软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(9):4036−4055 [doi: 10.13328/j.cnki.jos.007267] [CSTR: 32375.14.jos.007267]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于多模态融合的软件缺陷协同分派方法*

谢生龙[1,2,3], 李青山[1,3], 歹 杰[1,3], 崔 笛[1,3]

[1](西安电子科技大学 计算机科学与技术学院, 陕西 西安 710126)

[2](延安大学 数学与计算机科学学院, 陕西 延安 716000)

[3](西安市智能软件工程重点实验室 (西安电子科技大学), 陕西 西安 710126)

通信作者: 李青山, E-mail: qshli@mail.xidian.edu.cn

摘 要: 软件缺陷 (bug) 分派是将 bug 报告与适合解决该 bug 的开发人员进行匹配的过程, 能够使 bug 得到及时修复. 目前的 bug 分派研究大多集中于 bug 报告的文本分类, 但根据帕累托法则, 用以分类的 bug 报告存在数据分布不均衡现象, 容易对非活跃开发者产生较差的分派效果; 此外, 现有的分类模型忽视了对开发人员的建模且难以挖掘 bug 与开发人员之间的相关性, 影响了 bug 分派效能. 为此, 提出一种基于多模态融合的软件缺陷协同分派方法 CBT-MF (collaborative bug triaging method based on multimodal fusion). 该方法首先对 bug 报告进行预处理并构造 bug-开发人员二部图; 其次, 为了缓减 bug 修复记录分布不均衡性的影响, 通过 K-means 和正负采样的方法对二部图数据进行增强; 为了表征开发者信息, 基于图卷积模型提取二部图节点特征; 最后, 采用内积匹配的方法捕获 bug 与开发者的相关性, 并通过贝叶斯个性化排序实现 bug 报告与开发人员的推荐与分派. 在公开数据集上进行全面的实验评估, 实验结果表明, CBT-MF 在 bug 分派方面相较于多个现有先进方法表现出更优越的性能.

关键词: 缺陷分派; 不均衡性; 多模态融合; 图卷积

中图法分类号: TP311

中文引用格式: 谢生龙, 李青山, 歹杰, 崔笛. 基于多模态融合的软件缺陷协同分派方法. 软件学报, 2025, 36(9): 4036–4055. http://www.jos.org.cn/1000-9825/7267.htm

英文引用格式: Xie SL, Li QS, Dai J, Cui D. Collaborative Bug Triaging Method Based on Multimodal Fusion. Ruan Jian Xue Bao/Journal of Software, 2025, 36(9): 4036–4055 (in Chinese). http://www.jos.org.cn/1000-9825/7267.htm

Collaborative Bug Triaging Method Based on Multimodal Fusion

XIE Sheng-Long[1,2,3], LI Qing-Shan[1,3], DAI Jie[1,3], CUI Di[1,3]

[1](School of Computer Science and Technology, Xidian University, Xi'an 710126, China)

[2](College of Mathematics and Computer Science, Yan'an University, Yan'an 716000, China)

[3](Xi'an Key Laboratory of Intelligent Software Engineering (Xidian University), Xi'an 710126, China)

Abstract: Bug triaging is the process of assigning bug reports to developers suitable for resolving the reported bugs, ensuring timely fixes. Current research in bug triaging mainly focuses on the text classification of bug reports. However, according to the Pareto principle, the data distribution of bug reports used for classification is unbalanced, which may lead to ineffective triaging for inactive developers. Additionally, existing classification models often neglect to model developers and struggle to capture the correlations between bugs and developers, affecting the efficiency of bug triaging. To address these issues, this study proposes a collaborative bug triaging method based on multimodal fusion (CBT-MF). This method first preprocesses bug reports and constructs a bug-developer bipartite graph. To mitigate the impact of the unbalanced distribution of bug fix records, the bipartite graph data is enhanced using K-means clustering and positive-negative sampling. To represent developer information, node features are extracted from the bipartite graph using a graph convolutional

* 基金项目: 国家自然科学基金 (U21B2015, 62202357); 陕西省自然科学基础研究计划 (2023-JC-QN-0744); 陕西省科协青年人才托举计划 (20220113); 西安电子科技大学杭州研究院概念验证基金 (XJ2023230039)
收稿时间: 2023-10-25; 修改时间: 2024-01-09, 2024-05-30; 采用时间: 2024-08-04; jos 在线出版时间: 2025-01-24
CNKI 网络首发时间: 2025-01-26

network model. Finally, correlations between bugs and developers are captured by matching inner products, and Bayesian personalized ranking (BPR) is utilized for bug report recommendation and triaging. Comprehensive experiments conducted on publicly available datasets demonstrate that CBT-MF outperforms several state-of-the-art methods in bug triaging.

Key words: bug triaging; imbalance; multimodal fusion; graph convolution

随着现代软件工程技术的不断发展, 特别是开源软件生态的普及, 软件在商业、金融、医疗、电子政务、工业制造等领域的融合日益加深. 然而, 由于软件项目自身规模和用户需求复杂度的不断增长, 其内部存在着不可避免的缺陷 (bug)[1]. 根据 VulDB 漏洞资料库上的统计数据显示, 仅在 2022 年 1 月–10 月 5 日期间[2], Google Chromium 累计记录的安全 bug 数量达 303, 位居同类软件安全缺陷榜第 1 名, Mozilla Firefox 累计记录的安全 bug 数为 117. 目前, Chromium 项目已收到了超过 149 万个 bug 报告, Mozilla 项目已收到了超过 120 万个 bug 报告[3]. 因此, 软件的 bug 对政企核心数据安全保护、用户行为合规性审计、新型计算平台可靠性等方面形成了重大威胁[4]. 为了持续提升软件的服务性能和增强其安全性, 维护人员需要将发现的 bug 及时分派给开发人员, 以确保 bug 得到及时修复. 因此, bug 分派已成为软件安全维护过程中关键的活动之一. 软件 bug 被记录在如图 1 所示的 bug 报告中, 其中包含 bug ID、bug 修复状态、开发人员 ID、bug 摘要、bug 具体描述, 以及与 bug 有关的评论等关键信息. 这些信息有助于维护人员了解 bug 的严重程度、影响范围及可能的修复群体, 从而更有效地完成 bug 分派.

图 1 一个 Google Chromium bug 报告实例

在当前的 bug 分派研究工作中, 许多方法将其视为一个文本分类问题[3,5]. 例如, 基于规则的分派方法利用预定义的规则[6], 先根据 bug 报告文本构建语义空间, 再将一个 bug 映射为一组特征向量, 并根据规则分类的结果, 将 bug 分派给具有对应标签的开发人员[3,7–9]. 虽然这类方法很有效, 但是它们仍然存在一些不足: (1) bug 报告存在修复记录分布不均衡性. 由于 bug 修复记录具有集群特点[10], 且活跃度高的开发者修复 bug 记录多, 而活跃度低的开发者修复 bug 记录少, 导致 bug 数据集具有典型的帕累托分布. 因此, 用于分类器学习的修复记录具有潜在的开发者-bug 关联不均衡性. (2) 分类技术忽视了对开发人员的建模. 开发者与 bug 是两个对等的实体, 但基于 bug 报告文本的分类把开发人员简单地作为一个类标签, 进行 bug 报告文本多标签分类, 这导致 bug 分派时对 bug 和开发者建模失衡, 忽视了对可以揭示开发人员专业背景、兴趣及能力等关键信息的表征. (3) 分类模型难以捕捉 bug 与开发者的相关性. 分类模型训练时都基于一个共同的假设, 即把现实软件开发活动中密切相关的 bug 和开发人员看作两个独立的实体进行学习. 然而, 由于二者的语义向量函数缺乏对关键协作信号的显式编码, 导致其难以准确表达隐藏在 bug-开发者交互过程中的信号. 因此, 无法充分利用它们之间相关性进行 bug 和开发人员相似性匹配.

多模态表示学习[11]具有强大的机器学习能力, 能够使模型在学习多种模态数据后对目标对象进行互补性表示, 常见的文本和视频[12]、文本和图像[13]等结合的多模态学习表示已充分证明了这一点. 此外, 特别是协同表示

学习, 作为多模态表示学习的主要类别, 它在给定的约束下可以分别处理不同模式的数据, 并将它们带入协同空间. 我们最近的研究[14]也表明, 将 bug 报告文本和 bug-开发人员二部图进行融合, 比单一的文本信息更能够全面地表达 bug 报告和开发人员的语义以及他们之间的相关性. 因此, 为了解决上述问题, 本文受多模态机器学习和协同过滤 (collaborative filtering, CF) 在推荐系统及社交网络等领域对实体关系挖掘的预训练模型启发[15,16], 将 bug 报告文本数据和 bug-开发者二部图数据作为两种不同模态的数据, 提出了一种多模态融合的软件缺陷协同分派方法——CBT-MF, 如图 2 所示. 首先, 为了减小噪声影响, 通过文本清洗、Word2Vec[2]等技术进行 bug 报告文本预处理, 同时, 为了挖掘 bug 和开发者之间的相关性, 构建了 bug-开发人员二部图. 其次, 针对 bug 修复记录的不均衡性, CBT-MF 采用 K-means 和正负采样的方法对数据样本进行增强和二部图重构, 降低了由于 bug 修复记录不均衡性带来的过拟合风险. 随后, 为了丰富 bug 和开发者的表征信息并挖掘 bug 与开发者关系, 基于图卷积神经网络模型融合 bug 报告文本模态和二部图模态数据, 经多层卷积后捕获了二部图的节点语义和结构特征, 实现了 bug 和开发者的深度表征及二者关系的提取. 最后, 为了实现 bug 和开发者的相关性匹配, CBT-MF 将 bug 分派任务转化为图协同过滤 (graph collaborative filtering, GCF) 的 bug-开发人员二部图链接预测, 并基于内积匹配和贝叶斯个性化排序 (Bayesian personalized ranking, BPR) 得到了 bug 分派的推荐方案, 最终实现了 bug 分派推荐.

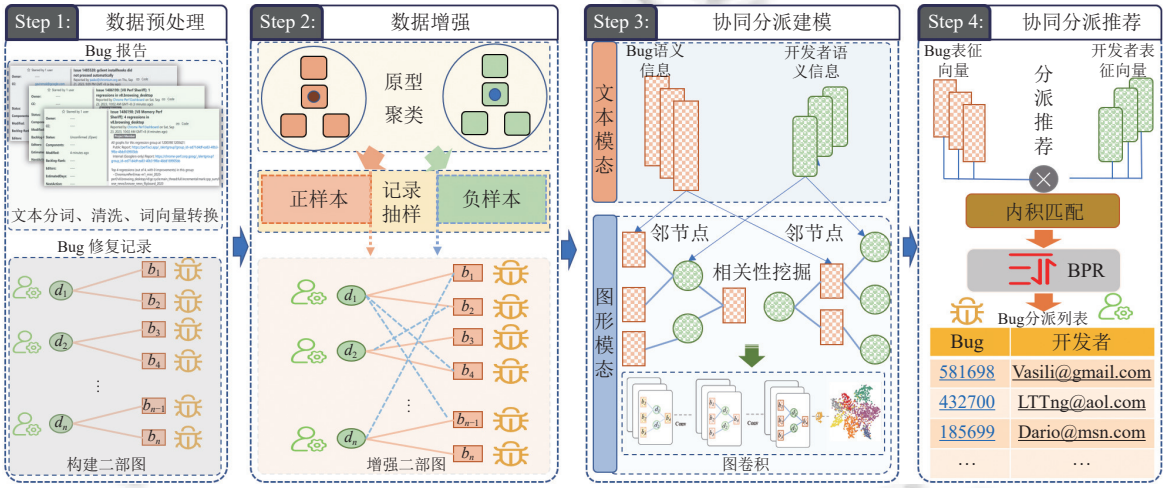


图 2 CBT-MF 缺陷分派思路

具体地讲, 本文主要研究贡献包含以下 4 点.

(1) 设计一个 CBT-MF 方法框架. 该框架由数据预处理、数据增强、多模态数据融合及 bug 分派推荐这 4 部分构成. 在该框架的指导下, 可以缓解 bug 修复记录不均衡性影响, 并充分地利用开发者信息挖掘、建模其与 bug 的相关性, 提高 bug 分派效能.

(2) 提出一个 bug 和开发者文本与图数据的增强方案. 一方面采用 K-means 对 bug 报告进行原型聚类; 另一方面, 通过正负采样的方式进行新记录抽样. 缓解了两种模态数据中修复记录不均衡性对分派预测结果的影响, 为深度挖掘 bug 和开发者之间的相关性提供了支持.

(3) 构建一个基于图卷积神经网络的协同分派模型. 通过定义节点之间的消息传递方式和聚合规则, 进行节点特征表示的更新, 并将 bug 报告和开发者的特征及其关系映射到同一个协同空间, 从而实现 bug 报告文本和二部图数据的融合, 丰富了 bug 和开发者表征信息, 更好地捕捉 bug 和开发者之间的相关性.

