数据挖掘课程实验
实验4 链接预测
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

使用DGL库进行探索 dgl.py

```
import dgl
import torch
import numpy as np

# 读取基因列表
with open('GeneList.txt', 'r') as f:
    gene_list = [line.strip() for line in f]
# 构建基因到索引的映射
gene_dict = {gene: idx for idx, gene in enumerate(gene_list)}

# 读取基因关系和置信分数
with open('Positive_LinkSL.txt', 'r') as f:
    edges = [line.strip().split() for line in f]
# 提取基因关系的源节点、目标节点和置信分数
src_nodes = [gene_dict[edge[0]] for edge in edges] + [gene_dict[edge[1]] for edge in edges]
dst_nodes = [gene_dict[edge[1]] for edge in edges] + [gene_dict[edge[0]] for edge in edges]
confidence_scores = [float(edge[2]) for edge in edges] + [float(edge[2]) for edge in edges]

# 读取特征
with open('feature1_go.txt', 'r') as file:
    feature1_go = np.array([list(map(float, line.split())) for line in file])
with open('feature2_ppi.txt', 'r') as file:
```

```python
    feature2_ppi = np.array([list(map(float, line.split())) for line in file])

# 构建图
edges = torch.tensor(src_nodes),torch.tensor(dst_nodes)
graph = dgl.graph(edges)
graph.edata['confidence'] = torch.tensor(confidence_scores,dtype=torch.float32)
graph.ndata['feature1_go'] = torch.tensor(feature1_go,dtype=torch.float32)
graph.ndata['feature2_ppi'] = torch.tensor(feature2_ppi,dtype=torch.float32)

"""print(graph)
# 输出边的权值值
edge_weights = graph.edata['confidence'].squeeze().numpy()
print("Edge Weights:")
print(edge_weights)
# 输出节点特征 'feature1_go'
feature1_go_values = graph.ndata['feature1_go'].squeeze().numpy()
print("Node Feature 'feature1_go':")
print(feature1_go_values)
# 输出节点特征 'feature2_ppi'
feature2_ppi_values = graph.ndata['feature2_ppi'].squeeze().numpy()
print("Node Feature 'feature2_ppi':")
print(feature2_ppi_values)"""

print(graph)


# 构建一个2层的GNN模型
import dgl.nn as dglnn
import torch.nn as nn
import torch.nn.functional as F
class SAGE(nn.Module):
    def __init__(self, in_feats, hid_feats, out_feats):
        super().__init__()
        # 实例化SAGEConve，in_feats是输入特征的维度，out_feats是输出特征的维度，aggregator_type是聚合函数的类型
        self.conv1 = dglnn.SAGEConv(
```

```
            in_feats=in_feats, out_feats=hid_feats, aggregator_type='mean')
        self.conv2 = dglnn.SAGEConv(
            in_feats=hid_feats, out_feats=out_feats, aggregator_type='mean')

    def forward(self, graph, inputs):
        # 输入是节点的特征
        h = self.conv1(graph, inputs)
        h = F.relu(h)
        h = self.conv2(graph, h)
        return h

def construct_negative_graph(graph, k):
    src, dst = graph.edges()

    neg_src = src.repeat_interleave(k)
    neg_dst = torch.randint(0, graph.num_nodes(), (len(src) * k,))
    return dgl.graph((neg_src, neg_dst), num_nodes=graph.num_nodes())

import dgl.function as fn
class DotProductPredictor(nn.Module):
    def forward(self, graph, h):
        # h是从5.1节的GNN模型中计算出的节点表示
        with graph.local_scope():
            graph.ndata['h'] = h
            graph.apply_edges(fn.u_dot_v('h', 'h', 'score'))
            return graph.edata['score']

def compute_loss(pos_score, neg_score):
    # 间隔损失
    n_edges = pos_score.shape[0]
    return (1 - pos_score.unsqueeze(1) + neg_score.view(n_edges, -1)).clamp(min=0).mean()

class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features):
        super().__init__()
```

```
        self.sage = SAGE(in_features, hidden_features, out_features)
        self.pred = DotProductPredictor()
    def forward(self, g, neg_g, x):
        h = self.sage(g, x)
        #return self.pred(g, h), self.pred(neg_g, h)
        pos_score = self.pred(g, h)
        neg_score = self.pred(neg_g, h)
        return pos_score, neg_score

node_features = graph.ndata['feature1_go']
n_features = node_features.shape[1]
k = 1
model = Model(n_features, 10, 5)
opt = torch.optim.Adam(model.parameters())
for epoch in range(1):
    negative_graph = construct_negative_graph(graph, k)
    pos_score, neg_score = model(graph, negative_graph, node_features)
    loss = compute_loss(pos_score, neg_score)
    opt.zero_grad()
    loss.backward()
    opt.step()
    print(f'Epoch {epoch + 1}, Loss: {loss.item()}')
```

