软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(5):2340−2358 [doi: 10.13328/j.cnki.jos.006992]

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

开源软件缺陷的跨项目相关问题推荐方法*

刘宝川, 张 莉, 刘桢炜, 蒋 竞

(软件开发环境国家重点实验室 (北京航空航天大学), 北京 100191)

通信作者: 蒋竞, E-mail: jiangjing@buaa.edu.cn

摘 要: GitHub 是著名的开源软件开发社区, 支持开发人员在开源项目中使用问题追踪系统来处理问题. 在软件缺陷问题的讨论过程中, 开发人员可能指出与该缺陷问题相关的其他项目问题 (称为跨项目相关问题), 为缺陷问题的修复提供参考信息. 然而, GitHub 平台中托管了超过 2 亿的开源项目和 12 亿个问题, 导致人工识别和获取跨项目相关问题的工作极其耗时. 提出为缺陷问题自动化推荐跨项目相关问题的方法 CPIRecom. 为了构建预选集, 采用项目之间历史相关问题对的数量和问题发布时间间隔筛选问题. 其次, 为了精准推荐, 采用 BERT 预训练模型提取文本特征, 分析项目特征. 然后使用随机森林算法计算预选问题与缺陷问题的相关概率, 最终根据相关概率排名得到推荐列表. 模拟 CPIRecom 方法在 GitHub 平台的使用情况. CPIRecom 方法的平均倒数排名达到 0.603, 前 5 项查全率达到 0.715.

关键词: 开源项目; 缺陷修复; 跨项目相关问题; 问题推荐

中图法分类号: TP311

中文引用格式: 刘宝川, 张莉, 刘桢炜, 蒋竞. 开源软件缺陷的跨项目相关问题推荐方法. 软件学报, 2024, 35(5): 2340–2358. http://www.jos.org.cn/1000-9825/6992.htm

英文引用格式: Liu BC, Zhang L, Liu ZW, Jiang J. Cross-project Issue Recommendation Method for Open-source Software Defects. Ruan Jian Xue Bao/Journal of Software, 2024, 35(5): 2340–2358 (in Chinese). http://www.jos.org.cn/1000-9825/6992.htm

Cross-project Issue Recommendation Method for Open-source Software Defects

LIU Bao-Chuan, ZHANG Li, LIU Zhen-Wei, JIANG Jing

(State Key Laboratory of Software Development Environment (Beihang University), Beijing 100191, China)

Abstract: GitHub is a well-known open-source software development community that supports developers using the issue tracking system in each open-source project on GitHub to address issues. During the discussion of an issue about a defect, the developer may point out issues from other projects correlated to the defect, which are called cross-project issues, so as to provide reference information for fixing the defect. However, there are more than 200 million open-source projects and 1.2 billion issues on the GitHub platform, making it time-consuming to identify and acquire cross-project issues manually. This study presents a cross-project issue recommendation method CPIRecom for open-source software defects. This study builds a pre-selection set by filtering issues based on the number of historical issue pairs and the time interval for reporting issues. Then, the study also proposes an accurate recommendation model, which extracts textual features based on the pre-trained model of BERT, analyzes features of projects, calculates the relevant probability between defects and issues from the pre-selection set based on a random forest classifier, and obtains the recommendation list according to the ranking. This study simulates the application of CPIRecom method on GitHub platform. The mean reciprocal rank of CPIRecom method reaches 0.603, and the Recall@5 reaches 0.715 on the simulative test set.

Key words: open-source project; defect fixing; cross-project issue; recommendation of issue

开源软件在大规模群体的协同开发过程中彼此关联、共同发展, 促进了软件供应链的形成和发展[1,2]. GitHub 是一个基于 Web 的项目托管平台[3−5], 提供了问题追踪系统 (issue tracking system) 辅助开发人员管理项目中出现

* 基金项目: 科技创新 2030—“新一代人工智能”重大项目 (2021ZD0112901); 国家自然科学基金 (62177003)
收稿时间: 2022-11-03; 修改时间: 2023-01-07, 2023-04-06; 采用时间: 2023-06-29; jos 在线出版时间: 2023-10-25
CNKI 网络首发时间: 2023-10-26

的问题[6,7]. 在问题追踪系统中, 用户可以报告开发过程中发现的软件缺陷, 其他开发人员可以围绕感兴趣的缺陷问题参与讨论. 开发人员可能在缺陷问题的讨论中指出与缺陷问题相关的来自其他项目中的问题 (跨项目相关问题)[8], 为缺陷问题的修复提供更多有效的信息. Li 等人分析了 GitHub 平台中的 16 584 个 Python 项目, 发现跨项目相关问题的链接比例达到 14.67%[9].

然而, GitHub 平台托管了全球超过 2 亿的开源项目和 12 亿个问题, 从海量的项目和问题中发现与缺陷问题相关的问题需要开发人员花费大量的时间和精力. 而且, 识别软件缺陷的跨项目相关问题往往需要依赖开发人员的开发经验和知识, 使得人工识别跨项目相关问题具有挑战性[10]. 在一个缺陷问题刚发布时, 开发人员不确定其是否会被后续缺陷的修改覆盖. 不管该缺陷问题涉及的修改是否被后续修改覆盖, 都需要尽快处理缺陷问题. 因此, 需要一种自动检测跨项目相关问题的方法, 能够针对一个新发布的缺陷问题及时地推荐跨项目相关问题, 帮助缺陷问题的修复.

Ma 等人[11]针对 Python 生态系统中 7 个开源项目的缺陷问题进行经验研究, 分析引用跨项目相关问题的缺陷问题的修复情况. Ma 等人[11]发现, 引用跨项目相关问题的软件缺陷的修复时间比其他缺陷问题的修复时间多 31%, 参与的开发人员数量也比其他缺陷问题中的开发人员数量多 24%. 通过调查问卷分析, 相对比其他缺陷问题, 40.74% 的开发人员认为引用跨项目相关问题的缺陷问题更严重, 47.6% 的缺陷问题需要开发人员花费 24 h 以上的时间才能追踪到跨项目相关问题. 因此, 通过对新发布的缺陷问题推荐跨项目相关问题, 可以减少开发人员人工追踪项目相关问题的时间, 跨项目相关问题可能帮助开发人员尽快修复缺陷问题. 另外, 跨项目相关问题中的参与人员可能对当前缺陷的修复有经验. 管理人员可以考虑把缺陷问题推荐给跨项目问题的参与人员, 邀请他们修复缺陷问题. 但是缺乏文献自动推荐跨项目相关问题. 目前, 针对问题之间关系的研究工作主要聚焦于检测同一项目内的重复问题. Alipour 等人[12]提出了一种基于上下文信息的重复问题检测方法. Thung 等人[13]考虑问题中的标题和正文信息, 通过计算问题之间文本信息的相似度, 检测重复的问题. 但是, 重复问题检测方法无法直接应用于检测并推荐缺陷问题的跨项目相关问题.

本文提出了一种自动化的跨项目相关问题推荐方法 CPIRecom (cross-project issue recommendation). 为了构建预选集, 本文使用项目之间历史相关问题对的数量和问题发布时间间隔筛选问题. 其次, 为了精准推荐, 本文采用 BERT 预训练模型提取文本特征, 分析项目特征. 然后使用随机森林算法计算预选问题与缺陷问题的相关概率, 最终根据相关概率排名得到推荐列表. 为了评估 CPIRecom 方法的推荐效果, 本文分别构建了调和数据集和模拟数据集, 然后分别在两个数据集上比较 CPIRecom 方法与现有方法 DupFinder 和 PrFinder 的推荐效果. 实验表明, 在调和测试集中, CPIRecom 方法的平均倒数排名达到 0.985, 前 5 项查全率达到 0.985; 在模拟测试集中, CPIRecom 方法的平均倒数排名达到 0.603, 前 5 项查全率达到 0.715. 与现有方法 DupFinder、PrFinder 相比, CPIRecom 方法在调和测试集上的平均倒数排名分别提升了 101.43% 和 83.09%, 前 5 项查全率分别提升了 77.48% 和 54.15%; CPIRecom 方法在模拟测试集上的平均倒数排名分别提升了 224.19% 和 197.04%, 前 5 项查全率分别提升了 243.75% 和 223.53%.

1 研究背景

GitHub 作为全球最大的开源软件开发社区之一, 促进了软件供应链的形成和发展. 开源项目之间往往彼此关联, 一个项目中软件缺陷的产生可能来源于它的上游项目, 上游项目的问题可能影响下游项目的软件缺陷[11]. 这不仅给开发人员的缺陷修复工作增加了难度, 同时也对维护软件供应链带来了挑战. 给定一个新发布且尚无人评论的缺陷问题, 如果针对缺陷问题的标题和正文能够及时发现与缺陷问题相关的其他问题并提供给开发人员, 可以提高修复缺陷问题的效率. 同时, 如果发现一个项目的软件缺陷和其上游项目的问题相关, 还可以提醒软件供应链中也依赖该上游项目的其他下游项目.

GitHub 平台提供了问题追踪系统用于管理用户报告的问题. 用户在问题追踪系统中可以报告软件缺陷, 其他开发人员可以围绕感兴趣的缺陷问题参与讨论. 在管理问题报告 (issue) 时, 管理人员通过添加标签的方式标识问题的类别. 例如: 管理人员会通过标签“bug”“defect”等同义词将该问题标识为软件缺陷[14]. 在问题发布后, 开发

人员发表评论 (comment), 讨论问题解决方案. 本文针对问题追踪系统中的缺陷问题展开研究. 以 tensorflow/tensorflow 项目中编号为 34456 的缺陷问题 (https://github.com/tensorflow/tensorflow/issues/34456) 为例, 它的基本结构如图 1 所示.

