金融异常检测任务作业报告

洪嘉栋 学号2246005 工程师学院

摘要

本报告详细描述了完成《金融异常检测任务》作业的过程，包括数据预处理、模型选择、模型训练与评估。通过训练`GraphSAGE`模型，本次作业在训练集上达到了78.079的准确度，在测试集上达到了0.783的`AUC`值，满足了最高分要求。

1. 数据预处理

1.1 图结构预处理

我们首先与示例代码中的处理相同，处理了边索引，并将数据进行了标准化处理，相比使用MLP模型的示例代码，我并没有改变数据处理的方式。

```
dataset_name='DGraph'
dataset = DGraphFin(root=path, name=dataset_name, transform=T.ToSparseTensor())
nlabels = dataset.num_classes
if dataset_name in ['DGraph']:
    nlabels = 2    #本实验中仅需预测类0和类1

data = dataset[0]
data.adj_t = data.adj_t.to_symmetric() #将有向图转化为无向图
data.validate()

if dataset_name in ['DGraph']:
    x = data.x
    x = (x - x.mean(0)) / x.std(0)
    data.x = x
if data.y.dim() == 2:
    data.y = data.y.squeeze(1)
```

2. 模型构建和评估

2.1 模型选择

我选择了`GraphSAGE`模型，训练速度较需要使用多头注意力机制的`GAT`模型更快，不管是训练还是推理任务上，运行时间都相较于`GAT`有优势。

```
# Model Definition
class GraphSAGE(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels, dropout):
        super(GraphSAGE, self).__init__()
        self.sageConv1 = SAGEConv(in_channels,hidden_channels)
        self.sageConv2 = SAGEConv(hidden_channels,out_channels)
        self.dropout   = dropout

    def reset_parameters(self):
        self.sageConv1.reset_parameters()
        self.sageConv2.reset_parameters()
```

```
    def forward(self,x,edge_index):
        x = self.sageConv1(x,edge_index)
        x = F.relu(x)
        x = F.dropout(x,p=self.dropout,training=self.training)
        x = self.sageConv2(x,edge_index)
        return F.log_softmax(x,dim=-1)

graphsage_parameters = {
    'lr': 1e-2
    , 'hidden_channels': 128
    , 'dropout': 0.0
    , 'weight_decay': 5e-7
                }
```

2.2 模型评估

本次作业使用了一个固定划分的训练集-验证集-测试集的结构，结果如下：

```
Epoch: 300, Loss: 0.0611, Train: 78.079, Valid: 76.221
```

主要评估手段用到了`AUC`，也就是`ROC`曲线下面积作为评测标准，相比于手动调整阈值使得模型达到高准确率，`AUC`对分类问题更有区分度。

3. 模型调优和实验

3.1 其他模型尝试

在模型选择上，我也尝试了`GAT`模型，因为`GAT`模型中引入了多头注意力机制，训练参数量比较大，训练速度较慢。在训练阶段放弃了进一步对`GAT`做优化，转向了`GraphSAGE`模型。

```
class GAT_classification(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels, heads=8,
dropout=0.6):
        super(GAT, self).__init__()
        self.conv1 = GATConv(in_channels, hidden_channels, heads=heads,
dropout=dropout)
        # On the final layer we use 1 head for concatenation
        self.conv2 = GATConv(hidden_channels * heads, out_channels, heads=1,
concat=False, dropout=dropout)
        self.dropout = dropout

    def reset_parameters(self):
        self.conv1.reset_parameters()
        self.conv2.reset_parameters()

    def forward(self, x, edge_index):
        x = F.dropout(x, p=self.dropout, training=self.training)
        x = F.elu(self.conv1(x, edge_index))
        x = F.dropout(x, p=self.dropout, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=-1)

 gat_parameters = {
```

```
    'lr': 0.001
    , 'hidden_channels': 128
    , 'heads' : 8
    , 'dropout': 0.8
    , 'weight_decay': 5e-7
                }
```

3.2 数据预处理尝试

尝试了关联原始图中的`data.edge_attr`与邻接矩阵`data.adj_t`，在处理的过程中发现`data.edge_attr`与`data.adj_t`在有向图中一一对应，但是始终无法在无向图中对应，于是我搁置了这个性质。由于在不考虑这个性质的情况下模型的性能已经非常好，于是并没对这个性质放入模型的方案做进一步探究。

同时，也探究了不使用邻接矩阵`adj_t`而是用对图结构拓扑动态性更强的`edge_index`作为网络输入，并且将节点`node_index`进行变换以满足输入要求，探究发现训练的效果并不如将邻接矩阵`adj_t`直接输入网络。

```
adj_t = data.adj_t
# 1. 从 SparseTensor 获取边索引
row, col, _ = adj_t.coo()
edge_index = torch.stack([row, col], dim=0)
print(len(edge_index[1]))
# 2. 应用训练掩码
src, dst = edge_index[0], edge_index[1]

train_mask = data.train_mask
mask = torch.isin(src, train_mask) & torch.isin(dst, train_mask)
# mask = train_mask[src] & train_mask[dst]
filtered_edge_index = edge_index[:, mask]

print(edge_index.shape)

data.x[train_idx].shape

edge_attr = data.edge_attr
print(edge_attr)
```

3.3 `molab`平台到本地平台测试优化

由于`molab`平台集成度十分有限，我在本地端根据文档环境自行搭建了docker镜像，在我自己的GPU上进行训练与推理。

```
# 使用 Python 3.10 的 slim 版本作为基础镜像
FROM python:3.10-slim

# 设置工作目录
WORKDIR /app

# 复制需求文件到容器中
COPY requirements.txt .
```

```
# 更新 pip 和安装依赖库
RUN pip install --upgrade pip && \
    pip install -r requirements.txt

# 复制应用代码到容器中（可选）
COPY . .
```

```
numpy==1.26.4
torch==2.3.1+cu118
torch-geometric==2.5.3
--extra-index-url https://download.pytorch.org/whl/cu118
```

最终提交测试的`predict`函数，我选择先将预测结果推理完成后缓存，再在提交界面直接读取缓存文件的方法减小测试的纯CPU运算负载。

```
dic={0:"正常用户",1:"欺诈用户"}
node_idx = 0
y_pred = gat_predict(data, node_idx)
print(y_pred)
print(f'节点 {node_idx} 预测对应的标签为:{torch.argmax(y_pred)}, 为
{dic[torch.argmax(y_pred).item()]}。')

node_idx = 1
y_pred = gat_predict(data, node_idx)
print(y_pred)
print(f'节点 {node_idx} 预测对应的标签为:{torch.argmax(y_pred)}, 为
{dic[torch.argmax(y_pred).item()]}。')

def gat_save_prediction(model, data):
    with torch.no_grad():
        model.eval()
        out = model(data.x, data.adj_t)
        y_pred = out.exp()

        torch.save(y_pred, save_dir + '/cached_prediction.pt')
        pass

def gat_predict_from_cache(node_id):
    cached_prediction = torch.load(save_dir + '/cached_prediction.pt')
    # out = cached_prediction["out"]
    y_pred = cached_prediction[node_id]
    return y_pred

gat_save_prediction(model, data)
gat_predict_from_cache(0)

def predict(data,node_id):
    cached_prediction = torch.load('./results/cached_prediction.pt',
map_location=torch.device('cpu'))
    # out = cached_prediction["out"]
    y_pred = cached_prediction[node_id]
    return y_pred
```

4. 结论

采用了GraphSAGE图神经网络模型，我成功构建了一个能够准确预测金融欺诈行为的的模型。我的模型在测试数据上的表现达到了最高分要求，证明了方法的有效性。