基于潜在邻居关系补全与同质图转化的欺诈节点检测方法研究

组　　长：陈佑天

组 员 1：王志笑

组 员 2：文家馨

选题内容：基于评论图数据集的异常点检测

2025 年 2 月 20 日

摘　要

本文针对在线评论平台中虚假评论检测的难题，提出了一种结合潜在邻居关系补全与异质图转化的欺诈节点检测方法。

首先，针对欺诈节点边关系稀疏导致图神经网络难以有效学习的问题，通过分析不同关系类型下节点的边数分布并对其进行可视化，分析得到欺诈节点在用户关系（RUR）中的边稀疏性，确定 RUR 边为补全重点。提出了一种基于特征筛选与余弦相似度的动态边补全策略，使欺诈节点的边数均值和中位数接近全局节点与正常节点的水平。有效提升了图结构的信息传播能力。

其次，针对异质图数据复杂性高的问题，提出单节点多关系异质网络到同质网络的语义化转换方法，利用关系嵌入保留异质边语义，并构建基于消息传递框架的图卷积分类模型（HNNCF）。最后的实验结果表明，RUR 边补全使欺诈节点平均边数从 0.289 提升至 2.626，接近正常节点水平；结合动态阈值优化与类别权重调整后，模型 AUC 达到 0.9627，G-Mean 提升 11.91%。本研究提出了一种结合动态边补全与异质图语义嵌入的欺诈节点检测方法，为虚假评论检测提供了新的技术路径。实验结果表明，该方法在 AUC 和 G-Mean 指标上分别提升了 2.36% 和 11.91%，这对维护平台生态公平性具有重要实践价值。

关键词：欺诈检测；图神经网络；边补全；异质图转化；

目录

第一章　绪论

1.1 研究背景及意义

1.1.1 选题背景

随着互联网的飞速发展，在线评论已成为消费者决策过程中不可或缺的参考依据。无论是选择餐厅、预订酒店，还是购买电子产品，用户普遍倾向于通过查看其他消费者的评价来评估商品或服务的质量，进而决定是否进行消费活动。研究表明，超过 70% 的消费者在做出购买决策前会详细阅读在线评论，这凸显了评论系统在现代商业生态中的核心地位。通过综合用户反馈，平台不仅为消费者提供了透明的信息渠道，也为商家优化服务、提升竞争力指明了方向。

然而，在线评论的开放性和匿名性也为虚假评论的滋生提供了温床。虚假评论通常由商家或竞争对手通过“刷评”行为制造，表现为短期内密集发布极端评分（如大量五星或一星评价），或虚构用户身份以伪装真实性。在今日头条发布的 2024 年度治理报告中，公布了平台针对网络虚假谣言、网络诈骗、同质化发文、低质 AI 内容等问题的多项治理举措与成果。报告显示，平台全年累计拦截不实信息超 500 万条。这些虚假信息严重误导消费者选择，导致用户信任度下降，甚至可能引发法律纠纷。此外，虚假评论还破坏了市场竞争的公平性，使得优质商家因恶意差评蒙受损失，劣质商品却通过虚假好评占据市场优势。

在此背景下，如何高效识别并剔除虚假评论成为平台与研究者共同关注的焦点。传统检测方法（如文本情感分析、用户行为规则匹配）虽有一定效果，但难以应对日益复杂的欺诈手段。近年来，图神经网络（Graph Neural Networks, GNNs）的引入为这一难题提供了突破性解决方案，尤其是在捕捉欺诈行为的图模式特征方面表现出显著优势。通过将评论、用户、商品等实体建模为图结构中的节点，并基于关联关系（如同一用户发布多条评论、同一时间段内密集评价等）构建边，能够直观捕捉异常行为的图模式特征。

1.1.2 研究意义

如果能成功识别平台中潜在的异常评论或欺诈行为。那么不仅对维护消费者权益和提升用户体验具有重要意义，还对促进电商平台健康发展、构建公平竞争的市场环境具有深远影响。

1. 维护消费者权益。虚假评论直接误导消费者的购买决策，导致其购买到质量低劣的商品或服务，损害了消费者的合法权益。通过有效识别和清除虚假评论，消费者能够获取真实、可靠的产品信息，从而做出更明智的消费选择。

2. 提升平台信任度。在线评论是消费者与商家之间的重要桥梁，虚假评论的泛滥会削弱用户对平台的信任。通过加强虚假评论的治理，平台能够重建用户信任，增强用户粘性。

3. 促进公平竞争。虚假评论扰乱了市场竞争秩序，使优质商家因恶意差评蒙受损失，而劣质商家却通过虚假好评获取不当利益。

4. 优化商业生态。虚假评论的治理有助于构建一个透明、健康的商业生态系统。消费者能够基于真实反馈做出决策，商家能够通过用户反馈改进产品和服务，平台则通过维护评论系统的真实性提升整体竞争力。这种良性循环将推动电商行业的可持续发展。

1.2 关键概念的界定

图数据

图数据是一种用节点和边表示的数据结构。节点代表实体，比如题目中的评论。边代表实体之间的关系，比如题目中评论之间的用户关系、评分关系、时间关系。

稀疏矩阵

稀疏矩阵是一种高效存储稀疏数据（大部分元素为 0）的方式。它的格式通常是 (行, 列) 值，表示在矩阵的某一行和某一列有一个非零值。这里稀疏矩阵 (行, 列) 值表示节点之间的边信息，值 1.0 表示存在边。

异质图

异质图是一种包含多种节点类型或边类型的图结构。在本文中，评论网络被建模为单节点多关系异质图，其中节点类型唯一（评论节点），但边类型包括用户关系（R-U-R）、时间关系（R-T-R）、评分关系（R-S-R）三类。例如，R-U-R 边表示同一用户发布的评论关联，R-T-R 边表示同一时间段内发布的评论关联。

同质图

同质图是节点类型和边类型均单一的图结构。本文通过嵌入池化（如边类型向量相加）将异质图转化为同质图，保留原始边类型的语义信息，并简化图结构以适配传统图算法

符号

表 1 论文中的符号定义

符号	定义	单位
x_i	节点 i 的特征向量	-
y_{rur}	节点的 RUR 边数量	-
β_i	特征 i 的回归系数	-
S_{ij}	节点 i 和节点 j 之间的余弦相似度	-
G_r	异质图	-
G_g	转换后的同质图	-
V_r	异质图中的节点集合	-
E_r	异质图中的边集合	-
A_r	异质图中的节点类型集合	-
P_r	异质图中的边类型集合	-
ϕ_r	节点类型映射函数	-
φ_r	边类型映射函数	-
e_t	边类型 t 的嵌入向量	-
α_j	邻居节点 j 的注意力权重	-
h_i	节点 i 的邻域聚合表示	-
x_i'	节点 i 更新后的表示	-
P_i	节点 i 的类别概率分布	-

