4.3 处理数据

(English Version)

用户可以在 `process()` 函数中实现数据处理。该函数假定原始数据已经位于 `self.raw_dir` 目录中。

图上的机器学习任务通常有三种类型：整图分类、节点分类和链接预测。本节将展示如何处理与这些任务相关的数据集。

本节重点介绍了处理图、特征和划分掩码的标准方法。用户指南将以内置数据集为例，并跳过从文件构建图的实现。 用户可以参考 1.4 从外部源创建图 以查看如何从外部数据源构建图的完整指南。

处理整图分类数据集

整图分类数据集与用小批次训练的典型机器学习任务中的大多数数据集类似。 因此，需要将原始数据处理为 `dgl.DGLGraph` 对象的列表和标签张量的列表。 此外，如果原始数据已被拆分为多个文件，则可以添加参数 `split` 以导入数据的特定部分。

下面以 `QM7bDataset` 为例：

```
from dgl.data import DGLDataset

class QM7bDataset(DGLDataset):
    _url = 'http://deepchem.io.s3-website-us-west-1.amazonaws.com/' \
           'datasets/qm7b.mat'
    _sha1_str = '4102c744bb9d6fd7b40ac67a300e49cd87e28392'

    def __init__(self, raw_dir=None, force_reload=False, verbose=False):
        super(QM7bDataset, self).__init__(name='qm7b',
                                          url=self._url,
                                          raw_dir=raw_dir,
                                          force_reload=force_reload,
                                          verbose=verbose)

    def process(self):
        mat_path = self.raw_path + '.mat'
        # 将数据处理为图列表和标签列表
        self.graphs, self.label = self._load_graph(mat_path)

    def __getitem__(self, idx):
        """ 通过idx获取对应的图和标签

        Parameters
        ----------
        idx : int
            Item index

        Returns
        -------
        (dgl.DGLGraph, Tensor)
        """
        return self.graphs[idx], self.label[idx]

    def __len__(self):
        """数据集中图的数量"""
        return len(self.graphs)
```

函数 `process()` 将原始数据处理为图列表和标签列表。用户必须实现 `__getitem__(idx)` 和 `__len__()` 以进行迭代。 DGL建议让 `__getitem__(idx)` 返回如上面代码所示的元组 `(图，标签)`。 用户可以参考 QM7bDataset源代码 以获得 `self._load_graph()` 和 `__getitem__` 的详细信息。

用户还可以向类添加属性以指示一些有用的数据集信息。在 `QM7bDataset` 中， 用户可以添加属性 `num_tasks` 来指示此多任务数据集中的预测任务总数：

```
@property
def num_tasks(self):
    """每个图的标签数，即预测任务数。"""
    return 14
```

在编写完这些代码之后，用户可以按如下所示的方式来使用 `QM7bDataset`：

```
import dgl
import torch

from dgl.dataloading import GraphDataLoader

# 数据导入
dataset = QM7bDataset()
num_tasks = dataset.num_tasks

# 创建 dataloaders
dataloader = GraphDataLoader(dataset, batch_size=1, shuffle=True)

# 训练
for epoch in range(100):
    for g, labels in dataloader:
        # 用户自己的训练代码
        pass
```

训练整图分类模型的完整指南可以在 5.4 整图分类 中找到。

有关整图分类数据集的更多示例，用户可以参考 5.4 整图分类：

- gindataset
- minigcdataset
- qm7bdata
- tudata

处理节点分类数据集

与整图分类不同，节点分类通常在单个图上进行。因此数据集的划分是在图的节点集上进行。 DGL建议使用节点掩码来指定数据集的划分。 本节以内置数据集 CitationGraphDataset 为例：

