数据挖掘课程实验
实验3 数据降维与可视化
实验手册

计科210X 甘晴void 202108010XXX

主要放实验报告上实现 效果的python源码

节点表征学习代码 **GAT1.py**

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch_geometric.data import Data
from torch_geometric.nn import GATConv
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# 读取基因列表
genes_df = pd.read_csv('List_Proteins_in_SL.txt', header=None,
names=['Gene'])
genes = genes_df['Gene'].tolist()

# 读取基因关系
relations_df = pd.read_csv('SL_Human_FinalCheck.txt', sep='\t',
header=None, names=['Gene1', 'Gene2', 'Confidence'])
relations_df['Confidence'] =
relations_df['Confidence'].astype(float)

# 将基因名称映射为数字标识
genes_dict = {gene: idx for idx, gene in enumerate(genes)}
relations_df['Gene1'] = relations_df['Gene1'].map(genes_dict)
relations_df['Gene2'] = relations_df['Gene2'].map(genes_dict)
```

```
# 构建图的 Data 对象
edge_index = torch.tensor(relations_df[['Gene1',
'Gene2']].values.T.astype(np.int64), dtype=torch.long)
x = torch.ones((len(genes), 1))  # 每个基因有一个1维的特征
data = Data(x=x, edge_index=edge_index)

# 定义图自编码器模型
class GraphAutoencoder(nn.Module):
    def __init__(self, in_features, hidden_size):
        super(GraphAutoencoder, self).__init__()
        self.encoder = GATConv(in_features, hidden_size, heads=1)
        self.decoder = GATConv(hidden_size, in_features, heads=1)

    def forward(self, data):
        encoded = self.encoder(data.x, data.edge_index)
        decoded = self.decoder(encoded, data.edge_index)
        return decoded

# 初始化图自编码器模型
model = GraphAutoencoder(in_features=1, hidden_size=2)

# 定义优化器
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 无监督学习任务：重构损失
num_epochs = 80
losses = []  # 用于存储每个epoch的损失值

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    reconstructed = model(data)

    # 选择适当的无监督损失函数，比如均方误差损失
    loss = nn.MSELoss()(reconstructed, data.x)

    loss.backward()
    optimizer.step()

    losses.append(loss.item())  # 记录每个epoch的损失值
    print(f'Epoch: {epoch + 1}, Loss: {loss.item()}')
```

```
# 在测试集上测试模型
model.eval()
with torch.no_grad():
    test_output = model(data)

# 绘制损失率的曲线图
plt.plot(losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Over Time')
plt.legend()
plt.show()

# 在训练后提取节点嵌入
with torch.no_grad():
    node_embeddings = model.encoder(data.x,
data.edge_index).numpy()

# 绘制节点嵌入的散点图
plt.rcParams['font.sans-serif'] = ['SimHei']  # 替换为系统中存在的中文字
体
plt.rcParams['axes.unicode_minus'] = False # 解决设置中文后符号无法显示
的问题

"""
plt.scatter(node_embeddings[:, 0], node_embeddings[:, 1],
alpha=0.7, c='blue', marker='.')
plt.title('节点嵌入')
plt.show()
"""



# 计算节点密度
node_density = np.zeros(len(genes))
for i in range(len(genes)):
    # 计算每个节点到其他节点的欧氏距离
    distances = np.linalg.norm(node_embeddings -
node_embeddings[i], axis=1)
    # 计算距离小于某个阈值的节点数，可以根据需要调整阈值
    density = np.sum(distances < 0.0000001)
    node_density[i] = density
```