(4) 在 Google Chromium、Mozilla Core 及 Mozilla Firefox 数据集上对 CBT-MF 进行了广泛的实验评估. 通过与文本模态分类、相关性匹配及基于 CF 的 3 类方法对比, 评估了 CBT-MF 对 bug 分派的性能; 又通过分组对比和消融实验证明了 CBT-MF 可以缓解数据不均衡性与文本数据模态对 bug 分派产生的影响; 此外, 还详细分析了 CBT-MF 对采样尺度、聚类质心数、图卷积层数的敏感性. 这些评估实验为 CBT-MF 后续研究与应用提供了建

设性参考.

以上为本文的研究背景. 本文第 1 节介绍 3 类 bug 分派方法与相关研究工作. 第 2 节介绍本文所需的基础知识, 包括 bug 分派的定义, bug 向量和开发人员向量的初始化等. 第 3 节详细介绍 CBT-MF 方法步骤. 第 4 节通过一系列实验验证所提方法的优越性和有效性. 第 5 节对本文的工作进行总结与展望.

1 相关工作

本文主要研究如何融合多模态数据进行软件 bug 的协同分派, 实现更高效、更准确的 bug 修复. 本节将从文本分类、相关性分析及推荐这 3 个方面的 bug 分派技术出发, 对相关工作进行阐述.

1.1 基于文本分类的 bug 分派

Bug 分派长期以来被认为是一种文本分类问题[17]. 早期就有学者研究了基于文本分类的 bug 分派方法, 他们将每个开发者看作是一个类标签, 而将 bug 报告作为分类的对象, 对其进行分类预测, 大量的实验也证明了该类方法的有效性[5,18]. 后来, 为了进一步提高 bug 分派的效率, 诸如支持向量机 (SVM)[7,18]等机器学习技术逐步应用于 bug 分派的研究中. 例如, CBR[5]将 bug 文本转换为向量, 并使用 SVM 分类器对它们进行分类后分派. 再后来, 随着自然语言处理技术的发展, 一些研究借助循环神经网络[19]等模型增强了 bug 报告的语义表示. 如, DBRNN-A[16]作为一种深度双向带注意机制的递归神经网络, 能够在更长的上下文中保留单词的顺序和语义关系. 然而, 此类方法多数是基于收集的 bug 报告数据集进行的, 其中普遍存在不均衡性问题[10]. 原因在于, bug 数据在软件模块中的分布大致符合帕累托法则. 例如, 在实际的 bug 数据集中, 高活跃度开发者的 bug 修复数量要远超过低活跃度开发者修复的 bug 数量, 这导致分类模型在预测时偏向高活跃度开发者, 而对低活跃度开发者的预测精度较低. 因此, 在分类的过程中难以对低活跃度开发者修复的 bug 进行有效的预测. 故这类方法只适用于 bug 数据较为均衡的情形. 此外, 在这些分类模型中, 通常将开发人员作为简单的标签, 事实上开发人员的编程兴趣可能比较广泛, 以至于用简单的标签不足以表征其 bug 修复的经验或特长. 于是部分研究工作就把希望寄托在了与开发人员有关的上下文信息, 比如他们在开源代码管理平台上的活动[20]和提交 bug 报告的历史记录[21]等. 尽管获取到的这些信息与开发人员的兴趣或能力有关, 但是仅凭分类模型还是难以深入地挖掘 bug 与开发人员之间的相关性, 这在很大程度上影响了 bug 分派效能.

1.2 基于相关性分析的 bug 分派

从分派的结果来看, bug 分派是 bug 报告与开发人员的匹配问题, 其主要任务是相关性分析. 因此, 有研究人员提出了诸如图学习等信息检索的方法来捕获他们之间的相关性. 例如, GRCNN[22]作为一个图循环卷积神经网络, 采用一种映射变化的方法来学习每个节点的时空隐含表示, 并取得了较好的效果; BERT[23]在对现有的目标模型微调后, 能够为手动分类的历史记录添加新开发人员; Yang 等人[24]提出了一种基于多特征模型和主题模型的 bug 严重性预测方法, 希望提升 bug 分派效能. 然而, 这类方法的基本观点是, 认为具有某种 bug 专业知识的开发人员就能够解决与其相似类型的新 bug. 因此, 不难看出这类技术同样是基于 bug 报告中开发人员过去修复的 bug 和新 bug 的相似性进行的, 这同样存在文本分类技术遇到的问题.

为了缓解这一问题, 近年来有部分研究将相关性分析工作的重点从 bug 报告文本转向了与其有关的图模态数据. 例如, PMGT[15]采用共同的用户活动来关联项目, 并基于多源辅助信息构建一个同质的项目关系图, 从而为项目的推荐提供了统一的相关性视图. 此外, 二部图作为一种特殊的图模态数据, 是软件维护过程中描述二元关系的有效工具, 其能够通过边来增强节点之间的语义关系, 进而支持实体相关性分析, 现已被广泛用于与相关性分析有关的场景中. 例如, 在电商平台中[25,26], 为了构建用户和商品之间的关系, 可以将用户和商品分别看作二部图的两个集合, 从而实现个性化商品匹配; 在社交网络中[27,28], 可以将所有用户分为两个不同的集合, 分别表示用户和其好友, 从而基于二部图来挖掘用户和好友之间的关系. 二部图在这些场景中的广泛应用, 为本文的研究工作提供了启发.

1.3 基于协同推荐的 bug 分派

推荐系统也常被用于 bug 报告的分派, 尤其是基于 CF 的推荐系统, 其可以将用户和项目视为独立的个体. 例

如, 通过矩阵分解将每个用户和项目的 ID 投影为特征向量, 并在它们之间进行卷积来预测交互[14]. 但在个别场景下由于缺乏与特定属性或与 bug 修复相关的数据, 导致推荐系统会遇到冷启动和数据稀疏性的问题[18], 致使推荐系统无法识别出合适的开发人员予以推荐. 此外, 虽然 CF 通过卷积可以强制将用户和项目向量映射到一个彼此观测到的范围, 但有时也难以挖掘二者的关系. 如 LightGCN 模型[29]表现出了较为突出的效率和性能, 但它的线性特征不足以揭示用户和项目之间复杂的非线性关系. 为此, 最近的研究集中于利用深度学习技术来强化交互函数. 例如, NCF[30]采用非线性神经网络作为交互函数, 捕获了用户和项目之间的非线性交互特征; CBCF[31]采用内容增强的 CF 将现有的 CBR[5,31]和 CF 推荐器相结合, 提高了 bug 分派推荐质量. 再后来, 为了增强协同过滤向量表征能力, 人们还投入了大量的精力来整合可用的项目或开发人员信息, 如项目内容[32,33]、社会关系[34]、项目关系[35]、用户评论[36]和外部知识图谱[37]等, 希望采用数据增强的方式来提高推荐效果.

早期的数据增强主要是对用户-项目矩阵进行数据推断[38], 最近的研究主要围绕 GAN 等深度模型[39]来生成新的数据. 后来, 在 GCF 的研究工作中出现了一些更为实用的数据增强技术. 例如, SGL[40]通过边缘缺失来生成结构特征; UserSim[41]采用监督式生成对抗网络的方式模拟用户. 在 GCF 中无论选择何种技术来解决数据稀疏性问题, 他们均表明, GCF 作为一种面向相关性推荐的范式[42,43], 具有处理图形等非结构化多模态数据的绝对优势. 为了实现更泛化的 bug 与开发人员数据增强, 已有研究利用 bug-开发人员之间的相关性揭示开发人员的专业能力, 以及在修复记录中的开发人员与 bug 的隐式关系. 例如, 通过计算 bug b_i 和已被开发人员 d_j 修复的 bug 之间的语义相似度, 实现 bug b_i 和开发人员 d_j 的匹配[14]. 然而, 在分析二者的相关性时, 完全丢弃 bug 报告文本数据而将重心仅放在 bug 与开发者所形成的图数据上时, 也会丢失图结构不能表达的重要信息, 如开发者专业背景、能力、兴趣及 bug 的严重程度等. 这使得最终的分派结果也会不尽人意. 因此, 在多模态机器学习技术兴起的浪潮下, 软件维护领域许多研究已将多模态技术用于挖掘和处理来自 bug 的多个模态数据信息. 例如, DeMoB[44]将 bug 报告和源文件视为两种模态, 并通过动态的神经网络捕获两种语言之间的差异, 以提高 bug 定位准确率. 此外, Wan 等人[45]基于文本语义、语法树结构、控制流图等多模态信息的代码表征学习, 开发了一种综合性的多模态表示方法来表示源代码的非结构化和结构化特性, 实现更全面的代码理解.

总结上述相关研究工作, 不难发现目前基于 bug 报告文本分类的方法受到 bug 数据不均衡性影响, 导致模型在预测时偏向多数类, 对少数类的预测精度较低; 而基于相关性分析的方法多是以文本数据模态为主, 通过信息检索的方式匹配, 这忽视了对开发者的建模; 基于协同推荐的 bug 分派受到冷启动和数据稀疏性等问题的影响, 难以直接应用于 bug 分派. 因此, 借鉴已有场景的解决方案, 本文创造性地将 bug 分派建模为二部图中的相关节点的匹配问题, 并基于协同分派模型融合 bug 报告文本和 bug-开发者二部图模态数据, 强化 bug 和开发者的语义信息, 为 bug 节点集合与开发人员节点集合建立联系, 深入挖掘 bug-开发者的相关性, 最后借助协同推荐技术实现 bug 分派. 这有望开辟软件维护工作的新思路.

2 基本定义

本文提出的 CBT-MF 方法主要基于 bug 报告文本和 bug-开发人员二部图两种模态数据进行分析, 旨在挖掘出 bug 和开发人员的相关性. 本节将先介绍 bug 分派的形式化定义, 以及表征 bug 与开发人员的向量等基础概念.

2.1 bug 分派的定义

本文对分派过程中两个关键对象集合: bug 集合 B 和开发人员集合 D, 有如下定义:

$$B=\{b_i \mid i=1,2,\ldots,n_B\},\quad D=\{d_j \mid j=1,2,\ldots,n_D\} \tag{1}$$

其中, n_B 和 n_D 分别表示 bug 和开发人员的数量. b_i 包含 bug 报告摘要、描述和开发人员 d_j 唯一的 E-mail 地址等信息; d_j 包含表示开发人员 ID 的 E-mail、性别、年龄及学历等信息. 此时, 可进一步将分派记录集 T 定义为:

$$T=\{t_{ij} \mid i=1,2,\ldots,n_B,\ j=1,2,\ldots,n_D\} \tag{2}$$

其中, 元素 t_{ij} 刻画了 bug b_i 和开发人员 d_j 的相关性, 表示二者之间是否存在分派关系, 具体地:

$$t_{ij}=\begin{cases}1, \text{ if } b_i \text{ is assigned to } d_j\\ 0, \text{ if } b_i \text{ is not assigned to } d_j\end{cases} \tag{3}$$

因此, 本文所述的 bug 分派任务是确定 bug b_i 和开发人员 d_j 之间的相关性 t_{ij}. 若分派预测结果 $t_{ij}=1$, 则表示将 bug b_i 分派给开发人员 d_j; 否则, 不予分派.

2.2 向量初始化

bug 报告文本通常由图 1 所示的相关字段组成. 在 bug 报告提交时, bug 跟踪系统会要求填写一个表单以说明与该 bug 有关的信息. bug 被分派后, 开发人员会根据表单中的信息去解决该 bug. 此类表单一般包含对 bug 观察到的摘要、详细描述、bug 报告或修改的时间戳, 有时还包括 bug 执行的堆栈等信息. 因此, bug 报告文本数据的规模较大, 可能会给预测模型带来噪声和维度灾难. 根据相关研究的建议[46], 本文先对 bug 报告文本做两个方面必要的预处理.

一方面, 首先将 bug 摘要和描述字段合并为一个文档, 作为新的 bug 报告文本; 其次, 通过删除十六进制代码和出现频率高但对预测模型毫无意义的停用词噪声文本, 例如, 删掉“the”“in”“that”等; 并使用自然语言工具包(NLTK) 对摘要和描述信息进行标点符号过滤, 例如, 将 Client_Conect_Server_Failure, 转换为 ClientConect-ServerFailure; 再根据 CamelCase 命名约定将其拆分为单个词, 例如, 将 ClientConectServerFailure 拆分为 Client、Conect、Server 和 Failure; 最后, 将所有文本都转换为小写. 另一方面, 从 bug 报告中提取开发人员的邮箱、性别、年龄及学历等字段信息, 并采用与 bug 报告类似的处理过程进行预处理. 例如, 邮箱地址 RonaldReagan890721@gmail.com, 经过预处理后得到了 ronald、reagan890721、gmail 及 com 等去噪后的开发者 ID 信息.