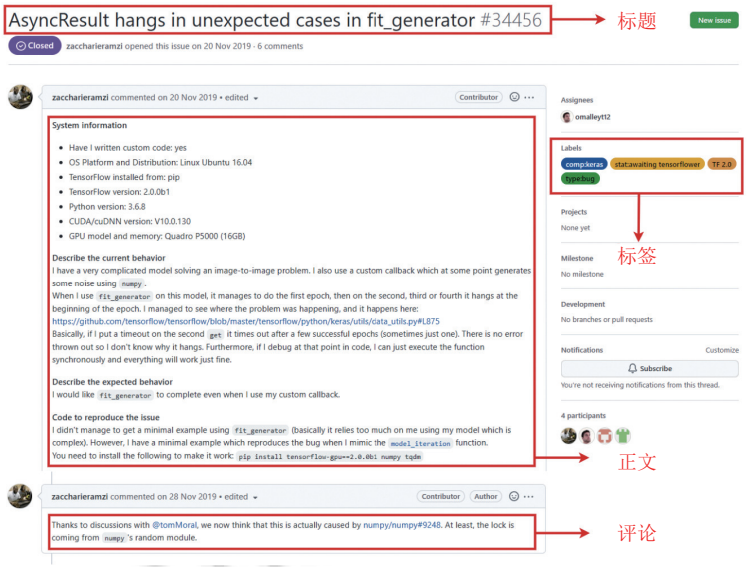

图 1 缺陷问题的样例

1) 标题, 即缺陷问题报告的题目.

2) 正文, 问题的主体部分, 包括对问题的具体描述.

3) 评论, 开发人员针对缺陷问题参与的讨论信息.

4) 标签, 管理人员添加标签来标识问题的类别.

如图 1 所示, 在 tensorflow/tensorflow 项目中编号为 34456 的缺陷问题中有开发人员指出该缺陷问题的根源可能是 numpy/numpy 项目中编号为 9248 的问题, 本文将 numpy/numpy 项目中编号为 9248 的问题称为该缺陷问题的跨项目相关问题.

2 跨项目相关问题推荐的相关工作

Ma 等人[11]针对 Python 生态系统中引用跨项目相关问题的缺陷问题进行经验研究, 发现相比其他缺陷问题, 引用跨项目相关问题的缺陷问题的修复时间要更长, 参与讨论的开发人员数量更多. Ma 等人同时向开发人员发放了调查问卷, 结果表明接近一半的缺陷问题需要开发人员花费 24 h 以上的时间才能发现到跨项目相关问题. 同时, 超过 60% 的开发人员表示发现缺陷问题的跨项目相关问题是困难的. 因此, 需要研究针对缺陷问题的跨项目相关问题自动推荐.

一些研究人员关注问题追踪系统中的重复问题检测工作[12−16]. Alipour 等人[12]提出了一种基于上下文信息的重复问题检测方法, 提取问题的功能性需求与非功能性需求信息、文本相似度、问题发布时间间隔以及问题类型特征, 使用逻辑回归模型训练重复问题检测模型. Thung 等人[13]提出了一种无监督重复缺陷问题检测方法 DupFinder, 使用词频-逆文档频率算法将问题标题和问题正文向量化, 利用余弦相似度计算问题标题向量的相似度与问题正文向量的相似度, 最终根据历史缺陷报告与新缺陷报告的相似度为开发人员推荐一个潜在重复的缺陷报告列表. Wang 等人[14]提出了一种用于检测项目内重复拉取请求的方法 PrFinder, 提取拉取请求的标题、描述、修改文件列表、引用的问题、时间间隔等特征, 使用自适应增强 (AdaBoost) 算法计算历史拉取请求和新提交的拉取请求的相似度, 最终通过排序推荐一个可能重复的拉取请求列表. 现有的重复问题检测方法旨在分析缺陷报

告之间的重复关系, 没有考虑缺陷问题和跨项目相关问题之间的相关关系, 但是对问题之间文本相似度的分析可以为跨项目相关问题的推荐提供技术支持.

3 CPIRecom 方法设计

针对人工识别并获取跨项目相关问题的困难, 我们提出了跨项目相关问题的自动化推荐方法 CPIRecom. 本节将介绍方法 CPIRecom 的设计思路和特征.

3.1 设计思路

为了研究跨项目相关问题的自动推荐工作, 首先需要收集预选问题, 然后计算缺陷问题与每个预选问题的相关度. 然而, 截止目前 GitHub 平台托管的开源项目超过 2 亿, 问题数量超过 12 亿. 针对某一缺陷问题, 理论上除缺陷问题所在项目以外的所有项目中的全部问题都在候选集中, 将候选集中海量的问题与该缺陷问题逐一组对进行分析是不现实的, 因而需要对海量的候选问题进行筛选和过滤. 缺陷问题与真实相关的跨项目问题所组建的问题对被称为相关问题对, 缺陷问题与其无关的跨项目问题所组建的问题对被称为无关问题对. 如果真实的相关问题对没有通过筛选未进入预选集, 那么将无法识别出正确的跨项目相关问题. 如果预选集的相关问题对数量较大, 那么将增加后续精准推荐的难度. 因此, 需要构建合适的预选集, 在减少预选集中相关问题对的数量的同时尽量保留真实的相关问题对.

在构造的预选集中, 大量问题对是无关的, 仅有少量问题对是相关的. 因此, 需要设计精准推荐, 从大量的预选问题对中精准地找出相关问题对. 预选集中的问题对分为相关和无关两类, 因此本文将精准推荐任务转换为二分类问题, 通过考虑不同维度的特征以及合适的分类算法设计精准推荐模型, 为缺陷问题自动推荐跨项目相关问题. CPIRecom 方法的整体架构如图 2 所示.

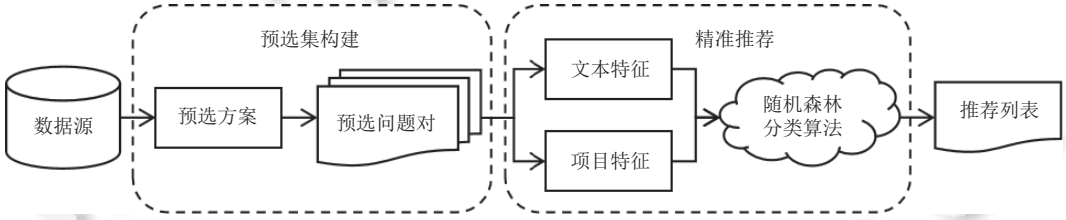

图 2 CPIRecom 方法整体框架

3.2 预选集构建

不同的缺陷问题所对应的跨项目相关问题也并不相同. 因此, 我们需要针对特定的缺陷问题构建预选集. 针对给定的缺陷问题, 本文首先筛选可能存在相关问题的项目, 然后对该项目的问题进行过滤. 顺利通过项目筛选和问题筛选的问题, 即为该缺陷问题对应的预选问题, 进入预选集. 具体步骤如下.

1) 项目筛选

筛选项目是指从所有的候选项目中选择符合条件的项目. 项目之间历史相关问题对的数量较多, 说明项目间可能有紧密的联系, 项目间的问题可能相关. 因此, 本文使用项目之间历史相关问题对的数量筛选项目. 针对一个项目, 本文仅保留和该项目的历史相关问题对超过特定阈值的项目.

2) 问题筛选

筛选问题是指针对特定的缺陷问题, 从符合条件的项目中提取符合要求的问题, 并和该缺陷问题组合构建预选问题对, 即<给定缺陷问题, 预选问题>. 所有预选问题对的集合即为给定缺陷问题的预选集. 问题之间的发布时间接近, 问题之间可能相关. 因此, 本文使用问题发布时间间隔作为预选条件筛选问题. 针对一个缺陷问题, 本文仅保留在该缺陷发布前特定阈值范围内的问题, 而过滤发布时间远早于该缺陷的问题. 由于希望在缺陷发布时推荐跨项目相关问题, 因此不考虑该缺陷发布后的问题.

图 3 展示了项目筛选和问题筛选流程的样例. 本文为项目 A 中的缺陷问题 a1 构建预选集. 首先, 根据项目之间历史相关问题对的数量阈值为 2 筛选项目. 其中, 项目 B (虚线框内的项目) 与项目 A 的历史相关问题对的数量为 1 个, 不满足该条件被过滤掉. 项目 C 与项目 A 的历史相关问题对数量为 8 个, 满足项目之间历史相关问题对的数量条件. 然后, 根据问题发布时间间隔阈值为 3 个月从项目 C 中筛选问题. 在项目 C 中, 问题 c2 和问题 c4 (虚线框内的问题) 与问题 a1 的发布时间间隔均超过 3 个月, 不满足该条件被过滤掉, 问题 c1 和问题 c3 与问题 a1 的发布时间间隔均在 3 个月之内, 满足问题发布时间间隔条件. 因此, 我们得到缺陷问题 a1 的预选问题 {c1,c3} , 最后, 本文将 2 个预选问题分别与缺陷问题 a1 组合, 得到缺陷问题 a1 的预选集, 即 {< a1,c1 >,< a1,c3 >} .

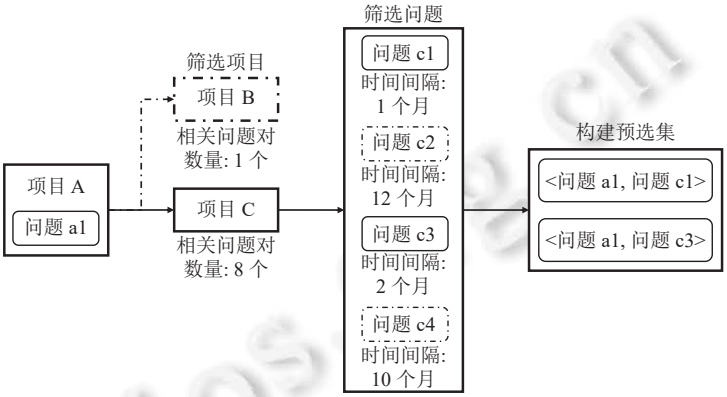

图 3 预选集构建样例

3) 求解最优方案

根据项目之间历史相关问题对的数量和问题发布时间间隔两个预选条件, 针对特定缺陷问题构建符合条件的预选问题对. 其中, 预选问题均为跨项目问题, 且预选问题可能与该缺陷问题相关, 也可能不相关. 随着预选条件越严格, 预选问题的数量越少. 在不断减少的预选问题中, 除了一些不相关的问题会被过滤掉, 一些相关的问题 (即真实的跨项目相关问题) 也会因为不满足预选条件被过滤掉. 当预选问题中真实的跨项目相关问题数量为 0 时, 再先进的推荐方法也检测不出跨项目相关问题. 因此, 本文预选问题时, 既要尽量减少预选集中问题对的数量, 同时也要尽量保留真实相关的跨项目问题.

