Extraction

The extraction chain uses the OpenAI `functions` parameter to specify a schema to extract entities from a document. This helps us make sure that the model outputs exactly the schema of entities and properties that we want, with their appropriate types.

The extraction chain is to be used when we want to extract several entities with their properties from the same passage (i.e. what people were mentioned in this passage?)

```
from langchain.chat_models import ChatOpenAI
from langchain.chains import create_extraction_chain, create_extraction_chain_pydantic
from langchain.prompts import ChatPromptTemplate
```

```
    /Users/harrisonchase/.pyenv/versions/3.9.1/envs/langchain/lib/python3.9/site-
packages/deeplake/util/check_latest_version.py:32: UserWarning: A newer version of deeplake (3.6.4) is
available. It's recommended that you update to the latest version using `pip install -U deeplake`.
      warnings.warn(
```

```
llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo-0613")
```

Extracting entities

To extract entities, we need to create a schema like the following, were we specify all the properties we want to find and the type we expect them to have. We can also specify which of these properties are required and which are optional.

```
schema = {
    "properties": {
        "person_name": {"type": "string"},
        "person_height": {"type": "integer"},
        "person_hair_color": {"type": "string"},
        "dog_name": {"type": "string"},
        "dog_breed": {"type": "string"},
    },
    "required": ["person_name", "person_height"],
}
```

```
inp = """
Alex is 5 feet tall. Claudia is 1 feet taller Alex and jumps higher than him. Claudia is a brunette and Alex is blonde.
Alex's dog Frosty is a labrador and likes to play hide and seek.
        """
```

```
chain = create_extraction_chain(schema, llm)
```

As we can see, we extracted the required entities and their properties in the required format:

```
chain.run(inp)
```

```
[{'person_name': 'Alex',
  'person_height': 5,
  'person_hair_color': 'blonde',
  'dog_name': 'Frosty',
  'dog_breed': 'labrador'},
 {'person_name': 'Claudia',
  'person_height': 6,
  'person_hair_color': 'brunette'}]
```

Pydantic example

We can also use a Pydantic schema to choose the required properties and types and we will set as 'Optional' those that are not strictly required.

By using the `create_extraction_chain_pydantic` function, we can send a Pydantic schema as input and the output will be an instantiated object that respects our desired schema.

In this way, we can specify our schema in the same manner that we would a new class or function in Python - with purely Pythonic types.

```
from typing import Optional, List
from pydantic import BaseModel, Field
```

```
class Properties(BaseModel):
    person_name: str
    person_height: int
    person_hair_color: str
```

```
    dog_breed: Optional[str]
    dog_name: Optional[str]
```

```
chain = create_extraction_chain_pydantic(pydantic_schema=Properties, llm=llm)
```

```
inp = """
Alex is 5 feet tall. Claudia is 1 feet taller Alex and jumps higher than him. Claudia is a brunette and Alex
is blonde.
Alex's dog Frosty is a labrador and likes to play hide and seek.
        """
```

As we can see, we extracted the required entities and their properties in the required format:

```
chain.run(inp)
```

```
    [Properties(person_name='Alex', person_height=5, person_hair_color='blonde', dog_breed='labrador',
dog_name='Frosty'),
     Properties(person_name='Claudia', person_height=6, person_hair_color='brunette', dog_breed=None,
dog_name=None)]
```