实验 2 图表示学习实验

一、图节点表示实验

（一）实验内容

在图节点表示学习的第一部分实验中，我们将进行节点表示学习的整个流程，具体分为以下三步：

- 第一步，我们先加载经典社交网络图 Karate Club Network。并探索这个图上的多种统计信息。
- 接着我们尝试将图结构转化成 tensor 形式，为我们进一步在图上进行机器学习算法做准备。
- 最后，我们将实现首个图算法：一个图节点表示模型。

（二）实验过程

1. Question1:karate club network 的 average degree(平均度)是多少？

由于 karate club network 是无向图，所以每一条边能贡献的度数为 2，由此可知，总的度数为总边数的两倍，然后再除以节点个数，最后对结果四舍五入到整数得到答案。

代码如下：

```
############  Your  code  here  ############
avg_degree  =  2*num_edges/num_nodes
avg_degree  =  round(avg_degree)
########################################
```

该 karate club network 的平均度如下：

```
Average degree of karate club network is 5
```

2. Question2:karate club 网络的平均聚类系数是多少 k?

这里直接调用自带的 average_clustering 函数进行计算，然后对结果保留两位小数，可以得到答案。

代码如下：

```
############  Your  code  here  ############
##  Note:
##  1:  Please  use  the  appropriate  NetworkX  clustering  function
avg_cluster_coef  =  nx.average_clustering(G)
avg_cluster_coef  =  round(avg_cluster_coef,  2)
########################################
```

平均聚类系数如下：

```
Average clustering coefficient of karate club network is 0.57
```

3. Question3:在一次 PageRank 迭代之后，节点 0(id 为 0 的节点)的 PageRank 值是多少?

PageRank 使用网络的链接结构来衡量图中节点的重要性。来自重要页面的“vote”更有价值。

PageRank 算法输出一个概率分布，表示随机浏览者点击链接到达任何特定页面的可能性。在每个时间步，随机的冲浪者有两个选择：

- β的概率随机跟随一个页面；
- $1-\beta$的概率随机跳转到一个页面。

因此，一个特定页面的重要性由下面的 PageRank 等式计算：

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

根据上述公式，我们可以得到如下代码：

```
############  Your  code  here  ############
##  Note:
##  1:  You  should  not  use  nx.pagerank
#  实现pagerank
r1  =  (1-beta)/G.number_of_nodes()
for  n  in  G.neighbors(node_id):
        r1  +=  beta  *  r0  /  G.degree(n)
r1  =  round(r1,  2)
########################################
```

首先，算法计算了一个常数(1-beta)/G.number_of_nodes()，其中 beta 是阻尼系数，G.number_of_nodes()是图中节点的数量。这个常数表示了随机跳转到任意节点的概率。

然后，算法遍历节点 node_id 的所有邻居节点，对于每个邻居节点 n，计算了 beta*r0/G.degree(n)，其中 beta 是阻尼系数，r0 是初始 PageRank 值，G.degree(n)是节点 n 的度数。这一部分表示了从邻居节点跳转到当前节点的概率。

最后，将这两部分的结果相加得到节点的一次迭代 PageRank 值 r1。

PageRank 第一次迭代结果如下：

```
The PageRank value for node 0 after one iteration is 0.13
```

4. Question4:获取 karate club network 的边列表并将其转换为 torch.LongTensor。pos_edge_index tensor 的 torch.sum 值是多少?

首先获取 G 的所有边，并将其转化为 list，如下图所示：

```
############# Your code here ############
edge_list = list(G.edges())
########################################
```

接着将边列表转化为 torch.LongTensor 类型，，最后计算 edge 的 shape 和 sun 值，如下图所示。

```
def edge_list_to_tensor(edge_list):
    # TODO: Implement the function that transforms the edge_list to
    # tensor. The input edge_list is a list of tuples and the resulting
    # tensor should have the shape [2, len(edge_list)].

    edge_index = torch.tensor([])

    ############# Your code here ############
    edge_index = torch.tensor(edge_list).t()
    edge_index.type(torch.long)
    ########################################

    return edge_index

pos_edge_list = graph_to_edge_list(G)
pos_edge_index = edge_list_to_tensor(pos_edge_list)
print("The pos_edge_index tensor has shape {}".format(pos_edge_index.shape))
print("The pos_edge_index tensor has sum value {}".format(torch.sum(pos_edge_index)))
```

结果如下所示：

```
The pos_edge_index tensor has shape torch.Size([2, 78])
The pos_edge_index tensor has sum value 2535
```

