数据完全存于内存的数据集类

引言

在上一节内容中，我们学习了基于图神经网络的节点表征学习方法，并用了现成的很小的数据集实现了节点分类任务。在此第6节的上半部分，我们将学习在PyG中如何自定义一个**数据完全存于内存的数据集类**。

`InMemoryDataset`基类简介

在PyG中，我们通过继承`InMemoryDataset`类来自定义一个数据可全部存储到内存的数据集类。

```
class InMemoryDataset(root: Optional[str] = None,
transform: Optional[Callable] = None, pre_transform:
Optional[Callable] = None, pre_filter:
Optional[Callable] = None)
```

`InMemoryDataset`官方文档：
`torch_geometric.data.InMemoryDataset`

如上方的`InMemoryDataset`类的构造函数接口所示，每个数据集都要有一个**根文件夹（`root`）**，它指示数据集应该被保存在哪里。在根目录下至少有两个文件夹：

- 一个文件夹为`raw_dir`，它用于存储未处理的文件，从网络上下载的数据集文件会被存放到这里；
- 另一个文件夹为`processed_dir`，处理后的数据集被保存到这里。

此外，继承`InMemoryDataset`类的每个数据集类可以传递一个`transform`**函数**，一个`pre_transform`**函数**和一个`pre_filter`函数，它们默认都为`None`。

- `transform`函数接受`Data`对象为参数，对其转换后返回。此函数在每一次数据访问时被调用，所以它应该用于数据增广（Data Augmentation）。

- `pre_transform`函数接受`Data`对象为参数，对其转换后返回。此函数在样本`Data`对象保存到文件前调用，所以它最好用于只需要做一次的大量预计算。
- `pre_filter`函数可以在保存前手动过滤掉数据对象。该函数的一个用例是，过滤样本类别。

为了创建一个`InMemoryDataset`，我们需要**实现四个基本方法**：

- `raw_file_names()`这是一个属性方法，返回一个文件名列表，文件应该能在`raw_dir`文件夹中找到，否则调用`process()`函数下载文件到`raw_dir`文件夹。
- `processed_file_names()`。这是一个属性方法，返回一个文件名列表，文件应该能在`processed_dir`文件夹中找到，否则调用`process()`函数对样本做预处理然后保存到`processed_dir`文件夹。
- `download()`: 将原始数据文件下载到`raw_dir`文件夹。
- `process()`: 对样本做预处理然后保存到`processed_dir`文件夹。

```
import torch
from torch_geometric.data import InMemoryDataset, download_url

class MyOwnDataset(InMemoryDataset):
    def __init__(self, root, transform=None, pre_transform=None, pre_filter=None):
        super().__init__(root=root, transform=transform, pre_transform=pre_transform, pre_filter=pre_filter)
        self.data, self.slices = torch.load(self.processed_paths[0])

    @property
    def raw_file_names(self):
        return ['some_file_1', 'some_file_2', ...]

    @property
    def processed_file_names(self):
        return ['data.pt']

    def download(self):
        # Download to `self.raw_dir`.
```

```
        download_url(url, self.raw_dir)
        ...

    def process(self):
        # Read data into huge `Data` list.
        data_list = [...]

        if self.pre_filter is not None:
            data_list = [data for data in data_list if
self.pre_filter(data)]

        if self.pre_transform is not None:
            data_list = [self.pre_transform(data) for
data in data_list]

        data, slices = self.collate(data_list)
        torch.save((data, slices),
self.processed_paths[0])

```

样本从原始文件转换成`Data`类对象的过程定义在`process`函数中。在该函数中，有时我们需要读取和创建一个`Data`对象的列表，并将其保存到`processed_dir`中。由于python保存一个巨大的列表是相当慢的，因此我们在保存之前通过`collate()`函数将该列表集合成一个巨大的`Data`对象。该函数还会返回一个切片字典，以便从这个对象中重构单个样本。最后，我们需要在构造函数中把这`Data`对象和切片字典分别加载到属性`self.data`和`self.slices`中。我们通过下面的例子来介绍**生成一个`InMemoryDataset`子类对象时程序的运行流程。**