为解决上述问题, 本文分别提出了预选问题对数量和相关问题对覆盖率两个指标度量预选集构建方案. 其中, 预选问题对数量是指针对所有缺陷问题的预选问题对数量求平均值. 首先, 针对每个特定缺陷问题, 我们计算为其构建的预选问题对数量. 然后, 计算所有缺陷问题的预选问题对数量的平均值. 缺陷问题和对应真实的跨项目相关问题组成的预选问题对被称为相关问题对. 缺陷问题和无关问题组成的预选问题对被称为无关问题对. 相关问题对覆盖率是指预选集中相关问题对的数量占预选前所有相关问题对数量的比例, 用于评估真实的跨项目相关问题通过预选的情况. 计算公式为:

$$\text{相关问题对覆盖率}=\frac{\text{预选集中的相关问题对数量}}{\text{相关问题对的总数量}} \tag{1}$$

例如, 给定缺陷问题 a1和a2 , 缺陷问题 a1 的真实跨项目相关问题为 b1 和 b2, 缺陷问题 a2 的真实跨项目相关问题为 b3 . 则缺陷问题 a1 的所有相关问题对是 < a1,b1 >,< a1,b2 > , 缺陷问题 a2 的所有相关问题对是 < a2,b3 > , 缺陷问题 a1和a2 的所有相关问题对数量为 3. 假设缺陷问题 a1 的预选集为 {< a1,b1 >,<a1,b4 >} , 缺陷问题 *a*2 的预选集为 {< a2,b3 >,<a2,b5 >} . 其中, 预选集中的真实相关问题对是 < a1,b1 > 和 < a2,b3 > , 数量为 2. 由计算公式可得, 相关问题对覆盖率为 66.67%.

为了在减少预选集中的问题对数量和保留真实的跨项目相关问题之间取得平衡, 本文基于帕累托最优理论[17]分析不同阈值条件下的预选集问题对数量与覆盖率. 作为博弈论中的重要概念, 帕累托最优的提出最早是为

了解决经济领域里资源配置的难题, 并且在经济学、工程学等社会科学中有着广泛的应用. 帕累托最优的思想是一种评价多目标问题解优劣的处理方法, 帕累托最优解集是指可行域中所有非劣解的集合, 帕累托最优前沿是帕累托最优解集对应目标值的集合. 即对帕累托最优前沿中的解 $A(x_1,y_1)$ 而言, 找不到非最优前沿中的任一解 $B(x_2,y_2)$, 使得 x_2 优于 x_1 且 y_2 优于 y_1.

因此, 基于预选问题对数量和相关问题对覆盖率构建的帕累托最优前沿中的任一解, 都是可选的预选集构建方案. 在实际项目中, 管理者可以根据个性化需求选择合适的预选集构建方案.

3.3 精准推荐

本文的精准推荐分为两个阶段: 训练阶段和检测阶段, 如图 4 所示. 接下来, 本文分别介绍训练阶段和检测阶段中详细的处理步骤.

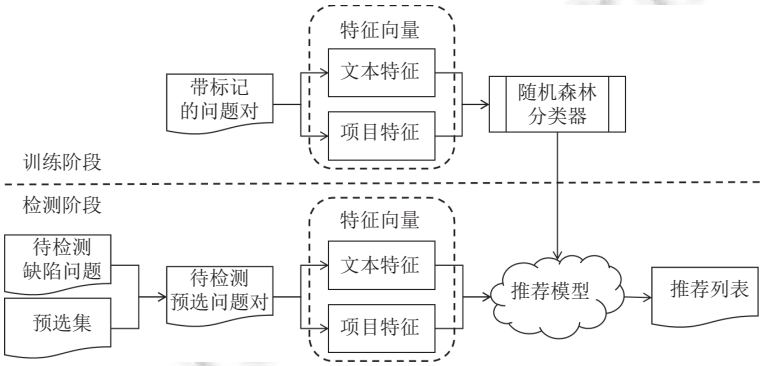

图 4 精准推荐的设计流程

在训练阶段, CPIRecom 方法首先基于历史数据将构建的问题对分别标记为“相关问题对”和“无关问题对”. 其次, 将带标记的问题对预处理后输入 BERT 预训练模型得到问题对的文本向量, 并计算问题对之间的文本相似度特征. 同时, 提取问题对所在项目之间的特征. 然后, 将问题对的文本特征和项目特征输入随机森林分类器中进行学习和训练, 获得推荐模型.

在检测阶段, 针对一个待检测缺陷问题, 首先基于预选方案为缺陷问题构建预选集, 得到待检测的预选问题对. 然后, 使用与训练阶段相同的特征提取策略提取出待检测预选问题对的文本特征和项目特征. 接下来, 输入文本特征和项目特征, 使用训练阶段基于随机森林分类器得到的推荐模型, 计算预选问题与缺陷问题的相关概率. 换句话说, 训练阶段使用随机森林分类器得到的推荐模型, 被应用到检测阶段. 最后, 按照相关概率对预选问题排序, 得到跨项目相关问题的推荐列表.

接下来, 本文将详细阐述特征提取方案.

3.3.1 文本特征

开发人员在报告问题时会使用标题信息和正文信息对该问题进行详细描述. 缺陷问题和跨项目相关问题之间可能存在文本相似性, 比如讨论相似的功能或技术手段. 例如: 项目 gatsbyjs/gatsby 中编号为#25405 的缺陷 (https://github.com/gatsbyjs/gatsby/issues/25405) 描述了一个标题为“Query.node defined in resolvers, but not in schema”的问题. 开发人员在评论信息中提及了项目 apollographql/apollo-server 中编号为#1534 的问题 (https://github.com/apollographql/apollo-server/issues/1534), “Quick googling shows that it is probably an apollo-server issue (at least there is a couple of similar reports in their repo): apollographql/apollo-server#1534”. 项目 apollographql/apollo-server 中问题#1534 的标题为“Upload defined in resolvers, but not in schema”. 结合标题和正文信息可以看出, 项目 gatsbyjs/gatsby 的缺陷问题#25405 与项目 apollographql/apollo-server 的问题#1534 存在文本相似性. 文本信息可能有助于推荐跨项目相关问题. 接下来介绍如何提取文本特征.

首先, 针对问题对的文本信息进行预处理. 然后, 使用 BERT 预训练模型将预处理后的文本信息向量化. 最后,

通过计算文本向量的余弦相似度、Jaccard 相似度和 *Dice* 相似度得到文本特征.

预处理步骤包括去除非文本信息, 统一为小写字母, 文本分割, 去除停用词, 词形还原. 其中, 本文去除标题中的非文本信息, 包括电话号码、电子邮件地址、表情等, 并将大写字母全部转化为小写字母; 使用空白符作为分隔符, 将文本信息划分为单词数组. 停用词是指类似限定词“the”“an”“those”等使用普遍, 与其他词相比提供了的信息十分有限, 不具备实际含义的词语, 在进行自然语言处理时, 通常需要去除停用词, 以节省存储空间和提升处理效率. 词形还原是指将一个单词转化为其根形式.

为了表示文本的语义, 本文采用预训练策略将预处理后的词数组转化为计算机可以直接利用的浮点型数组. 目前, 基于 BERT 的预训练模型是自然语言处理领域中的重要技术[18]. 我们使用基于 BERT 的预训练模型将词数组转化为能够表征语义信息的文本向量. 由于使用 BERT 向量化问题标题和描述的过程较为耗时, 因此在构建预选集时, 本文使用历史相关问题对数量筛选项目, 然后使用问题发布时间间隔筛选问题. 本文通过预选控制候选问题的数量, 从而减少 BERT 向量化的时间开销.

针对问题对的文本向量, 本文采用文本相似度计算问题的文本特征. 文本的相似度是把文本投影到向量空间, 通过比较计算向量的空间距离来比较文本的相似度. 本文分别计算了问题对的余弦相似度、Jaccard 相似度和 Dice 相似度.

1) 余弦相似度

余弦相似度指利用两个向量夹角的余弦值来预测它们之间的相似度. 余弦相似度的计算公式如下, 其中 a 表示缺陷问题的文本向量, b 表示跨项目问题的文本向量.

$$\cos(a,b)=\frac{a\cdot b}{\|a\|\|b\|} \tag{2}$$

2) Jaccard 相似度

Jaccard 相似度是指两个集合 a 和 b 的交集元素在 a 和 b 的并集中所占的比例, 可以用于衡量两个集合的相似程度. Jaccard 相似度的计算公式如下.

$$J(a,b)=\frac{|a\cap b|}{|a\cup b|} \tag{3}$$

3) *Dice* 相似度

Dice 相似度是一种集合相似度度量指标, 通常用于计算两个样本的相似度. 本文通过计算文本向量的 *Dice* 相似度来衡量文本信息的相似度. *Dice* 相似度的计算公式如下.

$$Dice(a,b)=\frac{2|a\cap b|}{|a|+|b|} \tag{4}$$

余弦相似度、Jaccard 相似度和 *Dice* 相似度是 3 种常见的用于计算文本相似度的方法. 其中, 余弦相似度相比 Jaccard 相似度最大的不同在于它考虑到了文本的词频, 而且关注文本向量在方向上的差异. 但是, 针对重复字符较多的文本, 余弦相似度的计算可能会受到词频的影响. Jaccard 相似度的计算方式和 *Dice* 相似度类似, 即两个集合中共有的属性越多, 二者越相似. Jaccard 相似度和 *Dice* 相似度只关注某一属性是否共有, 而忽视其出现的频率. 针对重复字符较多的文本, Jaccard 相似度和 *Dice* 相似度可以对文本进行去重处理, 获得很好的计算效果. 因此, 本文尝试通过组合 3 种相似度计算方法来评估文本之间的相似度以提高推荐效果, 并且第 5.3.3 节的结果也验证了同时使用 3 种相似度计算方法的必要性.