此外，DGL推荐重新排列图的节点/边，使得相邻节点/边的ID位于邻近区间内。这个过程 可以提高节点/边的邻居的局部性，为后续在图上进行的计算与分析的性能改善提供可能。 DGL提供了名为 `dgl.reorder_graph()` 的API用于此优化。更多细节，请参考 下面例子中的 `process()` 的部分。

```
from dgl.data import DGLBuiltinDataset
from dgl.data.utils import _get_dgl_url

class CitationGraphDataset(DGLBuiltinDataset):
    _urls = {
        'cora_v2' : 'dataset/cora_v2.zip',
        'citeseer' : 'dataset/citeseer.zip',
        'pubmed' : 'dataset/pubmed.zip',
    }

    def __init__(self, name, raw_dir=None, force_reload=False, verbose=True):
        assert name.lower() in ['cora', 'citeseer', 'pubmed']
        if name.lower() == 'cora':
            name = 'cora_v2'
        url = _get_dgl_url(self._urls[name])
        super(CitationGraphDataset, self).__init__(name,
                                                   url=url,
                                                   raw_dir=raw_dir,
                                                   force_reload=force_reload,
                                                   verbose=verbose)

    def process(self):
        # 跳过一些处理的代码
        # === 跳过数据处理 ===

        # 构建图
        g = dgl.graph(graph)

        # 划分掩码
        g.ndata['train_mask'] = train_mask
        g.ndata['val_mask'] = val_mask
        g.ndata['test_mask'] = test_mask

        # 节点的标签
        g.ndata['label'] = torch.tensor(labels)

        # 节点的特征
        g.ndata['feat'] = torch.tensor(_preprocess_features(features),
                                       dtype=F.data_type_dict['float32'])
        self._num_tasks = onehot_labels.shape[1]
        self._labels = labels
        # 重排图以获得更优的局部性
        self._g = dgl.reorder_graph(g)

    def __getitem__(self, idx):
        assert idx == 0, "这个数据集里只有一个图"
        return self._g

    def __len__(self):
        return 1
```

为简便起见，这里省略了 `process()` 中的一些代码，以突出展示用于处理节点分类数据集的关键部分：划分掩码。 节点特征和节点的标签被存储在 `g.ndata` 中。详细的实现请参考 CitationGraphDataset源代码 。

请注意，这里 `__getitem__(idx)` 和 `__len__()` 的实现也发生了变化， 这是因为节点分类任务通常只用一个图。掩码在PyTorch和TensorFlow中是bool张量，在MXNet中是float张量。

下面中使用 `dgl.data.CitationGraphDataset` 的子类 `dgl.data.CiteseerGraphDataset` 来演示如何使用用于节点分类的数据集：

```
# 导入数据
dataset = CiteseerGraphDataset(raw_dir='')
graph = dataset[0]

# 获取划分的掩码
train_mask = graph.ndata['train_mask']
val_mask = graph.ndata['val_mask']
test_mask = graph.ndata['test_mask']

# 获取节点特征
feats = graph.ndata['feat']

# 获取标签
labels = graph.ndata['label']
```

5.1 节点分类/回归 提供了训练节点分类模型的完整指南。

有关节点分类数据集的更多示例，用户可以参考以下内置数据集：

- citationdata
- corafulldata
- amazoncobuydata
- coauthordata
- karateclubdata
- ppidata
- redditdata
- sbmdata
- sstdata
- rdfdata

处理链接预测数据集

链接预测数据集的处理与节点分类相似，数据集中通常只有一个图。

本节以内置的数据集 KnowledgeGraphDataset 为例，同时省略了详细的数据处理代码以突出展示处理链接预测数据集的关键部分：

```
# 创建链接预测数据集示例
class KnowledgeGraphDataset(DGLBuiltinDataset):
    def __init__(self, name, reverse=True, raw_dir=None, force_reload=False, verbose=True):
        self._name = name
        self.reverse = reverse
        url = _get_dgl_url('dataset/') + '{}.tgz'.format(name)
        super(KnowledgeGraphDataset, self).__init__(name,
                                                    url=url,
                                                    raw_dir=raw_dir,
                                                    force_reload=force_reload,
                                                    verbose=verbose)

    def process(self):
        # 跳过一些处理的代码
        # === 跳过数据处理 ===

        # 划分掩码
        g.edata['train_mask'] = train_mask
        g.edata['val_mask'] = val_mask
        g.edata['test_mask'] = test_mask

        # 边类型
        g.edata['etype'] = etype

        # 节点类型
        g.ndata['ntype'] = ntype
        self._g = g

    def __getitem__(self, idx):
        assert idx == 0, "这个数据集只有一个图"
        return self._g

    def __len__(self):
        return 1
```

如代码所示，图的 `edata` 存储了划分掩码。在 KnowledgeGraphDataset 源代码 中可以查看完整的代码。下面使用 ``KnowledgeGraphDataset``的子类 `dgl.data.FB15k237Dataset` 来做演示如何使用用于链路预测的数据集：

```
from dgl.data import FB15k237Dataset

# 导入数据
dataset = FB15k237Dataset()
graph = dataset[0]

# 获取训练集掩码
train_mask = graph.edata['train_mask']
train_idx = torch.nonzero(train_mask, as_tuple=False).squeeze()
src, dst = graph.edges(train_idx)

# 获取训练集中的边类型
rel = graph.edata['etype'][train_idx]
```

有关训练链接预测模型的完整指南，请参见 5.3 链接预测。

有关链接预测数据集的更多示例，请参考DGL的内置数据集：

- kgdata
- bitcoinotcdata