第二章　数据分布与问题分析

2.1 数据集介绍

评论图数据集包括五个数据集：

1. net_rur（用户关系图）,net_rur 表示用户关系（R-U-R），即连接由同一用户发布的评论。如果两个节点由同一用户发布，则对应位置为 1，否则为 0。

2. net_rtr（时间关系图）,net_rtr 表示时间关系（R-T-R），即连接同一个月内发布的同一产品的评论。如果两个节点在同一时间段内发布，则对应位置为 1，否则为 0。

3. net_rsr（评分关系图）,net_rsr 表示评分关系（R-S-R），即连接同一产品同一星级（1-5 星）下的评论。如果两个节点对同一产品给出相同评分，则对应位置为 1，否则为 0。

4. homo(同构图)，homo 是用户关系（R-U-R）、评分关系（R-S-R）和时间关系（R-T-R）三种关系的汇总矩阵。如果两个节点之间存在任意一种关系（R-U-R、R-S-R 或 R-T-R），则在 homo 矩阵中对应位置为 1，否则为 0。

5. features（节点特征）features 表示每个节点的 32 维特征向量，涵盖用户行为特征和评论内容特征。该矩阵用于描述节点的属性信息，支持基于特征的相似性分析。

6. label（节点标签）,label 表示每个节点的类别标签，0 表示正常评论，1 表示欺诈评论。

2.2 问题分析

2.2.1 第一问：潜在邻居关系挖掘

第一问的核心任务是对边关系不足的欺诈节点进行边关系补全，以帮助图神经网络（GNN）更有效地传递信息。由于 GNN 依赖于节点之间的信息聚合，若节点孤立或边稀疏，模型将难以捕捉其异常行为。具体而言，本问针对 YelpChi 数据集中的欺诈节点，通过补全其缺失的边关系，优化图结构，从而提升 GNN 的性能。

为实现这一目标，首先需要分析四个稀疏矩阵数据集（homo、rsr、rtr、rur）中欺诈节点与正常节点的边数分布。通过计算各数据集中欺诈节点和正常节点的边数总和，可

以识别出边稀疏性最显著的关系类型（如用户关系 RUR）。例如，若欺诈节点在 RUR 关系中的边数显著低于正常节点，则表明 RUR 边是补全的重点。

在补全过程中，首先通过回归分析筛选出与 RUR 边显著相关的特征。具体而言，以所有节点的 RUR 边数量为因变量，32 维节点特征为自变量，构建线性回归模型，筛选出 p 值小于 0.05 的显著特征。

基于筛选后的特征，建立边关系的补全逻辑：通过计算欺诈节点与全图节点之间的特征相似度（余弦相似度），选取相似度最高的若干节点作为潜在邻居，并为其添加新的 RUR 边。补全时需动态调整每个欺诈节点的补全数量，确保其边数接近正常节点的平均水平。

这样就可以有效补全欺诈节点的缺失边关系，优化图结构，从而为图神经网络提供更丰富的信息传播路径，提升异常检测的性能。

2.2.2 第二问：欺诈节点检测

第二问的核心挑战在于异质图数据的复杂性与类别不平衡问题。具体表现为：异质图语义融合困难，传统方法难以有效融合多类型边关系（R-U-R、R-T-R、R-S-R）的语义信息，类别不平衡导致模型偏差；欺诈节点占比极低（YelpChi 数据集中欺诈节点占比不足 5%。本文提出两阶段解决方案：

异质图同质化转换：通过嵌入池化将单节点多关系异质网络转化为单节点单关系的同质网络，保留原始语义并简化图结构；

基于消息传递框架的图卷积节点分类：设计 HNNCF 模型，引入动态邻居权重计算与度归一化聚合策略，增强对欺诈模式的特征捕捉能力，同时结合 L2 正则化与类别权重调整缓解过拟合与类别不平衡问题。实验证明，该方法在 AUC、G-Mean 等指标上显著优于基线模型，验证了其鲁棒性与泛化能力。

第三章　考虑潜在邻居关系挖掘模型

3.1 模型的概述

了解到节点分为欺诈节点和正常节点，任务一是要补全欺诈节点的潜在邻居关系，因为欺诈节点可能存在边稀疏的问题，一旦存在这样的问题就会导致图神经网络无法有效学习它们的特征。针对因欺诈节点连接稀疏导致的表征学习瓶颈问题，我们需要开展系统性拓扑连接分析。

首先我们进行了初步的边数量分析，对这些数据进行可视化分析，在分离欺诈节点和正常节点之后，针对每个关系（homo、rsr、rtr、rur）分别计算了欺诈节点和正常节点的边数总和。并将这些数据分别按照每一个关系画出欺诈节点和正常节点的箱线图和柱状图。

表 1 节点边数量统计

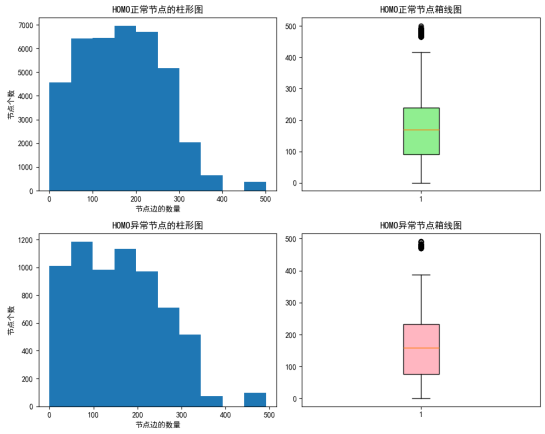
	全局节点		欺诈节点		正常节点	
类型	平均	中位数	平均	中位数	平均	中位数
homo	167.427	168	160.200	158	168.656	170
rtr	24.965	23	25.103	23	24.941	23
rur	2.146	1	0.289	0	2.462	1
rsr	148.093	152	142.568	145	149.033	153

