

# 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**

```
1  import dgl
2  import torch
3  import numpy as np
4
5  # 读取基因列表
6  with open('GeneList.txt', 'r') as f:
7      gene_list = [line.strip() for line in f]
8  # 构建基因到索引的映射
9  gene_dict = {gene: idx for idx, gene in enumerate(gene_list)}
10
11 # 读取基因关系和置信分数
12 with open('Positive_LinkSL.txt', 'r') as f:
13     edges = [line.strip().split() for line in f]
14 # 提取基因关系的源节点、目标节点和置信分数
15 src_nodes = [gene_dict[edge[0]] for edge in edges] +
16               [gene_dict[edge[1]] for edge in edges]
17 dst_nodes = [gene_dict[edge[1]] for edge in edges] +
18               [gene_dict[edge[0]] for edge in edges]
19 confidence_scores = [float(edge[2]) for edge in edges] +
20                      [float(edge[2]) for edge in edges]
21
22 # 读取特征
23 with open('feature1_go.txt', 'r') as file:
24     feature1_go = np.array([list(map(float, line.split())) for
25                             line in file])
26 with open('feature2_ppi.txt', 'r') as file:
```

```

23     feature2_ppi = np.array([list(map(float, line.split())) for
    line in file])
24
25 # 构建图
26 edges = torch.tensor(src_nodes), torch.tensor(dst_nodes)
27 graph = dgl.graph(edges)
28 graph.edata['confidence'] =
    torch.tensor(confidence_scores, dtype=torch.float32)
29 graph.ndata['feature1_go'] =
    torch.tensor(feature1_go, dtype=torch.float32)
30 graph.ndata['feature2_ppi'] =
    torch.tensor(feature2_ppi, dtype=torch.float32)
31
32 """print(graph)
33 # 输出边的权值值
34 edge_weights = graph.edata['confidence'].squeeze().numpy()
35 print("Edge weights:")
36 print(edge_weights)
37 # 输出节点特征 'feature1_go'
38 feature1_go_values =
    graph.ndata['feature1_go'].squeeze().numpy()
39 print("Node Feature 'feature1_go':")
40 print(feature1_go_values)
41 # 输出节点特征 'feature2_ppi'
42 feature2_ppi_values =
    graph.ndata['feature2_ppi'].squeeze().numpy()
43 print("Node Feature 'feature2_ppi':")
44 print(feature2_ppi_values)"""
45
46 print(graph)
47
48
49 # 构建一个2层的GNN模型
50 import dgl.nn as dglnn
51 import torch.nn as nn
52 import torch.nn.functional as F
53 class SAGE(nn.Module):
54     def __init__(self, in_feats, hid_feats, out_feats):
55         super().__init__()
56         # 实例化SAGEConv, 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
57         self.conv1 = dglnn.SAGEConv(

```

```

58         in_feats=in_feats, out_feats=hid_feats,
aggregator_type='mean')
59         self.conv2 = dglnn.SAGEConv(
60             in_feats=hid_feats, out_feats=out_feats,
aggregator_type='mean')
61
62     def forward(self, graph, inputs):
63         # 输入是节点的特征
64         h = self.conv1(graph, inputs)
65         h = F.relu(h)
66         h = self.conv2(graph, h)
67         return h
68
69 def construct_negative_graph(graph, k):
70     src, dst = graph.edges()
71
72     neg_src = src.repeat_interleave(k)
73     neg_dst = torch.randint(0, graph.num_nodes(), (len(src) *
k,))
74     return dglnn.graph((neg_src, neg_dst),
num_nodes=graph.num_nodes())
75
76 import dglnn.function as fn
77 class DotProductPredictor(nn.Module):
78     def forward(self, graph, h):
79         # h是从5.1节的GNN模型中计算出的节点表示
80         with graph.local_scope():
81             graph.ndata['h'] = h
82             graph.apply_edges(fn.u_dot_v('h', 'h', 'score'))
83             return graph.edata['score']
84
85 def compute_loss(pos_score, neg_score):
86     # 间隔损失
87     n_edges = pos_score.shape[0]
88     return (1 - pos_score.unsqueeze(1) +
neg_score.view(n_edges, -1)).clamp(min=0).mean()
89
90 class Model(nn.Module):
91     def __init__(self, in_features, hidden_features,
out_features):
92         super().__init__()