```
# 绘制节点嵌入的散点图，并使用节点密度作为颜色映射
plt.scatter(node_embeddings[:, 0], node_embeddings[:, 1],
alpha=0.7, c=node_density, cmap='viridis', marker='.')
plt.colorbar(label='Node Density')
plt.title('节点嵌入（根据节点密度着色）')
plt.show()
```

关联关系预测代码 **GAT2.py**

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch_geometric.data import Data
from torch_geometric.nn import GATConv
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# 读取基因列表
genes_df = pd.read_csv('List_Proteins_in_SL.txt', header=None,
names=['Gene'])
genes = genes_df['Gene'].tolist()

# 读取基因关系
relations_df = pd.read_csv('SL_Human_FinalCheck.txt', sep='\t',
header=None, names=['Gene1', 'Gene2', 'Confidence'])
relations_df['Confidence'] =
relations_df['Confidence'].astype(float)

# 将基因名称映射为数字标识
genes_dict = {gene: idx for idx, gene in enumerate(genes)}
relations_df['Gene1'] = relations_df['Gene1'].map(genes_dict)
relations_df['Gene2'] = relations_df['Gene2'].map(genes_dict)

# 移除带有非数字标识的行
relations_df = relations_df.dropna(subset=['Gene1', 'Gene2',
'Confidence'])

# 构建图的 Data 对象
```

```
edge_index = torch.tensor(relations_df[['Gene1',
'Gene2']].values.T.astype(np.int64), dtype=torch.long)
x = torch.ones((len(genes), 1))  # 每个基因有一个1维的特征
y = torch.tensor(relations_df['Confidence'].values,
dtype=torch.float).view(-1, 1)  # 置信分数作为目标

data = Data(x=x, edge_index=edge_index, y=y)

# 定义GAT模型
class GATModel(nn.Module):
    def __init__(self, in_features, out_features, num_heads):
        super(GATModel, self).__init__()
        self.conv1 = GATConv(in_features, out_features,
heads=num_heads)

    def forward(self, data):
        x = self.conv1(data.x, data.edge_index)
        return x

# 初始化模型
model = GATModel(in_features=1, out_features=1, num_heads=1)

# 定义优化器
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 回归任务的损失函数
loss_function = nn.MSELoss()

# 划分训练集和测试集
train_mask, test_mask = train_test_split(range(len(genes)),
test_size=0.2, random_state=42)

# 训练模型（回归任务）
num_epochs = 200
losses = []  # 用于存储每个epoch的损失值

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    output = model(data)

    # 使用均方误差损失
```

```
    loss = loss_function(output[train_mask], data.y[train_mask])
    loss.backward()
    optimizer.step()

    losses.append(loss.item())  # 记录每个epoch的损失值
    print(f'Epoch: {epoch + 1}, Loss: {loss.item()}')

# 在测试集上测试模型
model.eval()
with torch.no_grad():
    test_output = model(data)


# 绘制损失率的曲线图
plt.plot(losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Over Time')
plt.legend()
plt.show()


# 评估模型性能
# 在测试集上计算性能指标
from sklearn.metrics import mean_squared_error, mean_absolute_error

model.eval()
with torch.no_grad():
    test_output = model(data)

# 将tensor转换为numpy数组
test_output_np = test_output.cpu().numpy()
y_true_np = data.y[test_mask].cpu().numpy()

# 将test_output_np和y_true_np调整为相同的样本数
test_output_np = test_output[test_mask].cpu().numpy()

# 计算均方根误差和平均绝对误差
rmse = np.sqrt(mean_squared_error(y_true_np, test_output_np))
mae = mean_absolute_error(y_true_np, test_output_np)

print(f'Root Mean Squared Error (RMSE): {rmse}')
```

```
print(f'Mean Absolute Error (MAE): {mae}')


# 散点图（Scatter Plot）
plt.scatter(y_true_np, test_output_np)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Scatter Plot of Actual vs Predicted Values')
plt.show()


# 残差图（Residual Plot）
residuals = y_true_np - test_output_np
plt.scatter(y_true_np, residuals)
plt.xlabel('Actual Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--', linewidth=2)  # 添加水平
线表示残差为零
plt.title('Residual Plot')
plt.show()
```

多关系图的联合学习代码 **GAT3.py**

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from torch_geometric.data import Data
from torch_geometric.nn import GATConv
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import torch.nn.functional as F

# 读取基因列表
genes_df = pd.read_csv('List_Proteins_in_SL.txt', header=None,
names=['Gene'])
genes = genes_df['Gene'].tolist()

# 读取基因关系
```

```
relations_df = pd.read_csv('SL_Human_FinalCheck.txt', sep='\t',
header=None, names=['Gene1', 'Gene2', 'Confidence'])
relations_df['Confidence'] =
relations_df['Confidence'].astype(float)