针对缺陷问题的标题信息和正文信息、跨项目问题的标题信息和文本信息, 本文分别计算余弦相似度、Jaccard 相似度和 *Dice* 相似度的以下文本特征.

1) 标题相似度

标题描述问题信息, 本文分析缺陷问题标题与跨项目问题标题之间的文本相似度, 包括标题余弦相似度、标题 Jaccard 相似度和标题 *Dice* 相似度.

2) 正文相似度

正文是对问题的详细描述, 通常包括问题的现象, 如何复现问题等. 部分开发人员还会在问题正文中贴上代

码, 本文将正文中出现的代码作为文本信息进行统一处理. 本文分析缺陷问题正文与跨项目问题正文之间的文本相似度, 包括正文余弦相似度、正文 Jaccard 相似度和正文 *Dice* 相似度.

3) 缺陷问题标题和跨项目问题正文相似度

本文将缺陷问题和跨项目问题的标题和正文进行交叉组合, 计算缺陷问题标题和跨项目问题正文的余弦相似度、Jaccard 相似度和 *Dice* 相似度.

4) 缺陷问题正文和跨项目问题标题相似度

本文将缺陷问题和跨项目问题的正文和标题进行交叉组合, 计算缺陷问题正文与跨项目问题标题的余弦相似度、Jaccard 相似度和 *Dice* 相似度.

3.3.2 项目特征

针对问题对所在不同项目的特点, 本文提取问题对所在项目之间的特征, 包括编程语言相似度、历史相关问题对的数量等.

1) 编程语言相似度

项目使用的编程语言相近, 两个项目之间可能存在调用、依赖、参考等关系, 两个项目的问题之间可能存在关联. 因此本文采用项目使用的编程语言相似度特征. 首先, 本文统计数据集中所有项目使用的编程语言种类. 然后, 定义编程语言向量覆盖所有的编程语言, 即该向量的每一维度对应一种编程语言. 本文统计每个项目使用的各类编程语言及对应代码行数集合, 然后计算每种编程语言的代码行数占所有编程语言代码行数的比例, 得到该编程语言维度的数值, 从而构建该项目的编程语言向量. 最后, 利用余弦相似度可以计算向量之间方向差异的特点, 分析两个项目编程语言向量之间的相似度, 得到缺陷问题与跨项目问题所在项目之间使用编程语言的相似度.

2) 项目之间历史相关问题对的数量

对未来的推测是基于对历史数据的观察, 缺陷问题所在项目与跨项目问题所在项目之间的历史相关问题对的数量在一定程度上反映了两个项目间问题相关的可能性, 因此本文采用项目之间历史相关问题对的数量特征.

3) 重复贡献者占源项目贡献者比例

对某一项目进行过代码提交 (commit) 的开发人员被称为该项目的贡献者 (contributors). 重复贡献者占源项目贡献者比例是指缺陷问题所在项目 (源项目) 和跨项目问题所在项目 (目标项目) 之间重复贡献者的数量占缺陷问题所在项目 (源项目) 中贡献者数量的比例. 项目之间的贡献者重复度高说明项目之间可能存在紧密的关联, 两个项目的问题之间可能会相关.

4) 重复贡献者占目标项目贡献者比例

上一特征分析了重复贡献者占源项目贡献者比例, 本文同时考虑分析重复贡献者占目标项目贡献者比例. 本文统计源项目贡献者数量、目标项目贡献者数量, 获取源项目贡献者与目标项目贡献者的交集, 然后计算交集中贡献者数量占目标项目贡献者数量的比例.

5) 源项目和目标项目是否归属于同一用户

问题所在项目的全称为 username/reponame. 其中, username 为用户名, reponame 为项目名. 同一用户创建的不同项目之间可能存在紧密的关联, 两个项目的问题之间可能会相关. 因此, 本文考虑源项目和目标项目是否属于同一用户的特征.

4 数据集

为了研究缺陷问题的跨项目相关问题推荐工作, 本文需要收集缺陷问题及其跨项目相关问题的数据. 在本节中, 我们首先介绍如何收集数据. 然后, 人工分析基于链接提取的跨项目相关问题的类型. 接下来, 根据实际数据介绍如何设置预选集阈值. 最后, 基于实际的预选方案介绍如何构建数据集, 然后使用数据集进行特征显著性分析.

4.1 数据收集

本文首先收集 GitHub 平台中提及跨项目相关问题的缺陷问题, 然后分别收集缺陷问题和跨项目相关问题的文本信息和项目信息. 我们使用 GH Archive 和 GitHub Rest API 收集 2020 年 1 月 1 日–2020 年 12 月 31 日的信息. 其

中, GH Archive 记录了 GitHub 中的活动数据, 并以 1 h 作为单位切片存储在 JSON 文件中. REST API 是 GitHub 提供的一套 API 接口, 帮助开发人员通过访问对应的统一资源定位符获取所需的数据. 数据提取的具体流程如下.

首先, 本文从 GH Archive 中收集缺陷问题. 我们的研究对象是缺陷问题, 因此需要从问题追踪系统中抽取属于软件缺陷的问题. GitHub 支持开发人员通过打标签的方式为报告的问题标识类别. Zhang 等人[19]的研究发现, 开发人员标识缺陷问题所使用的关键词包括: bug、defect、flaw、fault、error、failure、problem、trouble. 我们在问题的标签中寻找缺陷关键词, 如果标签中包含上述任一关键词, 则判定该问题为缺陷问题. 以图 1 中 tensorflow/tensorflow 项目编号为 34456 的问题为例, 该问题的标签“type:bug”中包含关键词“bug”, 因此, 我们将该问题称为缺陷问题. 为了保证问题后续不会发生变化, 本文要求问题状态为已解决 (closed). 针对已解决的缺陷问题, 我们从 GH Archive 中收集缺陷问题的文本信息, 如标题、正文和评论. 本文从相关问题对的标题信息和正文信息提取文本特征、从相关问题对所在项目之间提取编程语言相似度等特征. 提取上述特征所需要的数据都在缺陷问题新发布时就能收集. 因此, 本文的方法能够帮助开放 (open) 状态的缺陷问题, 在其新发布时推荐跨项目相关问题.

其次, 本文通过从缺陷问题的标题、正文和评论信息中提取链接[9,11,20]以获取该缺陷问题的跨项目相关问题, 形成< 缺陷问题, 跨项目相关问题>. 链接格式为: username/reponame/issues#{int}或 username/reponame/issues/{int}. 其中 username/reponame 是项目全称, {int}是问题的编号. 通过链接中的项目全称和问题编号, 我们可以在 GitHub 中唯一标识一个具体的问题. 针对链接的跨项目相关问题, 我们采用同样的步骤从 GH Archive 中获取跨项目相关问题的文本信息. 我们把缺陷问题所在项目称为源项目, 跨项目相关问题所在的项目称为目标项目. 由于本文研究的是针对缺陷问题的跨项目相关问题推荐工作, 因此源项目和目标项目不能是同一项目.

接下来, 本文将收集到的缺陷问题和对应的跨项目相关问题组成相关问题对<缺陷问题, 跨项目相关问题>. 对于每个开源项目, 本文统计收集的相关问题对数量. 为了收集足够的数据, 本文过滤相关问题对数量小于 40 的项目, 只保留相关问题对数量大于等于 40 的项目. 经过数据收集及过滤, 本文最终在 30 个开源项目中收集了 1 818 个缺陷问题和 2 165 个相关问题对, 如表 1 所示. 针对 2 165 个相关问题对<缺陷问题, 跨项目相关问题>, 本文采用 GitHub REST API 分别提取相关问题对的其他信息, 如: 问题发布人员、评论信息的参与人员、问题所在项目的编程语言列表、问题所在项目的贡献者列表等. 我们统计数据集中缺陷问题从发布到开发人员引用相关问题的时间. 有 33.17% 的缺陷问题需要开发人员花费 1 个月以上的时间追踪跨项目相关问题. 跨项目相关问题推荐方法能够帮助到这些缺陷问题, 在其新发布且尚无人评论时推荐跨项目相关问题, 减少开发人员人工追踪跨项目相关问题的时间.

表 1　30 个开源项目的数据

项目名称	缺陷数量	相关问题对数量	项目名称	缺陷数量	相关问题对数量
kubeflow/kubeflow	36	40	xamarin/Xamarin.Forms	48	52
microsoft/vscode-go	34	40	FirebaseExtended/flutterfire	50	57
react-navigation/react-navigation	38	40	microsoft/vscode-js-debug	52	59
kubernetes-sigs/kind	27	41	rust-lang/rust	53	60
godotengine/godot	40	41	corona-warn-app/cwa-app-android	45	71
rancher/rancher	37	43	ansible/ansible	82	82
facebook/create-react-app	31	43	quarkusio/quarkus	78	86
Dart-Code/Dart-Code	35	43	microsoft/vscode	81	89
sedouard05/React	4	44	gatsbyjs/gatsby	78	91
Automattic/wp-calypso	46	47	NixOS/nixpkgs	89	106
jellyfin/jellyfin	44	49	tensorflow/tensorflow	96	108
corona-warn-app/cwa-app-ios	23	49	microsoft/vscode-python	101	123
aws/aws-cdk	45	51	nextcloud/server	125	138
mozilla-mobile/android-components	44	51	kubernetes/kubernetes	127	155
vector-im/element-web	43	51	mozilla-mobile/fenix	186	215

训练推荐模型不仅需要由相关问题对代表的正样本, 同时还需要负样本. 因此, 基于第 4.4.1 节的数据集构建过程, 本文采用同样的方式收集预选问题的信息. 最终得到了模型训练需要的所有正负样本的信息.

4.2 问题类型分析

本文使用现有文献[9,11,20]中的链接提取方式构建相关问题对<缺陷问题, 跨项目相关问题>. 其中, Li 等人不仅介绍了引用链接的方式, 也进一步手动分析了开发人员引用跨项目相关问题的原因, 并将缺陷问题和跨项目相关问题之间的关系划分为重复、相似、依赖等关系类型[20]. 本文基于链接的方式提取了缺陷问题对应的跨项目相关问题, 并组成相关问题对. 为了分析跨项目相关问题的作用, 本文随机选取了 100 对相关问题, 人工分析相关问题对的关系类型, 如表 2 所示.