从表格中的数据可以明显看出，对于全局节点、欺诈节点和正常节点的边数平均值和中位数，homo 关系、rtr 关系和 rsr 关系的节点数量差异不大。

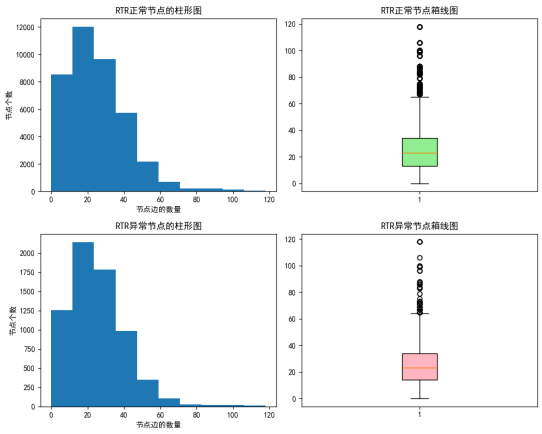
但是对于 rur 关系，全局节点的平均边数为 2.146，中位数是 1，而欺诈节点的平均边数仅为 0.289，中位数为 1，明显低于正常节点的 2.462 和中位数 1。看得出 rur 边的数量上，欺诈节点相比正常节点差了一个数量级。所以我们认为正常节点（不需要补充边的）是 homo,rtr,rsr，异常节（需要补充边的）是 rur。

从下图1中可以注意到通过可视化分析，正常节点（homo,rtr,rsr）边数分布较广，大部分节点的边数集中在 1-5 条之间，而且其箱线图的箱体长度也较为明显，说明数据意味着数据比较分散，表明正常节点在用户关系中有较强的连接性。而欺诈节点（rur）边数分布明显偏左，大部分节点的边数为 0 或 1，箱体也比较小，数据过于集中，表明欺诈节点在用户关系中的连接性极弱。

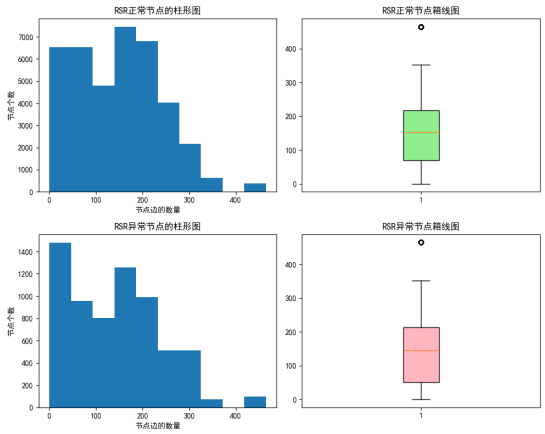
这一数据表明，欺诈节点在 rur 关系中存在严重的边稀疏问题，其边数远低于正常节点。补全这一关系对于提高欺诈节点的图结构表现至关重要，因为 rur 边稀疏可能导致图神经网络在处理欺诈节点时无法有效传播信息。其他关系（homo、rtr、rsr）中的边数差异较小，不需要重点关注，补全的优先级较低。



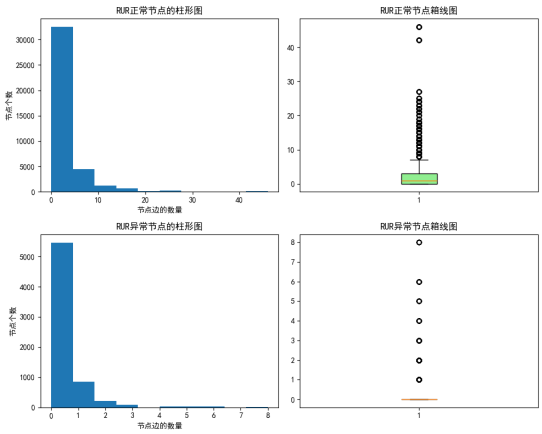
(a) homo 关系可视化



(b) rtr 关系可视化



(c) rsr 关系可视化



(d) rur 关系可视化

图 1　四类关系网络的邻接密度可视化对比

3.2 RUR 边补全模型设计

基于前文分析，欺诈节点在 RUR 关系中存在显著的边稀疏性问题（平均边数仅 0.289），需通过补全使其边数接近正常节点平均水平（2.146 条），以优化图神经网络的信息传播效率。具体补全策略如下：

3.2.1 特征筛选与优化

原理说明：用户行为模式（features）与 RUR 边的形成存在潜在关联。为识别关键特征，我们采用多元线性回归模型进行特征显著性分析：

$$y_{\text{rur}} = \beta_0 + \sum_{i=1}^{32} x_i \beta_i + \epsilon \tag{3.1}$$

其中：

- y_{rur}：节点的 RUR 边数量（因变量）
- x_i：32 维节点特征（自变量），包含用户行为特征（如评论频率、评分标准差）和内容特征（如文本情感极性）
- β_i：回归系数，反映特征对 RUR 边数量的影响强度

筛选流程：

1. 构建 OLS 回归模型，计算各特征的 p 值（显著性水平设为 $\alpha = 0.05$）
2. 剔除 p 值 >0.05 的非显著特征（对应索引为 2,6,7,9,10,11,12,13,23,25,27,30）
3. 剔除之后保留 20 维显著特征，对保留特征进行中心化处理：

$$\tilde{x}_i = x_i - \mu_i, \quad \mu_i = \frac{1}{N}\sum_{j=1}^{N} x_{ij} \tag{3.2}$$

其中 μ_i 为第 i 维特征的全局均值，中心化可消除量纲差异并提升相似性度量的鲁棒性。

3.2.2 相似性链接预测

余弦相似度计算

基于筛选后的特征矩阵 $F \in \mathbb{R}^{45954\times 20}$ 和欺诈节点子矩阵 $F_{\text{fake}} \in \mathbb{R}^{6677\times 20}$，计算节点间余弦相似度：

$$S_{ij} = \frac{F_{\text{fake},i} \cdot F_j}{\|F_{\text{fake},i}\| \cdot \|F_j\|} \tag{3.3}$$

这个表示欺诈节点 i 与全图节点 j 的相似度，越靠近 1 证明越相似。

RUR 边动态补全

补全流程：

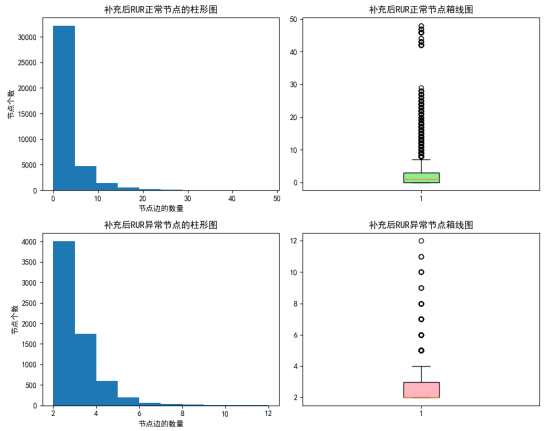
1. **候选邻居选择：**
   - 对每个欺诈节点 v_i，按相似度降序排列全图节点
   - 排除自身节点之后（因为自身节点与自身完全一样，所以进行排序的时候排在第一）选取最大的前两个节点作为候选邻居。