任务1 图卷积网络 test1.py

```
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.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, average_precision_score, roc_curve, auc
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# 读取数据
def read_data(file_path):
```

```python
    with open(file_path, 'r') as f:
        data = f.read().splitlines()
    return data

# 构建图数据
def build_graph_data(gene_list, link_list, feature1, feature2):
    edge_index = []
    edge_attr = []
    x1 = []
    x2 = []

    gene_dict = {gene: idx for idx, gene in enumerate(gene_list)}

    for link in link_list:
        gene1, gene2, confidence = link.split('\t')
        if gene1 in gene_dict and gene2 in gene_dict:
            edge_index.append([gene_dict[gene1], gene_dict[gene2]])
            edge_attr.append(float(confidence))

    edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()
    edge_attr = torch.tensor(edge_attr, dtype=torch.float).view(-1, 1)

    for gene in gene_list:
        if gene in gene_dict:
            x1.append(feature1[gene_dict[gene]])
            x2.append(feature2[gene_dict[gene]])

    x1 = torch.tensor(x1, dtype=torch.float)
    x2 = torch.tensor(x2, dtype=torch.float)

    data = Data(x1=x1, x2=x2, edge_index=edge_index, edge_attr=edge_attr)
    return data

# GAT 模型定义
class GATModel(nn.Module):
    def __init__(self, in_channels, out_channels, heads):
        super(GATModel, self).__init__()
```

```python
        self.conv1 = GATConv(in_channels, out_channels, heads=heads)

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

# 训练模型
def train(model, data, optimizer, criterion, epochs):
    model.train()
    losses = []  # 用于记录每个 epoch 的损失值
    for epoch in range(epochs):
        optimizer.zero_grad()
        out = model(data.x1, data.edge_index, data.edge_attr)
        loss = criterion(out, data.x2)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())  # 记录当前 epoch 的损失值
        print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss.item()}')

    # 绘制损失曲线图
    plt.plot(losses)
    plt.title('Training Loss Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()


# 评估链接预测结果
def evaluate(y_true, y_pred):
    y_true = (y_true > 0.5).int().cpu().numpy()
    y_pred = (y_pred > 0.5).int().cpu().numpy()

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='micro')
    recall = recall_score(y_true, y_pred, average='micro')
    f1 = f1_score(y_true, y_pred, average='micro')
    roc_auc = roc_auc_score(y_true, y_pred)
    aupr = average_precision_score(y_true, y_pred)

```

```
    return accuracy, precision, recall, f1, roc_auc, aupr

# 读取数据
gene_list = read_data('GeneList.txt')
link_list = read_data('Positive_LinkSL.txt')
feature1 = np.loadtxt('feature1_go.txt')
feature2 = np.loadtxt('feature2_ppi.txt')

# 划分数据集和测试集
train_gene_list, test_gene_list = train_test_split(gene_list,
test_size=0.2, random_state=42)

# 构建训练集和测试集的图数据
train_data = build_graph_data(train_gene_list, link_list,
feature1, feature2)
test_data = build_graph_data(test_gene_list, link_list,
feature1, feature2)

# 创建并训练 GAT 模型
model = GATModel(in_channels=128, out_channels=128, heads=1)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.MSELoss()

train(model, train_data, optimizer, criterion, epochs=200)

# 进行链接预测
pred_scores = model(test_data.x1, test_data.edge_index,
test_data.edge_attr)

# 评估链接预测结果
accuracy, precision, recall, f1, roc_auc, aupr =
evaluate(test_data.x2, pred_scores)
print(f'Accuracy: {accuracy} \nPrecision: {precision} \nRecall:
{recall} \nF1 Score: {f1}')
print(f'ROC AUC: {roc_auc} \nAUPR: {aupr}')




import networkx as nx
import torch
from torch_geometric.data import Data
```

```python


# 将 PyTorch Geometric 图数据转换为 NetworkX 图
G = nx.Graph()
G.add_nodes_from(range(test_data.num_nodes))
G.add_edges_from(test_data.edge_index.t().tolist())

# 使用 NetworkX 绘制图
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold',
node_color='lightblue', node_size=1000, font_size=8,
edge_color='gray')
plt.show()

```

任务2 多通道图卷积网络 **test2.py**

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, average_precision_score,
roc_curve, auc
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# 读取数据
def read_data(file_path):
    with open(file_path, 'r') as f:
        data = f.read().splitlines()
    return data