表 2 相关问题对的不同类型

类型	数量	比例 (%)
重复	34	34
相似	38	38
依赖	22	22
其他	6	6

在表 2 中, 有 34% 的相关问题对属于重复关系. 重复关系指缺陷问题和跨项目相关问题讨论的是同一问题, 具有相同的形成原因. 参考重复的跨项目相关问题可以为缺陷问题的修复提供更多的信息. 有 38% 的相关问题对属于相似关系. 相似关系指在缺陷问题和跨项目相关问题中讨论相似的功能或技术手段. 参考相似的跨项目相关问题可以促进缺陷问题的修复. 有 22% 的相关问题对属于依赖关系. 依赖关系是指缺陷问题的解决需要依赖跨项目相关问题, 即缺陷问题被跨项目相关问题所阻塞. 跨项目相关问题即为缺陷问题的上游问题, 只有解决了跨项目相关问题才能修复缺陷问题. 另外, 有 6% 的样本不属于这 3 种类型, 但这些跨项目相关问题仍然有助于缺陷问题的修复. 通过人工分析, 发现多种关系类型的跨项目相关问题均能为缺陷问题的解决提供参考信息. 因此, 基于链接提取的方式建立缺陷问题和跨项目问题之间的关系是有效的.

4.3 预选集阈值设置

本文在第 3.2 节介绍了预选集构建的通用方法. 本节针对第 4.1 节收集的 30 个开源项目介绍如何设置阈值.

本文首先统计了不同预选条件下的预选问题对数量和相关问题对覆盖率, 并对所有问题计算平均值. 如表 3 和表 4 所示, 当历史相关问题对数量为 1 且问题发布时间间隔为 1 时, 预选问题对数量为 3 650 个, 相关问题对覆盖率为 31%. 当问题发布时间间隔保持不变, 历史相关问题对数量增加到 10 时, 预选问题对数量下降到 642 个, 相关问题对覆盖率降低到 14%. 当历史相关问题对数量保持不变, 问题发布时间间隔为 12 时, 预选问题对数量增加到 41 245 个, 相关问题对覆盖率上涨到 59%. 结合表 3 和表 4 可得, 随着预选问题对数量的减少, 相关问题对覆盖率也在下降. 因此, 构造预选集需要权衡预选问题对数量和相关问题对覆盖率.

表 3 不同预选条件下的预选问题对数量

问题发布时间间隔	历史相关问题对数量									
	1	2	3	4	5	6	7	8	9	10
1个月以内	3 650	1 805	1 368	1 083	971	893	831	705	681	642
2个月以内	7 229	3 562	2 697	2 133	1 908	1 758	1 635	1 393	1 346	1 268
3个月以内	10 790	5 340	4 050	3 212	2 884	2 662	2 480	2 122	2 050	1 934
4个月以内	14 323	7 083	5 367	4 251	3 829	3 536	3 294	2 823	2 723	2 571
5个月以内	17 845	8 828	6 683	5 291	4 778	4 413	4 108	3 521	3 394	3 204
6个月以内	21 248	10 522	7 964	6 309	5 710	5 275	4 911	4 213	4 058	3 829
7个月以内	24 618	12 185	9 230	7 315	6 630	6 125	5 700	4 893	4 710	4 445
8个月以内	28 009	13 899	10 529	8 349	7 580	7 007	6 521	5 601	5 389	5 090
9个月以内	31 326	15 570	11 785	9 350	8 502	7 861	7 316	6 286	6 044	5 716
10个月以内	34 642	17 235	13 032	10 350	9 421	8 713	8 109	6 964	6 693	6 335
11个月以内	37 958	18 907	14 287	11 365	10 357	9 584	8 923	7 668	7 364	6 969
12个月以内	41 245	20 553	15 517	12 355	11 268	10 431	9 712	8 347	8 011	7 583

基于表 3 和表 4, 本文以预选问题对数量为横轴, 相关问题对覆盖率为纵轴绘制预选集构建方案的二维散点图, 如图 5 所示. 其中, 每个离散点代表一个预选集构建方案. 在图 5 中, 属于帕累托最优前沿的点共有 22 个 (黑线连接的点), 代表 22 种预选集构建方案. 本文进一步统计了 22 种预选集构建方案的详细指标, 包括: 项目之间历史相关问题对的数量、问题发布时间间隔、预选问题对数量和相关问题对覆盖率, 如表 5 所示.

在表 5 中, 从预选方案#1 到预选方案#8 共 8 个预选方案中的预选问题对数量不超过 3 000, 相关问题对覆盖率不超过 25%. 虽然这些预选方案满足预选问题对数量尽量少的需求, 但是相关问题对覆盖率较低. 这说明通过预选的真实的跨项目相关问题较少, 从而影响精准推荐模型的训练效果. 因此, 本文过滤掉前 8 个预选方案.

表 4 不同预选条件下的相关问题对覆盖率

问题发布时间间隔	历史相关问题对数量									
	1	2	3	4	5	6	7	8	9	10
1个月以内	31%	25%	22%	20%	18%	17%	16%	15%	14%	14%
2个月以内	38%	30%	27%	24%	23%	21%	20%	18%	18%	18%
3个月以内	42%	33%	29%	27%	25%	23%	22%	20%	20%	20%
4个月以内	45%	35%	31%	28%	26%	24%	23%	22%	21%	21%
5个月以内	48%	37%	33%	30%	28%	26%	25%	23%	22%	22%
6个月以内	50%	39%	35%	31%	29%	27%	26%	24%	24%	23%
7个月以内	52%	40%	36%	32%	30%	28%	26%	25%	24%	24%
8个月以内	54%	42%	37%	33%	31%	29%	27%	26%	25%	25%
9个月以内	55%	43%	38%	34%	32%	29%	28%	26%	25%	25%
10个月以内	57%	44%	38%	35%	32%	30%	28%	26%	26%	25%
11个月以内	58%	45%	39%	35%	33%	30%	29%	27%	26%	26%
12个月以内	59%	45%	40%	36%	33%	30%	29%	27%	26%	26%

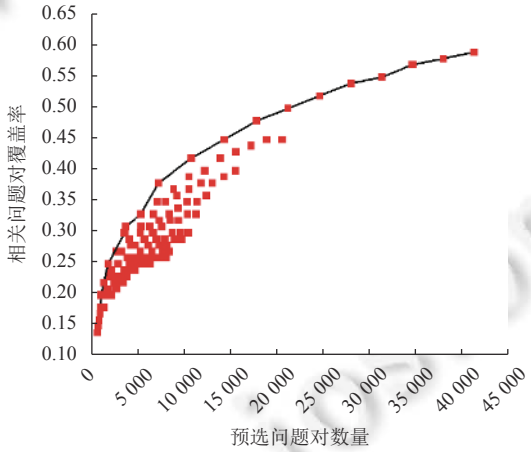

图 5 不同预选条件下的预选集构建方案

对于后面的 14 个预选方案, 即从预选方案#9 到预选方案#22, 预选问题对数量从 3 562 上涨到了 41 245, 同时相关问题对覆盖率从 30% 增加到 59%. 我们进一步分析了预选问题对数量和相关问题对覆盖率的上涨趋势, 发现预选问题对数量的上涨趋势远远大于相关问题对覆盖率的上涨趋势. 我们以预选方案#11 和预选方案#22 进行对比分析. 在预选方案#11 中, 历史相关问题对数量设置为 2, 问题发布时间间隔设置为 3, 此时预选问题对数量为 5 340, 相关问题对覆盖率为 33%. 在预选方案#22 中, 历史相关问题对数量设置为 1, 问题发布时间间隔设置为 12, 此时预选问题对数量为 41 245, 相关问题对覆盖率为 59%. 从预选方案#11 到预选方案#22, 预选问题对数量从 5 340 增加到 41 245, 增长率达到了 6.72 倍. 而相关问题对覆盖率从 33% 上涨到 59%, 增长率仅为 79%. 相关问题

对覆盖率的上涨满足了尽量有更多真实的跨项目相关问题通过预选的需求. 但是, 预选问题对数量的不断增加会加大推荐模型训练过程中的内存与计算开销. 因而, 本文过滤掉预选问题对数量超过 6000 的预选方案. 此时, 可选择的预选方案为#9、#10、#11.

表 5 基于帕累托最优的预选方案

#预选方案 (历史相关问题对数量, 问题发布时间间隔)	预选问题对数量	相关问题对覆盖率 (%)	#预选方案 (历史相关问题对数量, 问题发布时间间隔)	预选问题对数量	相关问题对覆盖率 (%)
#1(10, 1)	681	14	#12(1, 2)	7229	38
#2(8, 1)	705	15	#13(1, 3)	10790	42
#3(7, 1)	831	16	#14(1, 4)	14323	45
#4(6, 1)	893	17	#15(1, 5)	17845	48
#5(5, 1)	971	18	#16(1, 6)	21248	50
#6(4, 1)	1083	20	#17(1, 7)	24618	52
#7(3, 1)	1368	22	#18(1, 8)	28009	54
#8(2, 1)	1805	25	#19(1, 9)	31326	55
#9(2, 2)	3562	30	#20(1, 10)	34642	57
#10(1, 1)	3650	31	#21(1, 11)	37958	58
#11(2, 3)	**5340**	**33**	#22(1, 12)	41245	59

最终, 我们选择了预选方案#11, 因为该方案的相关问题对覆盖率高于预选方案#9、#10. 对于预选方案#11, 预选问题对的数量为 5340 个, 相关问题对数量占所有相关问题对总数量的 33%. 相对比 GitHub 平台中 12 亿个原始候选问题数量, 通过预选的问题数量缩减到原始候选集合的 5×10^{-6}. 此时, 预选条件为: 历史相关问题对数量设置为 2 个, 问题发布时间间隔设置为 3 个月.

4.4 数据集构建和特征显著性检验

基于实际的预选方案, 本节首先介绍如何构建数据集. 然后采用构建的数据集对第 3.3 节中选择的文本特征和项目特征进行显著性分析.