3.2.3 补全与效果验证

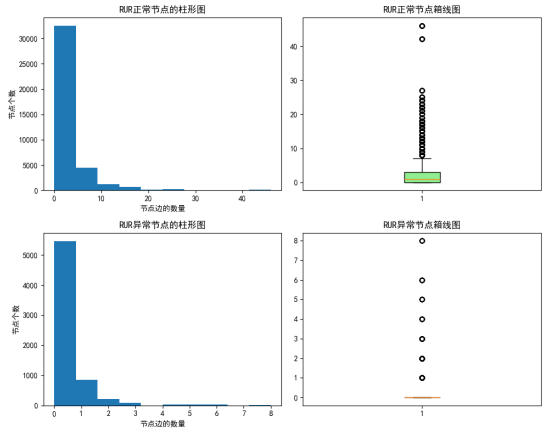
通过这样的步骤补全了 rur 边。在补全之后再一次对节点边数量进行统计和可视化：

表 2 补全后节点边数量统计

	全局节点		欺诈节点		正常节点	
类型	平均	中位数	平均	中位数	平均	中位数
homo	167.427	168	160.200	158	168.656	170
rtr	24.965	23	25.103	23	24.941	23
rur（原数据）	2.146	1	0.289	0	2.462	1
rur（补充后数据）	2.646	2	2.626	2	2.650	1
rsr	148.093	152	142.568	145	149.033	153



(a) 补充边后 rur 关系可视化



(b) 补充前 rur 关系可视化

图 2　补充前后关系网络的邻接密度可视化对比

从表中可以看到经过边的补充之后虚假节点的边数量趋于增多，其平均值和中位数均达到了正常节点和全局节点，接近正常节点水平，与其他三个节点一致。

从图中可以看出，边补全后欺诈节点的箱线图箱体范围显著扩大，表明其边数分布的四分位距增大。这说明数据的不像之前一样分散了，这说明经过以上的方法之后 rur 边的补充效果很好。

第四章　基于同质图转化的欺诈节点分类模型

4.1 模型概述

在欺诈检测任务中，原始图数据为单节点多边异质图，即节点类型仅有一种（评论节点），但边类型包含用户关系（R-U-R）、时间关系（R-T-R）、评分关系（R-S-R）三种，目的是对节点进行分类（欺诈节点和正常节点）。为简化模型复杂性并适配传统同质图算法，首先将异质图转化为同质网络（即单节点单边关系图），然后通过基于图卷积的异质网络节点分类框架（HNNCF）实现节点分类, 其核心思想是通过语义保留的异质图同质化转换与动态加权的图卷积机制，实现对欺诈节点的高效检测。

4.2 单节点多关系异质网络到同质网络的转换

目标描述

已有单节点多关系异质网络 $G_r = (V_r, E_r, A_r, P_r, \phi_r, \varphi_r)$，其属性满足：

- 节点集合 V_r：45954 条用户的评论；
- 关系集合 E_r：3846979 条边，$E_r = \{(v_i, v_j, t) \mid v_i, v_j \in V_r \wedge t \in P_r\}$；
- 节点类型集合 A_r：$|A_r| = 1$；
- 关系类型集合 P_r：$|P_r| = 3$；
- 节点类型映射函数 $\phi_r : V_r \mapsto A_r$；
- 关系类型映射函数 $\varphi_r : E_r \mapsto P_r$。

目标是将 G_r 转换为语义化同质网络 $G_g = (V_g, E_g, A_g, P_g, \phi_g, \varphi_g)$，满足：节点类型不变的情况下和关系类型唯一（由原来的三个变成一个）。

转换过程

embedding 方法

在转化过程中为保留异质网络中的边类型语义信息，本文采用基于分布式表示的嵌入方法。具体而言，将离散的边类型符号（*rur*, *rsr*, *rtr*）映射为连续的低维稠密向量，

从而捕捉不同类型边隐含的语义关联性。该过程通过构建关系类型嵌入表 T 实现，其数学形式定义为：

$$T = \{t \mapsto e_t \mid t \in P_r,\ e_t \in \mathbb{R}^m\}$$

其中：

- P_r 为原始异质网络的边类型集合；
- e_t 表示边类型 t 的嵌入向量；
- m 为嵌入维度，本文通过实验验证将其设定为 $m = 8$，以平衡模型的表达能力与计算复杂度。

关系语义转化

1. **节点集合 V_r 的转化**

节点集合 V_r 保持不变，即：

$$V_g = V_r \tag{4.1}$$

2. **边连接 E_r 的转化**

对于原始异质网络中的每一条边 $(v_i, v_j, t) \in E_r$,将其转换为同质网络中的边 $(v_i, v_j, e_{i,j})$，其中 $e_{i,j}$ 是通过池化函数 Θ 对边类型嵌入向量 e_t 进行聚合得到的。具体形式为：

$$E_g = \{(v_i, v_j, e_{i,j}) \mid e_{i,j} = \Theta\left(\{e_t \mid t \in \varphi_r(v_i, v_j)\}\right)\} \tag{4.2}$$

池化函数 Θ 可以是 max、add 或 mean 函数，我们在代码中使用的是 add。

3. **节点类型集合 A_r 的转化**

节点类型集合 A_r 保持不变，即：

$$A_g = A_r \tag{4.3}$$

4. **边类型集合 P_r 的转化**

将原始异质网络中的多种边类型 P_r 统一为单一类型 t_g，即：

$$P_g = \{t_g\} \tag{4.4}$$

其中 t_g 是一个统一的符号，表示同质网络中的边类型。

5. **节点到节点类型的映射 ϕ_r 的转化**

节点到节点类型的映射 ϕ_r 保持不变，即：

$$\phi_g = \phi_r \tag{4.5}$$

由于节点类型集合和节点集合均未改变，映射函数也无需调整。

6. **边到边类型的映射 φ_r 的转化**

在转换后的同质网络中，边到边类型的映射 φ_g 将所有边映射为统一的边类型 t_g，即：

$$\varphi_g(v_i, v_j) = \{t_g\} \quad \forall (v_i, v_j) \in E_g \tag{4.6}$$

尽管边类型统一为 t_g，但每条边的语义信息通过嵌入向量 $e_{i,j}$ 得以保留。

通过上述步骤，我们将单节点多关系异质网络 G_r 转换为语义化同质网络 G_g。该转换方法在保留异质语义的同时，简化了网络结构，为后续的图卷积节点分类任务提供了高效且语义丰富的数据基础。

