同济大学

TONGJI UNIVERSITY

实 验 报 告

实验名称: 金融中的动态社交网络研究

课程名称: 网络数据风控技术

专　　业:	数据科学与大数据技术
学生姓名:	胡逸凡
学　　号:	2153592
指导老师:	叶晨
成　　绩:	
实验日期:	2023 年 12 月

目录

1 实验背景与问题介绍 **2**
1.1 实验背景 . . . 2
1.2 问题分析 . . . 2
1.3 数据说明 . . . 2

2 国内外相关研究工作介绍 **3**

3 数据预处理 **3**
3.1 数据分布 . . . 3
3.2 离群点处理 . . . 5
3.3 归一化处理 . . . 5

4 特征工程 **6**
4.1 说明 . . . 6
4.2 Graph Embedding . . . 7
4.3 生成图信息相关特征 . . . 9
4.4 邻居采样 . . . 10
4.5 类别不平衡处理 . . . 11

5 建模与优化 **11**
5.1 实验环境 . . . 11
5.2 实验数据描述 . . . 11
5.3 CNN . . . 12
5.4 GCN . . . 13
5.5 GAT . . . 14
5.6 GIN . . . 15
5.7 SGC . . . 15
5.8 SAGE . . . 16
5.9 Care-GNN . . . 18
5.10 PC-GNN . . . 21
5.11 模型融合 . . . 22

6 实验结果 **23**

7 结论和展望 **23**

1 实验背景与问题介绍

1.1 实验背景

本次大作业的实验以反欺诈作为研究主题，这是金融领域中一个永恒的关注点。反欺诈旨在识别和预防欺诈行为，以确保金融交易的安全和可靠性。为了更好地理解和建模欺诈行为，我们采用了一个全连通的社交网络有向动态图作为我们的数据集。

数据集的抽样涵盖了企业不同业务时间段的数据，这使得我们能够在不同环境和条件下研究欺诈行为的变化和趋势。这种时序性的数据允许我们观察欺诈模式随时间的演变，为建立更为精准和实时的反欺诈系统提供了有力的支持。

社交网络的全连通性使得我们能够捕捉到广泛的关系和交互信息，这对于揭示欺诈网络的复杂性至关重要。有向图的使用使我们能够区分交易中的方向性关系，从而更好地理解资金流向和交易链路。动态图的特性则为我们提供了对欺诈行为随时间演变的全面洞察。

通过深入研究这一全连通的社交网络有向动态图，我们旨在发现隐藏在金融交易背后的模式和规律，以及可能的欺诈迹象。我们的目标是为金融领域提供更加智能和高效的反欺诈解决方案，以确保经济交易的安全性和信任度。

1.2 问题分析

在本题目的图数据中，节点代表注册用户，每个用户有 17 个节点特征，另外，从节点 A 指向节点 B 的有向边代表用户 A 与用户 B 有关系。图中的边有不同的类型，代表了不同关系分类。数据集通过 npz 方式储存。本题目的预测任务为识别欺诈用户的节点。在数据集中有两类节点，预测任务只需要将欺诈用户（Class 1）从正常用户（Class 0）中区分出来。

1.3 数据说明

数据集和测试集均在 npz 文件中，其中，训练集有 386104 条数据，测试集有 193053 条数据，大约为训练集的一半。观察数据可以发现，数据集中节点特征缺失值被-1 填充。另外，该分类数据集属于类别不平衡问题，负例是正例的 85 倍左右。

具体 npz 文件内容如下：

(1.) x (节点特征): 形状 (579157, 17) 表示该数据集包含 579157 个节点，每个节点有 17 维特征。这些特征可能包括有关节点的各种信息，如交易历史、用户属性、用户行为等等。

(2.) y (节点标签): 形状 (579157,) 表示每个节点的标签。这可能是一个二元标签，表示该用户是否属于欺诈用户。

(3.) edge_index (边索引): 形状 (167559, 2) 表示图的边索引。每一行 (i, j) 表示节点 i 到节点 j 之间存在一条有向边。这个数据结构用于表示节点之间的连接关系。

(4.) edge_type (边类型): 形状 (167559,) 表示对应于边索引的边的类型，共有 1,2,...,11 共 11 种类型的边。这可能是关于不同交易类型或不同用户关系的信息，用于更详细地描述节点之间的关联。

(5.) train_mask (训练集节点掩码): 形状 (386104,) 表示用于训练模型的节点的掩码。这个掩码通常用于从整个图中选择一部分节点作为训练集。

(6.) test_mask (测试集节点掩码): 形状 (193053,) 表示用于测试模型的节点的掩码。与训练集节点掩码类似，这个掩码用于选择一部分节点作为测试集。

2 国内外相关研究工作介绍

在本节中，我们将回顾一下传统的图神经网络方法，以及简要介绍了基于 GNN 的金融欺诈检测方法。

Research on Graph Representation Learning. 图神经网络（GNNs）通过聚合邻居信息来丰富节点表示。GNN 最核心的问题是如何设计卷积算子（聚合函数），它被用于构造节点关系之间的相关性。图卷积网络（GCN）[1] 是基于利用拉普拉斯矩阵构造聚合函数，在节点之间传递，无需参数学习。GAT [2] 在传递消息时，采用注意机制来学习相邻节点之间的重要性。GraphSAGE [3] 提出了随机邻居抽样，因为在现实中的图总是巨大的。MoNet [4] 和 MPNN [5] 为人们提供了设计基于 GNN 的方法的一般框架。HAN [6] 定义了不同的元路径来解决图中不同节点类型和边缘类型的问题。本次实验中我采用了多种 GNN 方法，并复现了两个 sota 模型 CARE-GNN 和 PC-GNN [7, 8]。

Research on Financial Fraud Detection. 金融欺诈检测最初是基于机器学习方法，如 SVM、贝叶斯网络和神经网络。后来人们引入图形方法来在欺诈检测场景中建模复杂的关系数据 [9]。主要分为三步：(a.) 构建图，主要通过后台的各种日志，比如说用户的日志，提取日志中用户的信息，组成图中的节点，通过用户之间的关系，组成图中的边，从而完成图的构建；(b.) 训练图神经网络学习图中信息，学习到的信息可以是节点的 Embedding、边的 Embedding 或者图的 Embedding。(c.) 基于 Embedding 训练分类器，基于第二步得到的 Embedding 和已知的一些标签信息，例如上图，已知红色节点为欺诈者，黑色节点标签未知，根据已知的标签训练分类器来得到对未知标签节点的预测。

近年来，国内外众多实验室在金融欺诈监测方面成果颇丰。首先就是 2018 年的 GraphRAD [10] (KDD'18)，它是第一篇将图神经网络应用到欺诈检测问题中的文章，来自亚马逊。CARE-GNN [7] 被选为 CIKM'20 影响力最高的 15 篇文章之一，基于 Yelp 和 Amazon 做了两个开源的数据集，并在这两个数据集上针对用户或者说欺诈者的伪装问题给出了相应的解决方案。PC-GNN [8] 通过下采样和过采样方法解决欺诈数据不平衡问题。[11] 引入了 GNN 方法来检测洗钱欺诈，而 EvolveGCN [12] 在欺诈检测任务中考虑了动态。

国内外也有很多知名的金融欺诈监测公司。例如课堂上给我们讲座分享的同盾科技，是国内数一数二的智能分析与预测公司，打造了基于人工智能的决策智能平台-智策和基于隐私计算的共享智能平台-智邦，聚焦金融风险、安全风险、政府治理风险三大场景，以决策智能技术体系全链路的支撑金融、互联网、大交通等多行业客户实现业务数字化转型。国际上在反欺诈领域最领先的公司是以色列的 Riskified、Forter 等，其中 Riskified 已与众多国际知名品牌，如古驰（Gucci）、普拉达（Prada）、Wish 和施华洛世奇（Swarovski）等达成合作，此外也与春秋航空（Spring Airlines）、宁波豪雅（Costway）等中国领先品牌携手，助力其金融安全。

3 数据预处理

3.1 数据分布

我通过 QQ 图可以看出数据分布，也检查数据和标准正态分布之间的差异大小，具体如下：如果数据点在 QQ 图上近似地落在一条直线上，表明数据可能符合正态分布；QQ 图中的斜率和截距提供了一个理论正态分布与实际数据分布之间的线性拟合，这些参数反映了数据偏离正态分布的程度；QQ 图上的 R 值表示实际数据分布与理论正态分布的相关性，R 值越接近 1，表明两者越趋近于线性关系，进一步支持数据的正态性。

