Using Resolvers

As PyKEEN is a heavily modular and extensible library, we make use of the `class_resolver` library to allow simple configuration of components. In this part of the tutorial, we explain how to use these configuration options, and how to figure out what values you can pass here.

We use the initialization method of the base model class `pykeen.models.base.Model()` and its handling of loss functions as an example. Its signature is given as

```
def __init__(
    self,
    *,
    ...,
    loss: HintOrType[Loss] = None,
    loss_kwargs: OptionalKwargs = None,
    ...,
) -> None:
```

Notice the two related parameters *loss: HintOrType[Loss] = None* and *loss_kwargs: OptionalKwargs = None*. The *loss_kwargs* thus takes a values of type *OptionalKwargs* as input, which is an abbreviation of *Union[None, Mapping[str, Any]]*. Hence, we can either pass a mapping of string keys to some values, or *None*. In the latter case of passing *None*, this is interpreted as an empty dictionary.

The *loss* parameter takes inputs of type *HintOrType[Loss]*. *HintOrType[Loss]* is a abbreviation of *Union[None, str, Type[Loss], Loss]*. Thus, we can either pass

1. an *instance* of the `pykeen.losses.Loss`, e.g., `pykeen.losses.MarginRankingLoss(margin=2.0)`. If an instance of `pykeen.losses.Loss` is passed, it is used without modification. In this case, *loss_kwargs* will be ignored.
2. a *subclass* of `pykeen.losses.Loss`, e.g., `pykeen.losses.MarginRankingLoss` In this case, the class is instantiated with the given *loss_kwargs* as (keyword-based) parameters. For instance,

   ```
   loss = MarginRankingLoss
   loss_kwargs = None  # equivalent to {}
   ```

   translates to *MarginRankingLoss()*. We can also choose different instantiation parameters by

```
loss = MarginRankingLoss
loss_kwargs = dict(margin=2)  # or {"margin": 2}
```

which translates to *MarginRankingLoss(margin=2)*

3. a *string*. This string is used to search for a matching class using the class-resolver's *lookup* function. The lookup function performs some string normalization and compares the resulting key to the normalized names of classes it is associated with. The found class is then used to instantiate the object as if we passed this class instead of the string. For instance, we can obtain instances *MarginRankingLoss* by passing *"MarginRankingLoss"*, *"marginrankingloss"*, *"marginranking"*, *"margin-ranking"*, or *"MRL"*.
4. *None*. In this case, we use the *default* class set in the class-resolver, which happens to be `MarginRankingLoss` for the *loss_resolver*. If no *default* is set, an exception will be raised.

Determining Allowed Inputs

To keep PyKEEN easily extensible and maintainable, we often use *None* for the choice, e.g., *loss*, and the keyword-based parameters. This can sometimes make it hard to read what default values are used, what valid choices are available, and what parameters are allowed with these different choices. In the following, we describe a few ways how to find this information.

First, you should take a look at the type annotation. *HintOrType[X] = None* tells you that you can pass any subclass of *X*. Moreover, you can always pass the string of the class name instead, which often is easier to setup for you result tracking, command line arguments, or hyperparameter search. All resolvers for classes used in PyKEEN are instantiated using the *ClassResolver.from_subclasses* factory function, which automatically registers all subclasses for a given base class as valid choices. Moreover, it will allow you to pass class names without the base class' name as suffix, e.g., *loss_resolver* accepts *MarginRanking* instead of *MarginRankingLoss*, since the base class' name *Loss* is removed as suffix during the normalization. To utilize this feature, we try to follow an appropriate naming scheme for all configurable parts, e.g., `pykeen.nn.representation.Representation`, or `pykeen.nn.modules.Interaction`.

The allowed parameters for *..._kwargs: OptionalKwargs* are a bit harder to determine, since they vary with your choice of the component! For instance, `MarginRankingLoss` has a *margin* parameter, while `pykeen.losses.BCEWithLogitsLoss` does not provide such. Hence, you should investigate the documentation of the individual classes to inform yourself about available parameters and allowed values.