软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(10):4558−4589 [doi: 10.13328/j.cnki.jos.007297] [CSTR: 32375.14.jos.007297]

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

SZZ 误标变更对移动 APP 即时缺陷预测性能和解释的影响*

李志强[1], 马 睿[1], 张洪宇[2], 荆晓远[3,4], 任 杰[1], 刘金会[5]

[1](陕西师范大学 计算机科学学院, 陕西 西安 710119)

[2](重庆大学 大数据与软件学院, 重庆 401331)

[3](武汉大学 计算机学院, 湖北 武汉 430072)

[4](广东石油化工学院 计算机学院, 广东 茂名 525011)

[5](西北工业大学 网络空间安全学院, 陕西 西安 710072)

通信作者: 张洪宇, E-mail: hyzhang@cqu.edu.cn

摘 要: 近年来, SZZ 作为一种识别引入缺陷的变更算法, 被广泛应用于即时软件缺陷预测技术中. 先前的研究表明, SZZ 算法在对数据进行标注时会存在误标问题, 这将影响数据集的质量, 进而影响预测模型的性能. 因此, 研究人员对 SZZ 算法进行了改进, 并提出多个 SZZ 变体. 然而, 目前尚未有文献研究数据标注质量对移动 APP 即时缺陷预测性能和解释的影响. 为探究 SZZ 错误标注的变更对移动 APP 即时软件缺陷预测模型的影响, 对 4 种 SZZ 算法进行广泛而深入的实证研究. 首先, 选取 GitHub 库中 17 个大型移动 APP 项目, 借助 PyDriller 工具抽取软件度量元. 其次, 采用 B-SZZ (原始 SZZ 版本)、AG-SZZ、MA-SZZ 和 RA-SZZ 这 4 种算法标注数据. 然后, 根据时间序列划分数据, 利用随机森林、朴素贝叶斯和逻辑回归分类器分别建立即时缺陷预测模型. 最后, 使用 *AUC*、*MCC*、*G-mean* 传统指标和 *F-measure*@20%、IFA 工作量感知指标评估模型性能, 并使用 SKESD 和 SHAP 算法对结果进行统计显著性检验与可解释性分析. 通过对比 4 种 SZZ 算法的标注性能, 研究发现: (1) 数据的标注质量符合 SZZ 变体之间的递进关系; (2) B-SZZ、AG-SZZ 和 MA-SZZ 错误标注的变更会造成 *AUC*、*MCC* 得分不同程度的下降, 但不会造成 *G-mean* 得分下降; (3) B-SZZ 会造成 *F-measure*@20% 得分下降, 而在代码审查时, B-SZZ、AG-SZZ 和 MA-SZZ 不会导致审查工作量的增加; (4) 在模型解释方面, 不同 SZZ 算法会影响预测过程中贡献程度排名前 3 的度量元, 并且 la 度量元对预测结果有重要影响.

关键词: 即时软件缺陷预测; 移动 APP; SZZ 算法; 挖掘软件存储库; 可解释性; 工作量感知; 实证软件工程

中图法分类号: TP311

中文引用格式: 李志强, 马睿, 张洪宇, 荆晓远, 任杰, 刘金会. SZZ误标变更对移动APP即时缺陷预测性能和解释的影响. 软件学报, 2025, 36(10): 4558–4589. http://www.jos.org.cn/1000-9825/7297.htm

英文引用格式: Li ZQ, Ma R, Zhang HY, Jing XY, Ren J, Liu JH. Impact of Mislabeled Changes by SZZ on Performance and Interpretation of Just-in-time Defect Prediction for Mobile APP. Ruan Jian Xue Bao/Journal of Software, 2025, 36(10): 4558–4589 (in Chinese). http://www.jos.org.cn/1000-9825/7297.htm

Impact of Mislabeled Changes by SZZ on Performance and Interpretation of Just-in-time Defect Prediction for Mobile APP

LI Zhi-Qiang[1], MA Rui[1], ZHANG Hong-Yu[2], JING Xiao-Yuan[3,4], REN Jie[1], LIU Jin-Hui[5]

[1](School of Computer Science, Shaanxi Normal University, Xi'an 710119, China)

[2](School of Big Data and Software Engineering, Chongqing University, Chongqing 401331, China)

* 基金项目: 国家自然科学基金 (61902228, 62176069, U23A20302); 陕西省自然科学基础研究计划 (2024JC-YBMS-497); 陕西省重点研发计划 (2023-YBGY-265)
收稿时间: 2023-09-28; 修改时间: 2023-12-26, 2024-04-07, 2024-06-13; 采用时间: 2024-10-02; jos 在线出版时间: 2025-02-19
CNKI 网络首发时间: 2025-02-19

[3](School of Computer Science, Wuhan University, Wuhan 430072, China)
[4](School of Computer, Guangdong University of Petrochemical Technology, Maoming 525011, China)
[5](School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: In recent years, as an algorithm for identifying bug-introducing changes, SZZ has been widely employed in just-in-time software defect prediction. Previous studies show that the SZZ algorithm may mislabel data during data annotation, which could influence the dataset quality and consequently the performance of the defect prediction model. Therefore, researchers have made improvements to the SZZ algorithm and proposed multiple variants of SZZ. However, there is no empirical study to explore the effect of data annotation quality by SZZ on the performance and interpretability of just-in-time defect prediction for mobile APP. To investigate the influence of mislabeled changes by SZZ on just-in-time defect prediction for mobile APP, this study conducts an extensive and in-depth empirical comparison of four SZZ algorithms. Firstly, 17 large-scale mobile APP projects are selected from the GitHub repository, and software metrics are extracted by adopting the PyDriller tool. Then, B-SZZ (original SZZ), AG-SZZ, MA-SZZ, and RA-SZZ are employed for data annotation. Then, the just-in-time defect prediction models are built with random forest, naive Bayes, and logistic regression classifiers based on the time-series data partitioning. Finally, the performance of the models is evaluated by traditional measures of *AUC*, *MCC*, and *G-mean*, and effort-aware measures of *F-measure*@20% and IFA, and a statistical significance test and interpretability analysis are conducted on the results by employing SKESD and SHAP respectively. By comparing the annotation performance of the four SZZ algorithms, the results are as follows. (1) The data annotation quality conforms to the progressive relationship among SZZ variants. (2) The mislabeled changes by B-SZZ, AG-SZZ, and MA-SZZ can cause performance reduction of *AUC* and *MCC* of different levels, but cannot lead to performance reduction of *G-mean*. (3) B-SZZ is likely to cause a performance reduction of *F-measure*@20%, while B-SZZ, AG-SZZ, and MA-SZZ are unlikely to increase effort during code inspection. (4) In terms of model interpretation, different SZZ algorithms will influence the three metrics with the largest contribution during the prediction, and the la metric has a significant influence on the prediction results.

Key words: just-in-time software defect prediction; mobile APP; SZZ method; mining software repository; interpretability; effort aware; empirical software engineering

随着互联网的快速发展, 智能手机已成为人们生活中不可或缺的必备工具. 截至目前, 全球移动用户数量已达到 30 亿 (https://newzoo.com/resources/trend-reports/newzoo-global-mobile-market-report-2019-light-version), 这极大地促进了移动应用市场的繁荣发展. 然而, 随着用户需求的不断提高, 应用程序的各种功能需要不断更新. 例如, 在移动 APP 的版本迭代过程中, 由于一些不可控因素, 新版本应用程序发布后可能会引入缺陷, 从而影响软件质量. 因此, 在发布新版本之前及时发现缺陷并反馈给相关开发人员进行修复已成为一项迫切需要解决的问题[1−4].

为了降低软件缺陷所带来的成本并提升软件质量, 研究人员提出了基于变更级的软件缺陷预测[5]. 近年来, 该技术越来越受到关注[6−8]. Kamei 等人[9]将该技术称为即时缺陷预测 (just-in-time defect prediction). 相较于预测文件或模块的缺陷倾向性[10,11], 即时缺陷预测可以帮助开发人员检查更少的风险代码, 在代码变更提交时即可进行预测, 以判定是否为缺陷引入的变更, 从而更容易进行缺陷定位, 便于开发人员及时地进行代码审查, 并能及早地在代码提交前发现缺陷[9]. 由于即时缺陷预测技术具有细粒度、即时性和易追溯的特点, 尤其适用于频繁进行更新且涉及大量的代码提交的软件产品, 例如移动 APP. 因此, 本文将重点研究面向移动 APP 的即时软件缺陷预测. 主要原因如下: (1) 移动 APP 的发布周期通常较短, 版本迭代速度较快, 这对于及时发现和修复缺陷至关重要, 以确保新版本的稳定性与质量; (2) 用户可以随时随地下载与使用移动 APP 应用, 意味着缺陷在任何时间都有可能发生, 在缺陷出现后若能尽快提供反馈, 这将有助于开发小组及时修复缺陷; (3) 用户体验至关重要, 及时发现和修复缺陷可以避免用户在使用 APP 应用时遇到问题, 进而提升用户满意度.