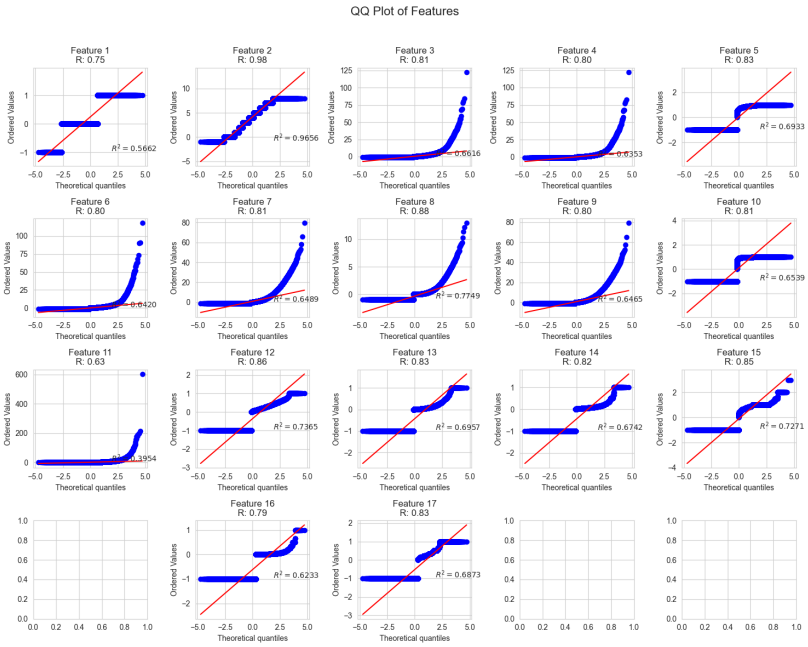


图 1: QQ Plot of Features

图中看出，大部分特征并没有很好的服从正态分布，因此我直接放弃了尝试传统的机器学习方法。而对于深度学习模型，数据不必严格服从正态分布，深度学习模型通常对数据的分布要求较低，因为它们能够自动学习复杂的特征表示。

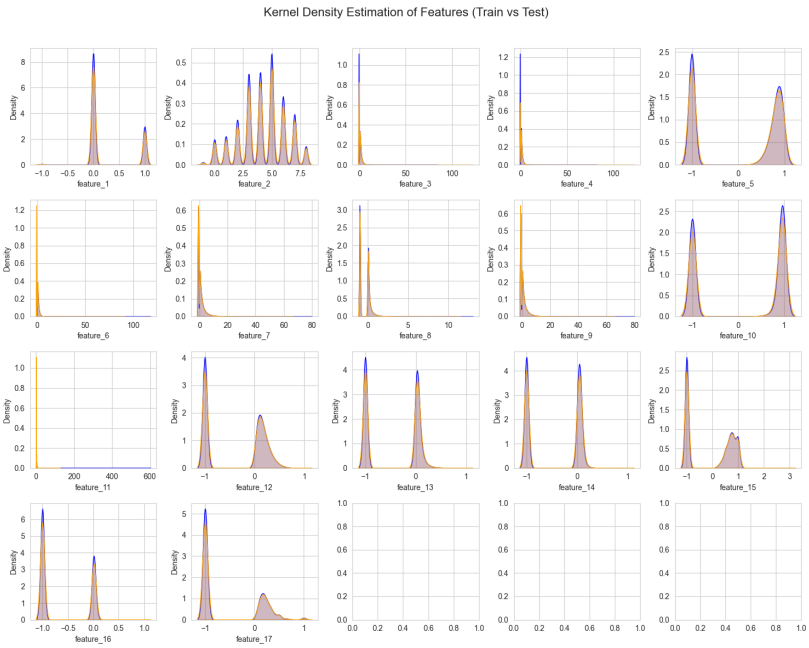


图 2: Kernel Density Estimation of Features (Train vs Test)

另外，我利用核密度图 (图 2) 对比训练集和测试集的特征分布。发现训练集和测试集特征的数据分布极为相似，这有助于减轻过拟合风险，相似的分布有助于模型更好地推广到新数据，也意味着模型在未知数据上的表现会更加可靠。

3.2 离群点处理

首先，通过绘制箱线图 (图 3)，我们发现个别特征有明显的离群点，由于本数据集为图数据，因此删除异常值记录是不合理的。因此直接由 3σ 准则，我将 3σ 以外的数据用上下界代替。

```
mean_x = np.mean(x)
std_x = np.std(x)                    # 计算均值和标准差
lower_bound = mean_x - 3 * std_x
upper_bound = mean_x + 3 * std_x  # 确定上下界
x_clipped = np.clip(x, lower_bound, upper_bound) # 替换离群点
```

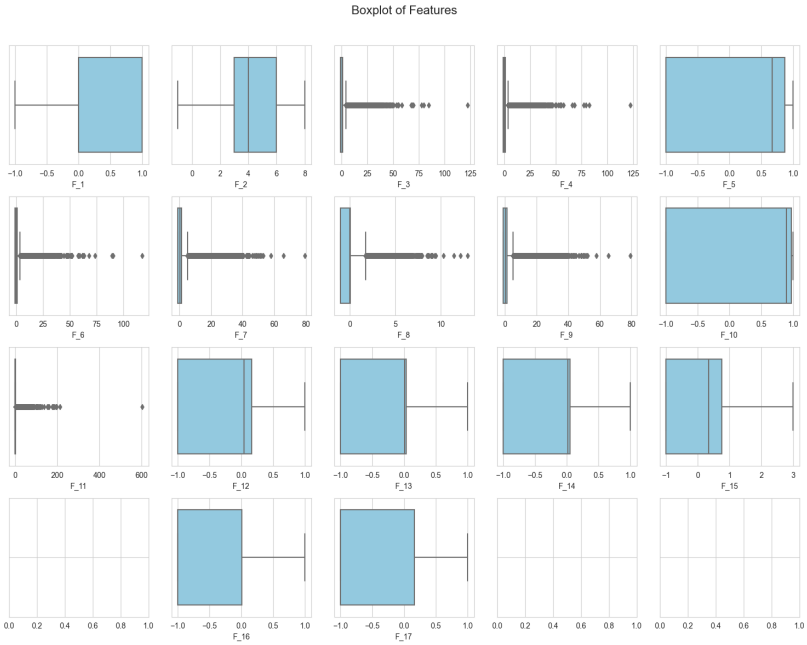


图 3: Boxplot of Features

3.3 归一化处理

首先我们通过 DataFrame 查看 x 特征数据分布，发现特征之间存在较大的量纲差距，这意味着它们的数值范围差异较大。可能会影响某些机器学习算法的性能，因此需要采取适当的标准化或缩放方法。对于数值型特征，采用 scaling，放缩到-1-1 之间。通过对数字型的归一化，我们消除不

```
df = pd.DataFrame(x, columns=[f'F_{i + 1}' for i in range(17)])
description = df.describe()
description
```

在 2023.12.23 15:54:20 于 722ms内执行

8行 × 17 列

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9
count	579157.000000	579157.000000	579157.000000	579157.000000	579157.000000	579157.000000	579157.000000	579157.000000	579157.000000
mean	0.248067	4.175918	0.430182	0.286104	0.043178	0.188256	0.891654	-0.328536	0.849458
std	0.442285	1.988596	1.992449	1.806118	0.911066	1.675974	2.944884	0.719256	2.885987
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.000000	3.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	0.000000	4.000000	0.190000	0.160000	0.675751	0.100000	0.142000	0.003000	0.130000
75%	1.000000	6.000000	1.095000	0.880000	0.875000	0.800000	1.474000	0.090000	1.412000
max	1.000000	8.000000	122.490000	122.085000	1.000000	117.600000	79.356000	12.933000	79.161000

图 4: Feature Describe

同量纲的影响，提高模型的泛化能力，另外，归一化也降低异常值的影响。但是，通过观察数据描述，由于所给的 npz 文件中数据的 75% 分位数量纲非常接近，只是个别数据由于最大值 max 的为异常值，导致在观察数据描述时显得有所不同。

```
if dataset in ['XYGraphP1']:
    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)
```

在实际应用中，通过模型预测发现，在不进行归一化的情况下，模型的性能表现更好。这可能是由于归一化过程中引入的信息损失，特别是在小数点后的数据范围较小的情况下。归一化的目的是为了消除不同特征之间的尺度差异，但在某些情况下，尤其是当数据的数量级已经很接近时，归一化可能会引入额外的噪音，导致模型性能下降。因此，综合观察数据描述和模型预测结果，我最终不进行归一化，保留数据的原始尺度，并最大限度地保留信息，以取得更好的模型性能。

4 特征工程

4.1 说明

最初，在没有进行特征工程的情况下，由于提供的数据较为完善且全为数值型数据，我初步认为只需要进行不平衡数据集的处理。然而，尽管在这种情况下最好的预测 AUC 指标仍然在约 0.75 左右，这可能表明数据的表达力受到了限制。

随后，通过查阅相关资料，认识到对图数据进行特征工程，尤其是在数据扩充和升维方面，可能对模型性能的提升有显著的影响。我采用了两种主要方法来处理图数据：

(1.) Graph Embedding：通过直接对 17 条数据进行图嵌入，即将图结构映射到低维度的向量空间中，以获取更富有表达力的特征表示。然而，这一方法的效果并不如预期。

(2.) 生成图信息相关特征：通过已知数据生成图信息相关的特征。这包括在节点层面、连接层面和全图层面来生成新的特征。这一方法表现出较好的效果，预测 AUC 指标超过了 0.8。

生成图信息相关特征可以从图 5 几个方面入手：

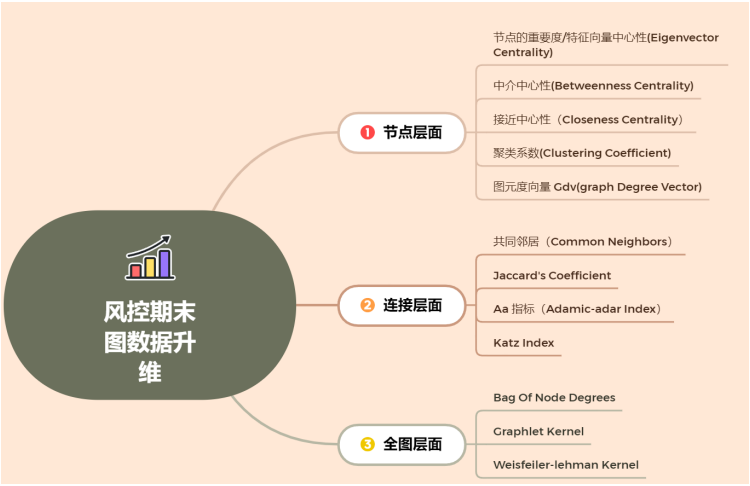


图 5: Graph Data Engineering

4.2 Graph Embedding

通过查找资料，在https://github.com/shenweichen/GraphEmbedding中有 5 种 Graph Embedding 方法具体介绍如下，DeepWalk, LINE (Large-scale Information Network Embedding), Node2Vec, SDNE (Structural Deep Network Embedding), Struc2Vec。（多层次的随机游走：不仅考虑单一层次的邻居节点，还考虑多个层次的邻居节点） [13–17]。

表 1: Comparison of 5 Graph Embedding Methods.

