An introduction to Supervised Learning with Scikit-learn

Section contents

In this section, we introduce the machine learning vocabulary that we use throughout scikit-learn and give a simple learning example.

Machine learning: the problem setting

In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or **features**.

We can separate learning problems in a few large categories:

- Supervised learning, in which the data comes with additional attributes that we want to predict. This problem can be either:
    - classification: samples belong to two or more classes and we want to learn from already labelled data how to predict the class of unlabelled data. An example of classification problem would be the handwritten digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.
    - regression: if the desired output consists of one or more continuous variables, then the task is called *regression*. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight.

- unsupervised learning, in which the training data consists of a set of input vectors x without any corresponding target values. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *visualization*

Training set and testing set

Machine learning is about learning some properties of a data set and applying them to new data. This is why a common practice in machine learning to evaluate an algorithm is to split the data at hand into two sets, one that we call the **training set** on which we learn data properties and one that we call the **testing set** on which we test these properties.

Loading an example dataset

scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the Boston house prices dataset for regression.

In the following, we start a Python interpreter from our shell and then load the `iris` and `digits` datasets. Our notational convention is that `$` denotes the shell prompt while `>>>` denotes the Python interpreter prompt:

```
$ python
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
```

A dataset is a dictionary-like object that holds all the data and some metadata about the data. This data is stored in the `.data` member, which is a `n samples, n features` array. In the case of supervised problem, one or more response variables are stored in the. target member. More details on the different datasets can be found in the dedicated section.

For instance, in the case of the digits dataset, `data` gives access to the features that can be used to classify the digits samples:

```
>>>
```

```
>>> print(digits.data)
```

and `digits.target` gives the ground truth for the digit dataset, that is the number corresponding to each digit image that we are trying to learn:

```
>>>
```

```
>>> digits.target
array([0, 1, 2, ..., 8, 9, 8])
```

Shape of the data arrays

The data is always a 2D array, shape (n_samples, n_features), although the original data may have had a different shape. In the case of the digits, each original sample is an image of shape (8, 8) and can be accessed using:

```
>>> digits.images[0]
```

The simple example on this dataset illustrates how starting from the original problem one can shape the data for consumption in scikit-learn.

5.6. Loading from external datasets¶

scikit-learn works on any numeric data stored as numpy arrays or scipy sparse matrices. Other types that are convertible to numeric arrays such as pandas DataFrame are also acceptable.

Here are some recommended ways to load standard columnar data into a format usable by scikit-learn:

- pandas.io provides tools to read data from common formats including CSV, Excel, JSON and SQL. DataFrames may also be constructed from lists of tuples or dicts. Pandas handles heterogeneous data smoothly and provides tools for manipulation and conversion into a numeric array suitable for scikit-learn.
- scipy.io specializes in binary formats often used in scientific computing context such as .mat and .arff
- numpy/routines.io for standard loading of columnar data into numpy arrays
- scikit-learn's **`datasets.load_svmlight_file`** for the svmlight or libSVM sparse format
- scikit-learn's **`datasets.load_files`** for directories of text files where the name of each directory is the name of each category and each file inside of each directory corresponds to one sample from that category

For some miscellaneous data such as images, videos, and audio, you may wish to refer to:

- skimage.io or Imageio for loading images and videos to numpy arrays
- scipy.misc.imread (requires the Pillow package) to load pixel intensities data from various image file formats
- scipy.io.wavfile.read for reading WAV files into a numpy array

Categorical (or nominal) features stored as strings (common in pandas data frames) will need converting to integers, and integer categorical variables may be best exploited when encoded as one-hot variables