在即时软件缺陷预测技术中, 从项目的代码变更历史中准确定位引入缺陷的变更是其中最关键的环节之一. 软件开发过程通常包含了大量的变更历史, 手动筛选引入缺陷的变更非常耗时且繁琐. 因此, 研究人员提出了 SZZ 算法, 旨在自动识别引入缺陷的变更[12−15]. SZZ 算法由 Sliwerski、Zimmermann 和 Zeller 这 3 位研究人员提出[12], 该算法首先通过缺陷关键词来定位引入缺陷的变更, 例如 bug、fix、crash、fault 等. 具体而言, SZZ 首先根据代码变更日志中包含这些关键词的变更来定位缺陷, 并将这些变更中所修改的代码行标注为缺陷行. 其次, SZZ 对这些

缺陷行进行溯源, 以识别出第 1 次引入这些缺陷行的变更. 这些变更被认为是引入缺陷的变更, 在获得引入缺陷的变更集合后, SZZ 对其进行过滤, 最终得到剩余的变更, 即为引入缺陷的变更.

目前, 许多即时软件缺陷预测研究都采用 SZZ 算法来识别引入缺陷的变更[6,8,9], 并将其用于数据标注的工作. 然而, 先前的研究发现 SZZ 算法将注释行或空白行的代码更改错误地视为有缺陷的变更, 从而影响了数据的缺陷标注质量[13−15]. 这些错误标注的变更包括假阳性和假阴性, 假阳性指该变更没有引入任何缺陷, 但错误地被认为引入了缺陷; 而假阴性则相反, 指该变更引入了缺陷, 但错误地被认为没有引入缺陷. 然而, 现有的即时软件缺陷预测研究大都仍采用原始的 SZZ 算法[6,8,16], 这可能会导致数据质量下降的问题, 进而影响缺陷预测模型的性能与解释.

为了解决原始 SZZ 算法中标注不精确的问题, 研究人员提出一系列的改进算法[13−15], 以提高数据的标注质量. 为了比较不同 SZZ 算法的标注性能, Fan 等人[5]面向开源框架应用进行了实证研究并对 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 进行了比较. 他们发现, B-SZZ、MA-SZZ 算法不太可能导致即时缺陷预测模型的性能显著降低, 而 AG-SZZ 则会导致模型性能显著下降. 然而, (1) 该研究采用 out-of-sample bootstrap 交叉验证算法进行随机数据划分, 其可能会使用当前的变更来预测之前的变更 (比如用 2023 年的变更预测 2021 年的变更), 这与实际软件开发过程不符. (2) 为解决缺陷数据的类不平衡问题, 该研究使用随机欠采样技术对训练数据进行类重平衡, 这可能会导致重要数据信息和关键变更的丢失[16]. (3) 该研究使用特征重要性算法仅对缺陷预测模型进行了全局解释, 而忽略了对模型进行局部解释. 全局解释侧重于模型对所有代码变更上的决策和预测结果进行解释, 局部解释侧重于对模型在单个变更上的具体预测结果进行解释. 事实上, 特征重要性只能解释软件度量元对模型的整体影响大小, 不能解释是对模型产生正类 (有缺陷) 还是负类 (无缺陷) 影响. 由于全局解释只能解释度量元对模型整体的影响, 不能解释模型在某个变更上的预测结果, 因此无法反映出在单个变更上, 各个度量元对模型的贡献大小. 总的来说, 随机进行数据划分可能导致使用当前的变更来预测之前的变更, 而下采样算法可能会降低预测模型的性能, 同时忽略个体预测结果的局部解释将不能很好地理解模型在单个变更上的决策, 以上这些因素会不利于即时缺陷预测模型在实践中的快速发展与应用.

据我们所知, 迄今为止, 还没有一项工作实证研究数据标注算法对移动 APP 即时缺陷预测模型的性能与解释的影响. 为了弥补这一空白, 并结合现有研究的不足之处, 本文大规模地评估了不同 SZZ 算法错误标注的变更对移动 APP 即时缺陷预测模型的影响. 具体地, 本文从 GitHub 上选择了 17 个大型移动 APP 项目, 其中 12 个项目参考先前的论文[17], 并抽取 Kamei 等人[9]提出的 14 个变更度量, 在此基础上使用 4 种 SZZ 算法进行数据标注; 采用时间序列划分方式进行数据划分, 以更加符合实际情况; 在类不平衡处理中, 使用 SMOTE 采样算法, 以避免重要数据信息与关键变更的丢失; 利用随机森林、朴素贝叶斯和逻辑回归分类器分别构建即时软件缺陷预测模型, 以评估各 SZZ 算法的标注质量对预测模型性能与解释的影响, 并使用 SKESD 和 SHAP 算法对预测结果进行统计显著性检验与可解释性分析. 本文发现, SZZ 错误标注的变更会对移动 APP 即时软件缺陷预测模型的性能与解释产生不同程度的影响.

本文的主要贡献可总结如下.

(1) 在 17 个项目上大规模实证比较了 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 算法的数据标注质量对移动 APP 即时缺陷预测性能与解释的影响.

(2) 采用时间序列交叉验证方法, 更好地捕捉数据中的时序特征, 提高实验结果的真实性, 并且可以避免使用当前的数据预测过去的数据, 以更加符合实际软件开发过程.

(3) 使用 SMOTE 采样算法可以有效地重平衡训练数据, 保留了原始数据的特征, 增加了多样性, 有助于模型更好地学习和泛化, 同时避免丢失重要数据信息和关键变更.

(4) 采用 SKESD 和 SHAP 算法对结果进行排序比较与解释分析, 发现 SZZ 错误标注的变更会影响移动 APP 即时缺陷预测的性能与解释, 特别是一些最为关键的度量元 (例如 la 和 nuc).

(5) 构建了移动 APP 即时缺陷预测数据集, 并公开本文的源程序以及详细的实验结果, 便于其他研究者进行后续研究 (https://github.com/Owner-MR/JIT-DP-for-mobile-APP.git).

本文第 1 节介绍研究背景及相关工作. 第 2 节介绍数据集的构建工作. 第 3 节描述本文实证研究的详细步骤. 第 4 节结合本文研究的问题和实验结果进行深入分析与总结. 第 5 节对实验进行补充并阐述实验的有效性. 第 6 节对全文进行总结, 并对未来的研究工作进行展望.

1 背景及相关工作

1.1 数据集质量与 SZZ 算法

近年来, 数据集的质量问题受到越来越多研究人员的关注. 一些研究表明, 数据集质量问题会对缺陷预测模型性能造成影响. Brid 等人[18]发现, 在软件开发过程中, 开发人员可能不会对修复缺陷的提交进行详细说明, 导致缺陷丢失或者在导入存储库后与实际缺陷描述不符. Bachmann 等人[19]对文件层次的缺陷预测模型涉及的缺陷差异问题进行了研究, 结果表明缺陷差异会对模型性能造成严重影响. 以上研究的数据来源于 Bugzilla 的 ITS 系统[18,19], 而本文的研究数据来源于 Git 版本控制系统.