Method	Source Journal	Advantages
DeepWalk	KDD 2014	简单有效，适用于大规模图的无监督学习任务，如节点聚类
LINE	WWW 2015	保留一阶和二阶相邻节点信息的灵活性，可以通过权衡这两者来调整嵌入的性质
Node2Vec	KDD 2016	平衡广度优先和深度优先的随机游走，兼顾全局和局部结构
SDNE	KDD 2016	通过多层非线性映射来捕捉图的结构信息，在小规模图上表现较好
Struc2Vec	KDD 2017	多层次的随机游走，保留图的结构信息，适用性广泛

Graph Embedding 的算法流程是先输入图数据；接着进行预处理，以确保数据的质量；然后，在图表示学习阶段，采用 Graph Embedding 算法（例如 DeepWalk、Node2Vec、LINE 等）对图进行学习，将节点映射到低维空间，捕捉图的结构和节点之间的关系；随后，通过后处理对嵌入进行进一步的处理，例如降维操作或聚类；最终，得到嵌入表示。Graph Embedding 的算法结构图如图 6。

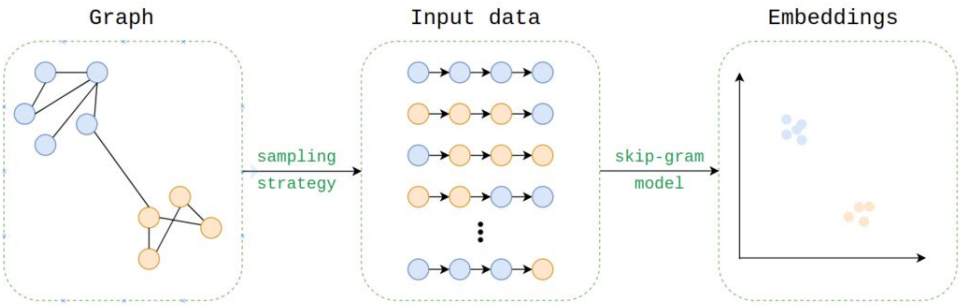
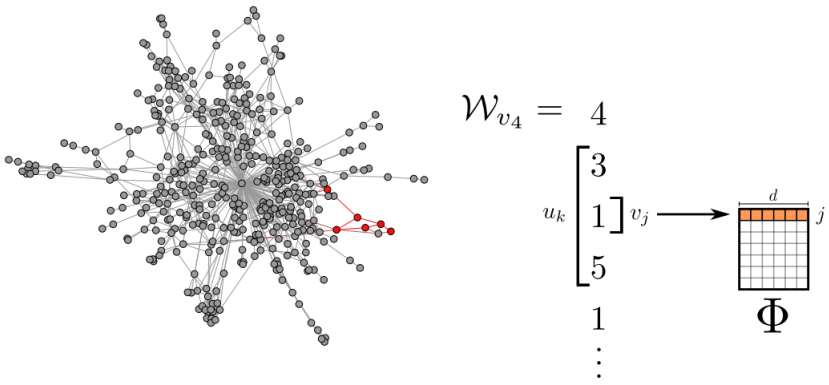


图 6: Graph Embedding Architecture

Random Walk 算法流程是在图数据输入后，选择一个起始节点作为随机游走的起点；然后，从该起始节点开始，根据一定的策略，随机选择邻近节点进行移动，形成一个节点序列；这一过程重复进行多次，每次从当前节点出发，按照预定义的策略选择邻居节点，直至达到预设的步数。在随机游走的过程中，节点的访问顺序不仅基于当前节点的邻居选择策略，还可能考虑一定的随机性，以增加序列的多样性。这样，通过多次随机游走，可以生成蕴含了图的结构信息的多个节点序列。随机游走的架构图如图 7。

如下为 Struc2Vec 的代码。

```
class Struc2Vec():
    def __init__(self, graph, walk_length=10, num_walks=100, workers=1, verbose=0,
        stay_prob=0.3, opt1_reduce_len=True,
            opt2_reduce_sim_calc=True, opt3_num_layers=None, temp_path='./temp_struc2vec/',
                reuse=False):
        self.graph = graph
        self.idx2node, self.node2idx = preprocess_nxgraph(graph)
        self.idx = list(range(len(self.idx2node)))
```



(a) Random walk generation. (b) Representation mapping.

图 7: Random Walk Architecture

```
7
8           self.opt1_reduce_len = opt1_reduce_len
9           self.opt2_reduce_sim_calc = opt2_reduce_sim_calc
10          self.opt3_num_layers = opt3_num_layers
11
12          self.resue = reuse
13          self.temp_path = temp_path
14
15          if not os.path.exists(self.temp_path):
16              os.mkdir(self.temp_path)
17          if not reuse:
18              shutil.rmtree(self.temp_path)
19              os.mkdir(self.temp_path)
20
21          self.create_context_graph(self.opt3_num_layers, workers, verbose)
22          self.prepare_biased_walk()
23          self.walker = BiasedWalker(self.idx2node, self.temp_path)
24          self.sentences = self.walker.simulate_walks(
25              num_walks, walk_length, stay_prob, workers, verbose)
26
27          self._embeddings = {}
28
29      ......
30
31  # 将我们的图信息插入
32  struc2vec = Struc2Vec(data)
33  walks = struc2vec.generate_walks()
34  struc2vec.learn_embeddings(walks)
```

通过 Graph Embedding 得到的升维数据预测效果并最好，在平台上 AUC 值达到了 0.76 以上。

4.3 生成图信息相关特征

我通过添加一系列由已知信息生成的图信息相关特征，最终将 (n, 17) 维的输入数据扩充到 (n, 33) 维，扩充大约一倍，其中 n 为训练集个数。这种特征工程的优势在于提升了模型对图结构的抽象能力，通过捕获更多的节点关系和结构信息，使得学得的图嵌入更为丰富和有意义。不仅有助于提高模型的泛化能力，适应更广泛的图结构，还能够解决数据稀疏性问题，为模型提供更多上下文信息，从而提高任务性能。

最终，这种通过添加图信息相关特征方式升维数据的预测效果比较好，在平台上 AUC 值超过 0.8。具体添加的特征介绍如下：

Bag of Node Degrees 这个特征构造方法借鉴了 NLP 中的词袋模型（BOW），只不过应用在了图中。当然，它也不是生搬硬套的统计相同节点的个数，而是统计出了一张图的所有节点的度。添加了与节点入度、出度相关的特征，代码如下

```
def degree_feat(edge_index, x):
    adj = csr_matrix(
        (np.ones(edge_index.shape[0]), (edge_index[:, 0], edge_index[:, 1])),
        shape=(x.shape[0], x.shape[0]))
    out_degree, in_degree = adj.sum(axis=1), adj.sum(axis=0).T
    return out_degree, in_degree
```

边特征编码 由于边的类型总共 11 中，我们添加 11 个 Int 类型变量，遍历每个节点，对于每个节点，统计与该节点相连的每种边类型的数量，记录在该矩阵中。代码如下

```
def edge_type_feat(edge_type, edge_index, x):
    edge_type_adj = csr_matrix(
        (edge_type, (edge_index[:, 0], edge_index[:, 1])),
        shape=(x.shape[0], x.shape[0]))
    edge_type_feat = np.zeros((x.shape[0], 11))
    data, indptr = edge_type_adj.data, edge_type_adj.indptr
    for i in range(x.shape[0]):
        row = data[indptr[i] : indptr[i + 1]]
        unique, counts = np.unique(row, return_counts=True)
        for j, k in zip(unique, counts):
            edge_type_feat[i, j - 1] = k
    return edge_type_feat
```