按照上述预处理过程得到的词规模仍然较大, 这会导致词向量矩阵的维数很高. 接下来, 采用 Word2Vec 方法[47]在维基百科语料库上预先训练的 Skip gram 模型[48]来获取词向量. 对于语料库中不存在的词, 采用随机化的方法进行初始化. 最终对于每个 bug b_i 都会提取一个 w 维语义向量 $p^0_{b_i}$. 本文参考文献 [49], 将 w 设置为 5. 同样, 对于开发人员向量, 用表示其 ID 的 E-mail 将其初始化为一个单向量 $q^0_{d_j}$. 如果还获取到了开发者性别、年龄、专业、学历及偏好等其他开发人员属性, 则根据需要, 也可以将这些属性按照同样的方式进行编码后连接在一起.

这种初始化方式虽然简单, 但可以从特定项目和维基百科语料库中补充与 bug 分派具有相关性的词向量, 有助于在后续的模型学习中获取与开发人员编程能力等有关的特征表示. 举一个简单的例子, 对于 Google Chromium bug 1485528 报告 (https://bugs.chromium.org/p/chromium/issues/detail?id=1485528) 的摘要“gclient installhooks did not proceed automatically”, 若将 Word2Vec 维度设置为 5 时, 其被转换后最终可获得如下所示的语义向量:

$$\begin{bmatrix}\text{gclient}\\ \text{installhooks}\\ \text{did}\\ \text{not}\\ \text{proceed}\\ \text{automatically}\end{bmatrix}=\begin{bmatrix}[-0.068\ 107\ 32 & -0.018\ 928\ 03 & 0.115\ 371\ 47 & -0.150\ 432\ 75 & -0.078\ 722\ 07]\\ [0.146\ 235\ 32 & 0.101\ 405\ 24 & 0.135\ 153\ 86 & 0.015\ 257\ 31 & 0.127\ 017\ 81]\\ [-0.036\ 320\ 35 & 0.057\ 531\ 6 & 0.019\ 837\ 47 & -0.165\ 704\ 3 & -0.188\ 976\ 36]\\ [0.147\ 610\ 1 & -0.030\ 669\ 43 & -0.090\ 732\ 26 & 0.131\ 081\ 03 & -0.097\ 203\ 21]\\ [-0.142\ 336\ 17 & 0.129\ 177\ 45 & 0.179\ 459\ 77 & -0.100\ 308\ 56 & -0.075\ 267\ 43]\\ [-0.010\ 724\ 54 & 0.004\ 728\ 63 & 0.102\ 066\ 99 & 0.180\ 185\ 47 & -0.186\ 059]\end{bmatrix}_{6\times 5}$$

3 CBT-MF 方法设计

如前文图 2 所示, 本文提出的 CBT-MF 方法核心包含了 4 个模块: 数据预处理、数据增强、协同分派建模及协同分派推荐. 其中, 每一个模块对应一个关键步骤, 接下来将对其进行逐一介绍.

3.1 数据预处理

CBT-MF 使用了 bug 报告的文本模态数据和 bug 跟踪系统中已修复记录所构造的 bug-开发人员二部图模态数据. 首先, 将全部的 bug 报告文本经过第 2.2 节描述的过程进行预处理并完成初始化, 最终转换为矩阵向量, 从而形成 bug 报告文本模态的数据表示; 然后, 根据 bug 跟踪系统中的修复记录, 将所有 bug 作为一个节点集 B, 所有开发人员作为另一个节点集 D, 并将每个 bug 节点 b_i 与相关开发人员节点 d_i 连接后形成代表修复关系的边 e_{ij}, e_{ij} 基于式 (3) 的 t_{ij} 表示了该 bug 与这些开发人员的相关性. 此时, 可以初步构建一个表达 bug 和开发人员信息及

关系的二部图模态数据 $H=(V, E)$. 其中, 节点集 $V=B\cup D$, 边集 $E=\{e_{ij}\,|e_{ij}=\langle b_i,t_{ij},d_j\rangle\}$.

3.2 数据增强

为了缓解 bug 报告文本与二部图模态数据中 bug 修复记录不均衡性对分派结果的影响, 在此模块中, 首先采用 K-means 方法对 bug 向量集进行聚类[50], 以发现与每个待分派的 bug 报告相似的 bug, 并将相似的 bug 分派给相应的开发人员; 同时进行正负采样, 进一步增强原始数据, 并实现对 bug-开发人员二部图的重构. 具体通过下述两个过程实施.

(1) 对 bug 向量原型聚类. 根据第 3.1 节的数据预处理, 对 bug 向量 $p_{b_i}^0$ 初始化后, 接着采用 K-means 聚类来计算每个 bug 的聚类质心, 挖掘出如下式表达的语义原型:

$$\sum_{B}\log P\left(p_{b_i}^0\,|\,\Theta\right)=\sum_{B}\log\sum_{C^B}P\left(p_{b_i}^0,c_k^i\,|\,\Theta\right) \tag{4}$$

其中, $p_{b_i}^0$ 为 bug b_i 的语义向量, 其对应的原型质心为 c_k^i, 且所有质心都位于空间 C^B 中, Θ 为相关参数集. 这里可以采用 Elbow[51]或枚举观测等方法来确定 bug 向量集合适的 k 值.

(2) 进行新记录抽样. 本文通过正采样和负采样两种方式生成多到多的 bug 修复记录, 进一步增强原始 bug 报告数据. 首先, 对于开发人员 d_j 和固定的 bug 集 B, 通过正采样收集 B 中与 b_i 相似的其他所有 bug, 并随机选择 s 个 bug 作为正样本; 其次, 通过负采样收集除 b_i 以外的所有原型 bug, 再随机选择 s 个 bug 作为负样本.

此时, 经过数据增强后, 一对多形式的 bug 修复记录将变成多对多, 有效地缓解了由于 bug 与开发人员特殊的交互模式所产生的数据不均衡性问题. 最近的研究已表明, 将不同的标签附加到不同类别数据上的方法对提高聚类质量是有效的[51]. 因此, 为了确保增强后的 bug 记录只出现在训练阶段, 本文设置了不同的 bug 标签来区分原始记录和增强记录[52]. 具体来讲, 这里对公式 (3) 分派集 T 中的 bug 分派记录 t_{ij}, 进行了如下改写:

$$t_{ij}=\begin{cases}3, \text{ if } b_i \text{ is originally assigned to } d_j\\ 2, \text{ if } b_i \text{ is a positive sample for } d_j\\ 1, \text{ if } b_i \text{ is a negative sample for } d_j\\ 0, \text{ if } b_i \text{ is not assigned to } d_j \text{ initially}\end{cases} \tag{5}$$

按照改写后的分派记录规则, 本文规定 $t_{ij}=\{3,2,1\}$ 只出现在训练阶段, 以降低数据不均衡带来的影响, 而 $t_{ij}=3$ 只能出现在验证和测试阶段, 以避免来自增强样本的任何干扰[53].

3.3 协同分派建模

协同分派建模的核心任务是构建图神经网络模型, 对 bug 和开发者语义及增强的 bug-开发者二部图信息进行融合, 强化 bug 和开发者表征并沿着图结构捕获 CF 信号, 为 bug 与开发者的相关性预测提供依据. 在本节中, 首先给出图神经网络一次卷积过程, 然后再推广至连续的多次卷积.

3.3.1 卷积计算

由于已修复的 bug 能够揭示开发者兴趣和能力特征, 且修复 bug 的开发者本身也可以被视为 bug 的一个关键特征, 被用于分析不同 bug 之间协作的相关性. 因此, 建模时将 bug 节点 b_i 初始化为 $p_{b_i}^0$, 开发人员节点 d_j 初始化为 $q_{d_j}^0$, 可以将这些语义信息完整地表达在各自的表示向量中, 并且能够灵活地与基于 bug 报告学习的多数模型集成在一起, 使之具有较好的泛化能力. 为此, 本文采用图卷积神经网络的消息传递机制实现增强二部图中的开发者和 bug 卷积过程. 该机制主要包含消息构建和消息聚合两个基本操作.

(1) 消息构建. 对于已连接的 bug-开发者 (b_i,d_j), 将从 b_i 到 d_j 的消息定义为:

$$m_{b_i\to d_j}=\psi(p_{b_i}^0,\eta_{t_{ij}},q_{d_j}^0) \tag{6}$$

其中, $m_{b_i\leftarrow d_j}$ 为消息向量, $\eta_{t_{ij}}$ 为相关性系数, 用来控制每个边传播 (b_i,d_j) 情况, $\psi(\cdot)$ 表示以向量 b_i 和 d_j 为输入消息的编码函数, 其具体形式如下:

$$\psi(b_i,\eta_{t_{ij}},d_j)=\eta_{t_{ij}}(\mathfrak{R}_1 p_{b_i}^0+\mathfrak{R}_2(p_{b_i}^0\odot q_{d_j}^0)) \tag{7}$$

其中, $\eta_{t_{ij}}=1/\sqrt{|Neigh_{b_i}||Neigh_{d_j}|}$, 是一个图拉普拉斯范数, $Neigh_{b_i}$ 和 $Neigh_{d_j}$ 表示 bug b_i 和开发者 d_j 的第一跳邻居. 从表示学习的角度来看, $\eta_{t_{ij}}$ 反映了历史 bug 对开发者能力或兴趣的贡献程度; 从消息传递的角度来看, $\eta_{t_{ij}}$ 可以被解释为一个折扣因子, 表示传播的消息随着路径长度的增加而衰减. $\mathfrak{R}_1$、$\mathfrak{R}_2\in\mathbb{R}^{d'\times d}$ 为卷积过程中的训练权值矩阵, 用于提取有用信息; d' 为卷积变换大小. 与只考虑 $p_{b_i}^0$ 贡献的传统图卷积网络不同, 这里特意将 $p_{b_i}^0$ 和 $q_{d_j}^0$ 之间的交互作用编码到基于 $p_{b_i}^0\odot q_{d_j}^0$ 传递的消息中. 其中, $\odot$表示元素乘积运算, 这使得消息依赖于 $p_{b_i}^0$ 和 $q_{d_j}^0$ 之间的相关程度, 进而可以通过相似的 bug 传递更多的消息. 这不仅提高了模型表达能力, 也提高了其预测性能.

(2) 消息聚合. 接下来, 需要聚合从 b_i 邻域传播的消息, 以强化 b_i 的表示. 本文将聚合函数定义为:

$$p_{b_i}^1=F_{\text{aggr}}\left(\mathfrak{R}_1p_{b_i}^0+\mathfrak{R}_2\sum_{d_j\in Neigh_{b_i}}m_{b_i\to d_j}\right)\tag{8}$$

其中, $p_{b_i}^1$ 表示在第 1 个向量卷积层得到 bug b_i 的表示, $F_{\text{aggr}}(\cdot)$ 为聚合函数. 这里选用可以将信息编码为正信号和小负信号的 LeakyReLU 作为激活函数. 公式 (8) 中除了聚合从邻居 $Neigh_{b_i}$ 传播的消息外, 还考虑了 $\mathfrak{R}_1p_{b_i}^0$, 即 b_i 的自连接 $m_{b_i\to b_i}$. 因此, 有效保留了 bug 原始特征的信息. 类似地, 进一步通过聚合其所连接的开发者传播消息来得到 d_j 的表示 $q_{d_j}^1$:

$$q_{d_j}^1=F_{\text{aggr}}\left(\mathfrak{R}_1q_{d_j}^0+\mathfrak{R}_2\sum_{b_i\in Neigh_{d_j}}m_{b_i\to d_j}\right)\tag{9}$$

其中, $q_{d_j}^1$ 表示在第 1 个向量卷积层得到开发者 d_j 的表示; $F_{\text{aggr}}(\cdot)$ 同样为聚合函数, 这里选仍然选用 LeakyReLU 作为激活函数. 向量卷积层在这里的优势在于显式地利用一次卷积信息来关联 bug 和开发者表示.

3.3.2 卷积推广

高阶连接对于协作信号编码和 bug-开发者之间的相关性分析是至关重要的. 因此, 通过一次连通性建模卷积的聚合表示后, 再采用堆叠的方式实现更多卷积层的计算, 从而达到探索高阶连通性信息表达的目的. 为此, 本文采用公式 (10) 所示的卷积函数对 bug 节点进行卷积推广:

$$p_{b_i}^l=W^S\left(F_{\text{aggr}}^S\left(q_{d_j}^{l-1}\mid d_j\in Neigh_{b_i}(D)\right)+p_{b_i}^{l-1}\right)\tag{10}$$

其中, $p_{b_i}^l\in\mathbb{R}^d$ 为第 l 层卷积的 bug 表示, $W^S\in\mathbb{R}$ 为卷积权值矩阵, $Neigh_{b_i}$ 表示包含与 d_j 相邻的 bug b_i, F_{aggr}^S 是一个线性聚合函数, 可以对所有的元素进行聚合操作.