为了探究数据噪音对文件层次以及变更层次的缺陷预测模型的影响, Kim 等人[20]随机地将假阳性 (标注为有缺陷但实际无缺陷) 和假阴性 (标注为无缺陷但实际有缺陷) 样本加入数据集, 结果发现数据集的错误标注样本数量占比在 25%–35% 时会严重影响缺陷预测模型的性能. Antoniol 等人[21]和 Kochhar 等人[22]发现部分问题报告被标注成了有缺陷, 但实际上并没有缺陷, 出现误报的情况. Herzig 等人[23]对 7 000 多个缺陷报告进行了人工检查, 他们发现缺陷报告并不会导致缺陷修复报告被误标, 但存在大部分的缺陷报告被误标的情况. 在此基础上, Tantithamthavorn 等人[24]深入研究被错误标注的缺陷报告对文件层缺陷预测模型的影响, 他们发现错误标注的缺陷报告只影响模型的召回率, 对模型的准确率无明显影响. 在实际开发过程中, 开发人员提交的一次代码变更可能涉及多个方面 (例如缺陷修复和代码重构), Herzig 等人[23]将这种变更称为复杂的代码变更[25]. 他们发现复杂的代码变更对代码的缺陷标注工作影响并不大, 只有少部分无缺陷的文件被错误的标注为有缺陷. 因此, 复杂的代码变更对缺陷预测模型不会造成显著影响. 此外, 他们进一步的研究发现, 复杂的代码变更会影响源文件 16.6% 的相关缺陷数量, 产生了数据噪音, 进而对预测文件的缺陷数量模型造成较大影响. 与此不同, 本文重点研究不同 SZZ 算法的数据标注过程中所产生的噪音对移动 APP 即时缺陷预测模型的影响, 这些噪音是由 SZZ 算法本身所造成的.

1.1.1 原始 SZZ (basic SZZ) 算法

为自动识别代码库中含有缺陷的变更, Sliwerski 等人[12]提出了 SZZ 算法, 称为原始 SZZ (B-SZZ). 它是一种基于版本控制历史的算法, 通过分析源代码的变更历史和缺陷报告之间的关联性, 来定位可能引入缺陷的代码变更. 图 1 展示了 B-SZZ 识别某款移动 APP 项目中一个缺陷的示例, 包括 4 个步骤.

● 从代码提交记录中检索修复该缺陷的提交报告. 此前许多项目使用问题跟踪系统 (issue tracking system, ITS) 来存储缺陷报告, 例如 JIRA, 每个缺陷对应一个唯一标识符, 其表示为 (项目名称-缺陷标识符). 一般情况下, 开发人员修复缺陷以后, 会将缺陷标识符记录在更改的提交信息中. 在 Git 版本控制系统中, B-SZZ 首先在检查变更日志中是否包含缺陷关键词, 如 Step 1 中, 变更信息包含缺陷关键词: “Fix”, 表示该次变更修复了缺陷.

● 识别有缺陷的代码行. B-SZZ 利用版本控制系统 (version control system, VCS) 中的 diff 命令来识别缺陷修复所更改的代码行. 在 Step 2 中, B-SZZ 算法识别出的缺陷修复的代码涉及 AudioPlayerFragment. java 文件中的一行 if 条件语句, 由于缺少条件导致了空指针异常, 因此 B-SZZ 算法认为该代码行存在缺陷.

● 识别引入缺陷的变更. B-SZZ 算法对含有缺陷的代码行进行溯源, 识别出这些代码被首次引入的变更记录, 而有缺陷的代码部分可能包含很多行, 识别出的提交记录也可能包含很多个. Sliwerski 等人[12]使用 VCS 的内置注释命令 (例如 git blame) 来对代码历史进行溯源. 如 Step 3 所示, B-SZZ 识别出变更 292c9bf15 引入了缺陷.

● 对识别出引入缺陷的变更进行过滤. B-SZZ 对引入了缺陷的变更进行筛选, 例如, 正确引入缺陷的变更应该发生在缺陷修复之前, 在缺陷修复变更 05606507b 的提交日期之后所识别的引入缺陷的变更不符合现实规律, 因此被去除. 变更 292c9bf15 的提交日期在缺陷修复变更 05606507b 之前, 并且引入了 05606507b 所修复的缺陷代码, 所以 B-SZZ 将其识别为引入了缺陷的变更.

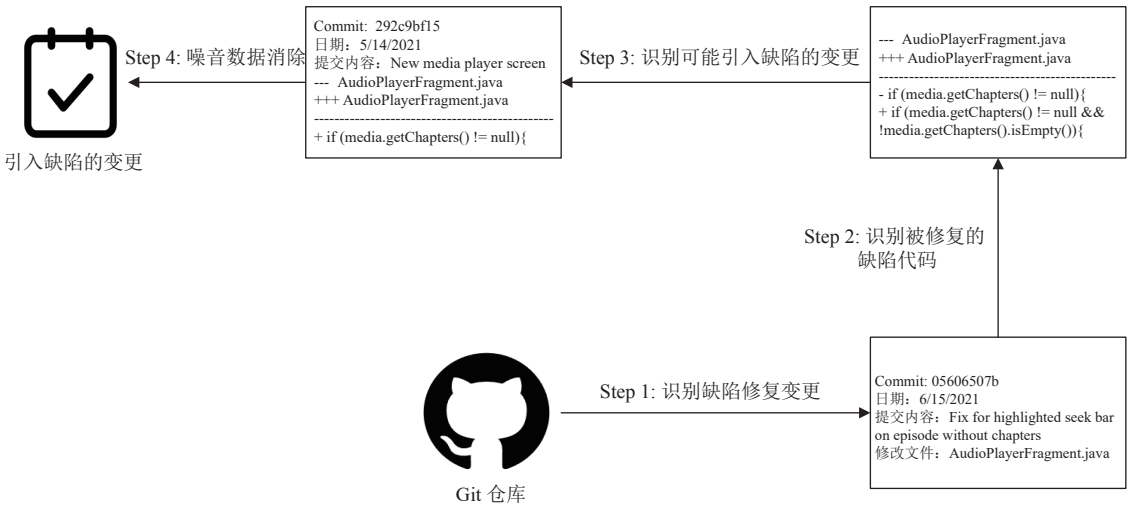


图 1 原始 SZZ 算法识别缺陷的一次示例

1.1.2 基于注释图的 SZZ 算法 (annotation graph SZZ, AG-SZZ)

Kim 等人[13]发现使用 B-SZZ 算法进行数据标注会产生大量噪音, 有相当多的变更被错误识别为引入了缺陷, 并且 B-SZZ 将注释行、空白行和格式修改 (如代码缩进) 等非语义行的修改错误识别为有缺陷. 事实上这些非语义代码行并不会引入缺陷, 然而 B-SZZ 算法未考虑与此. 特别是在 B-SZZ 算法的 Step 3 中, B-SZZ 错误地将涉及格式修改的代码识别为有缺陷. 因此, 格式修改的代码行可能会影响 B-SZZ 结果的正确性. 为了解决上述问题, Kim 等人提出了 AG-SZZ. 在 B-SZZ 算法 Step 3 中使用注释图[26]来溯源代码, 相比注释命令, 注释图可以提供更全面的代码修改信息 (例如增加和删除代码行).

1.1.3 基于元变化感知的 SZZ 算法 (meta-change aware SZZ, MA-SZZ)

Da Costa 等人[14]发现, 如果 B-SZZ 算法 Step 3 中涉及元更改 (分支更改、合并更改以及文件属性更改), 则会将代码的元更改识别为引入缺陷, 然而元更改并不属于代码缺陷, 不会对项目造成影响, 而 AG-SZZ 仍然会将其识别为引入了缺陷的变更. 由于 AG-SZZ 存在上述问题, Da Costa 等人[14]在 AG-SZZ 的基础上进行了改进, 提出基于元变化感知的 SZZ 算法, 即 MA-SZZ. 通过改进 AG-SZZ 中的注释图, 从而避免将元更改识别为代码缺陷.

1.1.4 基于代码重构感知的 SZZ 算法 (refactoring aware SZZ, RA-SZZ)

Neto 等人[15]观察到, 先前的 SZZ(B-SZZ、AG-SZZ 和 MA-SZZ) 算法会将重构代码行 (例如对函数名的修改) 识别为缺陷. 重构行会影响 SZZ 算法的 Step 2 和 Step 3, 其并不属于缺陷范畴, 所以在 Step 2 中应过滤掉重构行, 在 Step 3 中溯源缺陷代码时, 重构行会被作为缺陷进行溯源, 从而无法识别出正确的引入缺陷的更改. 为应对上述挑战, Neto 等人[15]提出了基于代码重构感知的 SZZ, 他们在 MA-SZZ 的基础上集成了一种名为 RefDiff 的 Java 代码重构检测工具[27], 通过该工具对代码行进行进一步筛选和处理, 不再将重构的代码行视为缺陷.

