# 数据完全存于内存的数据集类

## 引言

对于占用内存有限的数据集，我们可以将整个数据集的数据都存储到内存里。PyG为我们提供了方便的构造数据完全存于内存的数据集类，简称为InMemory数据集类，的方式，在此小节我们就将学习构造InMemory数据集类的方式。

内容安排如下：

- 首先，我们将学习PyG规定的使用数据的一般过程；
- 其次，我们将学习InMemoryDataset基类；
- 接着，我们将学习一个简化的InMemory数据集类；
- 最后，我们将学习一个InMemory数据集类实例，以及使用该数据集类时会发生的一些过程。

## 使用数据集的一般过程

**PyG定义了使用数据的一般过程**：

1. 从网络上**下载**数据原始文件；
2. 对数据原始文件做处理，为每一个图样本**生成**一个**Data对象**；
3. 对每一个Data对象**执行数据处理**，使其转换成新的Data对象；
4. **过滤Data对象**；
5. **保存Data对象到文件**；
6. 获取Data对象，在每一次获取Data对象时，都先对Data对象做数据变换（于是获取到的是数据变换后的Data对象）。

实际中并非需要严格执行每一个步骤，

以上步骤在特定的条件下可以被跳过，具体内容在下文里会学到。

# 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类初始化方法参数说明:

- root：字符串类型，**存储数据集的文件夹的路径**。该文件夹下有两个文件夹：
  - 一个文件夹为记录在**raw_dir**，它用于存储未处理的文件，从网络上下载的**数据集原始文件**会被存放到这里；
  - 另一个文件夹记录在**processed_dir**，**处理后的数据**被保存到这里，以后从此文件夹下加载文件即可获得Data对象。
  - 注：raw_dir和processed_dir是属性方法，我们可以自定义要使用的文件夹。
- transform：函数类型，一个数据转换函数，它接收一个Data对象并返回一个转换后的Data对象。**此函数在每一次数据获取过程中都会被执行**。获取数据的函数首先使用此函数对Data对象做转换，然后才返回数据。此函数应该用于数据增广（Data Augmentation）。该参数默认值为None，表示不对数据做转换。
- pre_transform：函数类型，一个数据转换函数，它接收一个Data对象并返回一个转换后的Data对象。**此函数在Data对象被保存到文件前调用**。因此它应该用于只执行一次的数据预处理。该参数默认值为None，表示不做数据预处理。
- pre_filter：函数类型，**一个检查数据是否要保留的函数**，它接收一个Data对象，返回此Data对象是否应该被包含在最终的数据集中。此函数也在Data对象被保存到文件前调用。该参数默认值为None，表示不做数据检查，保留所有的数据。

通过继承InMemoryDataset类来构造一个我们自己的数据集类，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**数据集原始文件**的文件名列表，数据集原始文件应该能在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**存储处理过的数据的文件**的文件名列表，存储处理过的数据的文件应该能在processed_dir文件夹中找到，否则调用process()函数对样本做处理，然后保存处理过的数据到processed_dir文件夹下的文件里。
- `download()`: **下载数据集原始文件**到raw_dir文件夹。
- `process()`: **处理数据，保存处理好的数据到processed_dir文件夹下的文件。**

# 一个简化的InMemory数据集类

以下是一个简化的自定义的数据集类的例子：

```
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):
```

```
23          # Read data into huge `Data` list.
24          data_list = [...]
25
26          if self.pre_filter is not None:
27              data_list = [data for data in data_list if
    self.pre_filter(data)]
28
29          if self.pre_transform is not None:
30              data_list = [self.pre_transform(data) for data in
    data_list]
31
32          data, slices = self.collate(data_list)
33          torch.save((data, slices), self.processed_paths[0])
34
```