# 构建图数据
def build_graph_data(gene_list, link_list, feature1, feature2):
    edge_index = []
    edge_attr = []
    x1 = []
    x2 = []
```

```

    gene_dict = {gene: idx for idx, gene in enumerate(gene_list)}

    for link in link_list:
        gene1, gene2, confidence = link.split('\t')
        if gene1 in gene_dict and gene2 in gene_dict:
            edge_index.append([gene_dict[gene1], gene_dict[gene2]])
            edge_attr.append(float(confidence))

    edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()
    edge_attr = torch.tensor(edge_attr, dtype=torch.float).view(-1, 1)

    for gene in gene_list:
        if gene in gene_dict:
            x1.append(feature1[gene_dict[gene]])
            x2.append(feature2[gene_dict[gene]])

    x1 = torch.tensor(x1, dtype=torch.float)
    x2 = torch.tensor(x2, dtype=torch.float)

    data = Data(x1=x1, x2=x2, edge_index=edge_index, edge_attr=edge_attr)
    return data

# Multi-Channel Graph Convolutional Network 模型定义
class MultiChannelGCN(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(MultiChannelGCN, self).__init__()
        self.conv1 = GCNConv(in_channels, out_channels)
        self.conv2 = GCNConv(in_channels, out_channels)

    def forward(self, x1, x2, edge_index, edge_attr):
        x1 = self.conv1(x1, edge_index, edge_attr)
        x2 = self.conv2(x2, edge_index, edge_attr)
        return x1, x2

# 训练模型
def train(model, data, optimizer, criterion, epochs):
```

```python
    model.train()
    losses = []  # 用于记录每个 epoch 的损失值
    for epoch in range(epochs):
        optimizer.zero_grad()
        out1, out2 = model(data.x1, data.x2, data.edge_index, data.edge_attr)
        loss1 = criterion(out1, data.x1)
        loss2 = criterion(out2, data.x2)
        loss = loss1 + loss2
        loss.backward()
        optimizer.step()
        losses.append(loss.item())  # 记录当前 epoch 的损失值
        print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss.item()}')

    # 绘制损失曲线图
    plt.plot(losses)
    plt.title('Training Loss Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()

# 评估链接预测结果
def evaluate(y_true, y_pred):
    y_true = (y_true > 0.3).int().cpu().numpy()
    y_pred = (y_pred > 0.3).int().cpu().numpy()

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='micro')
    recall = recall_score(y_true, y_pred, average='micro')
    f1 = f1_score(y_true, y_pred, average='micro')
    roc_auc = roc_auc_score(y_true, y_pred)
    aupr = average_precision_score(y_true, y_pred)

    return accuracy, precision, recall, f1, roc_auc, aupr

# 读取数据
gene_list = read_data('GeneList.txt')
link_list = read_data('Positive_LinkSL.txt')
feature1 = np.loadtxt('feature1_go.txt')
feature2 = np.loadtxt('feature2_ppi.txt')
```

```python

# 划分数据集和测试集
train_gene_list, test_gene_list = train_test_split(gene_list, test_size=0.2, random_state=42)

# 构建训练集和测试集的图数据
train_data = build_graph_data(train_gene_list, link_list, feature1, feature2)
test_data = build_graph_data(test_gene_list, link_list, feature1, feature2)

# 创建并训练 Multi-Channel GCN 模型
model = MultiChannelGCN(in_channels=128, out_channels=128)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.MSELoss()

train(model, train_data, optimizer, criterion, epochs=200)

# 进行链接预测
pred_scores1, pred_scores2 = model(test_data.x1, test_data.x2, test_data.edge_index, test_data.edge_attr)
pred_scores = (pred_scores1 + pred_scores2) / 2  # 取两个通道的平均值

# 评估链接预测结果
accuracy, precision, recall, f1, roc_auc, aupr = evaluate(test_data.x2, pred_scores)
print(f'Accuracy: {accuracy} \nPrecision: {precision} \nRecall: {recall} \nF1 Score: {f1}')
print(f'ROC AUC: {roc_auc} \nAUPR: {aupr}')



import networkx as nx
import torch
from torch_geometric.data import Data


# 将 PyTorch Geometric 图数据转换为 NetworkX 图
G = nx.Graph()
G.add_nodes_from(range(test_data.num_nodes))
G.add_edges_from(test_data.edge_index.t().tolist())
```

```python

# 使用 NetworkX 绘制图
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_color='lightblue', node_size=1000, font_size=8, edge_color='gray')
plt.show()

```

任务2 n通道图卷积网络 test2.2.py

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, \
    average_precision_score
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt


# 读取数据
def read_data(file_path):
    with open(file_path, 'r') as f:
        data = f.read().splitlines()
    return data


# 构建图数据
def build_graph_data(gene_list, link_list, feature1, feature2):
    edge_index = []
    edge_attr = []
    x1 = []
    x2 = []

    gene_dict = {gene: idx for idx, gene in enumerate(gene_list)}