4.4.1 数据集构建

为了进行训练以及验证, 我们需要在数据集中分别构建正样本和负样本, 即构建相关问题对和无关问题对. 然而, 一个缺陷问题往往只拥有少量的相关问题对以及大量的无关问题对. 任何分类模型只要返回问题对无关, 就能达到很高的准确率, 这会导致我们的推荐模型对样本较少的正样本有较差的预测和推荐效果. 因此, 本文采用欠采样的策略构建了调和数据集, 缓和了样本不均衡的情况. 调和数据集中正负样本的比例为 1:16.7, 与真实分布存在较大差异. 为了更好地模拟 CPIRecom 方法在 GitHub 平台的使用情况, 本文构建了模拟数据集, 正负样本比例达到了 1:4716. 在构建调和数据集和模拟数据集时, 所有正样本和负样本使用的文本信息仅包含标题和正文, 不考虑评论, 旨在模拟一个新的缺陷问题刚发布且尚无人评论时的真实环境.

1) 调和数据集

构建调和数据集的正样本取自数据收集阶段获取的 2165 个相关问题对. 在构建负样本时, 本文采取欠采样的策略. 针对正样本中的每个缺陷问题, 首先采用第 3.2 节中的预选方案构建预选集. 然后, 从中随机抽取 20 个与该缺陷无关的候选问题, 即在该缺陷问题的标题、正文及评论中没有出现这些问题的链接. 然后, 将这 20 个候选问题与缺陷问题逐一组对, 构建 20 个负样本, 以此平衡数据集. 通过上述步骤构建的数据集中每个缺陷问题都对应多个候选问题. 此外, 一个缺陷问题可能有多个正样本, 即多个相关问题. 本文将每个缺陷问题对应的候选问题整合到一起, 组成该缺陷问题的调和数据集. 如{缺陷问题 a1, {相关问题 b1, 相关问题 b2,…, 不相关问题 c1,…, 不相关问题 c20}}. 最终, 调和数据集中共包含 2165 个正样本和 36360 个负样本, 正负样本比为 1:16.7. 本文按照缺陷问题的发布时间将调和数据集切分为 5 份, 其中前 4 份用于训练得到推荐模型, 剩余 1 份用作测试.

2) 模拟数据集

本文采用第 3.2 节中为缺陷问题构建的预选集作为模拟数据集. 在预选集中, 一些缺陷问题的真实跨项目相

关问题可能因为没有通过预选被遗漏, 导致部分缺陷问题的预选问题中不含跨项目相关问题. 我们删除这些预选集不含跨项目相关问题的缺陷问题, 仅使用通过预选的真实跨项目相关问题对应的缺陷问题. 本文将这些缺陷问题对应预选集中相关的预选问题对作为正样本, 无关的预选问题对作为负样本. 最终, 模拟数据集中共包含 720 个正样本和 3 395 520 个负样本. 正负样本比为 1∶4716. 本文通过构建模拟数据集, 测试推荐模型在真实 GitHub 平台上的推荐效果.

4.4.2 特征显著性检验

基于调和数据集, 本文对第 3.3 节中选择的文本特征和项目特征进行了显著性分析. 调和数据集中共包含 2 165 个正样本和 36 360 个负样本, 即有 2 165 对相关问题和 36 360 对无关问题. 针对每个特征, 本文分别统计相关问题的特征均值与无关问题的特征均值, 并使用曼-惠特尼 U 检验分析了相关问题与无关问题在该特征上均值差异的显著性, 曼-惠特尼 U 检验结果越小, 说明差异越显著. 当结果小于 0.05 时, 就可以认为相关问题与无关问题在该特征上的均值差异显著. 统计结果如表 6 所示. 结果表明, 相关问题与无关问题在这 17 个特征上均值均存在显著的差异. 比如, 针对 12 个文本特征, 相关问题平均值远大于无关问题平均值. 在项目特征中, 相关问题与无关问题的特征均值也存在显著差异. 例如, 针对项目之间历史相关问题对数量特征, 相关问题的特征均值为 0.225, 无关问题的特征均值为 0.114. 这说明相关问题的项目之间历史相关问题对数量更高.

表 6 特征显著性检验

特征	相关问题平均值	无关问题平均值	曼-惠特尼U检验
标题余弦相似度	0.949	0.587	4.87E–274
正文余弦相似度	0.944	0.619	1.79E–202
缺陷问题标题和跨项目问题正文余弦相似度	0.925	0.592	1.31E–216
缺陷问题正文和跨项目问题标题余弦相似度	0.915	0.637	2.28E–267
标题Jaccard相似度	0.310	0.113	1.31E–91
正文Jaccard相似度	0.201	0.104	4.73E–73
缺陷问题标题和跨项目问题正文Jaccard相似度	0.070	0.037	1.55E–69
缺陷问题正文和跨项目问题标题Jaccard相似度	0.059	0.028	1.34E–70
标题*Dice*相似度	0.390	0.182	1.86E–91
正文*Dice*相似度	0.156	0.089	3.54E–64
缺陷问题标题和跨项目问题正文*Dice*相似度	0.091	0.053	9.69E–41
缺陷问题正文和跨项目问题标题*Dice*相似度	0.074	0.035	6.72E–46
编程语言相似度	0.479	0.424	0.033
项目之间历史相关问题对的数量	0.225	0.114	1.69E–18
重复贡献者占源项目贡献者比例	0.198	0.172	1.95E–10
重复贡献者占目标项目贡献者比例	0.334	0.217	1.32E–29
源项目和目标项目是否归属于同一用户	0.006	0.045	3.66E–05

5 实验评估

5.1 评估指标

针对跨项目相关问题的自动化推荐方法 CPIRecom, 本文分别采用平均倒数排名 (mean reciprocal rank) 和前 k 项查全率 (Recall@k) 两种评估指标评估推荐的效果.

1) 平均倒数排名 (mean reciprocal rank)

平均倒数排名是指真实的跨项目相关问题在推荐列表中排名倒数的平均值, 是推荐算法中常用的评价指标. 在推荐列表中, 真实的跨项目相关问题的排名越靠前, 平均倒数排名越高. 如果存在多个真实相关的跨项目问题, 那么选择排名最靠前的问题进行计算. 本文用 N 表示缺陷问题的数量, $rank(i)$ 表示针对第 i 个缺陷问题的推荐列

表中真实跨项目相关问题的排名. 平均倒数排名的计算公式如下:

$$平均倒数排名=\frac{1}{N}\sum_{i=0}^{N}\frac{1}{rank(i)} \tag{5}$$

2) 前 k 项查全率 (Recall@k)

k 为可配置选项. 前 k 项查全率指在排名前 k 项中真实跨项目相关问题的数量除以全部真实跨项目相关问题的数量[19]. 本文的研究目标是为缺陷问题推荐跨项目相关问题以辅助开发人员解决缺陷问题, 因此希望尽可能找出缺陷问题的跨项目相关问题, 故采用前 k 项查全率评估模型检测出跨项目相关问题的能力. 本文用 N 表示缺陷问题的数量, $related(i)_k$ 表示针对第 i 个缺陷问题的推荐列表前 k 项中真实的跨项目相关问题数量, $related(i)_{all}$ 表示针对第 i 个缺陷问题的所有真实的跨项目相关问题数量. 最后, 计算 N 个缺陷问题前 k 项查全率的平均值. 计算公式如下:

$$前k项查全率=\frac{1}{N}\sum_{i=1}^{N}\frac{related(i)_k}{related(i)_{all}} \tag{6}$$

5.2 研究问题

本文研究以下 3 个问题.

研究问题 1: 本文的推荐方法效果如何?

本文构建了调和数据集和模拟数据集, 并分别在调和数据集和模拟数据集中验证 CPIRecom 方法对跨项目相关问题的推荐效果.

研究问题 2: 基于哪种机器学习分类算法的推荐效果比较好?

本文基于机器学习分类算法来预测并推荐缺陷问题的跨项目相关问题. 我们分别尝试以下 6 种常见的分类算法——支持向量机 (support vector machine)[21]、随机森林 (random forest)[22]、逻辑回归 (logistic regression)、朴素贝叶斯算法 (naive Bayes)、XGBoost 算法和 AdaBoost 算法, 通过对比实验比较不同算法的推荐效果, 将效果最好的分类算法应用于 CPIRecom 方法.

研究问题 3: 不同特征的组合会提升 CPIRecom 方法的推荐效果吗?

本文提出的推荐方法考虑了不同维度的特征, 包括文本特征和项目特征, 其中文本特征分为余弦相似度特征、Jaccard 相似度特征和 *Dice* 相似度特征. 为了评估选择上述特征的必要性, 本文通过比较不同特征组合的推荐效果, 并选择合适的特征组合应用于 CPIRecom 方法.

5.3 评估结果

5.3.1 研究问题 1: CPIRecom 方法的推荐效果

为了评估 CPIRecom 方法的推荐效果, 本文分别构建了调和数据集和模拟数据集, 并分别在调和数据集和模拟数据集中计算 CPIRecom 方法的平均倒数排名和前 k 项查全率.

实验结果如表 7 所示. 在调和测试集中, CPIRecom 方法的平均倒数排名达到 0.985, 前 5 项查全率达到 0.985; 在模拟测试集中, CPIRecom 方法的平均倒数排名达到 0.603, 前 5 项查全率达到 0.715. 可以看出, 在模拟 GitHub 平台的真实情况下, CPIRecom 方法仍然能够达到不错的推荐效果.

研究问题 1: CPIRecom 方法具有好的跨项目相关问题推荐效果.