对开发人员节点进行卷积推广时, 这里同样采用了类似公式 (10) 的 bug 节点卷积函数. 通过增加图中隐藏的空间特征来强化开发人员的节点表示. 具体形式如下:

$$q_{d_j}^l=W^T\left(F_{\text{aggr}}^T\left(p_{b_i}^{l-1}\mid b_i\in Neigh_{d_j}(B)\right)+q_{d_j}^{l-1}\right)\tag{11}$$

其中, $q_{d_j}^1\in\mathbb{R}^d$ 为第 l 层卷积时的开发者表示, $Neigh_{d_j}$ 表示包含与 bug b_i 相邻的开发人员 d_j, F_{aggr}^T 同样为一个线性聚合函数. 根据公式 (11), 通过权值矩阵 $W^T\in\mathbb{R}^{d\times d}$ 变换后得到第 l 层开发人员节点的卷积向量表示.

3.4 协同分派推荐

本文将 bug 分派建模为 bug 与开发人员相关性的预测, 即图学习过程中两类节点之间链接的预测. 对于每个 bug 报告节点, 将其表示向量与所有开发者节点的表示向量进行内积运算, 得到 bug 报告与所有开发者之间的分派相关度评分. 最终根据得分对开发者进行推荐排序, 并将 bug 报告分派给排序列表中最前的开发者.

这里的预测是建立在 bug 和开发者最终的表示基础上. 通过 l 层卷积后, 得到了开发者的多个表示形式, 即 $\left\{q_{d_j}^1,q_{d_j}^2,\ldots,q_{d_j}^l\right\}$, 由于在不同卷积层中获得的开发者表示包含了不同连接传递的消息, 故其能够反映开发者的能力、兴趣及对 bug 报告的贡献. 因此, 将其连接起来, 构成开发者的最终表示向量 $q_{d_j}^*=q_{d_j}^0\|q_{d_j}^1\|\ldots\|q_{d_j}^l$; 类似地, 对 bug 也进行同样的操作, 将不同卷积层学习到的 bug 表示 $\left\{p_{b_i}^1,p_{b_i}^2,\ldots,p_{b_i}^l\right\}$ 连接在一起, 得到最终的 bug 向量表示 $p_{b_i}^*=p_{b_i}^0\|p_{b_i}^1\|\ldots\|p_{b_i}^l$. 此时, 再采用公式 (12) 进行内积运算, 以预测 d_j 修复 b_i 的可能性:

$$\hat{t}_{ij} = p_{b_i}^{*\mathrm{T}} \cdot q_{d_j}^{*} \tag{12}$$

其中, $\hat{t}_{ij}$ 为相关性评分, 表示开发人员与 bug 潜在的相关性. 具体来讲, $\hat{t}_{ij}$ 类似于原始数据集中的标注数据 t_{ij}, 一个未解决的 bug b_i 将被分配给与 b_i 具有最高相关性的开发人员 d_j. 为了能够从正样本和负样本中捕获相关性特征, CBT-MF 引入了 BPR[38] 作为预测时的损失函数:

$$\mathcal{L}_{\mathrm{BPR}} = \sum_{S} -\log\sigma(\hat{t}_{i,j} - \hat{t}_{k,j}) + \lambda\|\Xi\|_2^2 \tag{13}$$

其中, $\sigma(\cdot)$ 是 Sigmoid 函数, 而三元组集 $S = \{(i,j,k) \mid t_{i,j} = \{1,2\}, t_{k,j} = 3\}$ 表示了成对的训练数据, 包括为开发人员 d_j 提供的一个正采样、原始 bug b_i 及一个负采样 bug b_k, Ξ 表示所有可训练的模型参数, λ 为防止过拟合控制参数. 此时, 训练图神经网络的主要目标就是最小化 $\hat{t}_{i,j}$ 和 $\hat{t}_{k,j}$ 之间的距离, 即:

$$\arg\min_{\Lambda} \mathcal{L}_{\mathrm{BPR}} \tag{14}$$

其中, Λ 为网络参数. 这里采用 Adam[54] 对预测模型进行优化, 并更新模型参数. 当然, 对于一批随机抽样的三元组 $(b_i, t_{ij}, d_j) \in S$, 经过 l 次卷积后, 在已完成 $\left\{p_{b_i}^1, p_{b_i}^2, \ldots, p_{b_i}^l\right\}$ 和 $\left\{q_{d_j}^1, q_{d_j}^2, \ldots, q_{d_j}^l\right\}$ 表示的情况下, 也可以利用损失函数的梯度更新模型参数.

4 实验分析

为了全面评估本文提出的 CBT-MF 方法性能, 实验主要关注了以下 3 个研究问题 (RQ).

- RQ1: 与其他已有的主流 bug 分派方法相比, CBT-MF 对 bug 分派的性能表现如何?
- RQ2: CBT-MF 是否能够缓解由于 bug 数据分布不均衡性和文本数据模态单一对 bug 分派性能产生的影响?
- RQ3: 在 CBT-MF 方法中, 关键的超参数是如何影响其对 bug 分派性能的?

4.1 数据与指标

本文基于领域内广泛使用的 3 个公开数据集: Google Chromium (GC)、Mozilla Core (MC) 及 Mozilla Firefox (MF), 对 CBT-MF 进行上述 3 个研究问题的实验评估. 这 3 个数据集均有类似的 bug 报告记录, 包括 bug ID、摘要、描述、开发者 (开发人员的 E-mail) 及 bug 状态, 这为本文研究提供了一致性的实验评估基础. 但是, 它们在数据规模、开发者活跃度和数据质量等方面各具特色, 尤其是各数据集的 bug 报告质量层次不齐, 影响实验对比效果, 故对所选数据集做了必要的预处理. 首先, 为了防止 bug 报告重复或修改变化对评估结果产生影响, 实验按照不同的时间跨度筛选了状态为验证 (verified) 或修正 (fixed) 的 bug 报告记录, 得到的实验数据集基本特征如表 1 所示. 其次, 基于筛选后的 bug 报告记录 ID 和开发者字段构建 bug-开发人员二部图, 形成图模态数据, 表征“bug-开发者”的修复关联关系. 随后, 根据 CBT-MF 方法必要向量的初始化需求, 将符合状态要求的 bug 报告去除其他冗余属性后保留 bug 报告标题、描述、分派开发者这 3 项关键属性 (具体名称因跟踪系统而异), 作为 bug 报告文本源, 形成文本模态数据, 即通过 bug 报告属性筛选完成噪声过滤; 最后, 按照第 2.2 节描述的过程, 经过对标点符号、十六进代码及停用词过滤和大写字母转换, 对数据进一步清洗后进行 bug 语义表示. 最终, 将所选实验数据集均以 8:1:1 的比例分为训练集、验证集及测试集, 且为了消除开发人员的流动性对评估结果的影响, 划分的测试集时间跨度均控制在了 1 年以内.

表 1 实验数据集特征

特征	GC[55]	MC[56]	MF[57]
bug总数	383 104	314 388	162 307
开发者数	3 612	1 477	891
修复记录数	361 164	147 664	89 083
密度 (1E–4)	2.61	3.18	6.16
时间跨度	2008-08–2016-07	1998-04–2016-06	1999-07–2016-06

通常活跃的开发者贡献了多数修复记录; 反之, 不活跃的开发者修复记录则很少. 开发者和修复记录数据的这种不均衡性是数据集存在 bug 数据帕累托现象的一个直接原因. 因此, 实验分别采用 3 个数据集中记录数与 bug 和开发者数量乘积的比率所表示的密度, 即 密度 = *# of records*/(*# of bugs* ∗ *# of Developers*), 来反映其 bug 修复记录的不均衡性. 最终, 3 个数据集的密度均在 0.02%–0.07% 之间, 且密度越小表示数据不均衡性越强.

本文选择推荐问题研究时广泛采用的 *Recall@K* 和 *Hit@K* 作为性能评价的指标, 具体计算如公式 (15) 和公式 (16) 所示, 且 K 被设置为 5, 10,…, 25 共 5 种情况.

$$Recall@K = (\#\ of\ top\ K\ assignments\ that\ are\ relevant)/(\#\ of\ all\ relevant\ bugs) \tag{15}$$

其中, *# of top K assignments that are relevant* 表示前 K 个分派的结果与实际分派给相应开发者的 bug 相关的 bug 数量, 这里的“相关”通常指的是正确分派的 bug. *# of all relevant bugs* 表示实际上应该被分派给相应开发者的 bug 的总数. *Recall@K* 表示在前 K 个分派结果中, 有多少比例的 bug 被正确地分派给了相应的开发者. 通常, *Recall@K* 值在 0–1 之间, 越接近 1 表示方法在前 K 个分派结果中更好地覆盖了实际应该分派给相应开发者的 bug.

$$Hit@K = \frac{1}{|B|}\sum_{i=1}^{|B|} 1_{R(b_i)\leqslant K} \tag{16}$$

其中, $|B|$ 表示 bug 集合的大小, b_i 表示第 i 个 bug, $R(b_i)$ 表示 bug b_i 的推荐排名, $1_{R(b_i)\leqslant K}$ 为指标函数, 当 $R(b_i)\leqslant K$ 时, $1_{R(b_i)\leqslant K}=1$, 表示 b_i 在分派的前 K 个推荐中; 否则为 0, 即表示 b_i 不在分派的前 K 个推荐中. 因此, *Hit@K* 表示在前 K 个推荐中, 有多少比例的相关 bug 被成功地包含在内. 很显然, *Hit@K* 值也在 0–1 之间, 越接近 1 表示模型在前 K 个分派推荐中更好地命中了与开发人员相关的 bug.

4.2 比较基线

为了从多角度分析 CBT-MF 的性能, 本文选取了基于 bug 报告文本模态分类、相关性匹配及 CF 推荐 3 个类别, 共计 8 个较为先进的方法作为基线进行对比, 具体方法详见表 2.

表 2 基线方法

类别	方法
文本模态bug分类	CBR[5]、DBRNN-A[16]
相关性匹配	GRCNN[22]、BERT[23]
基于CF推荐	LightGCN[29]、NCF[30]、CBCF[31]、SGL[40]

在表 2 仅采用 bug 报告单一模态类方法中, CBR 将 bug 文本转换为向量, 并使用一种半自动化的方法学习每个开发人员解决的报告类型, 并将新记录的 bug 报告分派给合适的开发人员; DBRNN-A 提出了一种深度双向递归神经网络, 其以无监督的方式从 bug 报告长单词序列中学习语法和语义特征. 在相关性匹配类方法中, GRCNN 以不同的时间周期学习开发人员在协作网络上节点的时空特征, 以得到 bug-开发人员相关性的表示; BERT 通过调节所有层的左右上下文, 从未标记的数据中预训练深度双向表示, 帮助计算一个给定的 bug 和开发人员已修复 bug 之间的相关性; 在 CF 类方法中, NCF 可以使用一个神经网络模块, 比如多层感知神经网络, 来拟合相关匹配函数; CBCF 采用内容增强的协同过滤, 将 CBR 和 CF 推荐器相结合, 提高了推荐质量; LightGCN 提出了一种 GCF 方法, 能够有效地聚合线性邻域; SGL 将结构性对比学习与 LightGCN 集成, 使 LightGCN 的性能得到了进一步改进.

4.3 实验方案

实验利用 GC、MC 及 MF 数据集对选择的基线方法进行了复现, 并且为了不失公允性, 实验对所有比较的基线方法模型参数都进行了一定程度的优化. 同时, 为了尽可能消除因优化方式差异而对比较方法产生性能影响, 所有训练的模型均采用 Adam 优化器, 训练批大小设置为 8 192, 且采用默认的 Xavier 分布进行网络参数初始化, 向量大小均为 64, 采用 10 步提前停止的策略以防止过拟合. 对于 CBT-MF, 默认图卷积层为 2 层, 采样尺度 s 设置为 5. 此外, 为了在保持足够聚类细致度的同时, 避免设置过大的质心数及确保在整个方法中 bug 和开发者在聚类

过程中都得到同等程度的关注, 这里将聚类质心数分别设置为 bug 数和开发人员数的 30%, 并随机选择测试集中的样本作为初始簇中心. 所有实验均在配置为 16 GB 内存、英特尔酷睿 i7-7700、NVIDIA GTX 1080ti 的同一台 PC 机上进行.

4.4 结果与分析

● RQ1: 性能比较.

表 3 显示了 CBT-MF 与其他基线方法在 3 个所选数据集的测试集上, 进行 bug 分派后获得的评估指标统计结果. 其中, 带有下划线的结果为每个类别的最佳结果, 粗体显示结果为 CBT-MF 的性能评估情况.