在 SZZ 识别引入缺陷的变更过程中, 这 4 种 SZZ 算法都需执行如图 1 所示的 4 个步骤. 它们相同之处在于识别修复了缺陷的变更 (Step 1) 和去除掉不正确的引入缺陷的变更 (Step 4). 而不同之处在于, 识别缺陷代码行 (Step 2) 和识别引入了缺陷的变更 (Step 3) 时使用了不同的搜索策略. 4 种 SZZ 算法的概述参见表 1. 其中, 消除噪音的方法对应 SZZ 的 Step 2, 在修复缺陷的变更中识别出真正含有缺陷的代码行. 搜索策略应用于 SZZ 的 Step 3, 在 Step 2 的基础上进行回溯, 识别出引入了缺陷的变更.

1.2 即时软件缺陷预测

传统缺陷预测技术主要用于预测文件或模块的缺陷倾向性, 属于较粗粒度的软件实体. 然而粗粒度的预测模型可能会将包括数千行代码的文件识别为缺陷[28], 开发人员在此基础上检查并修复缺陷的工作量过于庞大. 此外, 一个文件可能经由多个甚至上百个开发人员修改, 很难找出引入了缺陷的开发人员.

表 1 各 SZZ 实现算法概要

SZZ	描述	消除噪音类型	搜索策略
B-SZZ	原始SZZ由Sliwerski等人[12]提出	缺陷修复涉及的所有代码行都被认为有缺陷	利用VCS中的注释命令识别修复缺陷的变更涉及的所有修改代码行的最近一次变更, 将这些变更标注为引入了缺陷的变更
AG-SZZ	Kim等人[13]在B-SZZ的基础上进行了改进	去除空行、注释行和涉及代码格式修改的非语义代码行, 其余代码行被视为有缺陷	利用注释图记录源文件中代码行的修改过程, 再对注释图进行深度优先搜索(忽略噪音类型的代码行), 识别引入了缺陷的变更
MA-SZZ	Da Costa等人[14]在AG-SZZ的基础上进行改进	去除非语义行和涉及元更改(分支更改、合并更改以及文件属性更改)的代码行	在AG-SZZ的基础上, 识别缺陷的过程中忽略涉及元变更的代码行
RA-SZZ	Neto等人[15]在MA-SZZ的基础上进行了改进	在MA-SZZ的基础上进一步去除涉及代码重构的行	在MA-SZZ的基础上, 识别缺陷的过程中忽略涉及代码重构的行

为解决上述问题, Mockus 等人[29]提出了代码变更级别的缺陷倾向性预测, 该粒度相比文件或模块更细化. 他们构建了初始维护请求 (IMR) 的缺陷预测模型, 一个 IMR 由多个变更组成, 跟踪系统用于记录这些 IMR 的故障记录, 以便于标注缺陷. Kim 等人[30]使用源代码变更日志构建缺陷预测模型, 用于识别多个开源软件项目是否存在缺陷, 平均结果达到了 78% 的准确率和 60% 的召回率. Shihab 等人[31]面向工业软件项目建立变更级模型, 以预测软件变更是否存在缺陷. 他们发现, 代码行与代码块的增加数量、被更改文件的缺陷倾向性、与变更相关的缺陷报告数量以及开发人员的经验是识别代码变更有无缺陷的主要度量. 随后, Kamei 等人[9]利用上述技术建立缺陷预测模型, 用于预测开发人员每次提交的代码变更是否引入了缺陷, 并将其称为即时软件缺陷预测. 具体地, 他们收集了 11 个由 C/C++或 Java 编写的大型项目, 包括 6 个知名度较高的大型开源项目和 5 个大型商业项目, 根据代码变更记录构建了 11 个数据集. 在此基础上, 采用逻辑回归分类器建立工作量感知模型 (EALR), 实验结果表明该模型的平均召回率为 0.64, 在 20% 的工作量中识别出了 35% 的错误代码提交. 该研究为后续即时软件缺陷预测提供了强有力的支撑.

在实际开发过程中, 为了在有限的时间和资源下帮助开发人员尽可能多的发现缺陷, 研究人员提出了工作量感知 (effort-aware) 的概念[32,33]. 工作量感知是指开发人员根据缺陷预测模型的结果, 通常只检查固定数量 (一般为 20%) 的代码行所能发现的缺陷数量及占比, 以便更高效地发现潜在的缺陷变更, 并采取相应的措施进行修复. Yang 等人[34]提出了一种基于单度量元排序的无监督工作量感知即时缺陷预测方法, 与多个有监督模型进行对比, 结果表明基于简单特征排序的无监督模型优于部分有监督模型. 随后, Liu 等人[35]提出基于代码 churn 的无监督工作量感知排序模型. 在 Yang 等人工作的基础上, Fu 等人[36]提出了一种名为 OneWay 的工作量感知即时缺陷预测方法. OneWay 首先使用 Yang 等人[34]的无监督模型来训练数据, 选择具有最佳工作效率的模型, 然后应用于待测数据并对其结果进行优先级排序. 此后, Huang 等人[37]提出了一种分类先于排序的工作量感知即时缺陷预测方法, 称为 CBS+. 该方法集成了无监督模型 LT 与有监督模型 EALR 的思想, 在 7 个开源项目上进行实验, 相比 LT 和 EALR, 所提出的有监督模型具有更高的预测精度. Chen 等人[38]采用多目标优化算法用于即时缺陷预测, 提出 MULTI 模型. 其中一个目标旨在最大化识别缺陷变更的数量, 另一个目标旨在最大程度减少代码审查的工作量. 结果表明 MULTI 在 Popt 和 *Recall*@20% 指标上优于对比方法. 最近, Li 等人[39]提出一种工作量感知的 tri-training 半监督即时缺陷预测方法, 实验结果表明所提出的模型在不同的标注比率上优于基线模型. 此外, 还有一些即时缺陷预测研究[40,41]在专有和 JavaScript 项目上实证比较有监督与无监督模型的性能.

随着深度学习技术的快速发展, 它在各个领域都展现出了巨大的潜力和广泛的应用前景. 深度学习模型以其强大的自动特征提取和模式识别能力, 也开始被应用于软件缺陷预测中[42−44]. Yang 等人[45]采用深度置信网络用于即时缺陷预测, 首先, 传统人工设计的特征被输入到网络中以学习深度特征表示, 然后, 来预测未知的代码变更. 与基线模型相比, 获得了较好的预测性能. Hoang 等人[46]提出了一种端到端的深度学习框架 (DeepJIT), 可从提交信息和代码变更中自动学习深度特征, 结果表明 DeepJIT 优于最先进的基线模型. 随后, 他们采用基于分层注意力网络的深度学习框架学习添加和删除代码的分布式向量表示, 由其附带的日志信息进行引导, 提出了一种 CC2Vec

模型[47]. 实验结果表明 CC2Vec 获得了最先进的即时缺陷预测性能. 最近, Zeng 等人[48]在大规模数据集上实证比较了 DeepJIT、CC2Vec 和传统的即时缺陷预测模型, 他们发现 CC2Vec 并不能始终优于 DeepJIT, 也不能总优于传统的即时缺陷预测模型. 于是他们提出一种基于增加的代码行特征的即时缺陷预测方法, 称为 LApredict. 结果表明, 在项目内与跨项目场景下, LApredict 的有效性和效率都大大优于基于深度学习的模型.

在即时缺陷预测技术可解释性方面, Pornprasit 等人[49]将局部可解释技术 LIME 用于即时缺陷预测中, 提出一种 JITLine 方法用于代码变更及相关联的代码行的预测与解释. 随后, Pornprasit 等人[50]提出一种基于局部规则的模型无关可解释算法, 命名为 PyExplainer, 用于生成即时缺陷预测模型的解释. 与 LIME 相比, PyExplainer 在相似度、局部模型精度、解释唯一性、与实际特征一致性等方面都有所提高. Lin 等人[51]对跨项目即时缺陷预测模型进行可解释性研究, 在 20 个开源项目上实验, 他们发现使用单个项目构建的模型其解释因项目而异, 不考虑项目级差异 (即全局模型) 的情况下, 从多个项目的混合数据池中训练的模型无法捕获项目级差异. 相反, 考虑项目级差异时该模型能够获得更好的解释, 并且不会牺牲性能, 特别是考虑项目的上下文时. Zheng 等人[52]使用随机森林建立即时缺陷预测模型, 并使用 LIME 进行解释. 结果表明, 通过这种方法可以达到原工作效果的 96%, 并减少开发人员 45% 的工作量. 陈丽琼等人[53]将 SHAP 技术用于即时缺陷预测, 他们首先使用 SHAP 分析初始数据集特征, 根据结果对数据集进行特征选择. 然后利用 SMOTEENN 算法对类不平衡数据进行正负实例均衡化, 并使用集成学习算法 XGBoost 进行建模. 最后采用 SHAP 对预测结果进行可解释性分析. Yang 等人[54]实证评价了 LIME、BreakDown 和 SHAP 这 3 种模型无关技术对于即时缺陷预测结果的解释. 在 6 个开源项目上的结果表明不同的变更有着不同的解释, 两个随机变更的度量元排序差异平均为 3. 对于一个给定的软件项目, 不同解释技术产生的 top-1 度量元排名具有很高的一致性, 然而其 top-3 度量元排名却具有较低的一致性. 在实际开发过程中, 他们建议使用模型无关解释技术来帮助开发人员更好地理解模型的预测结果.