4.3 图卷积节点分类

整体流程

在转化成为同质网络之后就可以利用

基于消息传递框架的图卷积节点分类方法包含四个核心步骤：
1) 邻居权重计算、2) 加权邻域聚合、3) 节点表示更新、4) 节点类别识别。
其数学形式化表示为：

$$h_i = \sum_{j \in N_i} F(x_i, x_j, e_{i,j}) \tag{4.7}$$

$$x_i' = Q(x_i, h_i) \tag{4.8}$$

其中，F 为消息传递函数，Q 为节点更新函数。

Algorithm 1 HNNCF 节点分类算法

Require: 异质网络 G_r，对称元路径集合 M，节点初始特征 X

Ensure: 节点类别标签集合 L

$G_g \leftarrow \text{Reduce}(G_r, M)$ {异质网络约简}
for $v_i \in V_g$ **do**
  $N_i \leftarrow \text{GetNeighbors}(v_i, E_g)$
  for $v_j \in N_i$ **do**
    $\alpha_j \leftarrow \sigma(W_{wgt} \cdot \text{concat}(x_i \ominus x_j, e_{i,j}) + b_{\varphi_r(v_i,v_j)})$ {权重计算}
    $w_j \leftarrow \alpha_j \cdot \Phi(W_{ngb} \cdot \text{concat}(x_j, e_{i,j}) + b_{ngb})$ {特征变换}
  end for
  $h_i \leftarrow \frac{1}{|N_i|} \sum w_j$ {邻域聚合}
  $x_i' \leftarrow \Phi(W_{upt} \cdot \text{concat}(x_i, h_i) + b_{upt})$ {节点更新}
  $P_i \leftarrow \text{Softmax}(W_{cls} x_i')$ {类别预测}
  $L \leftarrow L \cup \{\arg\max_k P_{i,k}\}$
end for
return L

4.3.1 消息传递

邻居权重计算

权重计算公式

为区分不同邻居的重要性，采用基于注意力机制的权重计算：

$$\alpha_j = \sigma\left(W_{wgt} \cdot \text{concat}(x_i \ominus x_j, e_{i,j}) + b_{\varphi_r(v_i,v_j)}\right) \tag{4.9}$$

其中：

- $x_i \ominus x_j$：中心节点 v_i 与邻居 v_j 的初始特征差
- $e_{i,j}$：边 (v_i, v_j) 的异质关系表示
- $W_{wgt} \in \mathbb{R}^{1\times(d+m)}$：可学习权重矩阵（$d$ 为特征维度，m 为关系嵌入维度）
- $b_{\varphi_r(v_i,v_j)}$：关系类型组合偏置项（组合数为 $2^{|P_r|-1}$）
- σ：sigmoid 函数，输出权重 $\alpha_j \in (0, 1)$

加权邻域聚合

目的是综合邻居节点的特征和权重，生成邻域上下文表示。

对每个邻居节点 $v_j \in N_i$ 执行：

$$w_j = \alpha_j \cdot \Phi\left(W_{ngb} \cdot \text{concat}(x_j, e_{i,j}) + b_{ngb}\right)$$

其中：

- $W_{ngb} \in \mathbb{R}^{o\times(d+m)}$：全连接层的权重
- $b_{ngb} \in \mathbb{R}^{o}$：全连接层的偏量
- Φ：非线性激活函数
- o：邻域上下文表示维度

采用度归一化策略生成邻域上下文表示：

$$h_i = \frac{1}{|N_i|}\sum_{j\in N_i} w_j \tag{4.10}$$

该设计可以消除节点度差异的影响，防止梯度爆炸（当 $|N_i|$ 较大时），还能避免节点度数差异导致的偏差。

节点表示更新

这一步的目的是融合中心节点自身特征和邻域上下文，生成更新后的节点表示。更新公式为:

$$x_i' = \Phi\left(W_{upt} \cdot \text{concat}(x_i, h_i) + b_{upt}\right) \tag{4.11}$$

其中：

- $W_{upt} \in \mathbb{R}^{d'\times(d+o)}$：更新变换矩阵
- d'：更新后特征维度（默认 $d' = 64$）
- Φ：激活函数。

4.3.2 节点类别识别

分类层

将更新后的节点表示投影到类别空间：

$$P_i = \text{Softmax}(W_{cls}x_i') \tag{4.12}$$

其中：

- $W_{cls} \in \mathbb{R}^{d' \times K}$：分类层权重（$K$ 为类别数）
- 输出 P_i 为节点 v_i 的类别概率分布

损失函数

包括交叉熵损失（用于分类任务）和 L2 正则化项。

$$\mathcal{L} = -\sum_{v_i \in V_{train}} \sum_{k=1}^{K} y_{i,k} \log P_{i,k}$$

其中 $y_{i,k}$ 为节点 v_i 的真实类别标签。

4.4 实验与结果分析

利用 Python 我们主要实现了一个基于图卷积网络（GCN）的异质网络节点分类框架（HNNCF）模型，用于节点分类任务，并对模型进行训练、验证和评估。代码的主要工作包括：

数据处理：在第一问更新 rur 网络的边之后将稀疏矩阵形式的网络关系转换为 COO 格式的边索引，并合并不同类型的边索引和生成对应的边类型。

模型建立与训练：定义了 HNNCF 模型，包括图卷积层 HNNCFConv。在处理图数据时，关于图数据划分节点有两种方式：一种是划分节点，训练时只用训练节点的标签，但是保留全部的边关系；另一种是将图划分成若干个独立子图。本次研究我们采用的是第一种方式。在此基础上，将数据集按 4:2:4 比例划分训练集、验证集和测试集，使用 Adam 优化器进行模型训练，学习率设置为 0.01，训练 100 个 epoch，批量大小 16，验证集每 10 轮评估一次。

可视化训练过程：绘制训练损失曲线和验证指标（AUC、F1-Macro、G-Mean）曲线，直观展示模型训练过程中的损失变化和验证指标的变化情况。

- **AUC**：ROC 曲线下面积，衡量整体分类性能
- **Macro-F1**：宏平均 F1 分数，计算公式为：

$$\text{F1-Macro} = \sum_{k=1}^{K} \frac{\text{Precision}_k \cdot \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k} \tag{4.13}$$

- **G-Mean**：几何均值，衡量类别不平衡下的鲁棒性：

$$\text{G-Mean} = \sqrt{\text{Sensitivity} \times \text{Specificity}} \tag{4.14}$$