5. Question5:请实现以下对负边进行采样的函数。然后回答哪些边(edge_1 到 edge_5)是 karate club network 中的负边?

两两配对 G 中的节点，由这两个节点构成的边如果不在 pos_edge_list 中，则为负边，接着对负边列表进行采样，得到结果。

代码如下：

```
############# Your code here ############
pos_edge_list = list(G.edges())
for n1 in G.nodes():
        for n2 in G.nodes():
                if n1 != n2 and ((n1, n2) not in pos_edge_list
                        neg_edge_list.append((n1, n2))
neg_edge_list = random.sample(neg_edge_list, num_neg_samples)
########################################
```

判断负边结果如下：

```
The neg_edge_index tensor has shape torch.Size([2, 78])
(7, 1) is a positive edge
(1, 33) is a negative edge
(33, 22) is a positive edge
(0, 4) is a positive edge
(4, 2) is a negative edge
```

6. Question6:训练表示：你能得到的最好表示是什么?请在实验报告上记录最佳损失和准确性。

在训练时，首先将需要训练的边进行 embedding，然后将两个节点的 embedding 结果进行点乘，经过 sigmoid 得到预测结果，将预测结果和真实结果计算 loss 和 accuracy，然后反向传播进行训练。

训练代码如下：

```
for i in range(epochs):

    ############# Your code here ############
    edge_embd = emb(train_edge)
    # print(edge_embd[0].shape)
    dot_product = torch.sum(edge_embd[0]*edge_embd[1], dim=1)
    pred = sigmoid(dot_product)
    loss = loss_fn(pred, train_label)
    acc = accuracy(pred, train_label)
    print(f"Epoch {i}, Loss: {loss}, Accuracy: {acc}")
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    ########################################
```

计算准确率代码如下：

```
############# Your code here ############
pred_label = (pred > 0.5).type(torch.long)
accu = (pred_label == label).type(torch.float).mean().item()
accu = round(accu, 4)
########################################
```

部分训练结果如下：

```
Epoch 495, Loss: 0.12296140938997269, Accuracy: 0.9231
Epoch 496, Loss: 0.12290618568658829, Accuracy: 0.9231
Epoch 497, Loss: 0.12285127490758896, Accuracy: 0.9231
Epoch 498, Loss: 0.1227966919541359, Accuracy: 0.9231
Epoch 499, Loss: 0.1227424293756485, Accuracy: 0.9231
```

二、GNN

（一）实验内容

在图节点表示学习的第二部分实验中，我们将使用 PyTorchGeometric(PyG)构建我们自己的图神经网络，然后将该模型应用于两个 OpengraphBenchmark(OGB)数据集。这两个数据集将用于在两个不同的基于图的任务上对模型的性能进行基准测试:

1) 节点属性预测，预测单个节点的属性;
2) 图属性预测，预测整个图或子图的属性。

以下是实验步骤：

- 首先，我们将学习 PyTorchGeometric 如何将图存储为 PyTorch 张量。
- 然后，我们将使用 ogb 包加载和检查开放图形基准(OGB)数据集之一。OGB 是一个现实的、大规模的、不同的基准数据集的集合，用于图上的机器学习。ogb 包不仅为每个数据集提供数据加载器，还提供模型评估器。
- 最后，我们将使用 PyTorchGeometric 构建我们自己的图神经网络。然后，我们将在 OGB 节点属性预测和图属性预测任务上训练和评估我们的模型。

（二）实验过程

1. Question1:ENZYMES 数据集中的类和特征的数量是多少?

对于类的数量，直接将数据集中的 num_classes 赋值给 num_classes；

对于特征的数量，直接将数据集中的 num_features 赋值给 num_features。

具体代码如下：

```
############# Your code here ############
## (~1 line of code)
## Note
## 1. Colab autocomplete functionality might be useful.
num_classes = pyg_dataset.num_classes
#########################################
```

```
############# Your code here ############
## (~1 line of code)
## Note
## 1. Colab autocomplete functionality might be useful.
num_features = pyg_dataset.num_features
#########################################
```

结果如下：

```
ENZYMES dataset has 6 classes
ENZYMES dataset has 3 features
```

2. Question2:在 ENZYMES 数据集中索引为 100 的图的标签是什么?

这里比较简单，直接找到 idx 索引下的样本，然后取出它的标签即可。

代码如下：

```
############# Your code here ############
## (~1 line of code)
label = pyg_dataset[idx].y[0]
#########################################
```

结果如下：

```
Data(edge_index=[2, 168], x=[37, 3], y=[1])
Graph with index 100 has label 4
```

3. Question3:索引为 200 的图有多少条边?

这里先找到 idx 索引下的样本并返回它的所有边索引，如果这个图是无向图，则两条相反的边只算一条边，所以需要去掉一些边后再计算该图具体有多少条边即可。

代码如下：

```
############## Your code here ############
## Note:
## 1. You can't return the data.num_edges directly
## 2. We assume the graph is undirected
## 3. Look at the PyG dataset built in functions
## (~4 lines of code)
edges = pyg_dataset[idx].edge_index
print(edges.shape)
edges = edges.t().tolist()
print(print(edges))
edges = [[i, j] for [i, j] in edges if [j, i] not in edges
num_edges = len(edges)
##########################################
```