此外, Pascarella 等人[55]对变更级缺陷粒度进行了研究, 有缺陷的提交通常由有缺陷和无缺陷的文件组成, 他们提出了更细粒度的文件级即时缺陷预测模型. 在 10 个开源项目上, 实验结果表明他们的模型可以获得更高的 *AUC* 分数. Cabral 等人[56]考虑到即时缺陷预测的类不平衡性, 选择了 10 个 GitHub 开源项目进行研究, 结果表明他们的模型比基线模型在 G-mean 指标上表现得更好. 关于即时软件缺陷预测技术的最新研究进展, 具体可参考文献 [3,57]. 然而, 与此不同, 本文重点关注错误标注的变更对移动 APP 即时缺陷预测性能与解释的影响.

1.3 移动 APP 即时缺陷预测

Scandariato 等人[58]首次研究基于移动 APP 的缺陷预测, 他们选择一个大型移动 APP 的 5 个版本进行实验, 利用支持向量机分类器在类层次上预测 APP 是否存在缺陷. Kaur 等人[59]的研究指出, 基于开发过程度量元的缺陷预测模型比基于移动 APP 的代码复杂性度量元具有更好的性能. Malhotra[60]使用 7 个移动 APP 构建了 18 个分类模型, 结果表明所建立模型之间的性能差异较大, 并且基于支持向量机的模型性能最差. Ricky 等人[61]指出, 在移动 APP 缺陷预测方面, 支持向量机比决策树的性能更佳. 此外, 一些研究利用日志文件和抽象语法树 (abstract syntax trees, AST)[62]来提取高频关键词, 并用于识别移动应用程序中含有缺陷的代码. 上述文献主要侧重于类层次上的 APP 缺陷预测, Catolino 等人[17]的研究表明, 变更层次更适用于移动 APP 缺陷预测.

Zhao 等人[43]提出了一种名为 simplified deep forest (SDF) 的方法, 用于移动 APP 即时缺陷预测. 该模型集成了传统森林的广度和深度特点, 形成了一个具有级联结构的森林模型. 在 10 个移动 APP 上进行实验, 结果表明 SDF 在 *F-measure*, *MCC* 以及 *AUC* 这 3 个性能指标上的表现明显优于基线模型. 为缓解类不平衡问题对预测模型的影响, Zhao 等人[44]提出了一种工作量感知即时缺陷预测模型—KPIDL. 该模型首先利用核主成分分析技术对原始数据进行预处理, 以获得更好的特征表示. 接着, 在深度神经网络中引入一种基于代价敏感的交叉熵损失函数, 通过考虑两类的先验概率来缓解类不平衡问题. 在 15 个移动 APP 上的实验结果表明, 相比于 25 种基线方法, 在大多数情况下, KPIDL 在 2 个工作量感知指标上的表现更为优异. 同样地, Cheng 等人[63]提出一种名为 KAL 的跨项目即时缺陷预测方法, 他们首先使用核主成分分析技术预处理原始数据, 然后使用深度对抗模型学习深度特征表示. 在 14 个移动 APP 上进行实验, 结果表明 KAL 的性能优于 20 种基线模型. Xu 等人[42]使用 19 个移动 APP

构建基于工作量感知的跨项目即时缺陷预测模型, 在 342 对移动 APP 上进行测试, 实验结果表明他们的模型比 14 个基线模型性能更好. 最近, 胡新宇等人[64]针对 Android 移动 APP 即时缺陷预测模型的可解释性展开研究, 通过差分进化算法对 LIME 进行超参优化得到 ExplainApp 方法. 在 14 个移动 APP 项目上进行实验, 结果表明 ExplainApp 方法可以解释移动 APP 即时缺陷预测模型得到的实例预测结果.

上述研究主要关注移动 APP 缺陷预测模型的性能, 然而数据的标注质量对模型性能与解释的影响不可忽视. 到目前为止, 还没有文献实证研究数据标注质量对移动 APP 即时缺陷预测的影响. 基于此, 本文就 SZZ 标注算法对移动 APP 即时缺陷预测模型性能与解释的影响开展广泛而深入的实证研究.

1.4 本文与文献 [5] 研究工作的区别

Fan 等人[5]在 10 个 Apache 开源项目上系统深入研究了 SZZ 错误标注的变更对即时缺陷预测模型的影响. 相较于该研究工作, 本文有以下不同之处.

(1) 实验对象方面. Fan 等人面向 Apache 开源项目进行实证研究, 而本文面向 17 个移动 APP 项目研究 SZZ 错误标注的变更对即时缺陷预测性能与解释的影响. 原因是: 1) 移动 APP 的发布周期通常较短, 版本迭代速度较快, 这对于及时发现和修复缺陷至关重要, 以确保新版本的稳定性与质量. 2) 用户可以随时随地下载与使用移动 APP 应用, 意味着缺陷在任何时间都有可能发生, 在缺陷出现后若能尽快提供反馈, 这将有助于开发小组及时修复缺陷. 3) 移动 APP 相较于其他应用拥有更多用户, 其体验至关重要, 及时发现和修复缺陷可以避免用户在使用 APP 应用时遇到问题, 进而提升用户满意度. 总之, 即时缺陷预测技术非常适用于检测移动 APP 应用中的缺陷.

(2) 数据划分方式. Fan 等人采用 out-of-sample bootstrap 交叉验证方法随机划分训练集与测试集, 其可能会使用当前的变更预测之前的变更 (比如用 2023 年的变更预测 2021 年的变更), 这通常与实际软件开发过程不符. 在即时缺陷预测技术研究方面, 先前工作表明[65]代码变更遵循时间顺序, 使用不考虑时间因素的交叉验证方法随机划分数据集不符合实际开发过程. 然而, Fan 等人使用 out-of-sample bootstrap 方法随机进行数据划分并没有考虑到实际情况. 与该研究工作不同, 本文采用 Yang 等人[34]的所提出的基于时间系列的数据划分方法用于弥补上述不足, 以更符合实际软件开发过程.

(3) 类不平衡处理方式. Fan 等人采用随机欠采样算法进行类重平衡, 没有深入对比不同数据采样算法对缺陷预测模型的性能与解释有何种影响. 由于欠采样算法会随机删除一部分无缺陷的代码变更, 可能导致重要数据信息和关键变更的丢失, 因此随机欠采样在该实验背景下是否真的最优有待进一步考量. 为此, 本文深入对比随机欠采样、随机过采样、合成少数类过采样 (SMOTE) 和随机过采样示例 4 种常用数据采样技术用于类重平衡的效果, 以比较不同数据采样算法对移动 APP 即时缺陷预测性能的影响. 结果发现 SMOTE 总体上表现最为出色, 为此本文采用 SMOTE 算法进行类重平衡, 即可保留原始数据的特征又可增加多样性.

(4) 模型解释方法. Fan 等人采用特征重要性方法对模型进行全局解释, 得出每个度量元的排名. 然而, 该工作有以下局限: 1) 特征重要性方法只能解释度量元对模型的整体影响大小, 不能解释是对模型产生正类 (有缺陷) 还是负类 (无缺陷) 影响. 例如, la (变更增加的代码行数) 度量元在特征重要性中排名较高, 只能得出 la 是重要的度量元, 但 la 对正向 (有缺陷) 或负向 (无缺陷) 的影响方向和程度, 这点无从而知. 2) 全局解释只能解释度量元对模型整体的影响, 不能解释模型在某个变更上的预测结果, 因此无法反映出在单个变更上各个度量元对模型的贡献大小. 3) 在类不平衡与重平衡情况下, 该工作没有探讨 SZZ 错误标注对模型的解释性影响有无异同. 为解决上述不足, 本文在类不平衡和重平衡情况下, 采用更为全面的 SHAP 方法进行解释. 该方法不仅能对整个模型的行为和结果进行全局解释, 还能对某个变更的预测结果进行局部解释, 更清晰地理解模型的决策.