表 3　3 类方法和两种消融模型的总体性能比较

数据集	指标	文本分类		相关性匹配		CF推荐					消融研究	
		CBR	DBRNN-A	BERT	GRCNN	CBCF	NCF	Light GCN	SGL	CBT-MF	CBT-G	CBT-P
GC	*Recall*@5	0.0015	0.0019	0.0011	0.0014	0.0018	0.0027	0.0029	0.0031	**0.0075**	0.0056	0.0041
	Recall@10	0.0022	0.0024	0.0016	0.0017	0.0024	0.0035	0.0035	0.0037	**0.0082**	0.0061	0.0051
	Recall@15	0.0025	0.0026	0.0024	0.0027	0.0037	0.0041	0.0047	0.0084	**0.0087**	0.0083	0.0071
	Recall@20	0.0027	0.0029	0.0029	0.0031	0.0043	0.0046	0.0051	0.0094	**0.0109**	0.0090	0.0085
	Recall@25	0.0031	0.0037	0.0035	0.0041	0.0058	0.0065	0.0072	0.0105	**0.0145**	0.0115	0.0102
	Hit@5	0.0196	0.0261	0.0244	0.0288	0.0297	0.0315	0.0361	0.0388	**0.0476**	0.0368	0.0332
	Hit@10	0.0284	0.0320	0.0323	0.0386	0.0332	0.0396	0.0436	0.0405	**0.0539**	0.0416	0.0396
	Hit@15	0.0330	0.0410	0.0408	0.0432	0.0418	0.0451	0.0491	0.0540	**0.0602**	0.0511	0.0459
	Hit@20	0.0457	0.0466	0.0469	0.0483	0.0499	0.0510	0.0520	0.0586	**0.0631**	0.0550	0.0525
	Hit@25	0.0629	0.0712	0.0585	0.0689	0.0680	0.0730	0.0902	0.1101	**0.1216**	0.0715	0.0651
MC	*Recall*@5	0.0003	0.0004	0.0007	0.0009	0.0010	0.0013	0.0014	0.0018	**0.0036**	0.0015	0.0012
	Recall@10	0.0006	0.0007	0.0009	0.0010	0.0011	0.0013	0.0016	0.0024	**0.0041**	0.0022	0.0021
	Recall@15	0.0007	0.0008	0.0009	0.0011	0.0010	0.0014	0.0017	0.0029	**0.0043**	0.0026	0.0023
	Recall@20	0.0008	0.0009	0.0010	0.0011	0.0012	0.0016	0.0018	0.0032	**0.0044**	0.0029	0.0024
	Recall@25	0.0010	0.0012	0.0025	0.0026	0.0019	0.0022	0.0027	0.0035	**0.0085**	0.0033	0.0029
	Hit@5	0.0514	0.0714	0.0616	0.0808	0.0936	0.1121	0.1274	0.1300	**0.1442**	0.1202	0.0987
	Hit@10	0.0809	0.0909	0.0970	0.1039	0.1102	0.1293	0.1305	0.1382	**0.1596**	0.1275	0.1220
	Hit@15	0.0922	0.1012	0.1022	0.1094	0.1151	0.1306	0.1312	0.1406	**0.1682**	0.1330	0.1329
	Hit@20	0.1015	0.1084	0.1109	0.1198	0.1260	0.1366	0.1332	0.1491	**0.1700**	0.1457	0.1366
	Hit@25	0.1320	0.1457	0.1353	0.1602	0.1311	0.1411	0.1547	0.1657	**0.2229**	0.1681	0.1563
MF	*Recall*@5	0.0012	0.0021	0.0023	0.0042	0.0054	0.0061	0.0068	0.0082	**0.0123**	0.0066	0.0052
	Recall@10	0.0025	0.0039	0.0059	0.0061	0.0066	0.0072	0.0065	0.0099	**0.0138**	0.0086	0.0076
	Recall@15	0.0036	0.0048	0.0060	0.0064	0.0070	0.0080	0.0072	0.0102	**0.0173**	0.0097	0.0082
	Recall@20	0.0048	0.0051	0.0063	0.0067	0.0073	0.0080	0.0078	0.0113	**0.0191**	0.0111	0.0130
	Recall@25	0.0064	0.0082	0.0087	0.0095	0.0082	0.0097	0.0095	0.0122	**0.0290**	0.0128	0.0135
	Hit@5	0.1762	0.1896	0.2159	0.2330	0.2602	0.2874	0.3024	0.3568	**0.3727**	0.2951	0.2717
	Hit@10	0.2072	0.2325	0.2537	0.2751	0.2880	0.3079	0.3285	0.3391	**0.3982**	0.3011	0.2920
	Hit@15	0.2515	0.2783	0.2615	0.2832	0.2936	0.3189	0.3385	0.3592	**0.4196**	0.3558	0.3173
	Hit@20	0.2800	0.2965	0.2799	0.2902	0.2998	0.3333	0.3467	0.3800	**0.4238**	0.3706	0.3455
	Hit@25	0.3015	0.3242	0.3057	0.3247	0.3108	0.3533	0.3680	0.4017	**0.4309**	0.4012	0.3813

从表 3 中可以观察到, 尽管 3 类不同的分派方法均采用了相同的优化方式, 但在 GC、MC 和 MF 数据集上的 *Recall@K* 指标表现差异仍然较大. 其中, CBT-MF 在所有 *Recall@K* 指标上表现得最好, 其次是 SGL 方法. 而在剩余方法中, NCF 和 LightGCN 方法表现较为接近. BERT 方法劣于 GRCNN, 且二者的 *Recall@K* 指标均低于 CF 推荐类的方法. 与 CBR 相比 DBRNN-A 表现出了一定的优势, 但均两类指标均低于相关性匹配和 CF 推荐类的方法. 这些方法性能表现结果差异很大程度上是因为, 基于单一模态 bug 分类、相关性匹配和基于 CF 的方法采用了不同的处理过程, 且方法模型的学习过程也不尽相同, 最终表现出了不同的数据拟合能力, 从而影响了方法的分派性能.

从表 3 中还可以看出, CBT-MF 方法在 GC、MC 和 MF 这 3 个数据集上的 *Hit@K* 指标也表现出了最好的性能. 尤其是在 MC 和 MF 数据集上的表现明显优于其他方法, 并且对任意两组数据集的同类指标进行方差分析后, 发现 $p\text{-}value < 0.05$, 这说明实验结果表现出了明显的统计学意义. 在其他方法中, SGL 方法在选定指标上表现相对较好, 而 CBCF、NCF 和 LightGCN 方法表现略差, 但明显优于文本分类的 CBR 和 DBRNN-A 方法, 以及相关性匹配的 BERT 和 GGRCNN 方法. 这是因为优化过后的各类模型参数对不同方法的性能会产生一定程度的影响, 例如迭代次数、正则化系数等, 最终影响到了其性能评估指标. 并且, 实验所选的 3 个数据集本身具有不同的特征, 如 bug 数据的密度、噪声、数据分布等, 这些特征也会对实验评估的各个方法性能产生影响.

当然, 还需要指出的是, 包括 CBT-MF 在内的上述方法在实验数据集上的 *Recall@K* 和 *Hit@K* 值并不高. 这个原因是多方面的, 其一, bug 分派场景中的 bug 数据存在天然的失衡, 缺陷报告文本丰富而开发者能力描述缺失; 其二, 数据集本身的复杂性和噪声问题可能会影响模型的学习效果, 尽管进行了数据清洗和预处理, 但仍可能存在一些难以完全消除的噪声和冗余信息. 此外, 虽然我们采用了先进的图神经网络和多模态融合方法实现 bug 与开发者复杂关系的捕获, 但是在处理一些噪声数据和异常情况时, 模型的鲁棒性仍有待提升. 然而, 这些方法所形成的指标差异性和低迷性, 并不影响我们归纳出下述 4 个基本结论.

(1) 基于相关性匹配的方法优于单一文本分类的方法. 虽然深度学习有助于 DBRNN-A 分派性能的提升, 最终会优于经典的 CBR, 但它们都比不上 BERT 和 GRCNN. 这是因为 BERT 和 GRCNN 从相关性匹配的角度分派时更能够挖掘 bug 和开发者之间的关系, 这种关系能够有力提升 bug 分配的准确性. 此外, 从表 3 所统计的实验结果中还可以看出, GRCNN 相对于 BERT 来讲性能更好, 这很大程度上是因为 GRCNN 中的深度图神经网络比 BERT 的左右上下文表示方法能够提取 bug 和开发者之间更多的相关性特征.

(2) CF 推荐的 bug 分派是分派性能较好的一类方法. CF 推荐类所包含的 4 个基线方法的两类评价指标都优于文本模态分类的基线方法. 特别地, LightGCN 在 bug 修复记录分布最不均衡的 GC 数据集上也相对优于 NCF. 因为 LightGCN 更适合处理存在不均衡性的数据, 这种优势在相对均衡的 MC 和 MF 数据集上更为明显, 因为 bug 修复记录相对均衡数据集通常可以给 LightGCN 带来更好的性能. 此外, SGL 出色的性能表明, 通过不断的节点特征学习可以将基于图学习的 CF 提升到更高的表达水平.

(3) 与所有的基线方法相比 CBT-MF 都表现出了最好的性能. 从表 3 中的实验结果可以出, 即使在像 GC 这样 bug 修复记录较不均衡的数据集中, CBT-MF 也取得了绝对的领先优势. 这一优势归因于 CBT-MF 方法的两个基本创新上. 一是聚类和重采样很大程度上缓减了 bug 数据分布不均衡性的影响; 二是以 bug 报告语义与 bug-开发者二部图的多模态融合有效表征了 bug 报告与开发者的相关性.

(4) 在 3 类方法中, MF 上的两种指标表现都明显优于 MC 和 GC. 这是因为 MF 数据集有相对大的密度, 表明该数据集中开发者的活跃度和缺陷修复记录的分布可能更为均衡, 这种均衡性有助于各类方法全面地学习开发者的修复行为和模式, 从而提高预测性能, 使得分派方法能够更有效地学习和预测. 此外, 在数据处理阶段, 对不同的数据集进行的数据清洗、缺陷报告过滤和属性筛选等步骤, 可能在 MF 数据集中更为有效地减少了噪声和冗余信息, 从而提升了数据的质量和一致性, 这进一步增强了这些方法的性能.

● RQ2: 数据不均衡性与文本数据模态单一影响的缓解.

CBT-MF 在进行 bug 分派时, 旨在缓解 bug 数据不均衡性和 bug 报告文本数据无法表达 bug 与开发者之间相关性产生的影响. 因此, 接下来再讨论, CBT-MF 将 bug 报告文本和二部图作为 bug-开发者相关性挖掘的多模态数据时, 能否缓解两个问题带来的分派性能影响. 实验分别构建了两个消融模型: CBT-P 和 CBT-G.

(1) CBT-P 仍然使用 GCF 模块的数据增强方案, 但不采取融合策略, 只对二部图进行图卷积计算, 且 GCF 模块的设置与 LightGCN 相同. 通过对表 3 中的 LightGCN 和 CBT-P 的比较, 可以看到本文设计的数据增强方案有效地缓解了 bug 数据的不均衡性影响, 并对评估指标带来了相当大的改进.

(2) CBT-G 不采用本文设计的数据增强方案进行数据增强, 仅使用 GCF 对 bug 文本语义和 bug-开发者二部图融合后的结果进行卷积计算. 比较表 3 中 CBT-MF 和 CBT-G 的实验结果, 可以看到, 由于 CBT-MF 将 bug 报告文本语义与 bug-开发者二部图捕获的结构特征同时作为链接预测的互补性依据, 故 CBT-MF 的分派性能表现优

于 CBT-G.

通过对比表 3 中的 CBT-P 和 CBT-G 可以发现, CBT-P 的各项评估指标略低于 CBT-G, 这说明即使 CBT-G 存在数据不均衡性问题的情况下, 两个模态数据的融合也能在一定程度上缓解 bug 报告文本数据模态单一对 bug-开发者相关性欠表征带来的影响, 这充分说明将两种模态数据融合分析可以对 CBT-MF 的 bug 分派性能提升产生积极的作用.

此外, 为了评估同一数据集不同程度修复记录关联分布不均衡性对 CBT-MF 性能的影响, 实验根据表 1 所示的修复记录数, 将 GC、MC、MF 这 3 个数据集中的所有 bug 报告记录和开发人员依次分成 G1–G5 这 5 个数据组, 保持每一组中的 bug 修复记录数和开发人员数量不变. 其中, 较小 ID 的数据组, 包含的 bug 报告较新, 需要修复的时间跨度大, 且数据组的 bug 报告记录规模大、数量多; 相同的 bug 修复记录数的条件下, 相对于其他同等数量开发者的大 ID 数据组来讲, 数据不均衡的可能性更大. 例如, 与大 ID 组相比, 在相同修复记录、相同开发人员数量下, 包含较多修复记录的 G1 表示在数据组中开发人员修复的 bug 相对较少, 意味着数据组的修复记录关联分布不均衡性更强. 各数据组仍然以 8:1:1 的比例分为训练集、验证集及测试集, 并与整体性能较优的 SGL 方法比较 *Recall*@20 和 *Hit*@20 指标, 最终在测试集上的实验统计结果如图 3 所示.