结果如下：

```
torch.Size([2, 106])
[[0, 21], [0, 22], [0, 23], [1, 7], [1, 15], [1, 23], [1, 24], [2, 3], [2, 5], [2, 13], [2, 26],
None
Graph with index 200 has 53 edges
```

4. Question4:在 ogbn-arxiv 图中有多少特征?

和 question1 类似，直接返回该图的特征数即可。

代码如下：

```
############## Your code here ############
## (~1 line of code)
num_features = data.num_features
##########################################
```

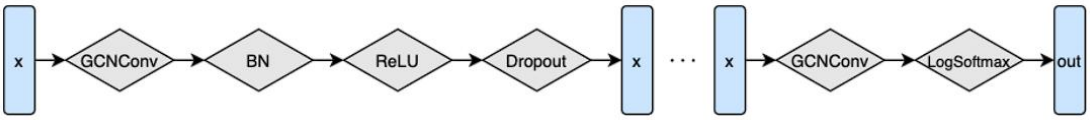
结果如下：

```
The graph has 128 features
```

5. Question5:你的 best_model 验证和测试精度是多少?

这一部分我们将使用 PyTorchGeometric 构建我们的第一个图神经网络。然后我们将其应用于节点属性预测(节点分类)任务。

首先构建模型，这里我们将自己构建 GCN 模型，具体结构如下：



GCN 模型的每一层由以下几个部分构成：

- GCNConv 层，也就是图卷积层，用于从图的结构中提取节点的特征信息。GCNConv 层使用输入特征 x 和图的邻接矩阵 adj_t 来计算节点的新的特征表示；
- BatchNorm1d 层，也就是批量归一化层，用于对每个节点的特征进行归一化；
- ReLU 激活函数；
- Dropout 层（在训练模式下），用于防止过拟合。它随机将一部分节点的特征值置为零，这样可以迫使模型更好地泛化。

最后一层经过一个图卷积层后进行 softmax 得到预测分类结果。

具体代码如下：

__init__函数：

```
self.convs = torch.nn.ModuleList(
        [GCNConv(in_channels=input_dim, out_channels=hidden_dim)] +
        [GCNConv(in_channels=hidden_dim, out_channels=hidden_dim)
                for i in range(num_layers-2)] +
        [GCNConv(in_channels=hidden_dim, out_channels=output_dim)]
)
self.bns = torch.nn.ModuleList([
        torch.nn.BatchNorm1d(num_features=hidden_dim)
                for i in range(num_layers-1)
])
self.softmax = torch.nn.LogSoftmax()
```

forward 函数：

```
for conv, bn in zip(self.convs[:-1], self.bns):
        x1 = F.relu(bn(conv(x, adj_t)))
        if self.training:
                x1 = F.dropout(x1, p=self.dropout)
        x = x1
x = self.convs[-1](x, adj_t)
out = x if self.return_embeds else self.softmax(x)
```

以下是我们的实验设置：

- Dataset：使用 ogbn-arxiv 数据集，并将数据集中的邻接矩阵转化为对称矩阵；
- 模型层数：设置为 3；

- 隐藏层维数：设置为 256；
- Dropout 参数：设置为 0.5；
- 学习率：设置为 0.01；
- Epoch 数：设置为 100；

接下来就可以开始训练了，训练时的每一步都需要计算 loss 并反向传播更新参数，还要进行测试，具体代码如下：

train 函数：

```
optimizer.zero_grad()
out  =  model(data.x,  data.adj_t)
loss  =  loss_fn(out[train_idx],  data.y[train_idx].reshape(-1))
```

test 函数：

```
out  =  model(data.x,  data.adj_t)
```

运行结果如下：

```
Epoch: 94, Loss: 0.9263, Train: 73.60%, Valid: 71.91% Test: 70.91%
Epoch: 95, Loss: 0.9239, Train: 73.54%, Valid: 71.97% Test: 71.23%
Epoch: 96, Loss: 0.9252, Train: 73.65%, Valid: 71.86% Test: 70.76%
Epoch: 97, Loss: 0.9200, Train: 73.75%, Valid: 71.51% Test: 70.04%
Epoch: 98, Loss: 0.9165, Train: 73.65%, Valid: 71.47% Test: 69.93%
Epoch: 99, Loss: 0.9167, Train: 73.59%, Valid: 71.56% Test: 70.37%
Epoch: 100, Loss: 0.9165, Train: 73.72%, Valid: 71.43% Test: 70.07%
```

best_model 验证和测试精度如下：

```
Saving Model Predictions
Best model: Train: 73.54%, Valid: 71.97% Test: 71.23%
```