```

```

93         self.sage = SAGE(in_features, hidden_features,
out_features)
94         self.pred = DotProductPredictor()
95         def forward(self, g, neg_g, x):
96             h = self.sage(g, x)
97             #return self.pred(g, h), self.pred(neg_g, h)
98             pos_score = self.pred(g, h)
99             neg_score = self.pred(neg_g, h)
100             return pos_score, neg_score
101
102 node_features = graph.ndata['feature1_go']
103 n_features = node_features.shape[1]
104 k = 1
105 model = Model(n_features, 10, 5)
106 opt = torch.optim.Adam(model.parameters())
107 for epoch in range(1):
108     negative_graph = construct_negative_graph(graph, k)
109     pos_score, neg_score = model(graph, negative_graph,
node_features)
110     loss = compute_loss(pos_score, neg_score)
111     opt.zero_grad()
112     loss.backward()
113     opt.step()
114     print(f'Epoch {epoch + 1}, Loss: {loss.item()}')

```

## 任务1 图卷积网络 test1.py

```

1  import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  from torch_geometric.data import Data
5  from torch_geometric.nn import GATConv
6  from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, average_precision_score,
roc_curve, auc
7  from sklearn.model_selection import train_test_split
8  import numpy as np
9  import matplotlib.pyplot as plt
10
11 # 读取数据
12 def read_data(file_path):

```

```

13     with open(file_path, 'r') as f:
14         data = f.read().splitlines()
15     return data
16
17 # 构建图数据
18 def build_graph_data(gene_list, link_list, feature1, feature2):
19     edge_index = []
20     edge_attr = []
21     x1 = []
22     x2 = []
23
24     gene_dict = {gene: idx for idx, gene in
25 enumerate(gene_list)}
26
27     for link in link_list:
28         gene1, gene2, confidence = link.split('\t')
29         if gene1 in gene_dict and gene2 in gene_dict:
30             edge_index.append([gene_dict[gene1],
31 gene_dict[gene2]])
32             edge_attr.append(float(confidence))
33
34     edge_index = torch.tensor(edge_index,
35 dtype=torch.long).t().contiguous()
36     edge_attr = torch.tensor(edge_attr,
37 dtype=torch.float).view(-1, 1)
38
39     for gene in gene_list:
40         if gene in gene_dict:
41             x1.append(feature1[gene_dict[gene]])
42             x2.append(feature2[gene_dict[gene]])
43
44     x1 = torch.tensor(x1, dtype=torch.float)
45     x2 = torch.tensor(x2, dtype=torch.float)
46
47     data = Data(x1=x1, x2=x2, edge_index=edge_index,
48 edge_attr=edge_attr)
49     return data
50
51 # GAT 模型定义
52 class GATModel(nn.Module):
53     def __init__(self, in_channels, out_channels, heads):
54         super(GATModel, self).__init__()