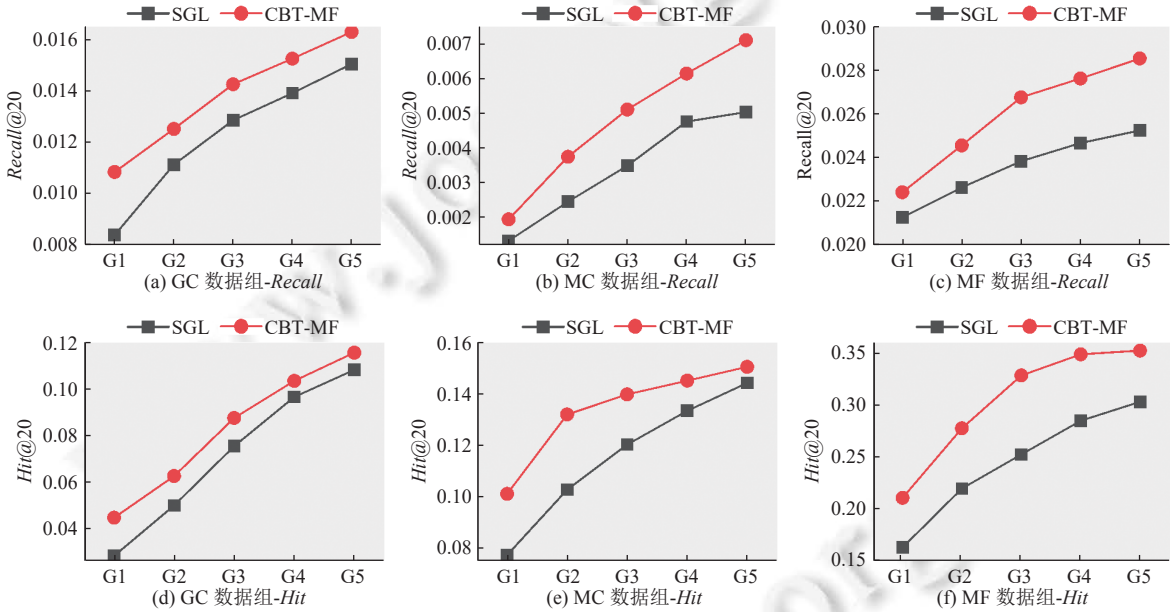


图 3 CBT-MF 不同均衡度级别的 *Recall*@20 和 *Hit*@20

在图 3(a)–(c) 及 (d)–(f) 中, 随着数据组 ID 的增大, 每个数据组中的开发人员修复的 bug 数量也逐渐增加, 表明数据组的数据不均衡性减弱. 从图 3 展示的实验结果也可以看出, CBT-MF 相对于 SGL 在不均衡数据的处理上具有更好的性能表现, 这一结论可以得到 GC、MC 及 MF 这 3 个不同数据集上实验结果验证. 因为在数据组 ID 增大的过程中, 无论是从 *Recall*@20 还是 *Hit*@20 来看, CBT-MF 的性能均有所提高, 这表明在不均衡数据组中, CBT-MF 对于处理较多 bug 数量的数据组具有更好的性能. 此外, 从子图 3(c) 和 (f) 中还可以看出, MF 数据组相对于 GC 和 MC 数据组中开发人员平均修复的 bug 数量较多, 数据不均衡性较弱, 于是 CBT-MF 在 MF 数据组中的性能更好. 这也再次表明数据增强可以使得 CBT-MF 对不均衡数据处理产生积极的影响. 因为, 随着 G1–G5 数据组不均衡性的减弱, 即开发人员修复 bug 数量的增多, CBT-MF 逐渐展示出了更好的性能表现.

● RQ3: 关键超参数敏感性分析.

为了充分探讨在不同情况下超参数会如何影响 CBT-MF 的性能, 下面采用网格搜索的思路进一步研究了 CBT-MF 对采样尺度、聚类质心数、图卷积层这 3 个关键参数变化的敏感性.

(1) 采样尺度的敏感性. 实验按照枚举的策略, 依次将 CBT-MF 采样尺度 s 设置为 1、3、5、7、9. 图 4 显示了在不同采样尺度 s 下, CBT-MF 在 GC、MC 及 MF 数据集的测试集上获得的 *Recall@K* 和 *Hit@K* 各 5 个, 共 10 个指标的实验评估结果.

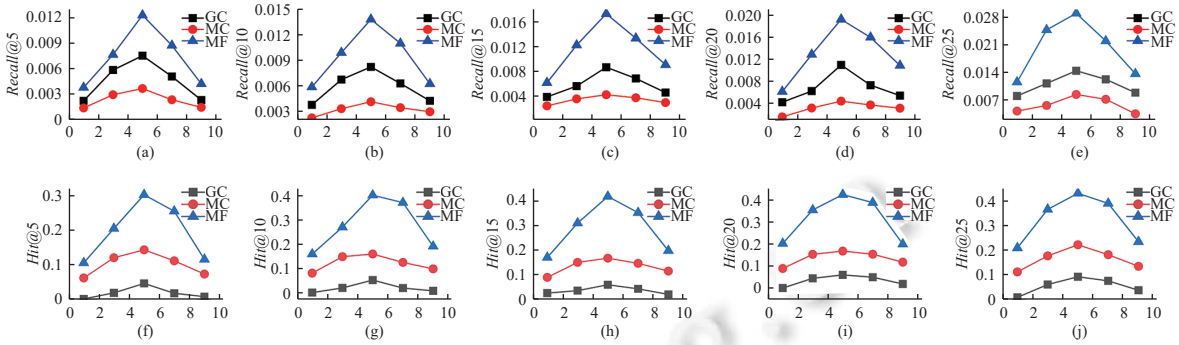


图 4 不同采样尺度对 *Recall@5*–*Recall@25* 和 *Hit@5*–*Hit@25* 的影响

从图 4(a)–(e) 显示的 *Recall@K* 实验结果可以看出, 随着采样尺度 s 的增加, *Recall@5*–*Recall@25* 的值也会增加. 但采样尺度 s 增大到一定程度时 ($s=5$), 各个指标达到了最大值. 随后续继续增大采样尺度 s, 各个指标并没有相应的提升, 相反地, 出现了下降的趋势. 例如, 在 GC 数据集上, 当采样尺度 s 从 1 增加到 5 时, *Recall@10* 从 0.003 7 增加到 0.008 2, *Recall@20* 从 0.004 2 增加到 0.010 9; 而当采样尺度 s 增加到 9 时, 二者分别降到了 0.002 3 和 0.005 4; 此外, 不同数据集之间的结果也有所不同. 例如, 在 *Recall@5* 指标上, MF 数据集的结果明显优于 GC 和 MC 数据集, 且在其他 *Recall@K* 指标上, 3 个数据集之间也存在类似的差异. 这些结果的形成可能与多种因素有关. 首先, 适当的采样尺度 s 可以提升 CBT-MF 对 bug 和开发者的覆盖率, 从而提高了 *Recall@K* 指标值; 其次, 不同数据集的特征和分布也会影响评估结果, 例如, MF 数据集密度大, 且其中的 bug 和开发者数量相对能够代表样本的多样性, 因此在适当的采样尺度影响下, 使 CBT-MF 在其上获得了更好的性能表现; 最后, CBT-MF 中采用聚类和采样的数据增强方式缓解数据不均衡性也对 *Recall@K* 评估指标产生了积极的影响.

从图 4(f)–(j) 显示的 *Hit@K* 指标统计结果可以看出, 首先对于每个指标和每个数据集, 同样随着采样尺度 s 在一定范围内 ($s \leqslant 5$) 的增加, CBT-MF 的性能也有所提升. 这说明采样尺度确实对 CBT-MF 性能有积极影响, 同时也证明了所设计的数据增强模块对提升 CBT-MF 的 bug 分派性能是有效的; 其次, 在每个数据集和每个 *Hit@K* 指标中, 采样尺度为 5 的时候, CBT-MF 的性能提升得最为明显. 以 *Hit@5* 指标为例, 当采样尺度 s 从 1 增加 3, 3 增加到 5 时, 对于 GC 数据集, CBT-MF 的性能分别提升了 0.003 6 和 0.001 7, 提升趋势逐渐减缓; 当采样尺度 s 继续增加时, CBT-MF 性能出现明显的下降趋势. 导致这个现象的原因是, 当采样尺度 s 过小时, 采样所得的样本集合不能全面代表原始测试集, 因此, CBT-MF 的性能一般; 当采样尺度 s 过大时, 由于 CBT-MF 需要处理的数据量增加, 引入的未知噪声也会随之增加, 导致 CBT-M 性能并不会有明显提升, 甚至在 3 个数据集上的评估指标出现了缓慢下降的趋势.

综上所述, 采样尺度 s 对 CBT-MF 性能有一定程度的影响, 增加 s 时 *Recall@K* 和 *Hit@K* 会随着增大, 表明更多的数据可以带来更多的学习特性; 然而, 较大的 s 会导致其性能退化, 这归因于实验测试集中的噪声在大采样尺度的情况下会导致 CBT-MF 过拟合. 此外, 从图 4 展示的实验结果中还发现, 数据集 GC 和 MC 的 *Hit@K* 结果与 *Recall@K* 结果趋势出现了反差. 这很大程度上是由于 GC 数据集修复记录更加不均衡, 而 MC 数据集相对均衡, 导致 CBT-MF 在 GC 数据集上拥有更高的 *Recall@K*, 这表明 CBT-MF 能够更好地召回 GC 数据集中的 bug. 但在 GC 数据集上的 *Hit@K* 较低, 表明 CBT-MF 在 GC 数据集中推荐出了更多的无关或不合适的 bug. 总体上讲, 采样尺度为 5 时, CBT-MF 性能提升最为明显, 考虑到这一点, 在后续的实验中, 把采样尺度 s 均设置为 5.

(2) 聚类质心数的敏感性. 为了分析聚类质心数对 CBT-MF 分派性能的影响, 本文构建了 CBT-MF 方法的变体 CBT-MF-*k*. 其中, k 依次设置为 10, 20,…, 50, 相应地将 bug 和开发人员的质心数设置为其总数的 k%. 且为了降

低算法对初始值的敏感性, 这里选用 K-means++算法初始化簇中心. 图 5 显示了在不同质心数 k 下, 使用 CBT-MF 方法的所有变体在 GC、MC 及 MF 数据集的测试集上获得的 *Recall@K* 和 *Hit@K* 两类共 10 个评估指标的实验结果.

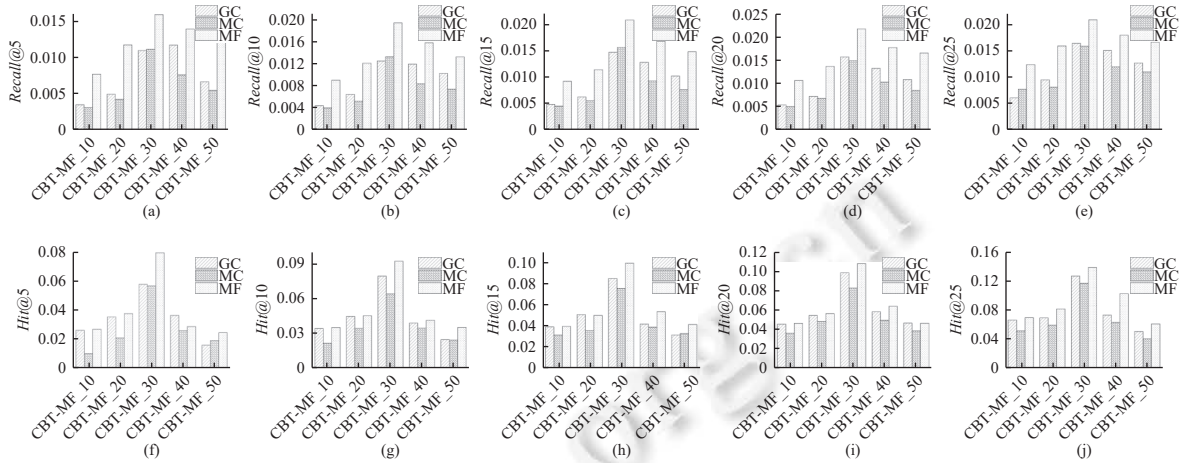


图 5 不同聚类质心对 *Recall@5*–*Recall@25* 和 *Hit@5*–*Hit@25* 的影响

从图 5(a)–(e) 显示的 *Recall@K* 实验结果可以看出, 在不同聚类质心数 k 下, 对于每个 *Recall@K* 的指标, 3 种数据集的表现趋势基本一致, 即随着聚类质心数 k 的增加, *Recall@K* 值逐渐提高. 其中, 在 MF 数据集上的表现最佳. 然而, 过大的聚类质心数 k 并没有使 *Recall@K* 指标继续增大, 反而出现了减小的现象. 例如, k=50 时的指标反而低于 k=40. 这是因为当聚类质心数较大时, CBT-MF 会更加关注 bug 和开发者之间的细节, 因此分派结果容易受到噪声干扰, 从而对性能指标产生了负面影响. 同时还可以看出, 相比于 GC 和 MC 数据集, CBT-MF 能够更好地捕捉 MF 数据集 bug 和开发者之间的相关性, 说明 CBT-MF 在数据不均衡性弱时性能表现更好. 此外, 不难发现, 对于任意一个 *Recall@K* 来讲, 聚类质心数为 k=30 时, 在 3 个数据集上均获得了最佳的性能表现, 其他过大或过小的聚类质心数会对 *Recall@K* 指标产生一定的负面影响. 因此, 在实际应用中, 需要根据具体情况选择合适的聚类质心数, 以达到最佳的分派效果.

