Bring Your Own Data

As an alternative to using a pre-packaged dataset, the training and testing can be set explicitly by file path or with instances of `pykeen.triples.TriplesFactory`. Throughout this tutorial, the paths to the training, testing, and validation sets for built-in `pykeen.datasets.Nations` will be used as examples.

Pre-stratified Dataset

You've got a training and testing file as 3-column TSV files, all ready to go. You're sure that there aren't any entities or relations appearing in the testing set that don't appear in the training set. Load them in the pipeline like this:

```
>>> from pykeen.triples import TriplesFactory
>>> from pykeen.pipeline import pipeline
>>> from pykeen.datasets.nations import NATIONS_TRAIN_PATH, NATIONS_TEST_PATH
>>> result = pipeline(
...     training=NATIONS_TRAIN_PATH,
...     testing=NATIONS_TEST_PATH,
...     model='TransE',
...     epochs=5,  # short epochs for testing - you should go higher
... )
>>> result.save_to_directory('doctests/test_pre_stratified_transe')
```

PyKEEN will take care of making sure that the entities are mapped from their labels to appropriate integer (technically, 0-dimensional `torch.LongTensor`) indexes and that the different sets of triples share the same mapping.

This is equally applicable for the `pykeen.hpo.hpo_pipeline()`, which has a similar interface to the `pykeen.pipeline.pipeline()` as in:

```
>>> from pykeen.hpo import hpo_pipeline
>>> from pykeen.datasets.nations import NATIONS_TRAIN_PATH, NATIONS_TEST_PATH,
NATIONS_VALIDATE_PATH
>>> result = hpo_pipeline(
...     n_trials=3,  # you probably want more than this
...     training=NATIONS_TRAIN_PATH,
...     testing=NATIONS_TEST_PATH,
...     validation=NATIONS_VALIDATE_PATH,
...     model='TransE',
...     epochs=5,  # short epochs for testing - you should go higher
... )
>>> result.save_to_directory('doctests/test_hpo_pre_stratified_transe')
```

The remainder of the examples will be for `pykeen.pipeline.pipeline()`, but all work exactly the same for `pykeen.hpo.hpo_pipeline()`.

If you want to add dataset-wide arguments, you can use the `dataset_kwargs` argument to the `pykeen.pipeline.pipeline` to enable options like `create_inverse_triples=True`.

```
>>> from pykeen.pipeline import pipeline
>>> from pykeen.datasets.nations import NATIONS_TRAIN_PATH, NATIONS_TEST_PATH
>>> result = pipeline(
...     training=NATIONS_TRAIN_PATH,
...     testing=NATIONS_TEST_PATH,
...     dataset_kwargs={'create_inverse_triples': True},
...     model='TransE',
...     epochs=5,  # short epochs for testing - you should go higher
... )
>>> result.save_to_directory('doctests/test_pre_stratified_transe')
```

If you want finer control over how the triples are created, for example, if they are not all coming from TSV files, you can use the `pykeen.triples.TriplesFactory` interface.

```
>>> from pykeen.triples import TriplesFactory
>>> from pykeen.pipeline import pipeline
>>> from pykeen.datasets.nations import NATIONS_TRAIN_PATH, NATIONS_TEST_PATH
>>> training = TriplesFactory.from_path(NATIONS_TRAIN_PATH)
>>> testing = TriplesFactory.from_path(
...     NATIONS_TEST_PATH,
...     entity_to_id=training.entity_to_id,
...     relation_to_id=training.relation_to_id,
... )
>>> result = pipeline(
...     training=training,
...     testing=testing,
...     model='TransE',
...     epochs=5,  # short epochs for testing - you should go higher
... )
>>> result.save_to_directory('doctests/test_pre_stratified_transe')
```

Warning

The instantiation of the testing factory, we used the `entity_to_id` and `relation_to_id` keyword arguments. This is because PyKEEN automatically assigns numeric identifiers to all entities and relations for each triples factory. However, we want the identifiers to be exactly the same for the testing set as the training set, so we just reuse it. If we didn't have the same identifiers, then the testing set would get mixed up with the wrong identifiers in the training set during evaluation, and we'd get nonsense results.

The `dataset_kwargs` argument is ignored when passing your own `pykeen.triples.TriplesFactory`, so be sure to include the `create_inverse_triples=True` in the instantiation of those classes if that's your desired behavior as in:

```
>>> from pykeen.triples import TriplesFactory
>>> from pykeen.pipeline import pipeline
>>> from pykeen.datasets.nations import NATIONS_TRAIN_PATH, NATIONS_TEST_PATH
>>> training = TriplesFactory.from_path(
...     NATIONS_TRAIN_PATH,
...     create_inverse_triples=True,
... )
>>> testing = TriplesFactory.from_path(
...     NATIONS_TEST_PATH,
...     entity_to_id=training.entity_to_id,
...     relation_to_id=training.relation_to_id,
...     create_inverse_triples=True,
... )
>>> result = pipeline(
...     training=training,
...     testing=testing,
...     model='TransE',
...     epochs=5,  # short epochs for testing - you should go higher
... )
>>> result.save_to_directory('doctests/test_pre_stratified_transe')
```

Triples factories can also be instantiated using the `triples` keyword argument instead of the `path` argument if you already have triples loaded in a `numpy.ndarray`.

Unstratified Dataset

It's more realistic your real-world dataset is not already stratified into training and testing sets. PyKEEN has you covered with `pykeen.triples.TriplesFactory.split()`, which will allow you to create a stratified dataset.

```
>>> from pykeen.triples import TriplesFactory
>>> from pykeen.pipeline import pipeline
>>> from pykeen.datasets.nations import NATIONS_TRAIN_PATH
>>> tf = TriplesFactory.from_path(NATIONS_TRAIN_PATH)
>>> training, testing = tf.split()
>>> result = pipeline(
...     training=training,
...     testing=testing,
...     model='TransE',
...     epochs=5,  # short epochs for testing - you should go higher
... )
>>> result.save_to_directory('doctests/test_unstratified_transe')
```

By default, this is an 80/20 split. If you want to use early stopping, you'll also need a validation set, so you should specify the splits:

```
>>> from pykeen.triples import TriplesFactory
>>> from pykeen.pipeline import pipeline
>>> from pykeen.datasets.nations import NATIONS_TRAIN_PATH
>>> tf = TriplesFactory.from_path(NATIONS_TRAIN_PATH)
>>> training, testing, validation = tf.split([.8, .1, .1])
>>> result = pipeline(
...     training=training,
...     testing=testing,
...     validation=validation,
...     model='TransE',
...     stopper='early',
...     epochs=5,  # short epochs for testing - you should go
...                # higher, especially with early stopper enabled
... )
>>> result.save_to_directory('doctests/test_unstratified_stopped_transe')
```

Bring Your Own Data with Checkpoints

For a tutorial on how to use your own data together with checkpoints, see Checkpoints When Bringing Your Own Data and Loading Models Manually.