总之, 在实验对象方面, 本文与 Fan 等人所采用的实验对象均采用 Java 语言编写, 差异主要表现在开发人员、编码风格、代码规范、编程经验以及缺陷修复的频率等不同. 在数据划分方面, 本文采用时间划分方式代替先前的随机划分方式, 更符合实际开发流程. 在类不平衡处理方面, 本文对比了多种采样算法, 使用性能最优的 SMOTE 算法进行类重平衡, 即可保留原始数据的特征又可增加多样性. 在模型解释方面, 本文引入了 SHAP 算法从全局和局部的角度对模型的预测结果进行了全面的解释, 能够更加清晰地理解模型的决策. 因此, 基于上述 4 个方面以及软件项目的不同, 从而导致本文的研究结果与 Fan 等人的研究结果存在差异.

2 数据集构建

2.1 数据集

本文使用的 17 个移动 APP 项目均来自 GitHub, 其中 12 个项目来自先前的研究[17], 其余来自 GitHub 中知名度较高、规模较大的移动项目. 表 2 给出了每个移动 APP 的项目信息、起止日期、变更数量和使用 RA-SZZ 算法标注的缺陷比例. 以上 17 个项目完整开发周期的历史数据全部基于 Git 版本控制系统.

表 2 移动 APP 数据集概要

项目	项目描述	起止日期	变更数量	缺陷比例 (RA-SZZ)(%)
Afwall	安卓系统防火墙	2012/12–2022/7	1815	35
Alfresco	企业内容管理	2012/8–2022/4	1 528	11
Android sync	同步适配器	2011/10–2015/7	3 220	43
Android walpaper	壁纸引擎	2014/4–2021/1	796	38
AnySoftKeyboard	安卓系统键盘APP	2009/5–2022/8	7 088	18
Atmosphere	网络事件驱动框架	2010/4–2022/5	6 074	34
Chat secure Android	加密通讯APP	2010/3–2018/1	2 910	30
Facebook Android	社交APP	2010/5–2022/8	2 718	16
Kiwix	离线维基百科阅读器	2012/2–2022/8	6 312	14
Own cloud	文件存储及共享APP	2011/8–2022/8	10 261	16
Page turner	多设备同步阅读器	2011/12–2021/10	1 265	18
Notify reddit	手表Reddit消息提醒	2014/7–2016/6	232	26
Conversations	即时通讯APP	2014/1–2022/8	6 879	25
AntennaPod	播客播放器APP	2011/12–2022/8	8 222	22
AndroidAPS	糖尿病患者血糖监测APP	2016/6–2022/8	14 053	13
k-9	电子邮件客户端	2008/10–2023/5	12 062	16
SeriesGuide	影视作品跟踪及管理APP	2011/7–2023/3	12 289	14

2.2 软件度量元的提取

本节对数据集中的变更度量元进行详细介绍. 本文使用 Kamei 等人[9]提出的 14 个变更度量元, 这些度量元已经被证明在即时缺陷预测模型上表现较好. 表 3 对 14 个度量元进行了描述, 将其分为 5 个维度, 分别为代码分布、规模、目的、历史和开发者经验.

表 3 软件度量元概要

维度	度量元	描述
代码分布	ns	变更修改的子系统数量 (number of subsystems)
	nd	变更修改的目录数量 (number of directories)
	nf	变更修改的文件数量 (number of files)
	entropy	变更修改的代码在相关文件中的分布 (信息熵)
规模	la	变更增加的代码行数 (lines of code added)
	ld	变更删除的代码行数 (lines of code deleted)
	lt	变更之前的代码行数 (lines of code of files touched by the change)
目的	fix	变更是否修复了缺陷
历史	ndev	对该变更相关的文件进行过修改的开发者数量 (number of developers)
	age	该变更相关文件的最近一次修改与该变更的平均时间间隔
	nuc	对该变更相关文件进行过修改的变更数量 (number of unique changes)
开发者经验	exp	开发者已提交变更数量 (experience)
	rexp	开发者近期提交变更数量 (recent experience)
	sexp	开发者已提交变更中影响到该变更相关子系统的数量 (subsystem experience)

代码分布维度将变更涉及的代码分布进行量化, 已有的研究表明代码分布与缺陷发生概率高度相关[66].

规模维度指代码变更的大小, 若代码有较大变更, 意味着代码的修改规模较大, 涉及的范围也更广, 因此更容易引入缺陷[67].

目的维度仅有一个 Fix 度量, 该度量元用来表示变更是否修复了缺陷. 此前的研究发现, 修复了缺陷的变更出现缺陷的概率更高, 并且很可能引入了新的缺陷[68,69].

历史维度用于描述开发人员修改代码所涉及的文件变更历史, 先前的研究中, 这些度量被证明在缺陷标注方面表现良好. Matsumoto 等人[70]的研究指出, 许多开发人员修改过的文件中存在较多缺陷.

开发者经验维度对变更历史中的开发人员信息进行了量化, 根据开发人员此前提交的变更历史构建的度量. Mockus 等人[29]的研究表明, 经验丰富的开发人员提交的变更一般不会引入缺陷.

在度量元的提取阶段, 首先, 使用 PyDriller[71]读取项目的 Git 仓库, 使用 traverse_commits 函数遍历 Git 日志, 这些日志基于时间序列分布, 提取出相关信息后构建树形结构, 每个变更表示一个节点, 其孩子节点分别表示提交中的信息, 例如每个变更的 Hash ID, 提交日期等, 并以 xml 格式保存, 便于后续复用. 其次, 遍历 xml 文件, 计算并生成度量元. 最后, 整合所有度量元信息, 结合进一步的数据标注结果, 生成数据集. 下节本文将介绍数据标注的具体步骤.

2.3 数据标注

本文使用 4 种不同的 SZZ 算法对数据进行标注, 即 B-SZZ、AG-SZZ、MA-SZZ 以及 RA-SZZ. 借助 PyDriller[71]工具对每个移动 APP 的 Git 代码变更日志进行检索, Git 库中包含了所有开发人员提交的代码变更历史, 使用正则表达式“(bug)|(fix)|(crash)|(fault)|(defect)|(problem)|(error)|(patch)|(wrong)”从中筛选出含有缺陷关键词 (例如 bug、fix、crash、fault 等) 的变更记录, SZZ 算法对这些已修复的缺陷变更进行溯源, 进而识别出含有缺陷的代码行, 并标注哪些变更引入了缺陷.

对于本文选取的 17 个移动 APP 项目, 使用 4 种 SZZ 算法对每个 APP 项目进行数据标注, 每个 APP 的数据集将包括 4 种 SZZ 算法的标注结果. Neto 等人[15]发现, RefDiff 工具在检查代码重构修改上的性能表现较好, RA-SZZ 比 MA-SZZ 在识别缺陷方面更加准确. Fan 等人[5]对 RefDiff 工具进行了验证, 其中两位作者具有多年的 Java 开发经验, 在他们的验证下发现 RefDiff 在 10 个项目中达到了 97%–99% 的精度, 相比 MA-SZZ, RA-SZZ 产生的噪音更小, 因此, 他们将 RA-SZZ 算法的标注结果作为真实标签实证比较不同 SZZ 算法的标注性能. 与该研究工作一样, 本文将 RA-SZZ 算法的标注结果作为真实标签数据集, 因为 RA-SZZ 所标注的数据准确率最高. 其余 3 种 SZZ 算法的标注数据作为模型的训练集, 使用 RA-SZZ 生成的测试集对其进行性能评估. 表 4 给出了利用 RA-SZZ 所标注的数据作为参照集, 统计 B-SZZ、AG-SZZ 和 MA-SZZ 错误标注的数量及占比. 从表 4 中可以看出, B-SZZ、AG-SZZ 和 MA-SZZ 的误标率依次递减, 基本符合表 1 所述.

