Machine learning

Machine learning is the discipline that studies and develops algorithms to learn from, and make predictions on, data. It is strongly related to data mining, and sometimes, the names of the two fields are used interchangeably. A common distinction between the two fields is roughly given as follows: machine learning focuses on predictions based on known properties of the data, while data mining focuses on discovery based on unknown properties of the data. Both fields borrow algorithms and techniques from their counterpart. One of the goals of this book is to be practical, so we acknowledge that academically the two fields, despite the big overlap, often have distinct goals and assumptions, but we will not worry too much about it.

Some examples of machine learning applications include the following:

- Deciding whether an incoming e-mail is spam or not
- Choosing the topic of a news article from a list of known subjects such as sport, finance, or politics
- Analyzing bank transactions to identify fraud attempts
- Deciding, from the *apple* query, whether a user is interested in fruit or in computers

Some of the most popular methodologies can be categorized into supervised and unsupervised learning approaches, described in the following section. This is an over simplification that doesn't describe the whole breadth and depth of the machine learning field, but it's a good starting point to appreciate some of its technicalities.

Supervised learning approaches can be employed to solve problems such as classification, in which the data comes with the additional attributes that we want to predict, for example, the label of a class. In this case, the classifier can associate each input object with the desired output. By inferring from the features of the input objects, the classifier can then predict the desired label for the new unseen inputs. Common techniques include **Naive Bayes** (**NB**), **Support Vector Machine** (**SVM**) and models that belong to the **Neural Networks** (**NN**) family, such as perceptrons or multi-layer perceptrons.

The sample inputs used by the learning algorithm to build the mathematical model are called **training data**, while the unseen inputs that we want to obtain a prediction on are called **test data**. Inputs of a machine learning algorithm are typically in the form of a vector with each element of the vector representing a *feature* of the input. For supervised learning approaches, the desired output to assign to each of the unseen inputs is typically called **label** or **target**.

Unsupervised learning approaches are instead applied to problems in which the data come without a corresponding output value. A typical example of this kind of problems is clustering. In this case, an algorithm tries to find hidden structures in the data in order to group similar items into clusters. Another application consists of identifying items that don't appear to belong to a particular group (for example, outlier detection). An example of a common clustering algorithm is k-means.

The main Python package for machine learning is **scikit-learn**. It's an open source collection of machine learning algorithms that includes tools to access and preprocess data, evaluate the output of an algorithm, and visualize the results.

You can install scikit-learn with the common procedure via the CheeseShop:

```
$ pip install scikit-learn
```

Without digging into the details of the techniques, we will now walkthrough an application of scikit-learn to solve a clustering problem.

As we don't have social data yet, we can employ one of the datasets that is shipped together with scikit-learn.

The data that we're using is called the Fisher's Iris dataset, also referred to as Iris flower dataset. It was introduced in the 1930s by Ronald Fisher and it's today one of the classic datasets: given its small size, it's often used in the literature for toy examples. The dataset contains 50 samples from each of the three species of Iris, and for each sample four features are reported: the length and width of petals and sepals.

The dataset is commonly used as a showcase example for classification as the data comes with the correct labels for each sample, while its application for clustering is less common, mainly because there are just two well-visible clusters with a rather obvious separation. Given its small size and simple structure, it makes the case for a gentle introduction to data analysis with scikit-learn. If you want to run the example, including the data visualization part, you need to install also the **matplotlib** library with `pip install matplotlib`. More details on data visualization with Python are discussed later in this chapter.

Let's take a look at the following sample code:

```
# Chap01/demo_sklearn.py
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

if __name__ == '__main__':
  # Load the data
  iris = datasets.load_iris()
```

```
X = iris.data
petal_length = X[:, 2]
petal_width = X[:, 3]
true_labels = iris.target
# Apply KMeans clustering
estimator = KMeans(n_clusters=3)
estimator.fit(X)
predicted_labels = estimator.labels_
# Color scheme definition: red, yellow and blue
color_scheme = ['r', 'y', 'b']
# Markers definition: circle, "x" and "plus"
marker_list = ['o', 'x', '+']
# Assign colors/markers to the predicted labels
colors_predicted_labels = [color_scheme[lab] for lab in
                           predicted_labels]
markers_predicted = [marker_list[lab] for lab in
                     predicted_labels]
# Assign colors/markers to the true labels
colors_true_labels = [color_scheme[lab] for lab in true_labels]
markers_true = [marker_list[lab] for lab in true_labels]
# Plot and save the two scatter plots
for x, y, c, m in zip(petal_width,
                      petal_length,
                      colors_predicted_labels,
                      markers_predicted):
  plt.scatter(x, y, c=c, marker=m)
plt.savefig('iris_clusters.png')
for x, y, c, m in zip(petal_width,
                      petal_length,
                      colors_true_labels,
                      markers_true):
  plt.scatter(x, y, c=c, marker=m)
plt.savefig('iris_true_labels.png')

print(iris.target_names)
```

Firstly, we will load the dataset into the `iris` variable, which is an object containing both the data and information about the data. In particular, `iris.data` contains the data itself, in the form of a NumPy array or arrays, while `iris.target` contains a numeric label that represent the class a sample belongs to. In each sample vector, the four values represent, respectively, sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. Using the slicing notation for the NumPy array, we extract the third and fourth element of each sample into `petal_length` and `petal_width`, respectively. These will be used to plot the samples in a two-dimensional representation, even though the vectors have four dimensions.

The clustering process consists in two lines of code: one to create an instance of the `KMeans` algorithm and the second to `fit()` the data to the model. The simplicity of this interface is one of the characteristics of scikit-learn, which in most cases, allows you to apply a learning algorithms with just a few lines of code. For the application of the k-means algorithm, we choose the number of clusters to be three, as this is given by the data. Keep in mind that knowing the appropriate number of clusters in advance is not something that usually happens. Determining the correct (or the most interesting) number of clusters is a challenge in itself, distinct from the application of a clustering algorithm *per se*. As the purpose of this example is to briefly introduce scikit-learn and the simplicity of its interface, we take this shortcut. Normally, more effort is put into preparing the data in a format that is understood by scikit-learn.

The second half of the example serves the purpose of visualizing the data using matplotlib. Firstly, we will define a color scheme to visually differentiate the three clusters, using red, yellow, and blue defined in the `color_scheme` list. Secondly, we will exploit the fact that both the real labels and cluster associations for each sample are given as integers, starting from 0, so they can be used as indexes to match one of the colors.

Notice that while the numbers for the real labels are associated to the particular meaning of the labels, that is, a class name; the cluster numbers are simply used to clarify that a given sample belongs to a cluster, but there is no information on the meaning of the cluster. Specifically, the three classes for the real labels are setosa, versicolor, and virginica, respectively-the three species of Iris represented in the dataset.

The last lines of the example produce two scatterplots of the data, one for the real labels and another for the cluster association, using the petal length and width as two dimensions. The two plots are represented in *Figure 1.6*. The position of the items in the two plots is, of course, the same, but what we can observe is how the algorithm has split the three groups. In particular, the cluster at the bottom left is clearly separated by the other two, and the algorithm can easily identify it without doubt. Instead, the other two clusters are more difficult to distinguish as some of the elements overlap, so the algorithm makes some mistakes in this context.