Extending the Datasets

While the core of PyKEEN uses the `pykeen.triples.TriplesFactory` for handling sets of triples, the definition of a training, validation, and testing trichotomy for a given dataset is very useful for reproducible benchmarking. The internal `pykeen.datasets.base.Dataset` class can be considered as a three-tuple of datasets (though it's implemented as a class such that it can be extended). There are several datasets included in PyKEEN already, each coming from sources that look different. This tutorial gives some insight into implementing your own Dataset class.

Pre-split Datasets

Unpacked Remote Dataset

Use this tutorial if you have three separate URLs for the respective training, testing, and validation sets that are each 3 column TSV files. A good example can be found at https://github.com/ZhenfengLei/KGDatasets/tree/master/DBpedia50. There's a base class called `pykeen.datasets.base.UnpackedRemoteDataset` that can be used to wrap it like the following:

```
from pykeen.datasets.base import UnpackedRemoteDataset

TEST_URL =
'https://raw.githubusercontent.com/ZhenfengLei/KGDatasets/master/DBpedia50/test.txt'
TRAIN_URL =
'https://raw.githubusercontent.com/ZhenfengLei/KGDatasets/master/DBpedia50/train.txt'
VALID_URL =
'https://raw.githubusercontent.com/ZhenfengLei/KGDatasets/master/DBpedia50/valid.txt'

class DBpedia50(UnpackedRemoteDataset):
    def __init__(self, **kwargs):
        super().__init__(
            training_url=TRAIN_URL,
            testing_url=TEST_URL,
            validation_url=VALID_URL,
            **kwargs,
        )
```

Unsplit Datasets

Use this tutorial if you have a single URL for a TSV dataset that needs to be automatically split into training, testing, and validation. A good example can be found at https://github.com/hetio/hetionet/raw/master/hetnet/tsv. There's a base class called `pykeen.datasets.base.SingleTabbedDataset` that can be used to wrap it like the following:

```python
from pykeen.datasets.base import SingleTabbedDataset

URL = 'https://github.com/hetio/hetionet/raw/master/hetnet/tsv/hetionet-v1.0-edges.sif.gz'

class Hetionet(SingleTabbedDataset):
    def __init__(self, **kwargs):
        super().__init__(url=URL, **kwargs)
```

The value for *URL* can be anything that can be read by `pandas.read_csv()`. Additional options can be passed through to the reading function, such as `sep=','`, with the keyword argument `read_csv_kwargs=dict(sep=',')`. Note that the default separator for Pandas is a comma, but PyKEEN overrides it to be a tab, so you'll have to explicitly set it if you want a comma. Since there's a random aspect to this process, you can also set the seed used for splitting with the `random_state` keyword argument.

Updating the `setup.cfg`

Whether you're making a pull request against PyKEEN or implementing a dataset in your own package, you can use Python entrypoints to register your dataset with PyKEEN. Below is an example of the entrypoints that register `pykeen.datasets.Hetionet`, `pykeen.datasets.DRKG`, and others that appear in the PyKEEN setup.cfg. Under the `pykeen.datasets` header, you can pick whatever name you want for the dataset as the key (appearing on the left side of the equals, e.g. `hetionet`) and the path to the class (appearing on the right side of the equals, e.g., `pykeen.datasets.hetionet:Hetionet`). The right side is constructed by the path to the module, the colon `:`, then the name of the class.

```ini
# setup.cfg
...
[options.entry_points]
console_scripts =
    pykeen = pykeen.cli:main
pykeen.datasets =
    hetionet         = pykeen.datasets.hetionet:Hetionet
    conceptnet       = pykeen.datasets.conceptnet:ConceptNet
    drkg             = pykeen.datasets.drkg:DRKG
    ...
```

If you're working on a development version of PyKEEN, you also need to run `pykeen readme` in the shell to update the README.md file.