```

```

50         self.conv1 = GATConv(in_channels, out_channels,
heads=heads)
51
52     def forward(self, x, edge_index, edge_attr):
53         x = self.conv1(x, edge_index, edge_attr)
54         return x
55
56 # 训练模型
57 def train(model, data, optimizer, criterion, epochs):
58     model.train()
59     losses = [] # 用于记录每个 epoch 的损失值
60     for epoch in range(epochs):
61         optimizer.zero_grad()
62         out = model(data.x1, data.edge_index, data.edge_attr)
63         loss = criterion(out, data.x2)
64         loss.backward()
65         optimizer.step()
66         losses.append(loss.item()) # 记录当前 epoch 的损失值
67         print(f'Epoch {epoch + 1}/{epochs}, Loss:
{loss.item()}')
68
69     # 绘制损失曲线图
70     plt.plot(losses)
71     plt.title('Training Loss Over Epochs')
72     plt.xlabel('Epoch')
73     plt.ylabel('Loss')
74     plt.show()
75
76
77 # 评估链接预测结果
78 def evaluate(y_true, y_pred):
79     y_true = (y_true > 0.5).int().cpu().numpy()
80     y_pred = (y_pred > 0.5).int().cpu().numpy()
81
82     accuracy = accuracy_score(y_true, y_pred)
83     precision = precision_score(y_true, y_pred,
average='micro')
84     recall = recall_score(y_true, y_pred, average='micro')
85     f1 = f1_score(y_true, y_pred, average='micro')
86     roc_auc = roc_auc_score(y_true, y_pred)
87     aupr = average_precision_score(y_true, y_pred)
88

```

```

89     return accuracy, precision, recall, f1, roc_auc, aupr
90
91 # 读取数据
92 gene_list = read_data('GeneList.txt')
93 link_list = read_data('Positive_LinkSL.txt')
94 feature1 = np.loadtxt('feature1_go.txt')
95 feature2 = np.loadtxt('feature2_ppi.txt')
96
97 # 划分数据集和测试集
98 train_gene_list, test_gene_list = train_test_split(gene_list,
99                                                    test_size=0.2, random_state=42)
100
101 # 构建训练集和测试集的图数据
102 train_data = build_graph_data(train_gene_list, link_list,
103                                feature1, feature2)
104 test_data = build_graph_data(test_gene_list, link_list,
105                               feature1, feature2)
106
107 # 创建并训练 GAT 模型
108 model = GATModel(in_channels=128, out_channels=128, heads=1)
109 optimizer = optim.Adam(model.parameters(), lr=0.001)
110 criterion = nn.MSELoss()
111
112 train(model, train_data, optimizer, criterion, epochs=200)
113
114 # 进行链接预测
115 pred_scores = model(test_data.x1, test_data.edge_index,
116                     test_data.edge_attr)
117
118 # 评估链接预测结果
119 accuracy, precision, recall, f1, roc_auc, aupr =
120 evaluate(test_data.x2, pred_scores)
121
122 print(f'Accuracy: {accuracy} \nPrecision: {precision} \nRecall:
123       {recall} \nF1 Score: {f1}')
124 print(f'ROC AUC: {roc_auc} \nAUPR: {aupr}')
125
126 import networkx as nx
127 import torch
128 from torch_geometric.data import Data

```

```

125
126
127 # 将 PyTorch Geometric 图数据转换为 NetworkX 图
128 G = nx.Graph()
129 G.add_nodes_from(range(test_data.num_nodes))
130 G.add_edges_from(test_data.edge_index.t().tolist())
131
132 # 使用 NetworkX 绘制图
133 pos = nx.spring_layout(G)
134 nx.draw(G, pos, with_labels=True, font_weight='bold',
135         node_color='lightblue', node_size=1000, font_size=8,
136         edge_color='gray')
137 plt.show()
138

```

## 任务2 多通道图卷积网络 test2.py

```

1  import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  from torch_geometric.data import Data
5  from torch_geometric.nn import GCNConv
6  from sklearn.metrics import accuracy_score, precision_score,
7  recall_score, f1_score, roc_auc_score, average_precision_score,
8  roc_curve, auc
9  from sklearn.model_selection import train_test_split
10 import numpy as np
11 import matplotlib.pyplot as plt
12
13 # 读取数据
14 def read_data(file_path):
15     with open(file_path, 'r') as f:
16         data = f.read().splitlines()
17     return data
18
19 # 构建图数据
20 def build_graph_data(gene_list, link_list, feature1, feature2):
21     edge_index = []
22     edge_attr = []
23     x1 = []
24     x2 = []