节点相似性 计算每个节点与其相邻节点的余弦相似度，之后加和，作为新特征添加。余弦相似度是一种衡量两个向量之间相似度的度量方法。在图嵌入领域，对于节点表示空间中的两个向量，余弦相似度可以告诉我们它们在方向上的相似程度；也能捕捉邻居节点关系，度量节点与其邻居节点之间的相似性；还能增强图嵌入表示。代码如下：

```
def cos_sim_sum(x, edge_index):
    row, col = edge_index
    sim = F.cosine_similarity(x[row], x[col])
    sim_sum = scatter.scatter(sim, row, dim=0, dim_size=x.size(0), reduce="sum")
    return torch.cat([x, torch.unsqueeze(sim_sum, dim=1)], dim=1)
```

AA 指标 AA 指标的定义如下：

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{ln|N(u)|}$$

这个指标代表你们的共同好友是不是社牛，$N(u)$ 代表的是节点 u 认识的好友数，而这里的 u 则是 x 与 y 的共同好友们。计算方式是对于两个节点之间的每一个共同邻居，将其度数的倒数（取对数）相加，换句话说，AA 指标更强调那些在图中度较低的节点，更关注那些“不社牛”的点。将 AA 指标作为附加的图结构特征，有助于模型更好地理解节点的环境和关联关系，从而提高模型的泛化能力。

```
def aa_index(x, edge_index):
    graph = nx.Graph()
    graph.add_edges_from(edge_index.cpu().numpy().T)
    aa_index = {}
    for node in graph.nodes():
        aa_index[node] = 0.0
        neighbors = set(graph.neighbors(node))
        for neighbor in neighbors:
            common_neighbors = set(graph.neighbors(neighbor)).intersection(neighbors)
            if len(common_neighbors) > 0:
                aa_index[node] += 1 / np.log(len(common_neighbors)) if len(common_neighbors) > 1
                    else 0

    aa_feature = np.array([aa_index[node] for node in graph.nodes()])
    aa_feature = aa_feature.reshape(-1, 1)
    x_with_aa = np.concatenate((x, aa_feature), axis=1)
    return x_with_aa
```

4.4 邻居采样

我还进行了邻居采样的尝试，该方法在每次训练中不是传入整个图，而是仅传入目标节点及其邻居节点。

邻居采样可以显著减少训练时传入模型的节点数量，从而降低计算复杂度。相较于在整个图上进行训练，仅传入目标节点及其邻居节点可以有效缩小训练规模，提高训练的效率；邻居采样使模型更专注于局部结构，有助于提升模型的泛化能力。通过关注目标节点的邻居，模型更容易捕捉节点在局部邻域内的结构和特征，使得模型更具适应性，能够更好地处理未见过的数据，进一步防止过拟合；许多图数据中存在着复杂的局部模式，邻居采样有助于模型更好地捕捉这些局部模式。通过仅传入目标节点及其邻居，模型更容易学到节点在其邻域内的特有模式，提高了对图中微观结构的敏感性。

采用邻居采样的方法给模型带来显著的性能改进。另外，我尝试了不同的采样方法和邻居层数，k_hop_subgraph 效果最好，表示获取指定节点集合的 k-hop 邻居，即在图中距离这些节点不超过 k 步的所有节点。random_walk_neighbor_sampling 是从起始节点开始，每一步都选择一个当前节点的邻居进行移动。通过重复这个过程，可以生成一个随机游走路径。这样得到的邻居采样是随机的，并不保证完全获取所有 k 阶邻居。

```
nodeandneighbor, edge_index, node_map, mask = k_hop_subgraph(
        train_idx, 4, data.edge_index, relabel_nodes=True, num_nodes=data.x.size(0)
    )
```

```
nodeandneighbor, edge_index, node_map, mask = random_walk_neighbor_sampling(
        train_idx, 3, data.edge_index
    )
```

4.5 类别不平衡处理

训练集中两类 label 的正负样本比例接近 85:1，但也出现了不平衡的问题，考虑将选择训练集使得比值小于 10。

我首先考虑了 imblearn 库中几种方法 [18]。我尝试了欠采样中的随机法，Prototype selection，NearMiss 以及过采样中的 Prototype generation，随机复制。然而，通过提交到网站的得分来看，尽管采用了这些处理方法，模型在评估指标上的表现并未如预期地提升，反而出现了不如不进行类别处理时的效果。这可能表明在当前场景下，这些处理方法并未显著改善模型的性能。或许数据的特殊性质或模型的复杂性使得传统的不平衡数据处理方法并不适用，需要更深入的分析和调整。

下采样的代码如下，通过修改 downsample_ratio 修改下采样比例

```
    # imbalance
    positive_indices = (y == 1).nonzero()[0]
    negative_indices = (y == 0).nonzero()[0]
    # 下采样类别为0的样本，使得0的个数是1的 downsample_ratio 倍
    num_negative_samples = int(len(positive_indices) * downsample_ratio)
    downsampled_negative_indices = torch.randint(len(negative_indices),
        size=(num_negative_samples,))
    # 映射下采样后的样本索引
    downsampled_indices = np.concatenate([positive_indices,
        negative_indices[downsampled_negative_indices]])

    num_samples = len(downsampled_indices)
    num_train_samples = int(num_samples * 0.8)
    indices_perm = torch.randperm(num_samples)
    train_mask = downsampled_indices[indices_perm[:num_train_samples]]
    valid_mask = downsampled_indices[indices_perm[num_train_samples:]]
```

5 建模与优化

5.1 实验环境

python version：3.10.13

pycharm version：PyCharm 2023.2.4 (Professional Edition)

pytorch version：2.1.0

CUDA version：12.2

5.2 实验数据描述

实验采用的图数据集如下，总共 11 种边。

表 2: Statistics of Edge Types in the Fraud Dataset.

Dataset	#Node	#Edge	Relations
Fraud	579157	167559	11

Edge Type	Number of Edges
1	18766
2	1758
3	2949
4	35076
5	63909
6	26108
7	107
8	5044
9	5589
10	5493
11	2760

5.3 CNN

开始时，我尝试了 CNN [19]，这只用到了节点的特征数据，并没有用到边数据。CNN 被设计用于图像处理，本赛题中 Conv 到底在卷什么？在预测数据中，卷积操作可以看作是对数据的滑动窗口进行特征提取，捕捉到输入序列中的局部模式。另外，CNN 参数共享，参数量较少，间接减少过拟合风险，虽然减少过拟合对本题作用不大。

我使用 CNN 的原因主要是在 2023CCF BDCI 竞赛中，我们使用 CNN 的预测效果好于使用 GNN，另外，有助于我充分了解特征信息。仅靠 CNN 利用未扩充前的 17 个特征做回归，AUC 已经能取得 0.73，在使用扩充的数据后，AUC 值能够达到 0.767，效果不错。说明所选取的特征与预测结果之间存在较强的相关性。

我使用的 CNN 架构图如下，两层卷积层包括池化层和两层全连接层，channels 视为 33，在 n 条数据之间做一维等宽卷积，卷到 channels 为 1 为止，具体见代码提交：

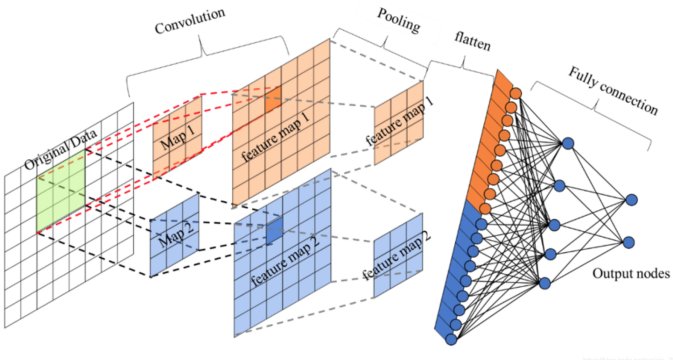


图 8: Model architecture of Fraud Detection CNN.

伪代码如 Algorithm1 中所示，在每一个 epoch 的训练中，我们首先从训练集中得到 $x^{(33)}, y$ 数据，之后进行前向传播输入模型，再通过反向传播计算损失函数并更新梯度。

```
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-7)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
```

另外我们采用 BCELoss() 即二元交叉熵损失作为 loss function，并采用 Adam 优化器来调参，并采用了学习率衰减（weight_decay）为 1e-7，以及 optim.lr_scheduler.StepLR 实现学习率的阶梯式衰减，即每 20 个 epoch，学习率乘 0.5，这样我们一开始可以使用较大的学习率。50 个 epoch 中，train loss 与 valid loss 的变化情况如图所示，loss 在不断波动，但并没有出现过拟合现象，最终在本地验证集上 AUC 值为 0.761。

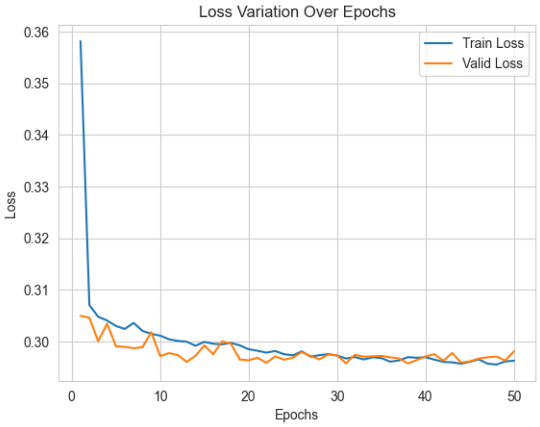


图 9: CNN Loss Variation Over Epochs.