定义一个`InMemoryDataset`子类

由于我们手头没有实际应用中的数据集，因此我们以公开数据集`PubMed`为例子。`PubMed`数据集存储的是文章引用网络，文章对应图的结点，如果两篇文章存在引用关系（无论引用与被引），则这两篇文章对应的结点之间存在边。该数据集来源于论文Revisiting Semi-Supervised Learning with Graph Embeddings。我们直接基于PyG中的`Planetoid`类修改得到下面的`PlanetoidPubMed`数据集类。

```
import os.path as osp

```

```
2
3  import torch
4  from torch_geometric.data import (InMemoryDataset,
   download_url)
5  from torch_geometric.io import read_planetoid_data
6
7  class PlanetoidPubMed(InMemoryDataset):
8      r"""The citation network datasets "PubMed" from
   the
9      `"Revisiting Semi-Supervised Learning with Graph
   Embeddings"
10     <https://arxiv.org/abs/1603.08861>`_ paper.
11     Nodes represent documents and edges represent
   citation links.
12     Training, validation and test splits are given by
   binary masks.
13
14     Args:
15         root (string): Root directory where the
   dataset should be saved.
16         split (string): The type of dataset split
17             (:obj:`"public"`, :obj:`"full"`,
   :obj:`"random"`).
18             If set to :obj:`"public"`, the split will
   be the public fixed split
19             from the
20             `"Revisiting Semi-Supervised Learning
   with Graph Embeddings"
21             <https://arxiv.org/abs/1603.08861>`_
   paper.
22             If set to :obj:`"full"`, all nodes except
   those in the validation
23             and test sets will be used for training
   (as in the
24             `"FastGCN: Fast Learning with Graph
   Convolutional Networks via
25             Importance Sampling"
   <https://arxiv.org/abs/1801.10247>`_ paper).
26             If set to :obj:`"random"`, train,
   validation, and test sets will be
```

```
            randomly generated, according to
:obj:`num_train_per_class`,
            :obj:`num_val` and :obj:`num_test`.
(default: :obj:`"public"`)
        num_train_per_class (int, optional): The
number of training samples
            per class in case of :obj:`"random"`
split. (default: :obj:`20`)
        num_val (int, optional): The number of
validation samples in case of
            :obj:`"random"` split. (default:
:obj:`500`)
        num_test (int, optional): The number of test
samples in case of
            :obj:`"random"` split. (default:
:obj:`1000`)
        transform (callable, optional): A
function/transform that takes in an
            :obj:`torch_geometric.data.Data` object
and returns a transformed
            version. The data object will be
transformed before every access.
            (default: :obj:`None`)
        pre_transform (callable, optional): A
function/transform that takes in
            an :obj:`torch_geometric.data.Data`
object and returns a
            transformed version. The data object will
be transformed before
            being saved to disk. (default:
:obj:`None`)
    """

    url =
'https://github.com/kimiyoung/planetoid/raw/master/da
ta'

    def __init__(self, root, split="public",
num_train_per_class=20,
                 num_val=500, num_test=1000,
transform=None,
```

```
                 pre_transform=None):

        super(PlanetoidPubMed, self).__init__(root,
transform, pre_transform)
        self.data, self.slices =
torch.load(self.processed_paths[0])

        self.split = split
        assert self.split in ['public', 'full',
'random']

        if split == 'full':
            data = self.get(0)
            data.train_mask.fill_(True)
            data.train_mask[data.val_mask |
data.test_mask] = False
            self.data, self.slices =
self.collate([data])

        elif split == 'random':
            data = self.get(0)
            data.train_mask.fill_(False)
            for c in range(self.num_classes):
                idx = (data.y ==
c).nonzero(as_tuple=False).view(-1)
                idx = idx[torch.randperm(idx.size(0))
[:num_train_per_class]]
                data.train_mask[idx] = True

            remaining =
(~data.train_mask).nonzero(as_tuple=False).view(-1)
            remaining =
remaining[torch.randperm(remaining.size(0))]

            data.val_mask.fill_(False)
            data.val_mask[remaining[:num_val]] = True

            data.test_mask.fill_(False)
            data.test_mask[remaining[num_val:num_val
+ num_test]] = True