从图 5(f)–(j) 显示的 *Hit@K* 实验结果可以看出, 在所有数据集上, 随着质心数 k 的增加, *Hit@K* 的值也会增加, 但增加的速率会变化. 例如, 在 GC 数据集上, 当质心数从 10 增加到 20 时, *Hit*@10 从 0.034 1 增加到 0.044 5, 增加了 30.5%; 而当质心数 k 从 40 增加到 50 时, *Hit*@10 从 0.038 8 降到 0.024 5, 出现了负增加现象. 此外, 不同数据集的结果也有所不同. 例如, 在 *Hit*@10 指标上, MF 数据集的结果较 GC 和 MC 数据集更好, 且在其内部的 *Hit@K* 指标随质心数的变化也都表现出了先升后降的趋势. 也就是说, 随着质心数 k 的增加, *Hit*@5–*Hit*@25 评估指标的值并未呈现出明显的上升趋势, 而是在 k=30 调整为 k=40 的情况下, 出现了 *Hit@K* 指标值下降的情况. 这说明增加质心数 k 并不一定能显著提高 CBT-MF 的性能, 有可能因为过大的质心导致了 CBT-MF 过拟合. 此外, 从图中还可以看出, 不同的评估指标之间存在较大的差异. 例如, 在同一数据集上, *Hit*@10 评估指标的值通常比 *Hit*@25 评估指标的值低很多, 这说明 CBT-MF 在较大列表长度的评估指标上, 更能显示其分派性能的优越性.

上述结果的形成与多种因素有关. 一方面, 质心数 k 在一定范围内的增加可以提高聚类的精度, 从而提高了 *Recall@K* 和 *Hit@K*; 另一方面, 不同数据集的特征和分布也会影响指标结果. 例如, GC 数据集中的 bug 和开发者数量相对较大, 因此可能需要更多的质心才能获得更好的结果. 此外, CBT-MF 方法其他的参数设置和实验环境也可能对评估结果产生了一定程度的影响. 但总体而言, 无论对那个指标来讲, CBT-MF-k 在 k=30 时, 取得了最好的性能. 因此, 说明在合适的聚类质心下 CBT-MF 能够捕获更好的语义特征.

(3) 图卷积层的敏感性. 在 GCF 模块中, 节点卷积可以包含多层相邻节点. 较多的层可以聚合更多的结构特征, 以及降低过度拟合的风险. 为了研究卷积层数 l 如何影响 CBT-MF 方法性能, 实验比较了其另外 3 种变体 CBT-

MF_L*n* 的性能. 图 6 中显示了在不同卷积神经网络层数 (1, 2, 3) 下, 使用 CBT-MF 方法在表 1 的 3 个不同数据集的测试集上获得的所有 *Recall@K* 和 *Hit@K* 评估结果.

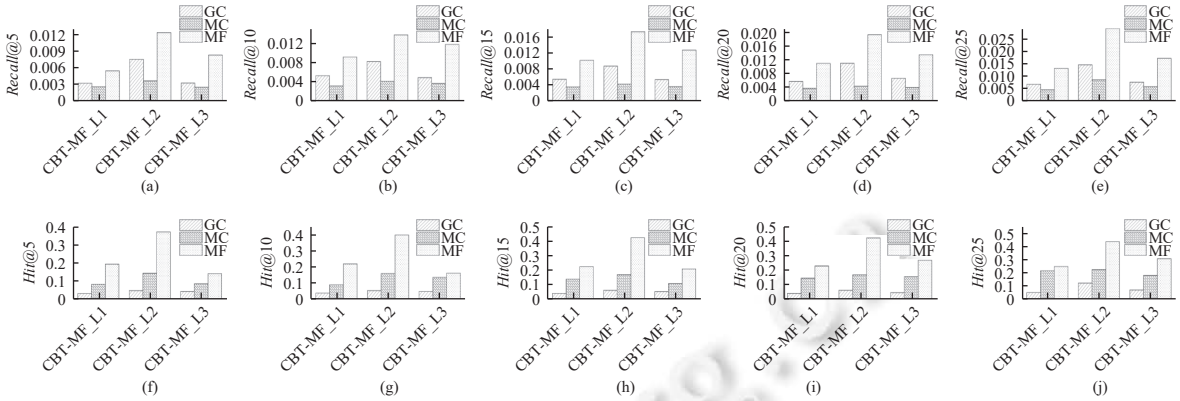


图 6 不同图卷积层数对 *Recall*@5–*Recall*@25 和 *Hit*@5–*Hit*@25 的影响

从图 6(a)–(e) 显示的 *Recall@K* 实验结果可以看出, 随着卷积神经网络层数的增加, *Recall*@5–*Recall*@25 的值也会增加. 然而, 对于同一 *Recall@K* 指标随着层数增加, 并没有出现持续上升的趋势. 例如, 在 MF 数据集上, 当卷积神经网络层数从 1 增加到 3 时, *Recall*@5 从 1 层时的 0.005 4 增加到 2 层时的 0.012 3, 而当在 3 层时又降至 0.008 3; 同时可以看出, 不同数据集之间的结果也有所不同. 例如, 在 *Recall*@20 指标上, MF 数据集的结果明显优于 GC 和 MC 数据集, 但 GC 和 MC 个别指标之间的差异不太明显. 这些结果的形成可能与多种因素有关, 尤其是不同数据集的特征和分布. 例如, GC 数据集中的 bug 与开发者数不均衡性较强, 对模型捕获二者的关系产生了消极的影响.

从图 6(f)–(j) 显示的 *Hit@K* 实验结果可以看出, 随着卷积神经网络层数的增加, *Hit*@5–*Hit*@25 的值也会增加, 而且, 对于同一 *Hit@K* 指标随着层数 1–3 的增加, 也出现了先增大后减小的变化趋势. 例如, 在 GC 数据集上, 当卷积层数从 1 增加到 3 时, *Hit*@20 从 0.040 2 增加到 0.063 1 后又降至 0.045 4; *Hit*@25 从 0.050 7 增加到 0.121 6 时又降至 0.070 1. 此外, 与 *Recall@K* 类似, 不同数据集之间的结果也呈现出了一定的差异. 例如, 在 *Hit*@25 指标上, MF 数据集的结果明显优于 GC 和 MC 数据集, 但在其他指标上, 基本保持了 MF 最好, MC 较好, GC 最差的态势. 这是因为对每个 *Recall@K* 来讲, 卷积神经网络层数的增加可以增加 CBT-MF 的 bug-开发者相关性的表达能力, 从而提高 CBT-MF 的分派性能, 但过大的卷积神经网络层数可能受到不同数据集的特征和分布影响, 存在过拟合的风险, 不能够保证其一直对评价指标的提升产生积极作用.

综上分析, 关键的超参数对 CBT-MF 方法的拟合与泛化能力有一定的影响, 但通过将参数调整到一定的范围可以缓解该问题. 也就是说, 如果为 CBT-MF 选择合理的参数配置, 就能够保证 CBT-MF 方法取得一个较好的分派结果. 因此, 在实践中如何设置 CBT-MF 关键超参数来提高 CBT-MF 的分派性能, 如 *Recall* 和 *Hit* 评估指标, 需要综合考虑不同数据集的特征和分布, 以及模型复杂度和计算效率之间的权衡, 来确定最合适的超参数配置方案.

5 总结与展望

基于 bug 报告文本分类的 bug 分派方法受到帕累托分布特征影响, 导致 bug 数据存在修复记录不均衡性. 目前的文本分类技术主要将开发者信息作为简单的标签, 忽视了对开发者更多信息的表征, 从而未充分利用 bug 与开发者之间的相关性. 因此, 本文提出了一种基于多模态融合的软件缺陷协同分派方法 (CBT-MF). 该方法受到文本数据和图结构数据融合训练模型启发, 将 bug 报告文本模态数据和 bug-开发者二部图模态数据融合在一起, 作为 bug 分派的分析依据. 通过引入数据增强方案重构了 bug-开发者二部图, 缓解了 bug 修复记录不均衡性的影响.

同时, 设计了一种 GCF 方法, 通过融合 bug 报告文本语义和 bug-开发者二部图两个模态数据重构节点特征, 并基于二部图结构捕获 bug-开发者之间的相关性. 最终, 将 bug 分派任务建模为 GCF 模块上的链接预测, 并根据预测结果得到了 bug 分派的推荐方案. 在 GC、MC 及 MF 这 3 个公开数据集上进行的大量实验分析表明, CBT-MF 对 bug 分派的性能较好, 整体表现出了明显的优越性. 该方法有望为软件 bug 分派开辟新的研究思路, 提高 bug 分派的准确性和效率.

CBT-MF 还有许多值得扩展的工作. 首先, 需要继续深入探究 bug-开发者相关性的有效表征, 并研究应用其他聚合器, 如最大池化、LSTM 等, 以提升 CBT-MF 方法的性能. 其次, 进一步研究自适应的数据增强方案, 通过基于自适应的数据增强方案对不均衡 bug 数据进行优化和调整, 以更好地适应不同分布和特征的数据. 第三, 不断探索 CBT-MF 在其他软件 bug 分派场景应用的可行性, 以提高 CBT-MF 方法的表达能力和泛化性. 最后, 计划探索处理重复报告和动态属性变化等问题的有效方法, 以提升 CBT-MF 的推广性和实用性. 此外, 在 CBT-MF 中, 主要强调 bug 和开发者文本表示向量和二部图数据的融合及特征提取, 因此在 bug 分派时选取了简单的内积交互函数进行相关性预测. 事实上, 还有其他更复杂的选择, 例如基于神经网络的交互函数, 其不仅可以通过向量传播层来丰富初始向量, 而且还允许通过参数调整来控制向量传播的范围. 因此, 探索更有效的分派预测方法也是未来研究工作的一部分.

References:

[1] Zhang TL, Chen R, Yang X, Zhu HY. Approach of bug reports classification based on cost extreme learning machine. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1386–1406 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5725.htm [doi: 10.13328/j.cnki.jos.005725]

[2] Sun JM, Xing ZC, Xia X, Lu QH, Xu XW, Zhu LM. Aspect-level information discrepancies across heterogeneous vulnerability reports: Severity, types and detection methods. ACM Trans. on Software Engineering and Methodology, 2023, 33(2): 49. [doi: 10.1145/3624734]

[3] Banerjee S, Syed Z, Helmick J, Culp M, Ryan K, Cukic B. Automated triaging of very large bug repositories. Information and Software Technology, 2017, 89: 1–13. [doi: 10.1016/j.infsof.2016.09.006]

[4] Binh NT, Tinh PD, Chien NH. BKiosk system—A solution for context-specific services supporting contextualization stage of E-government systems. In: Proc. of the 6th Int'l Conf. on E-commerce, E-business and E-government. Plymouth: ACM, 2022. 403–411. [doi: 10.1145/3537693.3537716]

[5] Anvik J, Hiew L, Murphy GC. Who should fix this bug? In: Proc. of the 28th Int'l Conf. on Software Engineering. Shanghai: ACM, 2006. 361–370. [doi: 10.1145/1134285.1134336]

[6] Sharma M, Tandon A, Kumari M, Singh VB. Reduction of redundant rules in association rule mining-based bug assignment. Int'l Journal of Reliability, Quality and Safety Engineering, 2017, 24(6): 1740005. [doi: 10.1142/S0218539317400058]

[7] Anvik J, Murphy GC. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Trans. on Software Engineering and Methodology (TOSEM), 2011, 20(3): 10. [doi: 10.1145/2000791.2000794]

[8] Xuan JF, Jiang H, Ren ZL, Zou WQ. Developer prioritization in bug repositories. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). Zurich: IEEE, 2012. 25–35. [doi: 10.1109/ICSE.2012.6227209]

[9] Jonsson L, Borg M, Broman D, Sandahl K, Eldh S, Runeson P. Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. Empirical Software Engineering, 2016, 21(4): 1533–1578. [doi: 10.1007/s10664-015-9401-9]

[10] He JY, Meng ZP, Chen X, Wang Z, Fan XY. Semi-supervised ensemble learning approach for cross-project defect prediction. Ruan Jian Xue Bao/Journal of Software, 2017, 28(6): 1455–1473 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5228.htm [doi: 10.13328/j.cnki.jos.005228]

[11] Baltrušaitis T, Ahuja C, Morency LP. Multimodal machine learning: A survey and taxonomy. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2019, 41(2): 423–443. [doi: 10.1109/TPAMI.2018.2798607]

[12] Akbari H, Yuan LZ, Qian R, Chuang WH, Chang SF, Cui Y, Gong BQ. VATT: Transformers for multimodal self-supervised learning from raw video, audio and text. In: Proc. of the 35th Int'l Conf. on Neural Information Processing Systems. Curran Associates Inc., 2021. 24206–24221.