6. Question 6: 你的 best_model 验证和测试 ROC-AUC 分数是多少?

在这一部分，我们将使用 GCN 创建一个用于图属性预测(图分类)的图神经网络。

首先构建模型，这里使用上面已经构建好的 GCN 模型，并进行进一步处理。具体结构如下：

- AtomEncoder 层，将原子特征编码为隐藏维度的特征向量。
- GCN 层，图卷积网络（GCN）用于从图的结构中提取节点的特征信息。
- Global Mean Pooling 层，对每个图中的节点特征进行平均池化，以获得图级别的特征表示。
- Linear 层，对每个图的特征表示进行线性变换，以预测图的属性。

具体代码如下：

__init__函数：

```
############  Your  code  here  ############
##  Note:
##  1.  Initialize  self.pool  as  a  global  mean
##  For  more  information  please  refer  to  the
##  https://pytorch-geometric.readthedocs.io/en/lat
self.pool  =  global_mean_pool
########################################
```

forward 函数：

```
embed  =  self.gnn_node(embed,  edge_index)
features  =  self.pool(embed,  batch)
out  =  self.linear(features)
```

以下是我们的实验设置：

- Dataset：使用 ogbg-molhiv 数据集；
- 模型层数：设置为 5；
- 隐藏层维数：设置为 256；
- Dropout 参数：设置为 0.5；
- 学习率：设置为 0.001；
- Epoch 数：设置为 30；

接下来就可以开始训练了，训练时的每一步都需要计算 loss 并反向传播更新参数，还要进行测试，具体代码如下：

train 函数：

```
optimizer.zero_grad()
out  =  model(batch)
loss  =  loss_fn(out[is_labeled],  batch.y[is_labeled].float())
```

运行结果如下：

```
Evaluating...
Iteration: 100% 1029/1029 [00:09<00:00, 128.96it/s]
Iteration: 100% 129/129 [00:01<00:00, 132.80it/s]
Iteration: 100% 129/129 [00:01<00:00, 132.55it/s]
Epoch: 30, Loss: 0.2382, Train: 84.68%, Valid: 77.05% Test: 75.06%
```

best_model 验证和测试 ROC-AUC 分数如下：

```
Iteration: 100% 1029/1029 [00:08<00:00, 130.39it/s]
Iteration: 100% 129/129 [00:01<00:00, 128.71it/s]
Saving Model Predictions
Iteration: 100% 129/129 [00:00<00:00, 130.48it/s]
Saving Model Predictions
Best model: Train: 82.08%, Valid: 79.07% Test: 73.50%
```

7. Question 7: 在 Pytorch Geometric 中使用另外两个全局池化层进行实验。

这里直接更改模型的 pool 层即可，分别改为 global_max_pool 和 global_add_pool 进行实验。

global_max_pool 具体代码如下：

```
from  torch_geometric.nn  import  global_max_pool
model.pool  =  global_max_pool
#  print(model.pool)
```

训练结果如下：

```
Iteration: 100% 1029/1029 [00:15<00:00, 56.08it/s]
Evaluating...
Iteration: 100% 1029/1029 [00:08<00:00, 127.77it/s]
Iteration: 100% 129/129 [00:01<00:00, 126.66it/s]
Iteration: 100% 129/129 [00:01<00:00, 126.85it/s]
Epoch: 30, Loss: 0.0228, Train: 84.22%, Valid: 76.90% Test: 74.30%
```

best_model 验证和测试 ROC-AUC 分数如下：

```
Iteration: 100% 1029/1029 [00:08<00:00, 134.16it/s]
Iteration: 100% 129/129 [00:00<00:00, 132.90it/s]
Saving Model Predictions
Iteration: 100% 129/129 [00:01<00:00, 130.25it/s]
Saving Model Predictions
Best model: Train: 82.84%, Valid: 80.76% Test: 75.01%
```

global_add_pool 具体代码如下：

```
from  torch_geometric.nn  import  global_add_pool
model.pool  =  global_add_pool
```

训练结果如下：

```
Iteration: 100% 1029/1029 [00:13<00:00, 79.72it/s]
Evaluating...
Iteration: 100% 1029/1029 [00:08<00:00, 128.82it/s]
Iteration: 100% 129/129 [00:01<00:00, 98.09it/s]
Iteration: 100% 129/129 [00:01<00:00, 87.16it/s]
Epoch: 30, Loss: 0.0289, Train: 84.56%, Valid: 77.71% Test: 75.14%
```

best_model 验证和测试 ROC-AUC 分数如下：

```
Iteration: 100% 1029/1029 [00:07<00:00, 132.83it/s]
Iteration: 100% 129/129 [00:00<00:00, 135.30it/s]
Saving Model Predictions
Iteration: 100% 129/129 [00:01<00:00, 132.17it/s]
Saving Model Predictions
Best model: Train: 83.70%, Valid: 80.42% Test: 74.01%
```