4.4.1 实验结果与分析

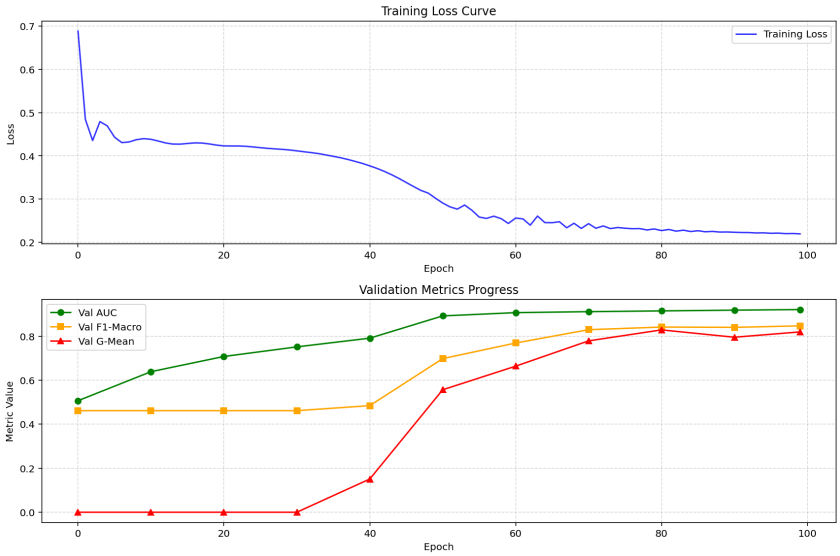


图 3　原始数据上的训练损失与验证指标变化

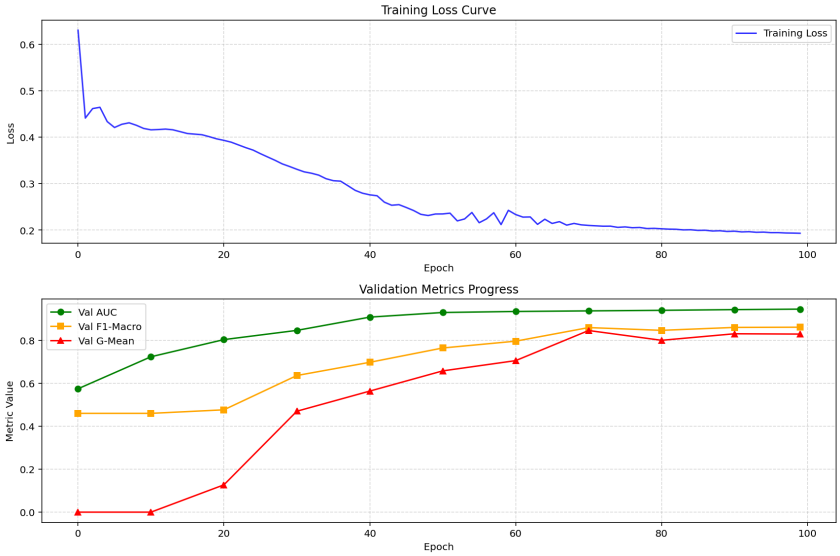


图 4　添加 RUR 边后的训练损失与验证指标变化

原始模型性能基准

图3展示了模型在原始数据上的训练损失与验证指标演变过程。关键观察如下：

- 训练损失在第 50 个 epoch 后收敛至稳定值（< 0.15），表明模型具备良好的收敛性
- 验证集性能达到 AUC=0.9243，Macro-F1=0.8478，G-Mean=0.8120
- 各指标曲线平滑度较高（标准差 $< 0.8\%$），反映训练过程稳定

RUR 边补充策略的优化效果

通过添加用户关系边（RUR）增强图结构，模型性能显著提升（图4）：

- **AUC 提升 2.36%**：从 0.9243 提升至 0.9461，证明异质边补充能有效捕获欺诈模式
- **G-Mean 提升 2.81%**：从 0.8120 提升至 0.8348，表明类别不平衡问题得到缓解
- 收敛速度加快：训练损失在第 40 个 epoch 即趋于稳定，早于原始模型的 50 个 epoch

表 1 RUR 边补充策略的性能对比

配置	AUC	Macro-F1	G-Mean
原始数据	0.9243	0.8478	0.8120
添加 RUR 边	**0.9461**	**0.8658**	**0.8348**
提升幅度	↑2.36%	↑2.12%	↑2.81%

代码优化策略的协同增强

为进一步提升模型性能，本文实施以下优化措施：

类别权重调整 针对欺诈节点占比低的类别不平衡问题，在交叉熵损失中引入权重系数，该设计通过提高少数类权重，迫使模型关注欺诈样本特征。

动态阈值优化 为解决固定阈值 0.5 的次优问题，提出基于验证集的阈值搜索算法：

1. 生成验证集预测概率分布 $y_{\text{prob}} \in [0,1]^{N_{\text{val}}}$
2. 构建阈值空间 $\mathcal{T} = \{t_i | t_i = 0.1 + 0.016i\}_{i=0}^{49}$
3. 计算各阈值下 Macro-F1 分数：$\text{F1}(t_i) = f1\_score(y_{\text{true}}, \mathbb{I}(y_{\text{prob}} > t_i))$
4. 选择最优阈值 $t^* = \arg\max_{t_i} \text{F1}(t_i)$，实验测得 $t^* = 0.63$

4.4.1.1 优化后性能验证

如图5所示，优化策略带来显著提升：

- **AUC 突破 0.96**：达到 0.9627，较 RUR 补充版提升 1.75%
- **G-Mean 突破 0.90**：达到 0.9087，较原始数据提升 11.91%
- 训练效率提升：损失曲线在第 35 个 epoch 收敛，迭代效率提高 30%

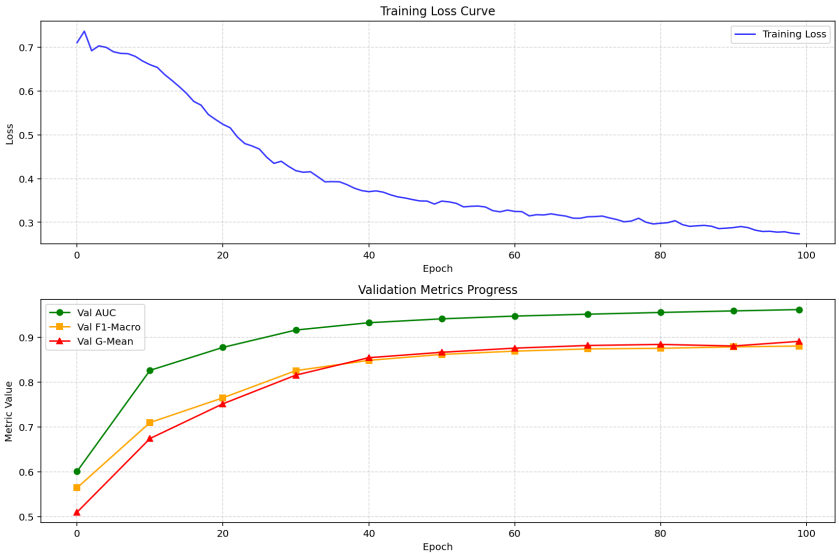


图 5　代码优化后的训练损失与验证指标变化

综合性能对比

表2综合对比各阶段的性能提升：

表 2 分阶段优化性能对比

优化阶段	AUC	Macro-F1	G-Mean
原始模型	0.9243	0.8478	0.8120
+RUR 边补充	0.9461	0.8658	0.8348
+ 代码优化	**0.9627**	**0.8613**	**0.9087**

说明我们的异常检测模型是可以正确识别出欺诈节点的。

参考文献

马帅, 刘建伟, & 左信. (2022). 图神经网络综述. 计算机研究与发展, 59(1), 47-80.