表 4　B-SZZ、AG-SZZ 和 MA-SZZ 的误标数量及占比

项目	B-SZZ		AG-SZZ		MA-SZZ	
	#误标数量	误标占比 (%)	#误标数量	误标占比 (%)	#误标数量	误标占比 (%)
Afwall	359	20	267	15	241	13
Alfresco	366	24	92	6	79	5
Android sync	420	13	301	9	270	8
Android walpaper	208	26	63	8	98	12
AnySoftKeyboard	1 329	19	1 174	16	1 079	15
Atmosphere	939	15	608	10	490	8
Chat secure Android	356	12	208	7	153	5
Facebook Android	399	15	282	10	254	9
Kiwix	1 211	19	1 096	17	1 007	16
Own cloud	1 959	19	1 658	16	1 519	15
Page turner	124	10	103	8	82	6
Notify reddit	34	15	21	9	19	8

表 4 B-SZZ、AG-SZZ 和 MA-SZZ 的误标数量及占比 (续)

项目	B-SZZ		AG-SZZ		MA-SZZ	
	#误标数量	误标占比 (%)	#误标数量	误标占比 (%)	#误标数量	误标占比 (%)
Conversations	1 058	15	898	13	782	11
AntennaPod	955	12	661	8	524	6
AndroidAPS	2 270	16	1 730	12	1 614	11
k-9	1 415	12	1 137	9	967	8
SeriesGuide	1 652	13	1 422	11	1 295	10

3 实证研究

为探究 4 种 SZZ 算法所标注的数据集对移动即时缺陷预测模型的影响, 本节对实验的相关设置进行详细描述. 基于此, 本文设计以下 5 个研究问题.

RQ1: SZZ 错误标注的变更是否影响类不平衡下的移动 APP 即时缺陷预测模型的性能?

RQ2: SZZ 错误标注的变更是否影响类平衡下的移动 APP 即时缺陷预测模型的性能?

RQ3: SZZ 错误标注的变更是否影响类不平衡下的移动 APP 即时缺陷预测模型的解释?

RQ4: SZZ 错误标注的变更是否影响类平衡下的移动 APP 即时缺陷预测模型的解释?

RQ5: 不同数据采样算法间的性能对比如何?

图 2 给出了本实证研究的主要流程, 共分为 8 个步骤.

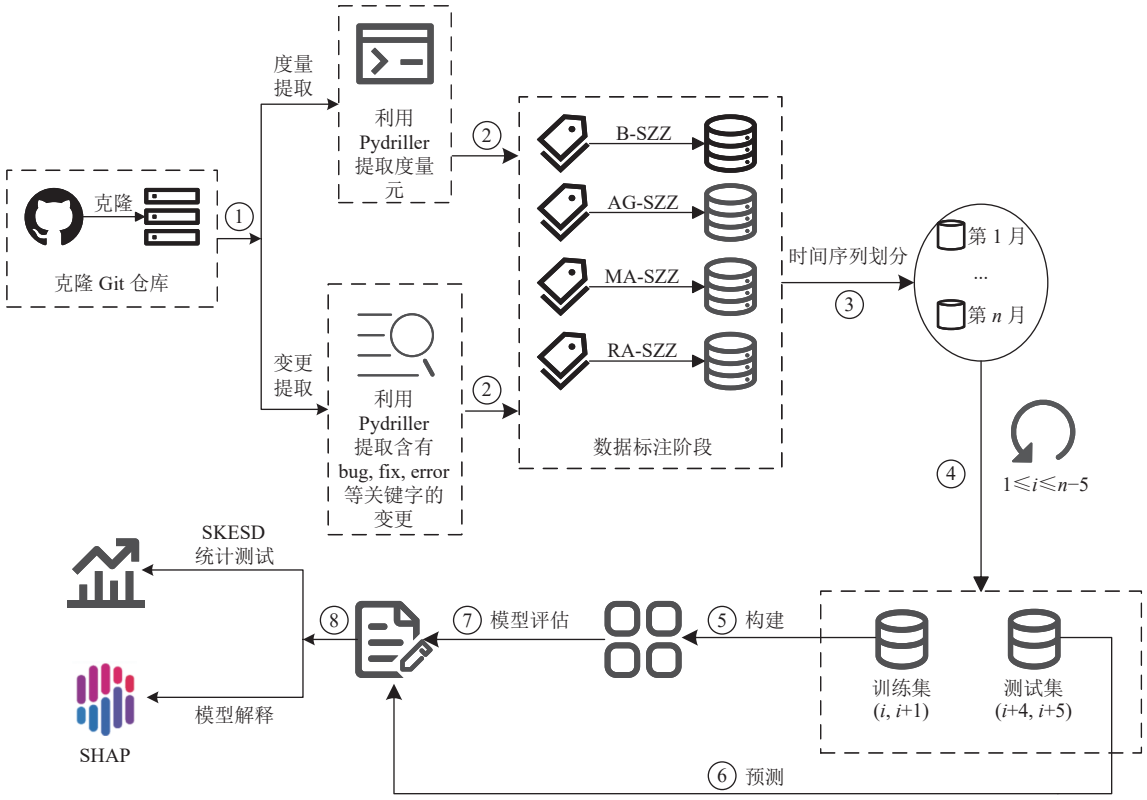


图 2 实验流程图

(1) 克隆移动 APP 远程 Git 仓库, 提取软件度量元.

(2) 提取含有缺陷关键词的变更, 使用 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 算法分别标注数据.

(3) 将所有变更按照提交时间排序, 划分训练集和测试集.

(4) 根据 4 种 SZZ 算法标注的训练集, 使用随机森林、朴素贝叶斯与逻辑回归分类器分别构建即时缺陷预测模型.

(5) 对测试集中每个变更 (即实例) 预测其是否引入了新的缺陷.

(6) 如本文第 2.4 节所述, 将 RA-SZZ 标注的数据集作为真实数据集.

(7) 对即时缺陷预测模型进行性能评估, 并计算评价指标.

(8) 最后使用 SKESD 和 SHAP 算法分别对预测结果进行排序与解释.

3.1 数据预处理

Tantithamthavorn 等人发现相关性较强的度量元会影响即时缺陷预测模型的性能与解释[72]. 因此, 直接使用这 14 个度量元可能会影响模型的性能表现或引起错误的模型解释. 基于先前的研究[9], 在构建即时缺陷预测模型前需要对数据集进行预处理, 预处理工作分为两步.

数据偏斜: 由于数据集中大部分度量值存在偏斜且较为分散, 因此需要对数据进行归一化处理. 为保持数据间的相对关系, 使得数据的分布更加均匀, 与先前研究一样[5], 本文采用 $\ln(x+1)$ 对数变换. 因为 Fix 是一个二进制度量元, 因此不对其进行处理.

度量元相关性: 先前的研究表明相关度量元会对缺陷预测模型产生影响[72]. 此外, 度量元的相关程度在不同项目上可能具有差异性. 因此, 本文对每个数据集的所有度量元进行相关性处理. 具体地, 采用 R 语言提供的 AutoSpearman 算法包[73]分别对每个项目进行 Spearman 相关性分析和方差分析, 以选取度量元.

3.2 数据集划分

对于即时缺陷预测中数据集的划分, 已有的研究表明[65,74], 代码变更遵循时间顺序, 而使用交叉验证随机划分数据集的方式并不符合实际情况. 例如, 用 2023 年提交的变更来预测 2021 年的变更是否引入了缺陷, 这并不符合常理. 此外, 还会导致模型准确度虚高. 为此, Yang 等人[34]提出了一种基于时间感知的数据集划分方法.

根据 Yang 等人[34]的时间序列划分方法, 对于每个项目, 本文将所有变更根据提交日期进行排序, 并将同一个月中的所有变更分为一组. 假设一个项目的所有变更被分为 n 组, 则训练集为第 i 和第 i+1 组, 测试集为第 i+4 和第 i+5 组 ($1 \leqslant i \leqslant n-5$), 从而保证每个训练集和测试集包含两个月的变更数据. 这样划分的理由如下: (1) 大多数项目的开发周期为 2 个月. (2) 每个训练集和测试集具有两个月的间隔, 确保开发人员有足够多的时间修复更多缺陷. (3) 确保训练集和测试集有足够的实例. (4) 使用与预测变更日期相近的变更数据确保实验的有效性.

3.3 模型构建

参考 Fan 等人[5]的研究工作, 本文使用随机森林、朴素贝叶斯和逻辑回归 3 种分类器进行模型构建, 在传统分类和工作量感知排序两种情况下分别构建即时缺陷预测模型. 特别地, 本文采用基于 CBS+[37]的工作量感知即时缺陷预测技术, 该模型目前被证明具有较优的性能. 在本文实验中, 随机森林、朴素贝叶斯和逻辑回归的实现使用 R 语言中的 randomForest 包[75], naivebayes 包[76]以及 glm 函数.

