7.4. Loading other datasets

7.4.1. Sample images

Scikit-learn also embeds a couple of sample JPEG images published under Creative Commons license by their authors. Those images can be useful to test algorithms and pipelines on 2D data.

`load_sample_images`()	Load sample images for image manipulation.
`load_sample_image`(image_name)	Load the numpy array of a single sample image.

Warning: The default coding of images is based on the `uint8` dtype to spare memory. Often machine learning algorithms work best if the input is converted to a floating point representation first. Also, if you plan to use `matplotlib.pyplpt.imshow`, don't forget to scale to the range 0 - 1 as done in the following example.

Examples:

- Color Quantization using K-Means

7.4.2. Datasets in svmlight / libsvm format

scikit-learn includes utility functions for loading datasets in the svmlight / libsvm format. In this format, each line takes the form `<label> <feature-id>:<feature-value> <feature-id>:<feature-value> ...`. This format is especially suitable for sparse datasets. In this module, scipy sparse CSR matrices are used for `X` and numpy arrays are used for `y`.

You may load a dataset like as follows:

```
>>> from sklearn.datasets import load_svmlight_file
>>> X_train, y_train = load_svmlight_file("/path/to/train_dataset.txt")
...
```

You may also load two (or more) datasets at once:

```
>>> X_train, y_train, X_test, y_test = load_svmlight_files(
...     ("/path/to/train_dataset.txt", "/path/to/test_dataset.txt"))
...
```

In this case, `X_train` and `X_test` are guaranteed to have the same number of features. Another way to achieve the same result is to fix the number of features:

```
>>> X_test, y_test = load_svmlight_file(
...     "/path/to/test_dataset.txt", n_features=X_train.shape[1])
...
```

Related links:

Public datasets in svmlight / libsvm format: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

Faster API-compatible implementation: https://github.com/mblondel/svmlight-loader

7.4.3. Downloading datasets from the openml.org repository

openml.org is a public repository for machine learning data and experiments, that allows everybody to upload open datasets.

The `sklearn.datasets` package is able to download datasets from the repository using the function `sklearn.datasets.fetch_openml`.

For example, to download a dataset of gene expressions in mice brains:

```
>>> from sklearn.datasets import fetch_openml
>>> mice = fetch_openml(name='miceprotein', version=4, parser="auto")
```

To fully specify a dataset, you need to provide a name and a version, though the version is optional, see Dataset Versions below. The dataset contains a total of 1080 examples belonging to 8 different classes:

Toggle Menu

```
>>> mice.data.shape
(1080, 77)
>>> mice.target.shape
(1080,)
>>> np.unique(mice.target)
array(['c-CS-m', 'c-CS-s', 'c-SC-m', 'c-SC-s', 't-CS-m', 't-CS-s', 't-SC-m', 't-SC-s'], dtype=object)
```

You can get more information on the dataset by looking at the `DESCR` and `details` attributes:

```
>>> print(mice.DESCR)
**Author**: Clara Higuera, Katheleen J. Gardiner, Krzysztof J. Cios
**Source**: [UCI](https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression) - 2015
**Please cite**: Higuera C, Gardiner KJ, Cios KJ (2015) Self-Organizing
Feature Maps Identify Proteins Critical to Learning in a Mouse Model of Down
Syndrome. PLoS ONE 10(6): e0129126...

>>> mice.details
{'id': '40966', 'name': 'MiceProtein', 'version': '4', 'format': 'ARFF',
'upload_date': '2017-11-08T16:00:15', 'licence': 'Public',
'url': 'https://www.openml.org/data/v1/download/17928620/MiceProtein.arff',
'file_id': '17928620', 'default_target_attribute': 'class',
'row_id_attribute': 'MouseID',
'ignore_attribute': ['Genotype', 'Treatment', 'Behavior'],
'tag': ['OpenML-CC18', 'study_135', 'study_98', 'study_99'],
'visibility': 'public', 'status': 'active',
'md5_checksum': '3c479a6885bfa0438971388283a1ce32'}
```

The `DESCR` contains a free-text description of the data, while `details` contains a dictionary of meta-data stored by openml, like the dataset id. For more details, see the OpenML documentation The `data_id` of the mice protein dataset is 40966, and you can use this (or the name) to get more information on the dataset on the openml website:

```
>>> mice.url
'https://www.openml.org/d/40966'
```

The `data_id` also uniquely identifies a dataset from OpenML:

```
>>> mice = fetch_openml(data_id=40966, parser="auto")
>>> mice.details
{'id': '4550', 'name': 'MiceProtein', 'version': '1', 'format': 'ARFF',
'creator': ...,
'upload_date': '2016-02-17T14:32:49', 'licence': 'Public', 'url':
'https://www.openml.org/data/v1/download/1804243/MiceProtein.ARFF', 'file_id':
'1804243', 'default_target_attribute': 'class', 'citation': 'Higuera C,
Gardiner KJ, Cios KJ (2015) Self-Organizing Feature Maps Identify Proteins
Critical to Learning in a Mouse Model of Down Syndrome. PLoS ONE 10(6):
e0129126. [Web Link] journal.pone.0129126', 'tag': ['OpenML100', 'study_14',
'study_34'], 'visibility': 'public', 'status': 'active', 'md5_checksum':
'3c479a6885bfa0438971388283a1ce32'}
```