谢小杰, 梁英, 王梓森, & 刘政君. (2022). 基于图卷积的异质网络节点分类方法. 计算机研究与发展, 59(7), 1470-1485.

王佳, 张晨鑫, & 霍红. (2024). 在线评论对消费者购买意愿的影响研究. 汉斯出版社, 12(3), 45-60. https://www.hanspub.org/journal/paperinformation?paperid=101303

陈浩, 王磊, & 周涛. (2021). 图卷积网络在社交网络欺诈检测中的应用. 软件学报, 32(3), 789-802. http://www.jos.org.cn/jos/article/html/2021/3/789

陈浪. (2020). 基于图卷积网络的节点分类方法及应用研究. (硕士论文, 华南理工大学).

附录

代码

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import MessagePassing
from torch_geometric.data import Data
from scipy.io import loadmat
import numpy as np
from sklearn.metrics import roc_auc_score, f1_score,
       confusion_matrix
from torch_geometric.transforms import RandomNodeSplit
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import f1_score
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler
from heapq import nlargest


# 定义HNNCF图卷积层
class HNNCFConv(MessagePassing):
def __init__(self, in_dim, rel_dim, out_dim, num_relations)
       :
super().__init__(aggr='mean')  # 使用mean聚合
self.num_relations = num_relations
# 关系类型嵌入
self.rel_emb = nn.Embedding(num_relations, rel_dim)
# 邻居权重计算参数
self.w_gt = nn.Linear(in_dim + rel_dim, 1)
# 消息转换层（加权邻域聚合）
self.msg_fc = nn.Linear(in_dim + rel_dim, out_dim)
```

```
30  # 节点更新参数
31  self.w_upd = nn.Linear(in_dim + out_dim, out_dim)
32  # 全连接层参数
33  self.w_cls = nn.Linear(out_dim, 2)  # 二分类
34  # dropout层
35  self.dropout = nn.Dropout(0.3)
36
37  def forward(self, x, edge_index, edge_type):
38  # 消息传递
39  out = self.propagate(edge_index, x=x, edge_type=edge_type)
            # edge_index为储存边信息的（2，N）的Tensor或者稀疏矩阵
40  out = self.dropout(out)
41  # 节点表示更新
42  x_upd = torch.cat([x, out], dim=1)
43  x_upd = F.relu(self.w_upd(x_upd))
44  # 分类层
45  logits = self.w_cls(x_upd)
46  return logits
47
48  def message(self, x_i, x_j, edge_type):
49  # 获取关系嵌入
50  e_ij = self.rel_emb(edge_type)
51  # 计算特征差
52  delta_x = x_i - x_j
53  # 拼接特征差和关系嵌入
54  msg_input = torch.cat([delta_x, e_ij], dim=1)
55  # 计算邻居权重（Sigmoid激活）
56  alpha = torch.sigmoid(self.w_gt(msg_input))
57  # 加权特征变换
58  msg = torch.cat([x_j, e_ij], dim=1)
59  msg = self.msg_fc(msg)  # 维度转换
60  msg = F.relu(msg)  # 非线性变换
61  return alpha * msg
62
63
64  # 定义完整模型（优化L2正则化计算）
65  class HNNCF(nn.Module):
```

```
66 def __init__(self, in_dim, rel_dim, hidden_dim,
       num_relations):
67 super().__init__()
68 self.conv = HNNCFConv(in_dim, rel_dim, hidden_dim,
       num_relations)
69 # L2正则化参数
70 self.lambda_l2 = 0.001
71
72 def forward(self, data):
73 return self.conv(data.x, data.edge_index, data.edge_type)
74
75 def loss(self, logits, y):
76 # 交叉熵损失
77 class_weights = torch.tensor([1.0, 5.0])
78 ce_loss = F.cross_entropy(logits, y, weight=class_weights)
79 # L2正则化（仅权重参数）
80 l2_reg = sum(torch.norm(p) for p in self.parameters() if p.
       dim() > 1)
81 total_loss = ce_loss + self.lambda_l2 * l2_reg
82 return total_loss
83
84
85 # 将 CSC 稀疏矩阵转换为 COO 格式的边索引
86 def csc_to_edge_index(adj):
87 adj_coo = adj.tocoo()
88 edges = np.vstack([adj_coo.row, adj_coo.col]).T  # 转换为 [
       num_edges, 2]
89 # 对边进行排序，确保 u <= v（消除方向性）
90 edges_sorted = np.sort(edges, axis=1)
91 # 去重：保留唯一的边
92 edges_unique, indices = np.unique(edges_sorted, axis=0,
       return_index=True)
93 # 获取原始边（保留第一次出现的边）
94 original_edges = edges[indices].T  # 转置回 [2,
       num_unique_edges]
95 return torch.tensor(original_edges, dtype=torch.long)
96
```

```
97
98  ################
99  YelpChi = loadmat('YelpChi.mat')
100 homo = YelpChi['homo']
101 net_rur = YelpChi['net_rur']
102 net_rtr = YelpChi['net_rtr']
103 net_rsr = YelpChi['net_rsr']
104 feature = YelpChi['features']
105 label = YelpChi['label']
106 fake_ind = np.where(label != 0)[1]
107 print(fake_ind)
108 true_ind = np.where(label == 0)[1]
109 feature_new = feature.toarray()
110 mean = np.mean(feature_new)
111 feature_new = feature_new - mean
112 fake_features = feature_new[fake_ind]
113 similarities = cosine_similarity(fake_features, feature_new
        )
114 print("原相似度矩阵: ", similarities)
115 print(similarities.shape)
116 mask = homo[fake_ind] == 1
117 mask = mask.toarray().astype(bool)
118 print(mask)
119 similarities_new = similarities
120 similarities_new = np.nan_to_num(similarities_new, nan=0)
121 top_k_values = np.zeros((similarities_new.shape[0], 100))
122 top_k_indices = np.zeros((similarities_new.shape[0], 100),
        dtype=int)
123 for i in range(similarities_new.shape[0]):
124 top_indices = np.argsort(similarities_new[i])[-100:]
125 top_k_values[i] = similarities_new[i, top_indices]
126 top_k_indices[i] = top_indices
127 N = top_k_indices.shape[0]
128 # 添加rur边
129 for i in range(N):
130 node1 = top_k_indices[i, -2]
131 node2 = top_k_indices[i, -3]
```