5.3.2 研究问题 2: 分类算法的选择

CPIRecom 方法使用机器学习分类算法来检测软件缺陷的跨项目相关问题. 本文尝试以下 6 种常见的分类算法——支持向量机 (support vector machine)、随机森林 (random forest)、逻辑回归 (logistic regression)、朴素贝叶斯算法 (naive Bayes)、XGBoost 算法和 AdaBoost 算法. 支持向量机是一种可以用于分类与回归分析数据的有监督学习算法, 其基本原理是寻找特征空间上间隔最大的线性分类器. 同时, 支持向量机提供了一种核方法, 可以将低维的空间映射到高维, 以解决低维空间线性不可分的情况. 基于决策树 (decision tree) 的随机森林使用了树形结

构, 在每个节点中基于某一属性值进行分支, 判断进入哪个子节点, 直至进入叶子节点得到最终的结果. 逻辑回归也是一种用于分类任务的有监督的机器学习算法. 逻辑回归算法通过将线性回归算法使用 Sigmoid 函数映射达到分类的效果. 朴素贝叶斯算法是基于贝叶斯定理和条件独立性假设的分类算法, 其基本原理是使用先验概率计算后验概率. XGBoost 是一种基于梯度提升树的集成学习算法, 通过使用二阶泰勒展开以及正则化来对普通的梯提升树 (GBDT) 进行优化. AdaBoost 是一种自适应增强算法. AdaBoost 的自适应在于前一个基本分类器分错的样本会得到加强, 加权后的全体样本再次被用来训练下一个基本分类器.

表 7　CPIRecom 方法在调和测试集与模拟测试集上的性能

数据集	平均倒数排名	前1项查全率	前2项查全率	前3项查全率	前4项查全率	前5项查全率
调和测试集	0.985	0.912	0.969	0.977	0.984	0.985
模拟测试集	0.603	0.461	0.568	0.632	0.683	0.715

为了选择合适的分类算法, 本文评估 6 种分类算法在调和测试集和模拟测试集中的检测效果, 实验结果如表 8 和表 9 所示. 结果显示, 基于随机森林的推荐方法在调和测试集和模拟测试集上的平均倒数排名均优于其他 5 种机器学习算法. 这表明采用基于随机森林的推荐方法找到的真实跨项目相关问题在推荐列表中的位置更靠前. 因此, 本文使用基于随机森林的推荐方法为缺陷问题推荐跨项目相关问题.

表 8　不同分类算法的比较 (调和测试集)

分类算法	平均倒数排名	前1项查全率	前2项查全率	前3项查全率	前4项查全率	前5项查全率
支持向量机	0.974	0.903	0.968	**0.981**	0.985	0.985
随机森林	**0.985**	**0.912**	0.969	0.977	0.984	0.985
逻辑回归	0.979	0.897	0.968	0.980	**0.986**	**0.988**
朴素贝叶斯	0.967	0.892	0.960	0.977	0.979	0.981
XGBoost	0.981	0.906	0.969	0.977	0.984	0.987
AdaBoost	0.984	0.907	**0.973**	**0.981**	0.984	0.986

表 9　不同分类算法的比较 (模拟测试集)

分类算法	平均倒数排名	前1项查全率	前2项查全率	前3项查全率	前4项查全率	前5项查全率
支持向量机	0.494	0.341	0.457	0.527	0.585	0.621
随机森林	**0.601**	**0.461**	**0.568**	0.632	0.683	0.715
逻辑回归	0.530	0.390	0.490	0.553	0.602	0.641
朴素贝叶斯	0.477	0.325	0.441	0.512	0.571	0.620
XGBoost	0.591	0.423	0.562	**0.656**	**0.704**	**0.756**
AdaBoost	0.570	0.428	0.529	0.597	0.644	0.678

研究问题 2: 基于随机森林推荐方法的平均倒数排名均优于其他 5 种机器学习分类算法.

5.3.3　研究问题 3: 特征组合的效果分析

本文的推荐方法 CPIRecom 分别提取文本特征和项目特征, 其中文本特征包括余弦相似度特征、Jaccard 相似度特征和 *Dice* 相似度特征. 为了评估选择上述特征的必要性, 本文基于随机森林分类算法分析使用不同特征组合的推荐效果.

本文定义去除文本特征的方法: 该方法不考虑文本特征, 只考虑项目特征, 并且在精准推荐环节使用随机森林分类算法. 同时, 本文定义去除项目特征的方法: 该方法不考虑项目特征, 只考虑文本特征, 并且在精准推荐环节使用随机森林分类算法. 然后, 对比去除文本特征的方法、去除项目特征的方法和 CPIRecom 方法在推荐效果上的差异, 如表 10、表 11 所示. 结果表明, CPIRecom 方法在调和测试集和模拟测试集上的平均倒数排名和前 k 项查全率均优于去除文本特征和去除项目特征的方法. 其中, 包含文本特征和项目特征的 CPIRecom 方法在调和测试

集和模拟测试集上的平均倒数排名和前 5 项查全率均明显优于去除文本特征的方法. 这表明使用基于文本相似度的文本特征有助于提高跨项目相关问题的推荐效果.

表 10 不同特征的比较 (调和测试集)

实验组	平均倒数排名	前1项查全率	前2项查全率	前3项查全率	前4项查全率	前5项查全率
去除文本特征	0.495	0.351	0.450	0.519	0.560	0.592
去除项目特征	0.978	0.904	0.968	0.976	**0.985**	**0.985**
CPIRecom	**0.985**	**0.912**	**0.969**	**0.977**	0.984	**0.985**

表 11 不同特征的比较 (模拟测试集)

实验组	平均倒数排名	前1项查全率	前2项查全率	前3项查全率	前4项查全率	前5项查全率
去除文本特征	0.025	0.005	0.010	0.012	0.018	0.023
去除项目特征	0.573	0.422	0.540	0.624	0.668	0.694
CPIRecom	**0.603**	**0.461**	**0.568**	**0.632**	**0.683**	**0.715**

在表 11 中, 去除文本特征方法的平均倒数排名为 0.025, 远低于去除项目特征方法的 0.573 和 CPIRecom 方法的 0.603. 这说明只使用项目特征的针对软件缺陷推荐跨项目相关问题是不够的. 项目特征指的是问题对所属两个项目之间的特征, 包括: 编程语言相似度、历史相关问题对的数量等. 给定项目 A 和项目 B, 项目 A 中的缺陷问题 a1 和项目 B 中的任何问题组成的问题对, 其项目特征都取决于项目 A 和项目 B, 和问题本身无关. 所以, 无法区分项目 B 中属于缺陷问题 a1 的相关问题或无关问题. 因此, 只考虑项目特征为软件缺陷推荐跨项目相关问题是不够的. 与项目特征不同, 文本特征考虑了两个问题的文本相似度, 分析了两个问题自身的特点. 因此基于文本特征的推荐效果优于基于项目特征的推荐效果.

CPIRecom 方法的平均倒数排名为 0.603, 说明 CPIRecom 方法在模拟数据集中的推荐效果还有待提高. 这是因为有些相关问题对之间的文本相似度不高, 使得基于 BERT 预训练模型的 CPIRecom 方法无法得到更好的效果, 例如相关问题对<tensorflow/tensorflow#35115, bazelbuild/bazel#7687>. 在项目 tensorflow/tensorflow 中, 编号#35115 问题 (https://github.com/tensorflow/tensorflow/issues/35115) 的标题为"Failed to build TF2.0 with TensorRT: undefined symbol: _ZN15stream_executor14 StreamExecutor18EnablePeerAccessTo EPS0_". 项目 bazelbuild/bazel 中编号#7687 问题 (https://github.com/bazelbuild/bazel/issues/7687) 的标题为"incompatible_ do_not_split_ linking_cmdline: Order of C++ link flags affected". 由于项目 tensorflow/tensorflow 使用了 bazel 模块的 0.27.1 版本, 导致软件缺陷#35115 遇到了与项目 bazelbuild/bazel 中编号#7687 相同的问题. 最终, 通过升级到 bazel 模块的 2.0.0 版本解决了编号#35115 的缺陷问题. 然而, 问题#35115 和问题#7687 的标题信息、描述信息之间没有明显相关的文本内容. 因此, 针对这种文本相关性不高的问题对, 使用基于 BERT 预训练模型的 CPIRecom 方法无法得到更好的效果.

然后, 针对文本特征中的 3 种相似度特征, 本文分别去除余弦相似度特征、Jaccard 相似度特征和 *Dice* 相似度特征, 并对比不同特征组合在推荐效果上的差异, 如表 12、表 13 所示. 结果表示, CPIRecom 方法在调和测试集上的平均倒数排名优于去除余弦相似度特征、去除 Jaccard 相似度特征和去除 *Dice* 相似度特征的方法. 在模拟测试集上, CPIRecom 方法的平均倒数排名和前 5 项查全率两项指标均优于去除余弦相似度特征、去除 Jaccard 相似度特征和去除 *Dice* 相似度特征的方法.

表 12 不同文本特征的比较 (调和测试集)

实验组	平均倒数排名	前1项查全率	前2项查全率	前3项查全率	前4项查全率	前5项查全率
去除余弦相似度特征	0.753	0.622	0.732	0.786	0.812	0.824
去除Jaccard相似度特征	0.978	0.905	0.963	**0.978**	0.984	**0.987**
去除*Dice*相似度特征	0.982	0.907	0.968	0.975	0.982	0.983
CPIRecom	**0.985**	**0.912**	**0.969**	0.977	0.984	0.985

表 13 不同文本特征的比较 (模拟测试集)

实验组	平均倒数排名	前1项查全率	前2项查全率	前3项查全率	前4项查全率	前5项查全率
去除余弦相似度特征	0.326	0.244	0.308	0.338	0.362	0.375
去除Jaccard相似度特征	0.561	0.415	0.525	0.595	0.642	0.677
去除*Dice*相似度特征	0.582	0.437	0.549	0.614	0.660	0.702
CPIRecom	**0.603**	**0.461**	**0.568**	**0.632**	**0.683**	**0.715**

综上, 相对于去除余弦相似度特征、Jaccard 相似度特征、*Dice* 相似度特征或项目特征, CPIRecom 方法在调和测试集和模拟测试集上均有更好的推荐效果, 这充分证明了本文选择上述特征的必要性.

研究问题 3: 组合文本特征 (余弦相似度特征、Jaccard 相似度特征和 *Dice* 相似度特征) 和项目特征可以提高推荐效果.

