# graph-sage code-update

> 我们的工作：https://github.com/Dodo-D-Caster/GraphSAGE-pytorch-inductive

# 1 inductive数据处理

cora Tensor版本数据处理：

https://gist.github.com/k1ochiai/d9c66fc50bf3f7181f9337753c68b80a

## 疑问1：如何确保训练过程中test和val数据集的不可见

以cora为例，cora有引用集合和节点的特征集，我们用引用集构建adj_list。预处理时，先把节点分成train，test和val三个集，test集就是未见的节点。问题来了，我们的图都是连接的，train在采样时，几乎一定能采到test 的节点，这样不就是transductive的了？所以，我们需要把这些点给排出去。

### 方法1

使用标签法，把节点给打上test和val的标签，test和val都是false的就是train集。在构建adj_list时，我们同样将引用集根据节点，给切分开。具体看以下代码(utils.py)：

```
for edge in G.edges():
        if (G.node[edge[0]]['val'] or G.node[edge[1]]['val'] or
            G.node[edge[0]]['test'] or G.node[edge[1]]['test']):
            G[edge[0]][edge[1]]['train_removed'] = True
        else:
            G[edge[0]][edge[1]]['train_removed'] = False
```

这里我们将图中test，val与train的关系断开，这样后续建立adj_list时，建不到test的节点了，彻底做到了inductive.

如何去构建adj_list(minibatch.py):

```
def construct_adj(self):
        adj = len(self.id2idx)*np.ones((len(self.id2idx)+1, self.max_degree))
        deg = np.zeros((len(self.id2idx),))

        for nodeid in self.G.nodes():
            if self.G.node[nodeid]['test'] or self.G.node[nodeid]['val']:
                continue
            neighbors = np.array([self.id2idx[neighbor]
                for neighbor in self.G.neighbors(nodeid)
                if (not self.G[nodeid][neighbor]['train_removed'])])
            deg[self.id2idx[nodeid]] = len(neighbors)
            if len(neighbors) == 0:
                continue
            if len(neighbors) > self.max_degree:
                neighbors = np.random.choice(neighbors, self.max_degree, replace=False)
            elif len(neighbors) < self.max_degree:
                neighbors = np.random.choice(neighbors, self.max_degree, replace=True)
            adj[self.id2idx[nodeid], :] = neighbors
        return adj, deg
```

### 方法2

在建立adj_lists的时候，对于验证集和测试集中的节点，直接建立边即可，而对于训练集，则需要判断两个节点是否都在训练集中，只有同时在训练集中才建立双向的边，这样就保证了在训练时只运用了训练集中的数据，从而确保是inductive方法。

```
# 对于测试集和验证集，直接建立adj_list
if node_map[pair[0]] not in train_index:
adj_lists[node_map[pair[0]]].add(node_map[pair[1]])
if node_map[pair[1]] not in train_index:
adj_lists[node_map[pair[1]]].add(node_map[pair[0]])
```

```
# 对于训练集，neighbor只能包含训练集中的节点
if node_map[pair[1]] in train_index and node_map[pair[1]] in train_index:
adj_lists[node_map[pair[0]]].add(node_map[pair[1]])
adj_lists[node_map[pair[1]]].add(node_map[pair[0]])
```

### 问题2：如何处理训练集中的孤立节点

由于我们将训练集中的节点邻居设置为训练集中的节点，那么就可能出现如下问题：某个训练集中的节点的所有邻居都在测试集或验证集中，导致在训练过程中，他实际上成了没有邻居的孤立节点。

孤立节点带来的问题是，当计算损失时，需要对节点进行正采样和负采样，我们无法对孤立节点进行正采样。

#### 方法1：增加自循环

在为节点建立adj_list的时候，为节点加上自循环。

```
if node_map[pair[0]] not in adj_lists: adj_lists[node_map[pair[0]]].add(node_map[pair[0]])
if node_map[pair[1]] not in adj_lists: adj_lists[node_map[pair[1]]].add(node_map[pair[1]])
```

#### 方法2：直接删除train中孤立节点

赋初值为空

```
if node_map[pair[0]] not in adj_lists: adj_lists[node_map[pair[0]]] = set()
if node_map[pair[1]] not in adists: adj_lists[node_map[pair[0]]] = set()in adj_lists: adj_lists[node_map[pair[0]]] = set()ists: adj_lists[n
```

删除孤立节点的adj_list和train_index

```
i = 0
while i < len(train_index):
    if i<len(train_index) and len(adj_lists[train_index[i]]) == 0:
        # print('--------------deleted--------------', i)
        del adj_lists[train_index[i]]
        train_index = np.delete(train_index, i)
        i -= 1
    i += 1
```

# 2 Sample方式

## 思路1：

按节点的度中心性进行排序，取top5和bottom5

## 思路2：

按节点的紧密中心性进行排序，取top10

# 3 结果

MEAN 10轮epoch

## sample - random

| | supervised | unsupervised | sup+unsup |
|---|---|---|---|
| inductive_delete | 0.835920 | 0.545455 | 0.761641 |
| | | | |

| | | | |
|---|---|---|---|
| inductive_selfloop | 0.853659 | 0.579823 | 0.808204 |
| transductive | 0.851441 | 0.701774 | 0.853659 |

## sample - 度中心性

取度最大的5个和最小的5个邻居

| | supervised | unsupervised | sup+unsup |
|---|---|---|---|
| inductive_delete | 0.843681 | 0.558758 | 0.819290 |
| inductive_selfloop | 0.859202 | 0.576497 | 0.812639 |
| transductive | 0.868071 | 0.729490 | 0.853659 |

## sample - 紧密中心性

取中心性最大的10个节点

$$\text{中心性} = 1 \div \frac{\sum \text{邻居的度}}{\text{邻居数量}} = \frac{\text{邻居数量}}{\sum \text{邻居的度}}$$

| | supervised | unsupervised | sup+unsup |
|---|---|---|---|
| inductive_delete | 0.848115 | 0.705100 | 0.783814 |
| inductive_selfloop | 0.873614 | 0.686253 | 0.786031 |
| transductive | 0.862528 | 0.742794 | 0.862528 |