Algorithm 1: The learning algorithm of CNN for Fraud Detection

Input : training set(n×33)
Parameters: learning rate η
Output : output label: The probability of belonging to fraudulent users

Initialize *learning rate* η as 0.0001;
Initialize *max_epoch* as 50;
for *epoch* = 1 *to max_epoch* **do**
  for *batch* = 1 *to n* **do**
    Get $x^{(30)}, y$ samples from the training set;
    Forward propagation;
    Calculate loss function : $L(y, p|\theta_t) = -\sum_i y_i log(p_i)$;
    Back propagation and update parameters : $\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$;
  end
end

5.4 GCN

GCN，图卷积神经网络，实际上跟 CNN 的作用一样，就是一个特征提取器，只不过它的对象是图数据。GCN 精妙地设计了一种从图数据中提取特征的方法，从而让我们可以使用这些特征去对图数据进行节点分类（node classification）、图分类（graph classification）、边预测（link prediction），还可以顺便得到图的嵌入表示（graph embedding） [1]。GCN 原论文中架构图如下：

GCN 的核心如下，我们手头有一批图数据，其中有 N 个节点（node），每个节点都有自己的特征，我们设这些节点的特征组成一个 N×D 维的矩阵 X，然后各个节点之间的关系也会形成一个 N×N 维的矩阵 A，也称为邻接矩阵（adjacency matrix）。X 和 A 便是我们模型的输入。GCN 也

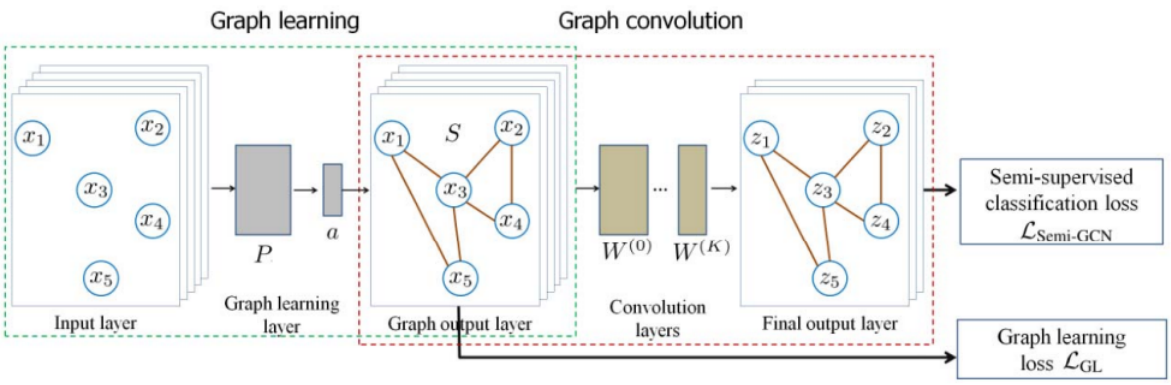


图 10: Model architecture of GCN for Semi-Supervised Learning.

是一个神经网络层，它的层与层之间的传播方式是：

$$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^l W^l)$$

其中，$\tilde{A}$ 是 $A+I$，其中 I 为单位矩阵，$\tilde{D}$ 是 $\tilde{A}$ 的度矩阵，即 $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$，H 是每一层的特征，对于输入层的话，H 就是 X，σ 是激活函数，$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ 用于提取图特征。上图中的 GCN 输入一个图，通过若干层 GCN 每个 node 的特征从 X 变成了 Z，但是无论中间有多少层，node 之间的连接关系，即 A，都是共享的。通过 GCN 可以训练一个 node classification 的模型了。由于即使只有很少的 node 有标签也能训练，也称为半监督分类。

在每一图卷积层输出，通过 ReLU 进行非线性变换，最后输入 softmax 进行分类的输出。

$$Z = f(X, A) = softmax(\tilde{A}\ ReLU(\tilde{A}XW^l)\ W^{l+1})$$

本次实验中，直接通过 PYG 中 GCNConv 即可，效果一般，调参后 AUC 值达到 0.71。

5.5 GAT

GCN 是处理 transductive 任务的一把利器（transductive 任务是指：训练阶段与测试阶段都基于同样的图结构），然而 GCN 有两大局限性是经常被诟病的：

(1.) 无法完成 inductive 任务，即处理动态图问题。inductive 任务是指：训练阶段与测试阶段需要处理的 graph 不同。通常是训练阶段只是在子图（subgraph）上进行，测试阶段需要处理未知的顶点。

(2.) 处理有向图的瓶颈，不容易实现分配不同的学习权重给不同的 neighbor。

GAT 原论文中作者提出，GAT 本质上可以有两种运算方式 [2]。

(a.) Global graph attention，顾名思义，就是每一个顶点都对于图上任意顶点都进行 attention 运算。它完全不依赖于图的结构，适合处理 inductive 任务，但是丢掉了图结构特征，效果可能会很差。

(b.) Mask graph attention，注意力机制的运算只在邻居顶点上进行。原论文中采用这种方式。

GAT 中 attention 的计算与所有 attention 方法类似，均分为两步，计算注意力系数和加权求和。

计算注意力系数：对于顶点 i，逐个计算它的邻居们（j∈N(i)）和它自己之间的相似系数：

$$e_{ij} = a([Wh_i||Wh_j]), where \quad j \in N(i)$$

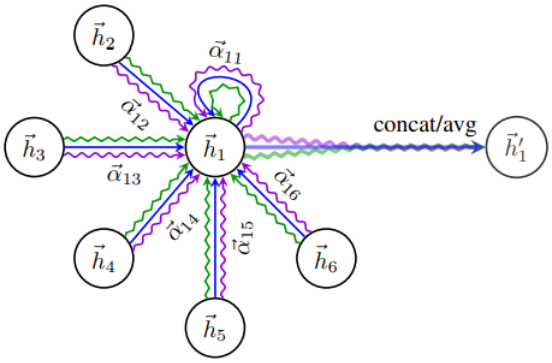


图 11: Model architecture of GAT.

其中共享一个参数 W 的线性映射对于顶点的特征进行了增维，是一种常见的特征增强（feature augment）方法；之后，$[\cdot||\cdot]$ 对于顶点 i, j 的变换后的特征进行了拼接；最后，a 将拼接后的高维特征映射到一个实数上，原论文通过 single-layer feedforward neural network 实现的。得到 e_{ij} 后输入 softmax 函数得到注意力系数 α_{ij}。

加权求和：根据计算好的注意力系数，把特征加权求和（aggregate）一下。

$$h_i^{'} = \sigma(\sum_{j \in N(i)} \alpha_{ij} W h_j)$$

Multi-head attention 可以理解成用了 ensemble 的方法，增加鲁棒性。

本次实验中，直接通过 PYG 中 GATConv 即可，效果与 GCN 差不多。

5.6 GIN

之前的 GNN 框架大多遵循递归邻域聚合（或者消息传递）框架，然而，新的 GNN 设计大多基于经验直觉、启发式和实验试错。目前，对神经网络的性质和局限性的理论认识较少，对神经网络表征能力的形式化分析也比较有限。GIN(图同构网络) 受到 Weisfeiler-Lehman (WL) 图同构测试的启发，WL 测试类似于 GNN，也通过聚合邻域节点特征来递归更新节点特征向量，以此来区分不同的图 [20]。

GIN 为了能够建模用于聚合的单射 multiset 函数，本文发明了 “deep multisets” 的理论，也就是用神经网络来参数化通用 multiset 函数。GIN 更新节点特征的模式如下

$$h_v^{(k)} = MLP^{(k)}\left((1+\epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}\right)$$

其中，在第一次迭代时，如果节点输入特征是 one-hot 向量则在相加前不需要 MLP，这是因为仅仅相加也可以保证单射。ϵ 可以是一个可学习的参数，也可以是一个固定的值。

整图分类任务，需要使用 readout 根据节点嵌入计算整图嵌入。本实验为节点分类，不再深入讨论。

$$h_G = CONCAT(READOUT(h_v^{(k)}|v \in G)|k = 0, 1, ..., K)$$

本次实验中，直接通过 PYG 中 GINConv 即可，效果与 GCN 差不多。

5.7 SGC

GCN 可以简单概括为三个步骤：邻居节点特征聚合、线性转换和非线性特征提取。SGC 论文中证明了 GCN 能 work 的原因不在于非线性特征提取，而仅仅是因为对邻居节点特征进行了聚合。

鉴于此，本文对 GCN 进行了简化，去掉了 GCN 中的非线性激活函数，将整个 GCN 简化为对预处理特征的直接多类逻辑回归，这种简化大大加速了训练过程，同时保持了不错的性能。 [21]

GCN 的卷积层有一个重要的功能：通过对邻居节点的特征进行平均以得到新的特征。这意味着，在 k 层之后，每一个节点将从它的所有 k 阶邻居节点获取特征信息，这种效果类似于卷积神经网络，GCN 深度增加可以增加内部特征的接受域。尽管卷积可以从增加深度中获益，但通常我们很难通过堆叠 MLP 来获得模型性能的提升。SGC 的作者大胆假设，GCN 中的非线性操作不是至关重要的，GCN 性能主要由局部邻居的特征平均来决定，也就是特征传播过程来决定。SGC 因此将 GCN 中的非线性操作去掉，只保留最终的 softmax 分类函数，以得到简化后的 GCN。值得注意的是，简化后的 GCN 是完全线性的。可以和 5.2 中 GCN 公式对比

$$Z_{SGC} = f(X, A) = softmax(\tilde{A}\tilde{A}...\tilde{A}XW^{(1)}W^{(2)}...W^{(l)})$$