```



```

23
24     gene_dict = {gene: idx for idx, gene in
enumerate(gene_list)}
25
26     for link in link_list:
27         gene1, gene2, confidence = link.split('\t')
28         if gene1 in gene_dict and gene2 in gene_dict:
29             edge_index.append([gene_dict[gene1],
gene_dict[gene2]])
30             edge_attr.append(float(confidence))
31
32     edge_index = torch.tensor(edge_index,
dtype=torch.long).t().contiguous()
33     edge_attr = torch.tensor(edge_attr,
dtype=torch.float).view(-1, 1)
34
35     for gene in gene_list:
36         if gene in gene_dict:
37             x1.append(feature1[gene_dict[gene]])
38             x2.append(feature2[gene_dict[gene]])
39
40     x1 = torch.tensor(x1, dtype=torch.float)
41     x2 = torch.tensor(x2, dtype=torch.float)
42
43     data = Data(x1=x1, x2=x2, edge_index=edge_index,
edge_attr=edge_attr)
44     return data
45
46 # Multi-Channel Graph Convolutional Network 模型定义
47 class MultiChannelGCN(nn.Module):
48     def __init__(self, in_channels, out_channels):
49         super(MultiChannelGCN, self).__init__()
50         self.conv1 = GCNConv(in_channels, out_channels)
51         self.conv2 = GCNConv(in_channels, out_channels)
52
53     def forward(self, x1, x2, edge_index, edge_attr):
54         x1 = self.conv1(x1, edge_index, edge_attr)
55         x2 = self.conv2(x2, edge_index, edge_attr)
56         return x1, x2
57
58 # 训练模型
59 def train(model, data, optimizer, criterion, epochs):

```

```

60     model.train()
61     losses = [] # 用于记录每个 epoch 的损失值
62     for epoch in range(epochs):
63         optimizer.zero_grad()
64         out1, out2 = model(data.x1, data.x2, data.edge_index,
data.edge_attr)
65         loss1 = criterion(out1, data.x1)
66         loss2 = criterion(out2, data.x2)
67         loss = loss1 + loss2
68         loss.backward()
69         optimizer.step()
70         losses.append(loss.item()) # 记录当前 epoch 的损失值
71         print(f'Epoch {epoch + 1}/{epochs}, Loss:
{loss.item()}')
72
73     # 绘制损失曲线图
74     plt.plot(losses)
75     plt.title('Training Loss Over Epochs')
76     plt.xlabel('Epoch')
77     plt.ylabel('Loss')
78     plt.show()
79
80 # 评估链接预测结果
81 def evaluate(y_true, y_pred):
82     y_true = (y_true > 0.3).int().cpu().numpy()
83     y_pred = (y_pred > 0.3).int().cpu().numpy()
84
85     accuracy = accuracy_score(y_true, y_pred)
86     precision = precision_score(y_true, y_pred,
average='micro')
87     recall = recall_score(y_true, y_pred, average='micro')
88     f1 = f1_score(y_true, y_pred, average='micro')
89     roc_auc = roc_auc_score(y_true, y_pred)
90     aupr = average_precision_score(y_true, y_pred)
91
92     return accuracy, precision, recall, f1, roc_auc, aupr
93
94 # 读取数据
95 gene_list = read_data('GeneList.txt')
96 link_list = read_data('Positive_LinkSL.txt')
97 feature1 = np.loadtxt('feature1_go.txt')
98 feature2 = np.loadtxt('feature2_ppi.txt')