# 将基因名称映射为数字标识
genes_dict = {gene: idx for idx, gene in enumerate(genes)}
relations_df['Gene1'] = relations_df['Gene1'].map(genes_dict)
relations_df['Gene2'] = relations_df['Gene2'].map(genes_dict)

# 构建图的 Data 对象
edge_index = torch.tensor(relations_df[['Gene1',
'Gene2']].values.T.astype(np.int64), dtype=torch.long)
x = torch.ones((len(genes), 1))  # 每个基因有一个1维的特征
data = Data(x=x, edge_index=edge_index)

# 读取额外的基因特征
feature_ppi = pd.read_csv('feature_ppi_128.txt', header=None,
sep='\s+')
feature_go = pd.read_csv('feature_go_128.txt', header=None,
sep='\s+')

# 将额外的特征合并到现有数据中
data.x = torch.cat([data.x, torch.tensor(feature_ppi.values,
dtype=torch.float32), torch.tensor(feature_go.values,
dtype=torch.float32)], dim=1)

# 修改 GATConv 层为 MultiHeadGATConv
class MultiHeadGATConv(nn.Module):
    def __init__(self, in_features, out_features, heads):
        super(MultiHeadGATConv, self).__init__()
        self.heads = heads
        self.attention_heads = nn.ModuleList([GATConv(in_features,
out_features, heads=1) for _ in range(heads)])
        self.out_linear = nn.Linear(heads * out_features,
out_features)  # 添加线性层

    def forward(self, x, edge_index):
        # 各个注意力头的输出
        head_outputs = [attention_head(x, edge_index) for
attention_head in self.attention_heads]
        # 拼接所有注意力头的输出
```

```
        x = torch.cat(head_outputs, dim=1)
        # 使用线性层调整输出维度
        x = F.relu(self.out_linear(x))
        return x


# 修改 GraphAutoencoder 类
class GraphAutoencoder(nn.Module):
    def __init__(self, in_features, hidden_size, heads):
        super(GraphAutoencoder, self).__init__()
        self.encoder = MultiHeadGATConv(in_features, hidden_size,
heads=heads)
        self.decoder = MultiHeadGATConv(hidden_size, in_features,
heads=heads)

    def forward(self, data):
        encoded = self.encoder(data.x, data.edge_index)
        decoded = self.decoder(encoded, data.edge_index)
        return decoded

# 初始化图自编码器模型
model = GraphAutoencoder(in_features=257, hidden_size=64, heads=2)

# 定义优化器
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 无监督学习任务：重构损失
num_epochs = 60
losses = []  # 用于存储每个epoch的损失值

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    reconstructed = model(data)

    # 选择适当的无监督损失函数，比如均方误差损失
    loss = nn.MSELoss()(reconstructed, data.x)

    loss.backward()
    optimizer.step()

    losses.append(loss.item())  # 记录每个epoch的损失值
```

```
    print(f'Epoch: {epoch + 1}, Loss: {loss.item()}')

# 在测试集上测试模型
model.eval()
with torch.no_grad():
    test_output = model(data)

# 绘制损失率的曲线图
plt.plot(losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Over Time')
plt.legend()
plt.show()

# 在训练后提取节点嵌入
with torch.no_grad():
    node_embeddings = model.encoder(data.x,
data.edge_index).numpy()

plt.rcParams['font.sans-serif'] = ['SimHei']  # 替换为系统中存在的中文字体
plt.rcParams['axes.unicode_minus'] = False # 解决设置中文后符号无法显示的问题

# 计算节点密度
node_density = np.zeros(len(genes))
for i in range(len(genes)):
    # 计算每个节点到其他节点的欧氏距离
    distances = np.linalg.norm(node_embeddings -
node_embeddings[i], axis=1)
    # 计算距离小于某个阈值的节点数，可以根据需要调整阈值
    density = np.sum(distances < 0.0000001)
    node_density[i] = density

# 绘制节点嵌入的散点图，并使用节点密度作为颜色映射
plt.scatter(node_embeddings[:, 0], node_embeddings[:, 1],
alpha=0.7, c=node_density, cmap='viridis', marker='.')
plt.colorbar(label='Node Density')
plt.title('节点嵌入（根据节点密度着色）')
plt.show()
```