-4 illustrates this for the case of classification on the `forge` dataset:

`In[10]:`

```
mglearn.plots.plot_knn_classification(n_neighbors=1)
```

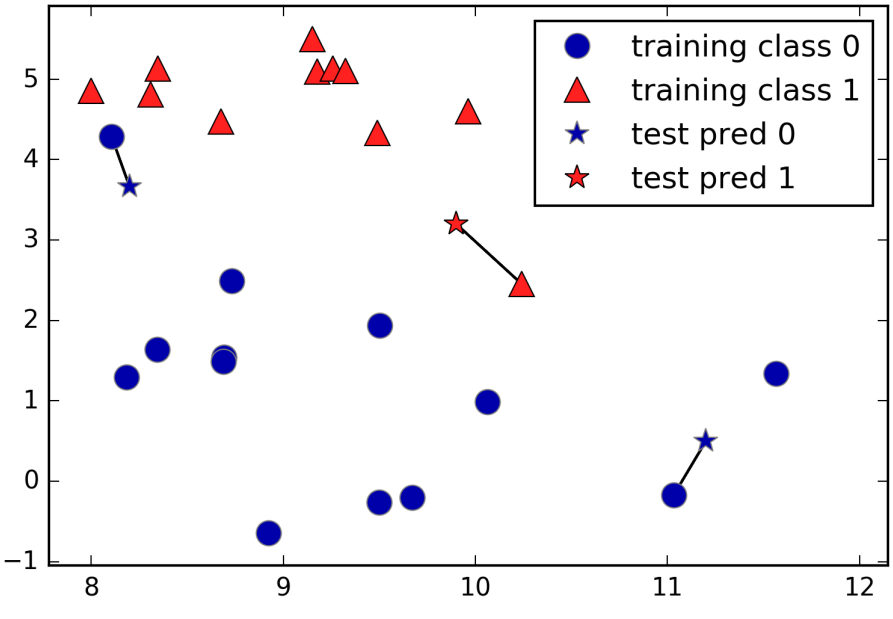

Figure 2-4. Predictions made by the one-nearest-neighbor model on the forge dataset

Here, we added three new data points, shown as stars. For each of them, we marked the closest point in the training set. The prediction of the one-nearest-neighbor algorithm is the label of that point (shown by the color of the cross).