通过 SGC，我领悟到，**这也可解释为何在图卷积网络中，通过堆叠 2 至 3 层便能取得良好效果而无需深层结构。其主要原因在于图卷积操作的局部性质，使得每一层能够逐步扩大节点的感受野（接受域），通过逐层的信息聚合，模型能够有效地捕捉到图结构中的特征和模式。在这种情境下，相较于进一步增加网络深度，更为重要的是通过有限的层数使得模型能够充分学习和表达图中的局部信息，而非深度的复杂性**。

本次实验中，直接通过 PYG 中 GGConv 即可，效果与 GCN 差不多，毕竟只是简化。

5.8 SAGE

大多数 graph embedding 框架是 transductive(直推式的), 只能对一个固定的图生成 embedding。这种 transductive 的方法不能对图中没有的新节点生成 embedding。相对的，GraphSAGE 是一个 inductive（归纳式）框架，能够高效地利用节点的属性信息对新节点生成 embedding [3]。

我们需要理解 transductive 与 inductive。统计机器学习可以分成两种: transductive learning, inductive learning，这里我们可以分别成为直推学习和归纳学习。transductive learning: To specific (test) cases, 指的是测试集是特定的 (固定的样本)。而 inductive learning: 测试集不是特定的。GCN 方法是 transductive learning，大多数节点嵌入模型都基于频谱分解/矩阵分解方法。而这些方法问题是矩阵分解方法本质上是 transductive 的。简而言之，transductive 方法在处理以前从未见过的数据时效果不佳。这些方法需要整个图形结构的节点在训练时都出现，以生成节点嵌入。如果之后有新的节点添加到 Gparh，则需要重新训练模型。而 GraphSAGE 方法学到的 node embedding，是根据 node 的邻居关系的变化而变化的，也就是说，即使是旧的 node，如果建立了一些新的 link，那么其对应的 embedding 也会变化，而且也很方便地学到。

GraphSAGE 的核心是，GraphSAGE 不是试图学习一个图上所有 node 的 embedding，而是学习一个为每个 node 产生 embedding 的映射。以论文中原图为例，图为红色的目标节点生成 embedding 的过程。k 表示距离目标节点的搜索深度，k=1 就是目标节点的相邻节点，k=2 表示目标节点相邻节点的相邻节点。GraphSAGE 第一步是采样，k=1 采样了 3 个节点，对 k=2 采用了 5 个节点；第二步是聚合邻居节点的信息，获得目标节点的 embedding；第三步是使用聚合得到的信息，也就是目标节点的 embedding，来预测图中想预测的信息。

GraphSAGE 的目标是基于参数 h 的相邻节点的某种组合来学习每个节点的表示形式。伪代码如算法 2。其算法每轮迭代两步 Aggregate 与 Update，论文中尝试了 Mean, Pool, LSTM 等多种 aggregator function。通过一轮 GraphSAGE 算法，我们将获得节点的新表示形式，原始图中的所有节点都遵循相同的过程。

我的 SAGE 模型如下，采用了 3 层图处理层，然后直接接 MLP，激活函数采用 ELU，防止 ReLU 陷阱。结果表现不错。

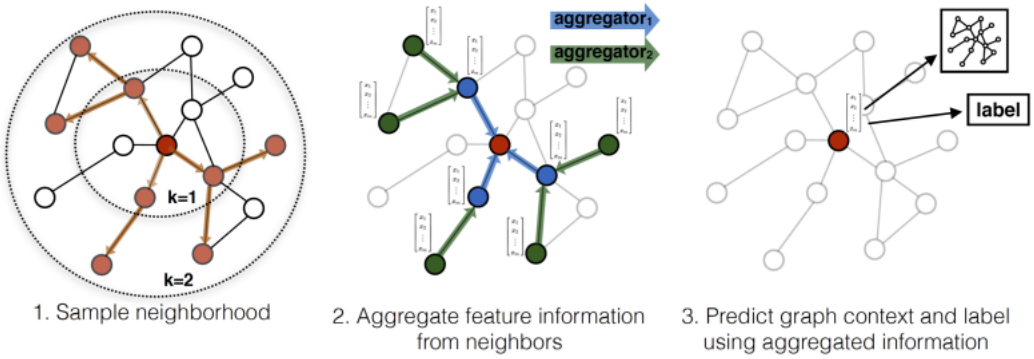


图 12: Visual illustration of the GraphSAGE sample and aggregate approach.

Algorithm 2: GraphSAGE embedding generation algorithm

Input : Graph $G(V, E)$ and related graph information, such as input features $x_v, \forall v \in \mathcal{V}$, neighborhood function $\mathcal{N} : v \rightarrow 2^v$

Output: Vector representations z_v for all $v \in \mathcal{V}$

$h_v^0 \leftarrow x_v$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $h_{\mathcal{N}_v}^k \leftarrow AGGREGATE_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $h_v^K \leftarrow \sigma(W^k \cdot CONCAT(h_v^{k-1}, h_{\mathcal{N}_v}^k))$;
  end
  $h_v^K \leftarrow h_v^K / ||h_v^K||_2, \forall v \in \mathcal{V}$;
end
$z_v \leftarrow h_v^K, \forall v \in \mathcal{V}$;

```python
class SAGE(nn.Module):
    def __init__(
        self,
        in_channels, hidden_channels, out_channels,
        edge_attr_channels=50, num_layers=3,
        dropout=0.1, bn=True,
        activation="elu",
    ):
        super().__init__()
        self.convs = nn.ModuleList()
        self.bns = nn.ModuleList()
        bn = nn.BatchNorm1d if bn else nn.Identity

        for i in range(num_layers):
            first_channels = in_channels if i == 0 else hidden_channels
            # second_channels = out_channels if i == num_layers - 1 else hidden_channels
            second_channels = hidden_channels
            self.convs.append(
                SAGEConv(
```

```
                (   first_channels + edge_attr_channels,
                    first_channels,),
                second_channels,
                )
            )
            self.bns.append(bn(second_channels))

        self.regression = nn.Sequential(
            nn.Linear(hidden_channels, 32),
            nn.Dropout(0.1),
            nn.ELU(),
            nn.Linear(32, out_channels),
        )
        self.dropout = nn.Dropout(dropout)
        self.activation = creat_activation_layer(activation)
        self.emb_type = nn.Embedding(12, edge_attr_channels)
        self.emb_direction = nn.Embedding(2, edge_attr_channels)
        self.reset_parameters()
```

另外我们采用 nll_loss() 即负对数似然损失作为 loss function，并采用 Adam 优化器来调参，并采用了学习率衰减（weight_decay）为 1e-7，以及 optim.lr_scheduler.StepLR 实现学习率的阶梯式衰减，即每 50 个 epoch，学习率乘 0.6，这样我们一开始可以使用较大的学习率。100 个 epoch 中，train auc 与 valid auc 的变化情况如图所示，出现了轻微的过拟合现象，最终在本地验证集上 AUC 值为 0.806。

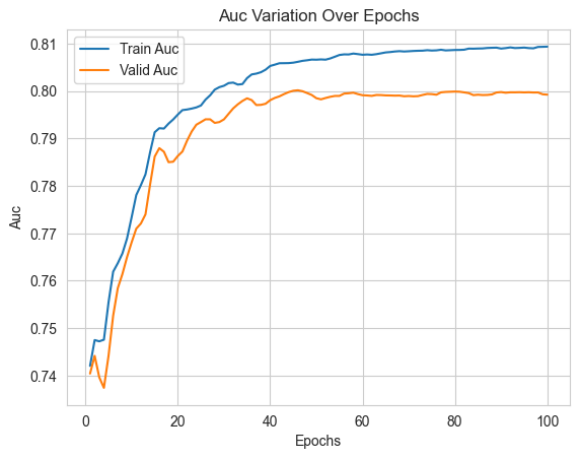


图 13: Auc Variation Over Epochs.