6 有效性分析

6.1 结构有效性威胁

结构有效性关注实验结果对于实验背后的概念或理论的普适性. 首先, 我们从 GitHub 平台上 2020 年 1 月 1 日–2020 年 12 月 31 日的数据中一共收集了 32 912 156 个问题. 大规模的数据具有一定代表性. 在未来的工作中, 我们考虑收集更多年份的问题, 进一步验证我们的方法. 然后, Li 等人的研究[23]发现一些过时问题的修改方案被覆盖或回退, 失去参考价值. 在未来的工作中, 参考文献 [23] 的过时问题识别方法, 预先删除候选问题集中的过时问题, 避免向缺陷问题推荐信息失效的过时问题. 最后, 本文构建的数据集与 GitHub 平台真实场景下的数据分布仍然存在不同. 在未来的工作中, 我们将分析真实数据集来评估本文的方法.

6.2 内部有效性威胁

内部有效性是研究在处置和结果之间建立可信的因果关系的程度. 首先, 本文在构建预选集方案时仅使用项目之间历史相关问题对数量筛选项目, 一些包含真实跨项目相关问题的项目可能会因为不符合项目之间历史相关问题对数量的条件没有通过预选, 导致这些真实的跨项目相关问题被过滤掉. 在未来的工作中, 本文将考虑更多的项目预选条件, 尽量让包含真实跨项目相关问题的项目通过预选. 例如, 考虑使用项目之间的代码依赖预选项目. 其次, 本文的推荐方法考虑了文本特征和项目特征. 在未来的工作中, 我们尝试提取更多特征并分析不同特征对跨项目相关问题推荐的影响.

6.3 外部有效性威胁

本文选取了 GitHub 平台研究跨项目相关问题的推荐工作, 但不确定该方法在其他开源平台中的效果. 在今后的工作中, 我们将考虑调整实验中的数据来源, 进一步验证 CPIRecom 方法. 同时, 由于本文的研究工作是为软件缺陷推荐来自项目外部的相关问题, 因此在预选问题时删除了不含跨项目相关问题的情况, 即过滤了项目内部问题. 在未来的工作中, 我们在使用本文方法为软件缺陷推荐相关问题时, 考虑将相关问题的项目来源从项目外部扩展到项目内部和项目外部. 同时, 采用现有文献[13,14]的测试集, 分析 CPIRecom 方法对项目内部相关问题的推荐效果.

7 总 结

本文提出了一种跨项目相关问题的自动推荐方法 CPIRecom. 为了构建预选集, 本文采用项目之间历史相关问题对的数量和问题发布时间间隔筛选问题. 其次, 为了精准推荐, 本文采用 BERT 预训练模型提取文本特征, 分析项目特征. 然后使用随机森林算法计算预选问题与缺陷问题的相关概率, 最终根据相关概率排名得到推荐列表. 实验表明: 在调和测试集中, CPIRecom 方法的平均倒数排名达到 0.985, 前 5 项查全率达到 0.985; 在模拟测试集中, CPIRecom 方法的平均倒数排名达到 0.603, 前 5 项查全率达到 0.715. 与现有方法 DupFinder、PrFinder 相比,

CPIRecom 方法在调和测试集上的平均倒数排名分别提升了 101.43% 和 83.09%, 前 5 项查全率分别提升了 77.48% 和 54.15%; CPIRecom 方法在模拟测试集上的平均倒数排名分别提升了 224.19% 和 197.04%, 前 5 项查全率分别提升了 243.75% 和 223.53%.

References:

[1] Dong RZ, Li BX, Wang LL, Li HW, Chen HL, Tan J. Review of research on software ecosystems. Chinese Journal of Computers, 2020, 43(2): 250–271 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.00250]

[2] He XX, Zhang YQ, Liu QX. Survey of software supply chain security. Journal of Cyber Security, 2020, 5(1): 57–73 (in Chinese with English abstract). [doi: 10.19363/J.cnki.cn10-1380/tn.2020.01.06]

[3] Dabbish L, Stuart C, Tsay J, Herbsleb J. Social coding in GitHub: Transparency and collaboration in an open software repository. In: Proc. of the 2012 ACM Conf. on Computer Supported Cooperative Work. Seattle: ACM, 2012. 1277–1286. [doi: 10.1145/2145204.2145396]

[4] Lima A, Rossi L, Musolesi M. Coding together at scale: GitHub as a collaborative social network. Proc. of the 2014 Int'l AAAI Conf. on Web and Social Media, 2014, 8(1): 295–304. [doi: 10.1609/icwsm.v8i1.14552]

[5] Yang B, Yu Q, Zhang W, Wu J, Liu C. Influence factors correlation analysis in GitHub open source software development process. Ruan Jian Xue Bao/Journal of Software, 2017, 28(6): 1330−1342 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5222.htm [doi: 10.13328/j.cnki.jos.005222]

[6] Bissyandé TF, Lo D, Jiang LX, Reveillere L, Klein J, Le Traon Y. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In: Proc. of the 24th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Pasadena: IEEE, 2013. 188–197. [doi: 10.1109/ISSRE.2013.6698918]

[7] Bertram D, Voida A, Greenberg S, Walker R. Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In: Proc. of the 2010 ACM Conf. on Computer Supported Cooperative Work. Savannah: ACM, 2010. 291–300. [doi: 10.1145/1718918.1718972]

[8] Zhang Y, Yu Y, Wang HM, Vasilescu B, Filkov V. Within-ecosystem issue linking: A large-scale study of rails. In: Proc. of the 7th Int'l Workshop on Software Mining. Montpellier: ACM, 2018. 12–19. [doi: 10.1145/3242887.3242891]

[9] Li LS, Ren ZL, Li XC, Zou WQ, Jiang H. How are issue units linked? Empirical study on the linking behavior in GitHub. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. Nara: IEEE, 2018. 386–395. [doi: 10.1109/APSEC.2018.00053]

[10] Zhang Y, Wu YW, Wang T, Wang HM. iLinker: A novel approach for issue knowledge acquisition in GitHub projects. World Wide Web, 2020, 23(3): 1589–1619. [doi: 10.1007/s11280-019-00770-1]

[11] Ma WWY, Chen L, Zhang XY, Zhou YM, Xu BW. How do developers fix cross-project correlated bugs? A case study on the GitHub scientific Python ecosystem. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering. Buenos Aires: ACM, 2017. 381–392. [doi: 10.1109/ICSE.2017.42]

[12] Alipour A, Hindle A, Stroulia E. A contextual approach towards more accurate duplicate bug report detection. In: Proc. of the 10th Working Conf. on Mining Software Repositories (MSR). San Francisco: IEEE, 2013. 183–192. [doi: 10.1109/MSR.2013.6624026]

[13] Thung F, Kochhar PS, Lo D. DupFinder: Integrated tool support for duplicate bug report detection. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 871–874. [doi: 10.1145/2642937.2648627]

[14] Wang QY, Xu BW, Xia X, Wang T, Li SP. Duplicate pull request detection: When time matters. In: Proc. of the 11th Asia-Pacific Symp. on Internetware. Fukuoka: ACM, 2019. 8. [doi: 10.1145/3361242.3361254]

[15] Yu Y, Li ZX, Yin G, Wang T, Wang HM. A dataset of duplicate pull-requests in GitHub. In: Proc. of the 15th Int'l Conf. on Mining Software Repositories. Gothenburg: ACM, 2018. 22–25. [doi: 10.1145/3196398.3196455]

[16] Kallis R, Di Sorbo A, Canfora G, Panichella S. Predicting issue types on GitHub. Science of Computer Programming, 2021, 205: 102598. [doi: 10.1016/j.scico.2020.102598]

[17] Bian GQ, Zhang WJ, Shao BL, Gong PJ. Research of cloud resource allocation based on Pareto optimality. Computer Engineering and Applications, 2014, 50(19): 70–73 (in Chinese with English abstract). [doi: 10.3778/j.issn.1002-8331.1211-0078]

[18] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol. 1 (Long and Short Papers). Minneapolis: Association for Computational Linguistics, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[19] Zhang JS, Chen SJ, Wang XK. Sustainable treatment of antibiotic wastewater using combined process of microelectrolysis and struvite crystallization. Water, Air & Soil Pollution, 2015, 226(9): 315. [doi: 10.1007/s11270-015-2581-5]

[20] Liu BC, Zhang L, Jiang J, Wang L. A method for identifying references between projects in GitHub. Science of Computer Programming, 2022, 222: 102858. [doi: 10.1016/j.scico.2022.102858]

[21] Noble WS. What is a support vector machine? Nature Biotechnology, 2006, 24(12): 1565–1567. [doi: 10.1038/nbt1206-1565]

[22] Pal M. Random forest classifier for remote sensing classification. Int'l Journal of Remote Sensing, 2005, 26(1): 217–222. [doi: 10.1080/01431160412331269698]

[23] Li ZX, Zhong H. An empirical study on obsolete issue reports. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 1317–1321. [doi: 10.1109/ASE51524.2021.9678543]

附中文参考文献:

[1] 董瑞志, 李必信, 王璐璐, 李宏伟, 陈海雷, Tan J. 软件生态系统研究综述. 计算机学报, 2020, 43(2): 250–271. [doi: 10.11897/SP.J.1016.2020.00250]

[2] 何熙巽, 张玉清, 刘奇旭. 软件供应链安全综述. 信息安全学报, 2020, 5(1): 57–73. [doi: 10.19363/J.cnki.cn10-1380/tn.2020.01.06]

[5] 杨波, 于茜, 张伟, 吴际, 刘超. GitHub开源软件开发过程中影响因素的相关性分析. 软件学报, 2017, 28(6): 1330–1342. http://www.jos.org.cn/1000-9825/5222.htm [doi: 10.13328/j.cnki.jos.005222]

[17] 边根庆, 张文敬, 邵必林, 龚培娇. 基于帕累托最优的云资源调度研究. 计算机工程与应用, 2014, 50(19): 70–73. [doi: 10.3778/j.issn.1002-8331.1211-0078]

刘宝川(1991－), 男, 博士生, 主要研究领域为开源软件, 软件库挖掘, 经验软件工程.

刘桢炜(2000－), 男, 硕士生, 主要研究领域为机器学习, 数据挖掘, 自然语言处理.

张莉(1968－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件建模与分析, 需求工程, 实证软件工程, 软件体系结构.

蒋竞(1985－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为智能软件工程, 经验软件工程, 开源软件, 软件库挖掘.