```

```

99
100 # 划分数据集和测试集
101 train_gene_list, test_gene_list = train_test_split(gene_list,
102                                                    test_size=0.2, random_state=42)
103
104 # 构建训练集和测试集的图数据
105 train_data = build_graph_data(train_gene_list, link_list,
106                                feature1, feature2)
107 test_data = build_graph_data(test_gene_list, link_list,
108                               feature1, feature2)
109
110 # 创建并训练 Multi-Channel GCN 模型
111 model = MultiChannelGCN(in_channels=128, out_channels=128)
112 optimizer = optim.Adam(model.parameters(), lr=0.001)
113 criterion = nn.MSELoss()
114
115 train(model, train_data, optimizer, criterion, epochs=200)
116
117 # 进行链接预测
118 pred_scores1, pred_scores2 = model(test_data.x1, test_data.x2,
119                                    test_data.edge_index, test_data.edge_attr)
120 pred_scores = (pred_scores1 + pred_scores2) / 2 # 取两个通道的平
121 均值
122
123 # 评估链接预测结果
124 accuracy, precision, recall, f1, roc_auc, aupr =
125 evaluate(test_data.x2, pred_scores)
126 print(f'Accuracy: {accuracy} \nPrecision: {precision} \nRecall:
127 {recall} \nF1 Score: {f1}')
128 print(f'ROC AUC: {roc_auc} \nAUPR: {aupr}')
129
130
131 import networkx as nx
132 import torch
133 from torch_geometric.data import Data
134
135 # 将 PyTorch Geometric 图数据转换为 NetworkX 图
136 G = nx.Graph()
137 G.add_nodes_from(range(test_data.num_nodes))
138 G.add_edges_from(test_data.edge_index.t().tolist())

```

```
134
135 # 使用 NetworkX 绘制图
136 pos = nx.spring_layout(G)
137 nx.draw(G, pos, with_labels=True, font_weight='bold',
138         node_color='lightblue', node_size=1000, font_size=8,
139         edge_color='gray')
```

## 任务2 n通道图卷积网络 test2.2.py

```
1  import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  from torch_geometric.data import Data
5  from torch_geometric.nn import GCNConv
6  from sklearn.metrics import accuracy_score, precision_score,
7  recall_score, f1_score, roc_auc_score, \
8  average_precision_score
9  from sklearn.model_selection import train_test_split
10 import numpy as np
11
12
13 # 读取数据
14 def read_data(file_path):
15     with open(file_path, 'r') as f:
16         data = f.read().splitlines()
17     return data
18
19
20 # 构建图数据
21 def build_graph_data(gene_list, link_list, feature1, feature2):
22     edge_index = []
23     edge_attr = []
24     x1 = []
25     x2 = []
26
27     gene_dict = {gene: idx for idx, gene in
28                 enumerate(gene_list)}
```

```

29     for link in link_list:
30         gene1, gene2, confidence = link.split('\t')
31         if gene1 in gene_dict and gene2 in gene_dict:
32             edge_index.append([gene_dict[gene1],
gene_dict[gene2]])
33             edge_attr.append(float(confidence))
34
35     edge_index = torch.tensor(edge_index,
dtype=torch.long).t().contiguous()
36     edge_attr = torch.tensor(edge_attr,
dtype=torch.float).view(-1, 1)
37
38     for gene in gene_list:
39         if gene in gene_dict:
40             x1.append(feature1[gene_dict[gene]])
41             x2.append(feature2[gene_dict[gene]])
42
43     x1 = torch.tensor(x1, dtype=torch.float)
44     x2 = torch.tensor(x2, dtype=torch.float)
45
46     data = Data(x1=x1, x2=x2, edge_index=edge_index,
edge_attr=edge_attr)
47     return data
48
49
50 # Multi-Channel Graph Convolutional Network 模型定义
51 class MultiChannelGCN(nn.Module):
52     def __init__(self, in_channels, out_channels,
num_channels):
53         super(MultiChannelGCN, self).__init__()
54         self.channels = nn.ModuleList([GCNConv(in_channels,
out_channels) for _ in range(num_channels)])
55
56     def forward(self, *inputs):
57         output_channels = [channel(x, inputs[-2], inputs[-1])
for channel, x in zip(self.channels, inputs[:-2])]
58         return output_channels
59
60
61 # 训练模型
62 def train(model, data, optimizer, criterion, epochs):
63     model.train()