5.9 Care-GNN

对比 PC-GNN 解决的是伪装欺诈者在图上的不平衡问题，CARE 首先解决伪装欺诈者的端到端识别问题 [7]。伪装欺诈者的伪装主要分为两部分：

(a.) Feature camouflage：聪明的骗子可能会调整他们的行为，在评论中添加特殊字符，或使用深度语言生成模型来掩盖明确的可疑结果，这些有助于绕过基于特征的检测器；

(b.) Relation camouflage：骗子将自己与良好的实体关联在一起（比如定期发布评论或与有信誉的用户联系），以此逃避检测。

为此，论文提出了三种解决办法

(a.) 标签相似感知模型（label-aware similarity measure）
无监督相似度度量手段难以将伪装的特征识别出来，所以需要一个有监督的参数化的相似度度量手段。使用一层的 MLP 作为标签预测器，使用 L1 范式度量相似度。

$$D^{(l)}(v,v^{'}) = ||\sigma(MLP^{(l)}(h_v^{(l-1)}) - \sigma(MLP^{(l)}(h_{v'}^{(l-1)})||_1$$

$$S^{(l)}(v,v^{'}) = 1 - D^{(l)}(v,v^{'})$$

为了直接训练到标签的相似度度量参数，单独设置一个交叉熵损失函数：

$$\mathcal{L}_{simi}^{(l)} = \sum_{v\in\mathcal{V}} -log(y_v \cdot \sigma(MLP^{l}(h_v^{(l)})))$$

(b.) 相邻节点相似度选择（similarity-aware neighbor selector）
Top-p Sampling：对于第 l 层关系 r，过滤的阈值为 $p_r^{(l)} \in (0,1)$，邻居个数选择 $p_r^{(l)} \cdot S^{(l)}(v,v^{'})$。那么如何选择阈值 $p_r^{(l)}$ 呢，由于不存在梯度，无法使用 BP 算法，作者提出使用强化学习，我理解的这是通过强化学习来模拟反向传播过程，十分巧妙。

Action. 作者设计的 action 为，每次给 $p_r^{(l)}$ 加或减一个非常小的固定常数 τ，在第 l 层关系 r 下的 action 记为 $a_r^{(l)}$，我不妨记增加为 $a_r^{(l)} = +$，减去为 $a_r^{(l)} = -$。

Reward. p_r^l 是为了在第 l 层关系 r 下的所有节点中找到最相似的邻居，可以理解为距离 $D^l(v,v^{'})$ 尽量小。作者设计定义一个 epoch e 内的平均距离

$$G(D_r^{(l)})^{(e)} = \frac{\sum_{v\in V_{train}} D_r^{(l)}(v,v^{'})^{(e)}}{|V_{train}|}$$

之后，通过平均距离在每一个 epoch 间变化定义 Reward Function，两个 epoch 间，当平均距离减小时采取 $a_r^l = +$，平均距离增大时采取 $a_r^l = -$。

$$f(p_r^l, a_r^l)^{(e)} = \begin{cases} +1, & G(D_r^l)^{(e-1)} - G(D_r^l)^{(e)} \geq 0. \\ -1, & G(D_r^l)^{(e-1)} - G(D_r^l)^{(e)} < 0. \end{cases}$$

Terminal. 10 个 epoch 内总波动小于 2，即 $|\sum_{e-10}^{e} f(p_r^l, a_r^l)^{(e)}| \leq 2, where\ e \geq 10$。

(c.) 关系感知节点整合模型（relation-aware neighbor aggregator）
intra-relation(关系内聚合)，$\mathcal{E}_r^{(l)}$ 表示第 l 层关系 r 下的边。

$$h_{v,r}^{(l)} = ReLU\left(AGG_r^{(l)}\left(\{h_{v'}^{(l-1)} : (v,v^{'}) \in \mathcal{E}_r^{(l)}\}\right)\right)$$

inter-relation(关系间聚合)，采用对称差集得到关系间信息。

$$h_v^{(l)} = ReLU\left(AGG_{all}^{(l)}\left(h_{v'}^{(l-1)} \oplus \{p_r^{(l)} \cdot h_{v,r}^{(l)}\}|_{r=1}^{R}\right)\right)$$

AGG_{all} 可以是任意种聚合器，作者在论文里测试了 mean 聚合器、Weight 聚合器、attention 聚合器、GNN 聚合器，在 AUC 指标评价中，GNN 聚合器效果最好，mean 次之，本次实验中也采用了 GNN 聚合器。

Care-GNN 模型架构图如下，在每一层中，对于每一个节点 v_i，首先通过标签相似感知模型，寻找其相邻节点的相似度，之后我们通过相邻节点相似度选择模型，对每种关系过滤掉不相似的相邻节点，该选择模型使用强化学习的手段进行优化，在整合这一步，我们先进行内部关系整合，在进行关系交互整合。聚合完成后，对于节点 v，通过 L 层之后最终 embedding 为 $z_v = h_v^{(L)}$，最后使用 MLP 对节点嵌入 z_v 进行二分类，使用交叉熵损失函数。

$$\mathcal{L}_{GNN} = \sum_{v \in V} -log(y_v \cdot \sigma(MLP(z_v)))$$

Care-GNN 的损失函数也包括三部分的加和，分别是 $\mathcal{L}_{simi}$，$\mathcal{L}_{GNN}$ 以及 L2 正则化参数。

$$\mathcal{L}_{CARE} = \mathcal{L}_{GNN} + \lambda_1 \mathcal{L}_{simi} + \lambda_2 ||\Theta||_2$$

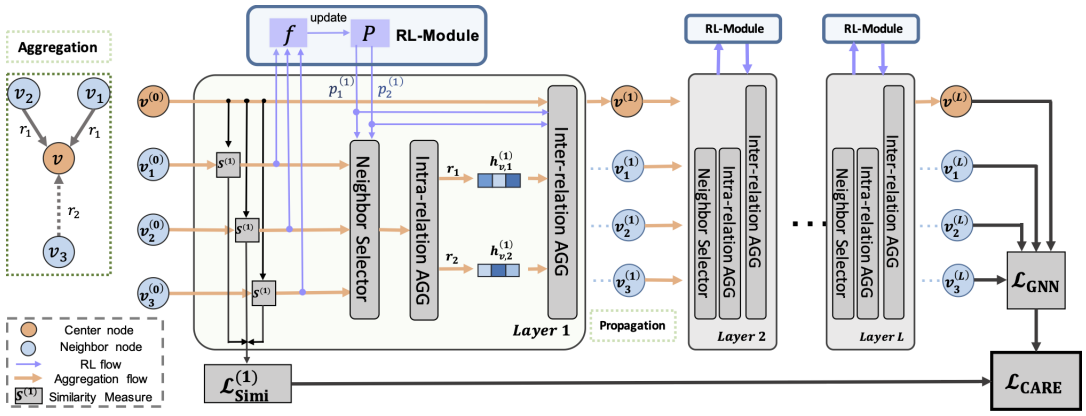


图 14: The aggregation process of proposed CARE-GNN at the training phase.

需要注意的是，Care-GNN 的输入包含由评论出发生成的三种关系图，以 Yelp 数据集 (包含用户评论) 为例，其中包含 R-U-R(同一用户发布的评论), R-T-R(同一产品相同星级评论，即 1-5 星) 以及 R-S-R(同一产品同一个月内评论) 三种图。本次实验中我需要处理的即是如何生成这三种图。我思考出了四种方法：

(a.) 全部采用 ALL 的数据图，即三种图全为全部的数据图。

(b.) 由于我不清楚我们实验数据集中 11 类边的具体含义，我通过数量聚合不同类型的边得到三种图。具体来说，我尝试了三种图的边数几乎相同以及三种图边数在 100 倍差距左右。

(c.) 计算边相似度，按照边相似度将 11 类边聚合为 3 类边，最后得到 3 张图输入。

(d.) 将模型的 3 种边增加到 11 种边，充分挖掘边的信息。

(a.), (b.) 两种方法最终得到的 AUC 结果差不多，均为 0.748 左右，Care-GNN 训练时 loss 变化如图，loss 波动较大我分析时由于每个 epoch 中数据几乎完全不同，由于数据的随机性或采样的影响，导致每个训练批次的数据分布差异较大。这样的情况可能导致模型在每个 epoch 内面临不同的样本分布，使得损失函数的优化过程受到更大的挑战，表现为损失的较大波动。在复现 Yelp 数据集以及 Amazon 数据集时也有同样的现象。

(c.) 方法中，我首先尝试了 Jaccard 相似度与余弦相似度，然后通过 K-Means 聚类将 11 类边聚类为 3 类边，然后作为 R-U-R, R-T-R 以及 R-S-R 三种图输入模型，效果仍一般。

(d.) 方法由于 11 种边中有的边个数极少，数据极不平衡，而且修改代码复杂期末时间短，并没有采用，这应该是在未来工作中重点尝试的，能否修改模型使得适应与任意种类数的边。

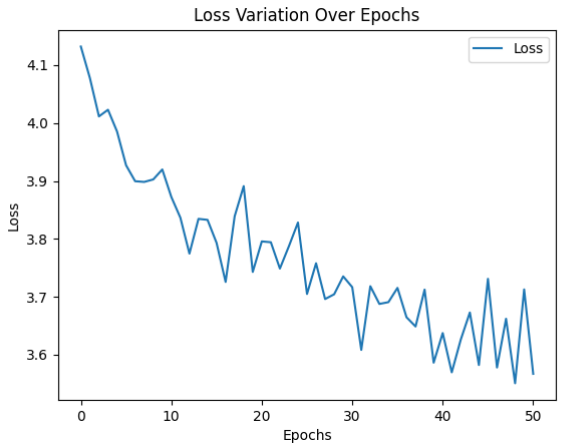


图 15: Loss Variation Over Epochs via method (a.)&(b.).