```

```python
    for link in link_list:
        gene1, gene2, confidence = link.split('\t')
        if gene1 in gene_dict and gene2 in gene_dict:
            edge_index.append([gene_dict[gene1], gene_dict[gene2]])
            edge_attr.append(float(confidence))

    edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()
    edge_attr = torch.tensor(edge_attr, dtype=torch.float).view(-1, 1)

    for gene in gene_list:
        if gene in gene_dict:
            x1.append(feature1[gene_dict[gene]])
            x2.append(feature2[gene_dict[gene]])

    x1 = torch.tensor(x1, dtype=torch.float)
    x2 = torch.tensor(x2, dtype=torch.float)

    data = Data(x1=x1, x2=x2, edge_index=edge_index, edge_attr=edge_attr)
    return data


# Multi-Channel Graph Convolutional Network 模型定义
class MultiChannelGCN(nn.Module):
    def __init__(self, in_channels, out_channels, num_channels):
        super(MultiChannelGCN, self).__init__()
        self.channels = nn.ModuleList([GCNConv(in_channels, out_channels) for _ in range(num_channels)])

    def forward(self, *inputs):
        output_channels = [channel(x, inputs[-2], inputs[-1]) for channel, x in zip(self.channels, inputs[:-2])]
        return output_channels


# 训练模型
def train(model, data, optimizer, criterion, epochs):
    model.train()
```

```
    losses = []  # 用于记录每个 epoch 的损失值
    for epoch in range(epochs):
        optimizer.zero_grad()
        output_channels = model(data.x1, data.x2, data.edge_index, data.edge_attr)

        # Assuming that data.x1 and data.x2 are the target values for each channel
        loss = sum(criterion(output, data.x1 if i == 0 else data.x2) for i, output in enumerate(output_channels))

        loss.backward()
        optimizer.step()
        losses.append(loss.item())  # 记录当前 epoch 的损失值
        print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss.item()}')

    # 绘制损失曲线图
    plt.plot(losses)
    plt.title('Training Loss Over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()


# 评估链接预测结果
def evaluate(y_true, y_pred):
    y_true = (y_true > 0.3).int().cpu().numpy()
    y_pred = (y_pred > 0.3).int().cpu().numpy()

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='micro')
    recall = recall_score(y_true, y_pred, average='micro')
    f1 = f1_score(y_true, y_pred, average='micro')
    roc_auc = roc_auc_score(y_true, y_pred)
    aupr = average_precision_score(y_true, y_pred)

    return accuracy, precision, recall, f1, roc_auc, aupr


# 读取数据
```

```python
gene_list = read_data('GeneList.txt')
link_list = read_data('Positive_LinkSL.txt')
feature1 = np.loadtxt('feature1_go.txt')
feature2 = np.loadtxt('feature2_ppi.txt')

# 划分数据集和测试集
train_gene_list, test_gene_list = train_test_split(gene_list, test_size=0.2, random_state=42)

# 构建训练集和测试集的图数据
train_data = build_graph_data(train_gene_list, link_list, feature1, feature2)
test_data = build_graph_data(test_gene_list, link_list, feature1, feature2)

# 创建并训练 Multi-Channel GCN 模型
num_channels = 150 # Set the number of channels (adjust as needed)
model = MultiChannelGCN(in_channels=128, out_channels=128, num_channels=num_channels)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.MSELoss()

train(model, train_data, optimizer, criterion, epochs=200)

# 进行链接预测
pred_scores_list = model(test_data.x1, test_data.x2, test_data.edge_index, test_data.edge_attr)
pred_scores = torch.stack(pred_scores_list).mean(dim=0)  # Take the mean across channels

# 评估链接预测结果
accuracy, precision, recall, f1, roc_auc, aupr = evaluate(test_data.x2, pred_scores)
print(f'Accuracy: {accuracy} \nPrecision: {precision} \nRecall: {recall} \nF1 Score: {f1}')
print(f'ROC AUC: {roc_auc} \nAUPR: {aupr}')

import networkx as nx
import torch
from torch_geometric.data import Data

```

```python
# 将 PyTorch Geometric 图数据转换为 NetworkX 图
G = nx.Graph()
G.add_nodes_from(range(test_data.num_nodes))
G.add_edges_from(test_data.edge_index.t().tolist())

# 使用 NetworkX 绘制图
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold', node_color='lightblue', node_size=1000, font_size=8,
        edge_color='gray')
plt.show()

```