[13] Anand V, Ramesh R, Jin BS, Wang ZY, Lei XX, Lin CY. Multimodal language modelling on knowledge graphs for deep video understanding. In: Proc. of the 29th ACM Int'l Conf. on Multimedia. ACM, 2021. 4868–4872. [doi: 10.1145/3474085.3479220]

[14] Dai J, Li QS, Xue H, Luo Z, Wang YL, Zhan SY. Graph collaborative filtering-based bug triaging. Journal of Systems and Software,

2023, 200: 111667. [doi: 10.1016/j.jss.2023.111667]

[15] Liu Y, Yang SS, Lei CY, Wang GX, Tang HH, Zhang JY, Sun AX, Miao CY. Pre-training graph Transformer with multimodal side information for recommendation. In: Proc. of the 29th ACM Int'l Conf. on Multimedia. ACM, 2021. 2853–2861. [doi: 10.1145/3474085.3475709]

[16] Mani S, Sankaran A, Aralikatte R. DeepTriage: Exploring the effectiveness of deep learning for bug triaging. In: Proc. of the 2019 ACM India Joint Int'l Conf. on Data Science and Management of Data. Kolkata: ACM, 2019. 171–179. [doi: 10.1145/3297001.3297023]

[17] Čubranić D, Murphy GC. Automatic bug triage using text categorization. In: Proc. of the 16th Int'l Conf. on Software Engineering & Knowledge Engineering. Banff, 2004. 92–97.

[18] Nagwani NK, Suri JS. An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review. Int'l Journal of Information Management Data Insights, 2023, 3(1): 100153. [doi: 10.1016/j.jjimei.2022.100153]

[19] Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078, 2014.

[20] Zaidi SFA, Woo H, Lee CG. A graph convolution network-based bug triage system to learn heterogeneous graph representation of bug reports. IEEE Access, 2022, 10: 20677–20689. [doi: 10.1109/ACCESS.2022.3153075]

[21] Al-Batlaa A, Abdullah-Al-Wadud M, Hossain MA. A review on recommending solutions for bugs using crowdsourcing. In: Proc. of the 21st Saudi Computer Society National Computer Conf. (NCC). Riyadh: IEEE, 2018. 1–4. [doi: 10.1109/NCG.2018.8593018]

[22] Wu HR, Ma YT, Xiang ZL, Yang C, He KQ. A spatial-temporal graph neural network framework for automated software bug triaging. Knowledge-based Systems, 2022, 241: 108308. [doi: 10.1016/j.knosys.2022.108308]

[23] Zaidi SFA, Woo H, Lee CG. Toward an effective bug triage system using transformers to add new developers. Journal of Sensors, 2022, 2022: 4347004. [doi: 10.1155/2022/4347004]

[24] Yang G, Zhang T, Lee B. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In: Proc. of the 38th IEEE Annual Computer Software and Applications Conf. Vasteras: IEEE, 2014. 97–106. [doi: 10.1109/COMPSAC.2014.16]

[25] Li Z, Shen X, Jiao YH, Pan XM, Zou PC, Meng XL, Yao CW, Bu JJ. Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In: Proc. of the 36th IEEE Int'l Conf. on Data Engineering (ICDE). Dallas: IEEE, 2020. 1677–1688. [doi: 10.1109/ICDE48307.2020.00149]

[26] Ren YX, Zhu H, Zhang JW, Dai P, Bo LF. EnsemFDet: An ensemble approach to fraud detection based on bipartite graph. In: Proc. of the 37th IEEE Int'l Conf. on Data Engineering (ICDE). Chania: IEEE, 2021. 2039–2044. [doi: 10.1109/ICDE51399.2021.00197]

[27] Al-Eidi S, Chen YZ, Darwishand O, Alfosool AMS. Time-ordered bipartite graph for spatio-temporal social network analysis. In: Proc. of the 2020 Int'l Conf. on Computing, Networking and Communications (ICNC). Big Island: IEEE, 2020. 833–838. [doi: 10.1109/ICNC47757.2020.9049668]

[28] Hafsi A, Gamha Y, Ben Njima C, Ben Romdhane L. BIG-SWSDM: BIpartite graph based social Web service discovery model. In: Proc. of the 23rd Int'l Conf. on Business Information Systems. Colorado Springs: Springer, 2020. 307–318. [doi: 10.1007/978-3-030-53337-3_23]

[29] He XN, Deng K, Wang X, Li Y, Zhang YD, Wang M. LightGCN: Simplifying and powering graph convolution network for recommendation. In: Proc. of the 43rd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM, 2020. 639–648. [doi: 10.1145/3397271.3401063]

[30] He XN, Liao LZ, Zhang HW, Nie LQ, Hu X, Chua TS. Neural collaborative filtering. In: Proc. of the 26th Int'l Conf. on World Wide Web. Perth: Int'l World Wide Web Conf. Steering Committee, 2017. 173–182. [doi: 10.1145/3038912.3052569]

[31] Park JW, Lee MW, Kim J, Hwang SW, Kim S. CosTriage: A cost-aware triage algorithm for bug reporting systems. In: Proc. of the 25th AAAI Conf. on Artificial Intelligence. San Francisco: AAAI, 2011. 139–144. [doi: 10.1609/aaai.v25i1.7839]

[32] Chen JY, Zhang HW, He XN, Nie LQ, Liu W, Chua TS. Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In: Proc. of the 40th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. Shinjuku: ACM, 2017. 335–344. [doi: 10.1145/3077136.3080797]

[33] Wang H, Wang NY, Yeung DY. Collaborative deep learning for recommender systems. In: Proc. of the 21st ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Sydney: ACM, 2015. 1235–1244. [doi: 10.1145/2783258.2783273]

[34] Wang X, He XN, Nie LQ, Chua TS. Item Silk Road: Recommending items from information domains to social users. In: Proc. of the 40th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. Shinjuku: ACM, 2017. 185–194. [doi: 10.1145/3077136.3080771]

[35] Xin X, He XN, Zhang YF, Zhang YD, Jose J. Relational collaborative filtering: Modeling multiple item relations for recommendation. In:

Proc. of the 42nd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. Paris: ACM, 2019. 125–134. [doi: 10.1145/3331184.3331188]

[36] Cheng ZY, Ding Y, Zhu L, Kankanhalli M. Aspect-aware latent factor model: Rating prediction with ratings and reviews. In: Proc. of the 2018 World Wide Web Conf. Lyon: Int'l World Wide Web Conf. Steering Committee, 2018. 639–648. [doi: 10.1145/3178876.3186145]

[37] Wang X, Wang DX, Xu CR, He XN, Cao YX, Chua TS. Explainable reasoning over knowledge graphs for recommendation. In: Proc. of the 33rd AAAI Conf. on Artificial Intelligence. Honolulu: AAAI, 2019. 5329–5336. [doi: 10.1609/aaai.v33i01.33015329]

[38] Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L. BPR: Bayesian personalized ranking from implicit feedback. In: Proc. of the 25th Conf. on Uncertainty in Artificial Intelligence. Montreal: AUAI Press, 2009. 452–461.

[39] Ma H, King I, Lyu MR. Effective missing data prediction for collaborative filtering. In: Proc. of the 30th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. Amsterdam: ACM, 2007. 39–46. [doi: 10.1145/1277741.1277751]

[40] Wu JC, Wang X, Feng FL, He XN, Chen L, Lian JX, Xie X. Self-supervised graph learning for recommendation. In: Proc. of the 44th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM, 2021. 726–735. [doi: 10.1145/3404835.3462862]

[41] Zhao XY, Xia L, Zou LX, Liu H, Yin DW, Tang JL. UserSim: User simulation via supervised generativeadversarial network. In: Proc. of the 2021 Web Conf. Ljubljana: ACM, 2021. 3582–3589. [doi: 10.1145/3442381.3450125]

[42] Dai J, Li QS, Xie SL, Li DZ, Chu H. PCG: A joint framework of graph collaborative filtering for bug triaging. Journal of Software: Evolution and Process, 2024, 36(9): e2673. [doi: 10.1002/smr.2673]

[43] Goldberg D, Nichols D, Oki BM, Terry D. Using collaborative filtering to weave an information tapestry. Communications of the ACM, 1992, 35(12): 61–70. [doi: 10.1145/138859.138867]

[44] Zhu ZY, Li Y, Wang Y, Wang YJ, Tong HH. A deep multimodal model for bug localization. Data Mining and Knowledge Discovery, 2021, 35(4): 1369–1392. [doi: 10.1007/s10618-021-00755-7]

[45] Wan Y, Shu JD, Sui YL, Xu GD, Zhao Z, Wu J, Yu P. Multi-modal attention network learning for semantic source code retrieval. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 13–25. [doi: 10.1109/ASE.2019.00012]

[46] Ye X, Shen H, Ma X, Bunescu R, Liu C. From word embeddings to document similarities for improved information retrieval in software engineering. In: Proc. of the 38th Int'l Conf. on Software Engineering. Austin: ACM, 2016. 404–415. [doi: 10.1145/2884781.2884862]

[47] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv:1301.3781, 2013.

[48] Zheng W, Chen JZ, Wu XX, Chen X, Xia X. Empirical studies on deep-learning-based security bug report prediction methods. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1294–1313 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5954.htm [doi: 10.13328/j.cnki.jos.005954]

[49] Guo SK, Zhang XY, Yang X, Chen R, Guo C, Li H, Li TT. Developer activity motivated bug triaging: Via convolutional neural network. Neural Processing Letters, 2020, 51(3): 2589–2606. [doi: 10.1007/s11063-020-10213-y]

[50] Bock HH. Clustering methods: A history of K-means algorithms. In: Brito P, Cucumel G, Bertrand P, Carvalho F, eds. Selected Contributions in Data Analysis and Classification. Berlin: Springer, 2007. 161–172. [doi: 10.1007/978-3-540-73560-1_15]

[51] Syakur MA, Khotimah BK, Rochman EMS, Satoto BD. Integration K-means clustering method and elbow method for identification of the best customer profile cluster. IOP Conf. Series: Materials Science and Engineering, 2018, 336: 012017. [doi: 10.1088/1757-899X/336/1/012017]

[52] Dai J. Research on self-supervised learning for collaborative triaging of software bugs [Ph.D. Thesis]. Xi'an: Xidian University, 2023 (in Chinese with English abstract). [doi: 10.27389/d.cnki.gxadu.2023.000028]

[53] Ma QL, Zheng ZJ, Zheng JW, Li S, Zhuang WQ, Cottrell GW. Joint-label learning by dual augmentation for time series classification. In: Proc. of the 35th AAAI Conf. on Artificial Intelligence. AAAI, 2021. 8847–8855. [doi: 10.1609/aaai.v35i10.17071]

[54] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv:1412.6980, 2017.

[55] Google. Issues—Chromium. 2023. https://bugs.chromium.org/p/chromium/issues/list

[56] Mozilla. Components for core. 2023. https://bugzilla.mozilla.org/describecomponents.cgi?product=Core

[57] Mozilla. Components for Firefox. 2023. https://bugzilla.mozilla.org/describecomponents.cgi?product=Firefox

附中文参考文献:

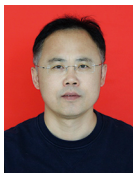
[1] 张天伦, 陈荣, 杨溪, 祝宏玉. 基于代价极速学习机的软件缺陷报告分类方法. 软件学报, 2019, 30(5): 1386–1406. http://www.jos.org.cn/1000-9825/5725.htm [doi: 10.13328/j.cnki.jos.005725]

[10] 何吉元, 孟昭鹏, 陈翔, 王赞, 樊向宇. 一种半监督集成跨项目软件缺陷预测方法. 软件学报, 2017, 28(6): 1455–1473. http://www.jos.

org.cn/1000-9825/5228.htm [doi: 10.13328/j.cnki.jos.005228]
[48] 郑炜, 陈军正, 吴潇雪, 陈翔, 夏鑫. 基于深度学习的安全缺陷报告预测方法实证研究. 软件学报, 2020, 31(5): 1294–1313. http://www.jos.org.cn/1000-9825/5954.htm [doi: 10.13328/j.cnki.jos.005954]
[52] 歹杰. 面向软件缺陷协同分派的自监督学习研究 [博士学位论文]. 西安: 西安电子科技大学, 2023. [doi: 10.27389/d.cnki.gxadu.2023.000028]

谢生龙(1989－), 男, 博士生, CCF 高级会员, 主要研究领域为软件运维, 软件缺陷定位与分派, 软件自优化.

歹杰(1993－), 男, 博士生, 主要研究领域为软件缺陷分派, 智能推荐, 图神经网络.



李青山(1973－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为智能软件工程, 自适应软件演化, 开源基础软件.

崔笛(1994－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为代码分析与软件安全, 软件架构自适应与演化.