import numpy as np  
import networkx as nx  
from sklearn.preprocessing import MinMaxScaler  
from node2vec import Node2Vec  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
from torch\_geometric.data import Data  
from torch\_geometric.nn import GCNConv  
from tqdm import tqdm  
from scipy.io import mmread  
  
# ======================== 数据准备 ========================  
def load\_dolphin\_network():  
 G = nx.read\_gml("dolphins.gml")  
 # 将字符串节点名映射为整数  
 node\_mapping = {node: idx for idx, node in enumerate(G.nodes())}  
 G = nx.relabel\_nodes(G, node\_mapping) # 重命名节点  
 adj = nx.to\_numpy\_array(G)  
 return G, adj  
  
  
# 示例：加载其他网络（此处以空手道俱乐部网络为例）  
def load\_karate\_club():  
 G = nx.karate\_club\_graph()  
 node\_mapping = {node: idx for idx, node in enumerate(G.nodes())}  
 G = nx.relabel\_nodes(G, node\_mapping) # 重命名节点  
 adj = nx.to\_numpy\_array(G)  
 return G, adj  
  
  
# ===================== Node2Vec嵌入 ======================  
def generate\_node2vec\_embeddings(G, dimensions=50):  
 *"""生成Node2Vec嵌入向量"""* node2vec = Node2Vec(G, dimensions=dimensions, walk\_length=80, num\_walks=10)  
 model = node2vec.fit(window=10, min\_count=1)  
 embeddings = np.array([model.wv[str(node)] for node in G.nodes()])  
 return embeddings  
  
  
# ===================== VGAE模型定义 ======================  
class VGAE(nn.Module):  
 *"""变分图自编码器"""* def \_\_init\_\_(self, in\_dim, hidden\_dim, out\_dim):  
 super(VGAE, self).\_\_init\_\_()  
 # 编码器  
 self.gcn1 = GCNConv(in\_dim, hidden\_dim)  
 self.gcn\_mu = GCNConv(hidden\_dim, out\_dim)  
 self.gcn\_logvar = GCNConv(hidden\_dim, out\_dim)  
  
 def encode(self, x, edge\_index):  
 h = F.relu(self.gcn1(x, edge\_index))  
 mu = self.gcn\_mu(h, edge\_index)  
 logvar = self.gcn\_logvar(h, edge\_index)  
 return mu, logvar  
  
 def reparameterize(self, mu, logvar):  
 if self.training:  
 std = torch.exp(0.5 \* logvar)  
 eps = torch.randn\_like(std)  
 return mu + eps \* std  
 else:  
 return mu  
  
 def forward(self, x, edge\_index):  
 mu, logvar = self.encode(x, edge\_index)  
 z = self.reparameterize(mu, logvar)  
 return z, mu, logvar  
  
  
# ====================== 节点重要性计算 ======================  
def calculate\_weight(z):  
 *"""将潜在特征映射为权重"""* global\_context = torch.mean(z, dim=0)  
 weights = torch.sigmoid(torch.tanh(global\_context) @ z.T)  
 return weights.detach().numpy()  
  
  
# 示例：改进的度中心性计算  
def improved\_degree\_centrality(G, weights):  
 dc = nx.degree\_centrality(G)  
 nodes = list(G.nodes())  
 improved\_dc = {node: dc[node] \* weights[i] for i, node in enumerate(nodes)}  
 return improved\_dc  
  
  
# ====================== 评估指标 ========================  
def SIR\_simulation(G, source, beta, gamma=1.0, iterations=100):  
 *"""SIR传播模型模拟"""* infected = set([source])  
 recovered = set()  
 for \_ in range(iterations):  
 new\_infected = set()  
 for node in infected:  
 neighbors = list(G.neighbors(node))  
 for neighbor in neighbors:  
 if neighbor not in infected and neighbor not in recovered:  
 if np.random.rand() < beta:  
 new\_infected.add(neighbor)  
 recovered.update(infected)  
 infected = new\_infected  
 return len(recovered)  
  
  
# ====================== 主流程 ========================  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 加载数据  
 dolphin\_G, dolphin\_adj = load\_dolphin\_network()  
  
 # 生成Node2Vec嵌入  
 embeddings = generate\_node2vec\_embeddings(dolphin\_G)  
  
 # 转换为PyG数据格式  
 edge\_index = torch.tensor(list(dolphin\_G.edges()), dtype=torch.long).t().contiguous()  
 data = Data(x=torch.tensor(embeddings, dtype=torch.float),  
 edge\_index=edge\_index)  
  
 # 初始化并训练VGAE  
 model = VGAE(in\_dim=50, hidden\_dim=32, out\_dim=16)  
 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)  
  
 for epoch in tqdm(range(100)):  
 model.train()  
 z, mu, logvar = model(data.x, data.edge\_index)  
  
 # 重构损失  
 recon\_loss = F.mse\_loss(z @ z.T, torch.tensor(dolphin\_adj, dtype=torch.float))  
  
 # KL散度  
 kl\_loss = -0.5 \* torch.mean(torch.sum(1 + logvar - mu.pow(2) - logvar.exp(), dim=1))  
  
 loss = recon\_loss + kl\_loss  
 optimizer.zero\_grad()  
 loss.backward()  
 optimizer.step()  
  
 # 提取节点权重  
 with torch.no\_grad():  
 z, \_, \_ = model(data.x, data.edge\_index)  
 weights = calculate\_weight(z)  
  
 # 计算改进后的节点重要性  
 improved\_dc = improved\_degree\_centrality(dolphin\_G, weights)  
  
 # 与其他网络比较（示例：空手道俱乐部网络）  
 karate\_G, \_ = load\_karate\_club()  
 # 重复上述流程计算改进前后的指标...  
  
 # 结果可视化与比较  
 print("海豚网络改进度中心性示例:", sorted(improved\_dc.items(), key=lambda x: x[1], reverse=True)[:5])