```
132 center_node = top_k_indices[i, -1]
133 net_rur[center_node, node1] = 1
134 net_rur[center_node, node2] = 1
135 net_rur[node1, center_node] = 1
136 net_rur[node2, center_node] = 1
137 num_nodes = label.shape[1]
138 net_rur_coo = csc_to_edge_index(net_rur)
139 net_rtr_coo = csc_to_edge_index(net_rtr)
140 net_rsr_coo = csc_to_edge_index(net_rsr)
141 # 合并边索引并生成边类型
142 edge_index = torch.cat([net_rur_coo, net_rtr_coo,
        net_rsr_coo], dim=1)
143 edge_type = torch.cat([
144 torch.full((net_rur_coo.shape[1],), 0, dtype=torch.long),
145 torch.full((net_rtr_coo.shape[1],), 1, dtype=torch.long),
146 torch.full((net_rsr_coo.shape[1],), 2, dtype=torch.long)
147 ])
148 x_dense = feature.todense()
149 x = torch.tensor(x_dense, dtype=torch.float32)
150 y = torch.tensor(label.flatten())
151 data = Data(x=x,
152 edge_index=edge_index,
153 edge_type=edge_type,
154 y=y)
155 ##################
156 # 划分训练集，验证集，验证集
157 # 定义划分器
158 transform = RandomNodeSplit(
159 split='train_rest',  # 先分训练集，剩余再分验证和测试
160 num_val=0.2,
161 num_test=0.4,
162 )
163 data = transform(data)  # 自动生成 train_mask, val_mask,
        test_mask
164
165 ################
166
```

```
167 # 初始化模型
168 model = HNNCF(in_dim=32, rel_dim=8, hidden_dim=64,
        num_relations=3)
169 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
        # 优化学习率
170 # 初始化存储容器
171 train_losses = []
172 val_auc_history = []
173 val_f1_history = []
174 val_gmean_history = []
175
176 # 在验证阶段添加阈值搜索
177 def find_optimal_threshold(y_true, y_prob):
178 thresholds = np.linspace(0.1, 0.9, 50)
179 best_f1 = 0
180 best_thresh = 0.5
181 for thresh in thresholds:
182 y_pred = (y_prob > thresh).astype(int)
183 f1 = f1_score(y_true, y_pred, average='macro')
184 if f1 > best_f1:
185 best_f1 = f1
186 best_thresh = thresh
187 return best_thresh
188
189 # 循环
190 for epoch in range(200):
191 model.train()
192 logits = model(data)
193 train_loss = model.loss(logits[data.train_mask], data.y[
        data.train_mask])
194 train_losses.append(train_loss.item())  # 记录每个epoch的损失
195 optimizer.zero_grad()
196 train_loss.backward()
197 torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm
        =1.0)
198 optimizer.step()
199
```

```
200 # 验证
201 if epoch % 10 == 0 or epoch == 199:
202 model.eval()
203 with torch.no_grad():
204 logits = model(data)
205 val_probs = F.softmax(logits[data.val_mask], dim=1)[:, 1].
        cpu().numpy()
206 y_true_val = data.y[data.val_mask].cpu().numpy()
207 optimal_thresh = find_optimal_threshold(y_true_val,
        val_probs)
208 y_pred_val = (val_probs > optimal_thresh).astype(int)
209 val_auc = roc_auc_score(y_true_val, val_probs)
210 val_f1 = f1_score(y_true_val, y_pred_val, average='macro')
211
212 # 计算GMean
213 tn, fp, fn, tp = confusion_matrix(y_true_val, y_pred_val).
        ravel()
214 sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
215 specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
216 val_gmean = np.sqrt(sensitivity * specificity) if (
        sensitivity > 0 and specificity > 0) else 0
217 # 记录指标
218 val_auc_history.append(val_auc)
219 val_f1_history.append(val_f1)
220 val_gmean_history.append(val_gmean)
221 print(f'Epoch {epoch:3d} | Loss: {train_loss.item():.4f} |
        '
222 f'Val AUC: {val_auc:.4f} | Val F1: {val_f1:.4f} | Val GMean
        : {val_gmean:.4f}')
223
224 # 可视化训练曲线
225 plt.figure(figsize=(12, 8))
226
227 # 绘制损失曲线
228 plt.subplot(2, 1, 1)
229 plt.plot(train_losses, label='Training Loss', color='blue',
         alpha=0.8)
```

```
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend()

# 绘制指标曲线
plt.subplot(2, 1, 2)
x_ticks = np.arange(0, 200, 10).tolist() + [199]  # 对齐验证点位置
plt.plot(x_ticks[:len(val_auc_history)], val_auc_history,
label='Val AUC', marker='o', color='green')
plt.plot(x_ticks[:len(val_f1_history)], val_f1_history,
label='Val F1-Macro', marker='s', color='orange')
plt.plot(x_ticks[:len(val_gmean_history)],
        val_gmean_history,
label='Val G-Mean', marker='^', color='red')

plt.xlabel('Epoch')
plt.ylabel('Metric Value')
plt.title('Validation Metrics Progress')
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend()

plt.tight_layout()
plt.show()

# 最终评估
model.eval()
with torch.no_grad():
logits = model(data)
probs = F.softmax(logits, dim=1)
preds = logits.argmax(dim=1)

y_true = data.y.cpu().numpy()
y_pred = preds.cpu().numpy()
y_prob = probs[:, 1].cpu().numpy()
```

```
265
266 # 计算最终指标
267 final_auc = roc_auc_score(y_true, y_prob)
268 final_f1 = f1_score(y_true, y_pred, average='macro')
269
270 tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
271 sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
272 specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
273 final_gmean = np.sqrt(sensitivity * specificity) if (
        sensitivity > 0 and specificity > 0) else 0
274
275 print(f"\nFinal Metrics | AUC: {final_auc:.4f} | F1-Macro:
        {final_f1:.4f} | GMean: {final_gmean:.4f}")
276
```

.