```

```

64     losses = [] # 用于记录每个 epoch 的损失值
65     for epoch in range(epochs):
66         optimizer.zero_grad()
67         output_channels = model(data.x1, data.x2,
data.edge_index, data.edge_attr)
68
69         # Assuming that data.x1 and data.x2 are the target
values for each channel
70         loss = sum(criterion(output, data.x1 if i == 0 else
data.x2) for i, output in enumerate(output_channels))
71
72         loss.backward()
73         optimizer.step()
74         losses.append(loss.item()) # 记录当前 epoch 的损失值
75         print(f'Epoch {epoch + 1}/{epochs}, Loss:
{loss.item()}')
76
77         # 绘制损失曲线图
78         plt.plot(losses)
79         plt.title('Training Loss Over Epochs')
80         plt.xlabel('Epoch')
81         plt.ylabel('Loss')
82         plt.show()
83
84
85 # 评估链接预测结果
86 def evaluate(y_true, y_pred):
87     y_true = (y_true > 0.3).int().cpu().numpy()
88     y_pred = (y_pred > 0.3).int().cpu().numpy()
89
90     accuracy = accuracy_score(y_true, y_pred)
91     precision = precision_score(y_true, y_pred,
average='micro')
92     recall = recall_score(y_true, y_pred, average='micro')
93     f1 = f1_score(y_true, y_pred, average='micro')
94     roc_auc = roc_auc_score(y_true, y_pred)
95     aupr = average_precision_score(y_true, y_pred)
96
97     return accuracy, precision, recall, f1, roc_auc, aupr
98
99
100 # 读取数据

```

```

101 gene_list = read_data('GeneList.txt')
102 link_list = read_data('Positive_LinkSL.txt')
103 feature1 = np.loadtxt('feature1_go.txt')
104 feature2 = np.loadtxt('feature2_ppi.txt')
105
106 # 划分数据集和测试集
107 train_gene_list, test_gene_list = train_test_split(gene_list,
108                                                    test_size=0.2, random_state=42)
109
110 # 构建训练集和测试集的图数据
111 train_data = build_graph_data(train_gene_list, link_list,
112                                feature1, feature2)
113 test_data = build_graph_data(test_gene_list, link_list,
114                               feature1, feature2)
115
116 # 创建并训练 Multi-Channel GCN 模型
117 num_channels = 150 # Set the number of channels (adjust as
118                    # needed)
119 model = MultiChannelGCN(in_channels=128, out_channels=128,
120                          num_channels=num_channels)
121 optimizer = optim.Adam(model.parameters(), lr=0.001)
122 criterion = nn.MSELoss()
123
124 train(model, train_data, optimizer, criterion, epochs=200)
125
126 # 进行链接预测
127 pred_scores_list = model(test_data.x1, test_data.x2,
128                           test_data.edge_index, test_data.edge_attr)
129 pred_scores = torch.stack(pred_scores_list).mean(dim=0) # Take
130                # the mean across channels
131
132 # 评估链接预测结果
133 accuracy, precision, recall, f1, roc_auc, auapr =
134 evaluate(test_data.x2, pred_scores)
135
136 print(f'Accuracy: {accuracy} \nPrecision: {precision} \nRecall:
137       {recall} \nF1 Score: {f1}')
138 print(f'ROC AUC: {roc_auc} \nAUPR: {auapr}')
139
140 import networkx as nx
141 import torch
142 from torch_geometric.data import Data
143

```

```
134 # 将 PyTorch Geometric 图数据转换为 NetworkX 图
135 G = nx.Graph()
136 G.add_nodes_from(range(test_data.num_nodes))
137 G.add_edges_from(test_data.edge_index.t().tolist())
138
139 # 使用 NetworkX 绘制图
140 pos = nx.spring_layout(G)
141 nx.draw(G, pos, with_labels=True, font_weight='bold',
142         node_color='lightblue', node_size=1000, font_size=8,
143         edge_color='gray')
143 plt.show()
144
```