Develop a New Operator

In the previous notebooks, we are already familiar with the basic classes of each type of OPs. In this notebook, we will show you how to develop your own OP.

Basically, we only need to implement the `__init__`, `compute_xxx`, `process` methods (if required) after inheriting from these basic classes. Here we take Filter as an example. Compared with Mapper and Selector, Filter and Deduplicator need to compute some stats or hash values for processing, so there are some extra steps to implement them.

Coding for Your Filter

As we said before, we only need to inheriting from the basic Filter class, implement `__init__`, `compute_stats`, `process` methods. However, due to Filter would introduce stats, we need to define a new stats field key for your new Filter first.

For example, we want to implement our own Filter to filter out the samples with too long or too short texts (Acutally, Data-Juicer already contain an OP with the same function: `TextLengthFilter`). We need a new field key to store this new stats. Assuming we call this new field key "my_text_len", we need to:

- Add this new field key to `StatsKeys` class in `data_juicer/utils/constant.py` to store the statistical variable of the new OP.
  - It's worth noticing that when developing your own Deduplicator and you need a new kind of hash value, you also need to add a new hash field key to the `HashKeys` class in `data_juicer/utils/constant.py`. For Mapper and Selector, we don't need this step because they don't depend on stats usually.

In [1]:
```
class StatsKeys(object):
    # ... other keys
    my_text_len = 'my_text_len'
```

Then we can create a new OP file, such as `my_text_length_filter.py` in the corresponding `data_juicer/ops/filter/` directory as follows.

In [2]:
```
import sys
from jsonargparse.typing import PositiveInt

from data_juicer.utils.constant import Fields
# NOTE: use the new StatsKeys definition above in this example notebook.
# from data_juicer.utils.constant import StatsKeys
from data_juicer.ops.base_op import OPERATORS, Filter

# register into the OP list
```

```python
@OPERATORS.register_module('my_text_length_filter')
# inheriting from the basic Filter class
class MyTextLengthFilter(Filter):
    """Filter to keep samples with total text length within a specific
    range."""

    # implement the __init__ method
    def __init__(self,
                 min_len: PositiveInt = 10,
                 max_len: PositiveInt = sys.maxsize,
                 *args,
                 **kwargs):
        """
        Initialization method.

        :param min_len: The min text length in the filtering. samples
            will be filtered if their text length is below this
            parameter.
        :param max_len: The max text length in the filtering. samples
            will be filtered if their text length exceeds this
            parameter.
        :param args: extra args
        :param kwargs: extra args
        """
        super().__init__(*args, **kwargs)
        self.min_len = min_len
        self.max_len = max_len

    # implement the compute_stats method
    def compute_stats(self, sample):
        # check if it's computed already
        if StatsKeys.my_text_len in sample[Fields.stats]:
            return sample

        # compute the stats and store it into the stats dict with the new
        sample[Fields.stats][StatsKeys.my_text_len] = len(sample[self.tex
        return sample

    # implement the process method
    def process(self, sample):
        # only keep those samples whose text length is in the expected ra
        if self.min_len <= sample[Fields.stats][StatsKeys.my_text_len] <=
            return True
        else:
            return False
```

```
/usr/local/python310/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWar
ning: IProgress not found. Please update jupyter and ipywidgets. See http
s://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

- After implemention, add it to the OP dictionary and `__all__` list in the `__init__.py` file in `data_juicer/ops/filter/` directory.

```python
from . import  my_text_length_filter
...
from .my_text_length_filter import MyTextLengthFilter
...
__all__ = [
```

```
    ...
    'MyTextLengthFilter',
    ...
]
```

Now we can try our own OP after importing it.

In [3]:
```
# import our own new OP from data-juicer. Here in this example notebook,
# from data_juicer.ops.filter import MyTextLengthFilter

# initialize this OP. We only keep samples with text length >= 20
my_op = MyTextLengthFilter(
    min_len=20,
)

# create a example dataset
from data_juicer.core import NestedDataset
samples = [
    {'text': 'Data Juicer'},  # len = 11
    {'text': 'Welcome to Data Juicer Playground'}  # len = 33
]
ds = NestedDataset.from_list(samples)
# Add a new column to the dataset to store the stats of the Filter operat
from data_juicer.utils.constant import Fields
ds = ds.add_column(name=Fields.stats, column=[{}] * ds.num_rows)
print(ds)
```

```
Dataset({
    features: ['text', '__dj__stats__'],
    num_rows: 2
})
```

Then use this new OP process this example dataset.

In [4]:
```
out_ds = my_op.run(ds)

print(out_ds)
print(f'Number of samples of output dataset : {len(out_ds)}')
for sample in out_ds:
    print(sample)
```

```
num_proc must be <= 2. Reducing num_proc to 2 for dataset of size 2.
my_text_length_filter_compute_stats (num_proc=2): 100%|██████████| 2/2 [0
0:00<00:00, 21.40 examples/s]
num_proc must be <= 2. Reducing num_proc to 2 for dataset of size 2.
my_text_length_filter_process (num_proc=2): 100%|██████████| 2/2 [00:00<0
0:00, 24.64 examples/s]
Dataset({
    features: ['text', '__dj__stats__'],
    num_rows: 1
})
Number of samples of output dataset : 1
{'text': 'Welcome to Data Juicer Playground', '__dj__stats__': {'my_text_l
en': 33}}
```

How to contribute new OPs in Data-Juicer style

After you develop a new OP, you might want to contribute to Data-Juicer repository to share this new OP with other developers in the open-source community.

As we know, good coding style can make code more readable and easier to maintain and follow. So you need some extra works before completing the contribution, including writing unit tests, add corresponding documents, and check coding style for your new OP before it's merged into Data-Juicer.

For more details, please refer to the DeveloperGuide doc.

Conclusion

In this notebook, we learn how to develop a new OP and make it available in Data-Juicer. Also, we also suggest and encourage developers to modify their new OP to a good coding style and contribute it to Data-Juicer for open-source community.