5.10 PC-GNN

PC-GNN 使用 pick and choose 的方法解决的金融关系拓扑图中类别不平衡的问题 [8]。作者提出了两个角度的三个挑战分别是:

- 应用角度: 一是关系上的冗余连接，与良好的实体的连接；二是缺少与欺诈节点直接连接（relation camouflage）;
- 算法角度: GNN 依赖于对邻居特征的聚合，类别数量不均衡会导致少数类特征被稀释。

根据论文，模型主要分为 pick、choose、aggregate 三部分，分别针对节点的平衡采样、节点采样后得到图的对邻居节点的上采样和下采样、节点嵌入表达。PC-GNN 的框架图如下，在第 l 层中，显示了一个包含 11 个节点的示例图。实线和虚线表示这些节点之间的两种关系。灰色节点为欺诈节点，白色节点为良性节点。

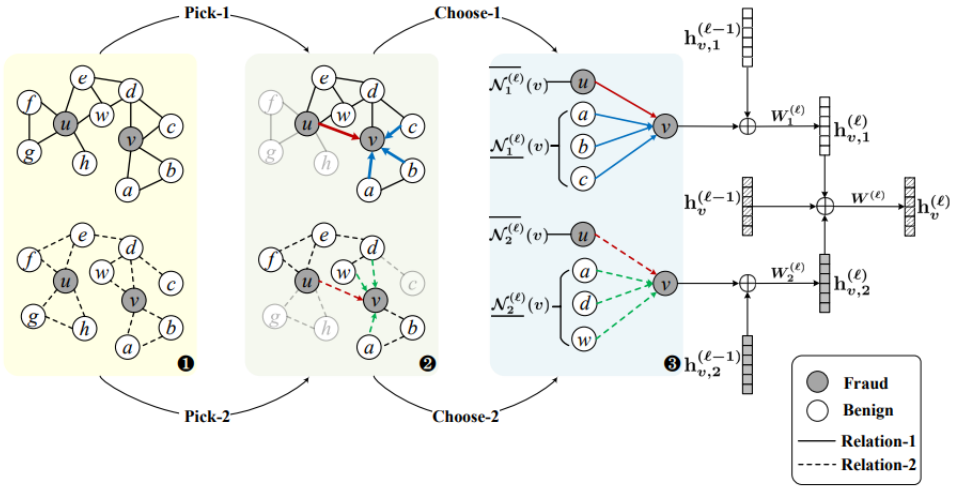


图 16: The figure demonstrates the l-th layer of the proposed PC-GNN framework on an example graph.

(a.) Pick: 进行标签平衡采样，对图中节点进行采样，每个节点的采样概率如公式，从公式可以发现，每个节点采样的概率和节点与其他节点的连接程度呈正比，和类别占比的多少呈反比，这

样可以采样更多邻居多且属于少数类别的节点，达到标签平衡采样的目的。采样完的节点连同它们的一阶邻居构成新的图。

$$P(v) \propto \frac{||\tilde{A}(:,v)||^2}{LF(C(v))}$$

其中，$\tilde{A}$ 为所有关系的邻接矩阵之和的归一化，$LF(C(v))$ 为 v 对应的类别标签的频率。

(b.) Choose：对图上的标签进行平衡采样后，节点附近仍然存在冗余的相似邻居而缺少和欺骗节点直接的联系，不利于节点嵌入的学习，所以要进行节点邻居的选择。对于属于多数类别的目标节点，只需下采样；对于少数类别的目标节点，同时进行上采样和下采样。$A_r(v,u)$ 表示邻接矩阵相连，$C(u)$ 表示类别标签频率。

下采样：

$$\underline{\mathcal{N}_r^l(v)} = u \in V | A_r(v,u) > 0 \ \ and \ \ D_r^{(l)}(v,u) < \rho_-$$

上采样：

$$\overline{\mathcal{N}_r^l(v)} = u \in V | C(u) = C(v) \ \ and \ \ D_r^{(l)}(v,u) < \rho_+$$

定义的距离函数 $D_r^{(l)}(v,u)$，使用了 L1 范数衡量节点嵌入之间的距离，该定义与 Care-GNN 类似，是有 MLP 输出得到的。

(c.) Aggregate：将每个关系下的得到的 l-1 层节点嵌入聚合得到各个关系的 l 层节点嵌入，之后再将所有关系下的节点嵌入聚合在一起形成总的 l 层嵌入表达。聚合也类似于 Care-GNN 操作。

本次实验中，PC-GNN 的输入与 Care-GNN 完全相同，最终得到的 AUC 结果为 0.751 左右，PC-GNN 训练时 loss 变化如图。

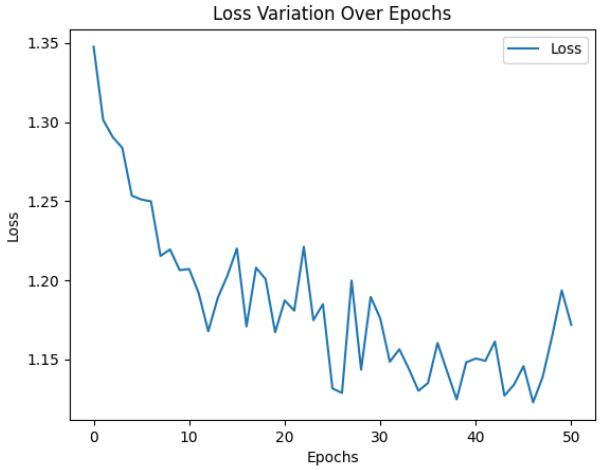


图 17: Loss Variation Over Epochs via PC-GNN.

5.11 模型融合

我考虑了两种模型融合的方法：其一是在看到题目中数据包含 timestamp 时，我考虑每个 timestamp 建立一个静态图模型，然后将 graph embedding 输入 LSTM 中最后进行分类，我在 2023CCF BDCI 中就采用了这种建模方法，效果较好；其二是每个图卷积最后输入 MLP 进一步学习高级特征和非线性关系，从而进行分类。

6 实验结果

SAGE 的效果最佳，其他 GNN 模型的效果相近，但均优于 CNN，此处所有模型用的数据集均为图数据工程添加特征后的数据集，且训练时采用邻居采样，具体结果如表所示。

表 3: Performance comparison on Fraud for opinion fraud detection.

Fraud	CNN	GCN	GAT	GIN	SGC	SAGE	Care-GNN	PC-GNN
AUC	0.7672	0.7865	0.7949	0.7898	0.7875	**0.8069**	0.7494	0.7513

从实验得出，是否进行邻居采样是影响模型性能的关键因素，本应效果最好的 Care-GNN 与 PC-GNN 在训练时直接采用的 mini_batch，每次采用小批量数据训练，但是并没有采用邻居采样，效果不尽人意。通过图特征工程添加数据也有一定提升。optim.lr_scheduler.StepLR 也实现学习率的阶梯式衰减，即每 50 个 epoch，学习率乘 0.6，使得在后期训练时不至于因为梯度大而出现严重波动和无法收敛的情况。

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=l2)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.6)
```

7 结论和展望

本次实验，在平台封榜前，我训练的模型已经达到满分（>78），并已经超过 80 分。在之前的实验过程中，我也遇到了很多问题，尤其在没有尝试特征工程数据升维前一度陷入僵局，不管尝试何种建模方法，都没有什么起色。后来通过搜集资料，了解到图特征工程，通过尝试 Graph Embedding 和增加构建图相关特征以及邻居采样，分类效果大大提升。

在实验中，我也学到了不同图神经网络模型的优缺点以及它们的适用场景。了解熟悉了 GCN、GraphSAGE 等等模型的原理和应用，这样的学习过程使我能够更明晰地选择适用于不同问题的图神经网络模型，并灵活运用它们来解决实际挑战。

另外，我还复现了 sota 模型作为 baseline 进行尝试，虽然效果不是最好，但这过程让我更深入地理解了论文中的算法和实验细节。这对于理解并应用前沿的研究成果有很大帮助。

总的来说，通过这次实验，我不仅提高了对图数据处理的能力，还学到了很多关于图神经网络的知识，这将对我的未来研究和工作选择产生积极的影响。

参考文献

[1] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11305–11312, 2019.

[2] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[4] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns, 2016.

[5] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

[6] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, WWW '19, page 2022–2032, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S. Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 315–324, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Pick and choose: A gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the Web Conference 2021*, WWW '21, page 3168–3177, New York, NY, USA, 2021. Association for Computing Machinery.

[9] Ronald D. R. Pereira and Fabrício Murai. Quão efetivas são redes neurais baseadas em grafos na detecção de fraude para dados em rede? In *Anais do X Brazilian Workshop on Social Network Analysis and Mining (BraSNAM 2021)*, BraSNAM 2021. Sociedade Brasileira de Computação - SBC, July 2021.

[10] Jun Ma and Danqing Zhang. Graphrad : A graph-based risky account detection system. 2018.

[11] Mark Weber, Jie Chen, Toyotaro Suzumura, Aldo Pareja, Tengfei Ma, Hiroki Kanezashi, Tim Kaler, Charles E. Leiserson, and Tao B. Schardl. Scalable graph learning for anti-money laundering: A first look, 2018.

[12] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs, 2019.

[13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[14] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1225–1234, New York, NY, USA, 2016. Association for Computing Machinery.

[16] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[17] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. Struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 385–394, New York, NY, USA, 2017. Association for Computing Machinery.

[18] 王改改. 【图文结合】一文详解数据不平衡问题. `https://zhuanlan.zhihu.com/p/296632599`.

[19] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[20] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ArXiv*, abs/1810.00826, 2018.

[21] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks, 2019.