```

```
            self.data, self.slices =
self.collate([data])

    @property
    def raw_dir(self):
        return osp.join(self.root, 'raw')

    @property
    def processed_dir(self):
        return osp.join(self.root, 'processed')

    @property
    def raw_file_names(self):
        names = ['x', 'tx', 'allx', 'y', 'ty',
'ally', 'graph', 'test.index']
        return ['ind.pubmed.{}'.format(name) for name
in names]

    @property
    def processed_file_names(self):
        return 'data.pt'

    def download(self):
        for name in self.raw_file_names:
            download_url('{}/{}'.format(self.url,
name), self.raw_dir)

    def process(self):
        data = read_planetoid_data(self.raw_dir,
'pubmed')
        data = data if self.pre_transform is None
else self.pre_transform(data)
        torch.save(self.collate([data]),
self.processed_paths[0])

    def __repr__(self):
        return '{}()'.format(self.name)

```

在我们生成一个`PlanetoidPubMed`类的对象时，**程序运行流程**如下：

- 首先**检查数据原始文件是否已下载**：

    - 检查`self.raw_dir`目录下是否存在`raw_file_names()`属性方法返回的每个文件，
    - 如有文件不存在，则调用`download()`方法执行原始文件下载。
    - 其中`self.raw_dir`为`osp.join(self.root, 'raw')`。
- 其次**检查数据是否经过处理**：
    - 首先**检查之前对数据做变换的方法**：检查`self.processed_dir`目录下是否存在`pre_transform.pt`文件：如果存在，意味着之前进行过数据变换，则需加载该文件获取之前所用的数据变换的方法，并检查它与当前`pre_transform`参数指定的方法是否相同；如果不相同则会报出一个警告，“The pre_transform argument differs from the one used in ……”。
    - 接着**检查之前的样本过滤的方法**：检查`self.processed_dir`目录下是否存在`pre_filter.pt`文件，如果存在，意味着之前进行过样本过滤，则需加载该文件获取之前所用的样本过滤的方法，并检查它与当前`pre_filter`参数指定的方法是否相同，如果不相同则会报出一个警告，“The pre_filter argument differs from the one used in ……”。其中`self.processed_dir`为`osp.join(self.root, 'processed')`。
    - 接着**检查是否存在处理好的数据**：检查`self.processed_dir`目录下是否存在`self.processed_paths`方法返回的所有文件，如有文件不存在，意味着不存在已经处理好的样本的文件，如需执行以下的操作：
        - 调用`process`方法，进行数据处理。
        - 如果`pre_transform`参数不为`None`，则调用`pre_transform`方法进行数据处理。
        - 如果`pre_filter`参数不为`None`，则进行样本过滤（此例子中不需要进行样本过滤，`pre_filter`参数始终为`None`）。
        - 保存处理好的数据到文件，文件存储在`processed_paths()`属性方法返回的路径。如果将数据保存到多个文件中，则返回的路径有多个。这些路径都在`self.processed_dir`目录下，以`processed_file_names()`属性方法的返回值为文件名。
        - 最后保存新的`pre_transform.pt`文件和`pre_filter.pt`文件，其中分别存储当前使用的数据处理方法和样本过滤方法。

现在让我们**查看这个数据集**：

```
dataset = PlanetoidPubMed('../dataset/Planetoid/PubMed')
print(dataset.num_classes)
print(dataset[0].num_nodes)
print(dataset[0].num_edges)
print(dataset[0].num_features)

# 3
# 19717
# 88648
# 500
```

可以看到这个数据集包含三个分类任务，共19,717个结点，88,648条边，节点特征维度为500。

参考资料

- `InMemoryDataset`官方文档：`torch_geometric.data.InMemoryDataset`
- `Data`官方文档：`torch_geometric.data.Data`
- 提出PubMed数据集的论文：Revisiting Semi-Supervised Learning with Graph Embeddings
- `Planetoid`官方文档：torch_geometric.datasets.Planetoid