Bringing features onto the same scale

Feature scaling is a crucial step in our **preprocessing** pipeline that can easily be forgotten. Decision trees and random forests are one of the very few machine learning algorithms where we don't need to worry about feature scaling. However, the majority of machine learning and optimization algorithms behave much better if features are on the same scale, as we saw in *Chapter 2*, *Training Machine Learning Algorithms for Classification*, when we implemented the **gradient descent** optimization algorithm.

The importance of feature scaling can be illustrated by a simple example. Let's assume that we have two features where one feature is measured on a scale from 1 to 10 and the second feature is measured on a scale from 1 to 100,000. When we think of the squared error function in **Adaline** in *Chapter 2*, *Training Machine Learning Algorithms for Classification*, it is intuitive to say that the algorithm will mostly be busy optimizing the weights according to the larger errors in the second feature. Another example is the **k-nearest neighbors** (**KNN**) algorithm with a Euclidean distance measure; the computed distances between samples will be dominated by the second feature axis.

Now, there are two common approaches to bringing different features onto the same scale: **normalization** and **standardization**. Those terms are often used quite loosely in different fields, and the meaning has to be derived from the context. Most often, normalization refers to the rescaling of the features to a range of [0, 1], which is a special case of min-max scaling. To normalize our data, we can simply apply the min-max scaling to each feature column, where the new value $x^{(i)}_{norm}$ of a sample $x^{(i)}$ can be calculated as follows:

$$x^{(i)}_{norm} = \frac{x^{(i)} - x_{\min}}{x_{\max} - x_{\min}}$$

Here, $x^{(i)}$ is a particular sample, $x_{\min}$ is the smallest value in a feature column, and $x_{\max}$ the largest value, respectively.

The min-max scaling procedure is implemented in scikit-learn and can be used as follows:

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> mms = MinMaxScaler()
>>> X_train_norm = mms.fit_transform(X_train)
>>> X_test_norm = mms.transform(X_test)
```


www.it-ebooks.info

Although normalization via min-max scaling is a commonly used technique that is useful when we need values in a bounded interval, standardization can be more practical for many machine learning algorithms. The reason is that many linear models, such as the logistic regression and SVM that we remember from *Chapter 3*, *A Tour of Machine Learning Classifiers Using Scikit-learn*, initialize the weights to 0 or small random values close to 0. Using standardization, we center the feature columns at mean 0 with standard deviation 1 so that the feature columns take the form of a normal distribution, which makes it easier to learn the weights. Furthermore, standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling, which scales the data to a limited range of values.

The procedure of standardization can be expressed by the following equation:

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

Here, μ_x is the sample mean of a particular feature column and σ_x the corresponding standard deviation, respectively.

The following table illustrates the difference between the two commonly used feature scaling techniques, standardization and normalization on a simple sample dataset consisting of numbers 0 to 5:

input	standardized	normalized
0.0	-1.336306	0.0
1.0	-0.801784	0.2
2.0	-0.267261	0.4
3.0	0.267261	0.6
4.0	0.801784	0.8
5.0	1.336306	1.0

Similar to `MinMaxScaler`, scikit-learn also implements a class for standardization:

```
>>> from sklearn.preprocessing import StandardScaler
>>> stdsc = StandardScaler()
>>> X_train_std = stdsc.fit_transform(X_train)
>>> X_test_std = stdsc.transform(X_test)
```


www.it-ebooks.info