- 在raw_file_names属性方法里，也就是第11行，写上数据集原始文件有哪些，在此例子中有some_file_1, some_file_2等。
- 在processed_file_names属性方法里，也就是第15行，处理过的数据要保存在哪些文件里，在此例子中只有data.pt。
- 在download方法里，我们实现下载数据到self.raw_dir文件夹的逻辑。
- 在process方法里，我们实现数据处理的逻辑：
  - 首先，我们从数据集原始文件中读取样本并生成Data对象，所有样本的Data对象保存在列表data_list中。
  - 其次，如果要对数据做过滤的话，我们执行数据过滤的过程。
  - 接着，如果要对数据做处理的话，我们执行数据处理的过程。
  - 最后，我们保存处理好的数据到文件。但由于python保存一个巨大的列表是相当慢的，我们需要先将所有Data对象合并成一个巨大的Data对象再保存。collate()函数接收一个列表的Data对象，返回合并后的Data对象以及用于从合并后的Data对象重构各个原始Data对象的切片字典slices。最后我们将这个巨大的Data对象和切片字典slices保存到文件。

## InMemoryDataset数据集类实例

我们以公开数据集PubMed为例子，进行InMemoryDataset数据集实例分析。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。PyG中的Planetoid数据集类包含了数据集PubMed的使用，因此我们直接基于Planetoid类进行修改，得到PlanetoidPubMed数据集类。

我们将首先学习PlanetoidPubMed数据集类的构造，其次学习使用PlanetoidPubMed数据集类时会发生的过程。

## *PlanetoidPubMed数据集类的构造*

PlanetoidPubMed数据集类如下所示：

```
import os.path as osp

import torch
from torch_geometric.data import (InMemoryDataset, download_url)
from torch_geometric.io import read_planetoid_data

class PlanetoidPubMed(InMemoryDataset):
    r""" 节点代表文章，边代表引文关系。
         训练、验证和测试的划分通过二进制掩码给出。
    参数:
        root (string): 存储数据集的文件夹的路径
        transform (callable, optional): 数据转换函数，每一次获取数据时被调用。
        pre_transform (callable, optional): 数据转换函数，数据保存到文件前被调用。
    """

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

    def __init__(self, root, transform=None, pre_transform=None):

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

```
22
23      @property
24      def raw_dir(self):
25          return osp.join(self.root, 'raw')
26
27      @property
28      def processed_dir(self):
29          return osp.join(self.root, 'processed')
30
31      @property
32      def raw_file_names(self):
33          names = ['x', 'tx', 'allx', 'y', 'ty', 'ally', 'graph',
    'test.index']
34          return ['ind.pubmed.{}'.format(name) for name in names]
35
36      @property
37      def processed_file_names(self):
38          return 'data.pt'
39
40      def download(self):
41          for name in self.raw_file_names:
42              download_url('{}/{}'.format(self.url, name),
    self.raw_dir)
43
44      def process(self):
45          data = read_planetoid_data(self.raw_dir, 'pubmed')
46          data = data if self.pre_transform is None else
    self.pre_transform(data)
47          torch.save(self.collate([data]), self.processed_paths[0])
48
49      def __repr__(self):
50          return '{}()'.format(self.name)
51
```

该类初始化方法的参数说明见代码。代码中还实现了raw_dir()和processed_dir()两个属性方法，通过修改返回值我们就可以修改要使用的文件夹。

# *该数据集类的使用*

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。
  - 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，接着需要加载该文件，以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 ……”。
    - self.processed_dir为osp.join(self.root, 'processed')。
  - 其次，**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，则加载该文件并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 ……”。
  - 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_file_names属性方法返回的所有文件，如有文件不存在，则需要执行以下的操作：
    - 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()函数进行数据处理。
    - 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()**属性方法返回的文件路径。如果将数据保存到多个文件中，则返回的路径有多个。

    - **`processed_paths()`属性方法是在基类中定义的**，它对`self.processed_dir`文件夹与`processed_file_names()`属性方法的返回每一个文件名做拼接，然后返回。
- 最后保存新的`pre_transform.pt`文件和`pre_filter.pt`文件，它们分别存储当前使用的数据处理方法和样本过滤方法。

最后让我们来**查看这个数据集**：

```
dataset = PlanetoidPubMed('dataset/PlanetoidPubMed')
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