7.4.3.1. Dataset Versions

A dataset is uniquely specified by its `data_id`, but not necessarily by its name. Several different "versions" of a dataset with the same name can exist which can contain entirely different datasets. If a particular version of a dataset has been found to contain significant issues, it might be deactivated. Using a name to specify a dataset will yield the earliest version of a dataset that is still active. That means that `fetch_openml(name="miceprotein", parser="auto")` can yield different results at different times if earlier versions become inactive. You can see that the dataset with `data_id` 40966 that we fetched above is the first version of the "miceprotein" dataset:

```
>>> mice.details['version']
'1'
```

In fact, this dataset only has one version. The iris dataset on the other hand has multiple versions:

Toggle Menu

```
>>> iris = fetch_openml(name="iris", parser="auto")
>>> iris.details['version']
'1'
>>> iris.details['id']
'61'

>>> iris_61 = fetch_openml(data_id=61, parser="auto")
>>> iris_61.details['version']
'1'
>>> iris_61.details['id']
'61'

>>> iris_969 = fetch_openml(data_id=969, parser="auto")
>>> iris_969.details['version']
'3'
>>> iris_969.details['id']
'969'
```

Specifying the dataset by the name "iris" yields the lowest version, version 1, with the `data_id` 61. To make sure you always get this exact dataset, it is safest to specify it by the dataset `data_id`. The other dataset, with `data_id` 969, is version 3 (version 2 has become inactive), and contains a binarized version of the data:

```
>>> np.unique(iris_969.target)
array(['N', 'P'], dtype=object)
```

You can also specify both the name and the version, which also uniquely identifies the dataset:

```
>>> iris_version_3 = fetch_openml(name="iris", version=3, parser="auto")
>>> iris_version_3.details['version']
'3'
>>> iris_version_3.details['id']
'969'
```

References:

- Vanschoren, van Rijn, Bischl and Torgo. "OpenML: networked science in machine learning" ACM SIGKDD Explorations Newsletter, 15(2), 49-60, 2014.

7.4.3.2. ARFF parser

From version 1.2, scikit-learn provides a new keyword argument `parser` that provides several options to parse the ARFF files provided by OpenML. The legacy parser (i.e. `parser="liac-arff"`) is based on the project LIAC-ARFF. This parser is however slow and consume more memory than required. A new parser based on pandas (i.e. `parser="pandas"`) is both faster and more memory efficient. However, this parser does not support sparse data. Therefore, we recommend using `parser="auto"` which will use the best parser available for the requested dataset.

The `"pandas"` and `"liac-arff"` parsers can lead to different data types in the output. The notable differences are the following:

- The `"liac-arff"` parser always encodes categorical features as `str` objects. To the contrary, the `"pandas"` parser instead infers the type while reading and numerical categories will be casted into integers whenever possible.
- The `"liac-arff"` parser uses float64 to encode numerical features tagged as 'REAL' and 'NUMERICAL' in the metadata. The `"pandas"` parser instead infers if these numerical features corresponds to integers and uses panda's Integer extension dtype.
- In particular, classification datasets with integer categories are typically loaded as such `(0, 1, ...)` with the `"pandas"` parser while `"liac-arff"` will force the use of string encoded class labels such as `"0"`, `"1"` and so on.
- The `"pandas"` parser will not strip single quotes - i.e. `'` - from string columns. For instance, a string `'my string'` will be kept as is while the `"liac-arff"` parser will strip the single quotes. For categorical columns, the single quotes are stripped from the values.

In addition, when `as_frame=False` is used, the `"liac-arff"` parser returns ordinally encoded data where the categories are provided in the attribute `categories` of the `Bunch` instance. Instead, `"pandas"` returns a NumPy array were the categories. Then it's up to the user to design a feature engineering pipeline with an instance of `OneHotEncoder` or `OrdinalEncoder` typically wrapped in a `ColumnTransformer` to preprocess the categorical columns explicitly. See for instance: Column Transformer with Mixed Types.

7.4.4. 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.
- Toggle Menu ecializes 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 into numpy arrays
- scipy.io.wavfile.read for reading WAV files into a numpy array

Categorical (or nominal) features stored as strings (common in pandas DataFrames) will need converting to numerical features using **`OneHotEncoder`** or **`OrdinalEncoder`** or similar. See Preprocessing data.

Note: if you manage your own numerical data it is recommended to use an optimized file format such as HDF5 to reduce data load times. Various libraries such as H5Py, PyTables and pandas provides a Python interface for reading and writing data in that format.

Toggle Menu