3.4 性能评价指标

为评估缺陷预测模型的性能, 本文使用 *AUC*、*MCC* 和 *G-mean* 这 3 个非工作量感知的评估指标, 其已被证明在类不平衡场景下能够保持稳定[77]. 此外, 使用 *F-measure*@20% 和 *IFA* 两个工作量感知的评估指标[37].

AUC (area under curve), 即曲线下的面积, 其指受试者工作曲线 (receiver operating characteristic, *ROC*), *ROC* 曲线绘制了在所有阈值下真阳性率 (true positive rate, *TPR*) 关于假阳性率 (false positive rate, *FPR*) 的函数曲线. *AUC* 的取值范围为 [0, 1], 若 *AUC* 得分越高, 则该模型表现越好, 若 *AUC* 得分小于或等于 0.5, 则说明该模型的预测没有优于随机预测. 目前, *AUC* 已广泛应用于缺陷预测研究[6,9,78]. 由于缺陷数据普遍具有类不平衡的特点, 而 *AUC* 不受类别分布以及数据的类不平衡影响, 因此其常作为类不平衡场景下的评价指标.

在二分类场景中, *MCC* (Matthews correlation coefficient)[4]用于计算预测值和真实值之间的关系, 其取值范围

为 [−1, 1]. 当值为−1 时, 该模型的预测值与实际值完全不同; 当值为 0 时, 该模型的预测值不如随机预测; 当值为 1 时, 则认为该模型的预测结果与实际情况完全一致. 由此可见, *MCC* 越接近 1 则模型表现越好. 由于 *MCC* 指标考虑到了混淆矩阵的所有结果, 即 *TP*、*TN*、*FP* 和 *FN*, 所以该指标是一个较为均衡的评价指标, 计算公式如下:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{1}$$

G-mean[4]是一种衡量正类和负类性能之间平衡的指标, 适用于类不平衡的数据分类中. 它综合考虑了模型在不同类别上的表现, 通过计算几何平均值来评估模型的性能. *G-mean* 的取值范围为 [0, 1], 其值越高表示模型在不同类别上的表现越好. 当存在类别不平衡时, *G-mean* 能够平衡少数类和多数类的分类性能, 计算公式如下:

$$G\text{-}mean = \sqrt{Recall \times (1 - Pf)} \tag{2}$$

其中, *Recall* 表示被正确预测为缺陷的变更占实际缺陷变更的比率, *Pf* 表示无缺陷的变更被预测为缺陷的变更占实际无缺陷变更的比率.

F-measure@20%[37]是一种基于工作量感知的评价指标, 指使用 20% 的总代码行 (lines of code, *LOC*) 来衡量开发人员审查代码的工作量, 为 *Precision*@20% 和 *Recall*@20% 的调和平均数. 此前的研究指出, 在大多数情况下, 缺陷出现在 20% 的文件或代码行中[15], 因此本文使用 *F-measure*@20% 指标来评估基于工作量感知的即时缺陷预测模型. 具体计算公式如下:

$$F\text{-}measure@20\% = \frac{2 \times Precision@20\% \times Recall@20\%}{Precision@20\% + Recall@20\%} \tag{3}$$

其中, *Precision*@20% 指在使用 20% 的总代码行下被正确预测为缺陷的变更占预测为缺陷变更的比率. *Recall*@20% 指在使用 20% 的总代码行下被正确预测为缺陷的变更占实际缺陷变更的比率. *F-measure*@20% 综合考虑了在工作量感知情况下模型的精确率和召回率.

IFA[37]是指在缺陷检测过程中, 在发现第一个实际缺陷之前遭遇的误报数量, 用于评估缺陷预测的准确性和效率, 较低的初始误报数量表示缺陷检测机制或流程更准确, 产生较少的误报. 假设预测模型所推荐的前 *k* 个变更都是误报, 开发人员会感到沮丧, 不太可能继续审查其他变更. *IFA* 计算如下:

$$IFA = k \tag{4}$$

3.5 统计测试

为了检验不同 SZZ 标注算法之间的性能差异是否具有统计学意义, 本文使用 SKESD (scott-knott effect size difference)[79]算法进行统计显著性测试. SKESD 是一种均值比较算法, 它结合了 Cohen's delta 效应大小和层次聚类分析技术, 用于将一组均值划分为在统计学上具有显著差异的不同组 (α<0.05). 与先前的缺陷预测工作一样[78−80], 本文执行两轮 SKESD 测试. 首先, 对每种 SZZ 标注算法在每个项目上多次运行结果的均值进行 SKESD 测试, 得到每种 SZZ 算法的等级排序列表. 基于此, 再执行第 2 轮 SKESD 测试, 得到每种 SZZ 算法在所有项目上的等级排名, 排名越高则标注性能越好. 通过执行两轮 SKESD 测试, 可得到每种 SZZ 算法在所有项目上的排名, 这样可以综合考虑不同项目中的性能表现. 这种方法能够全面比较和评估各种 SZZ 算法, 考虑到不同项目中的变化和差异.

3.6 可加性解释模型 SHAP

SHAP (shapley additive explanations) 是一种与模型无关的解释方法[81], 它基于博弈论中的 Shapley 值概念, 可适用于对黑盒模型的全局解释或局部解释. 其值提供了一种解释模型预测结果的方法, 可以帮助人们理解模型是如何利用每个特征来进行预测的. 它可以揭示特征对于预测结果的贡献程度, 以及特征值的变化对于预测的影响. 这样的解释有助于人们理解模型的决策过程, 解释模型的预测结果, 并帮助做出可信的决策. 本文使用 SHAP 方法的原因如下: (1) SHAP 一种比较强大的模型无关解释方法, 既可以对缺陷预测模型进行全局解释, 也可以对单个预测结果进行局部解释. 通过全局解释, 可以得出哪些度量元对模型的预测最重要, 哪些度量元对模型的预测影响较小, 有助于了解模型的整体性能和特征的相对重要性. 通过局部解释, 揭示模型为什么对某个变更做出了特定的预测, 有助于发现模型的行为是否符合预期. (2) SHAP 具有理论基础, 可以保证生成最佳的特征重要性排名. 近年

来, 越来越多的软件缺陷预测研究[7,53,54]采用 SHAP 来计算特征重要性排名.

具体地, 对于每个变更, 模型都会输出一个预测值, SHAP 构建一个可加性模型, 将所有特征都作为贡献者, 计算每一个特征对模型输出的贡献. 若训练集有 k 个特征, 假设第 i 个变更为 x_i, 第 i 个变更的第 j 个特征为 x_{ij}, 模型对该变更的预测值为 y_i, 令所有变更目标变量的均值为模型的基准值, 记为 y_{base}, 则第 i 个特征的 SHAP 值为:

$$y_i = y_{\text{base}} + f(x_{i1}) + f(x_{i2}) + \ldots + f(x_{ik}) \tag{5}$$

为了对不同 SZZ 算法标注数据的预测结果进行解释, 本文采用 PyPI (Python package index) 库中的 SHAP 算法包 (https://github.com/shap/shap).

4 实验结果

RQ1: SZZ 错误标注的变更是否影响类不平衡下的移动 APP 即时缺陷预测模型的性能?

研究动机: 在实际项目开发过程中, 即时缺陷预测模型可以帮助开发人员迅速地识别哪些变更可能引入了缺陷, 以便及时进行修复, 同时也避免了将更多的缺陷引入到项目中, 从而提高开发效率. 然而, 目前还未有工作研究不同 SZZ 算法错误标注的数据对移动 APP 即时缺陷预测模型性能的影响. 因此, 在类不平衡情况下, 本文深入研究 SZZ 错误标注的变更对移动 APP 即时缺陷预测模型性能的影响.

研究方法: 如图 2 所示, 将 RA-SZZ 标注的数据作为测试集, 分别使用 B-SZZ、AG-SZZ 和 MA-SZZ 构建的模型与使用 RA-SZZ 构建的模型进行比较 (RA-SZZ 作为基准模型), 以此来探究这 3 种 SZZ 算法对移动即时缺陷预测模型性能的影响. 采用随机森林、朴素贝叶斯和逻辑回归分类器训练模型, 分别记为 B、AG、MA 和 RA, 使用 *AUC*、*MCC*、*G-mean* 以及工作量感知指标 *F-measure*@20% 和 *IFA* 指标来评估模型, 最后应用 SKESD 统计测试对结果进行比较分析.

实验结果: 表 5 列出了各分类器对应的 4 种 SZZ 算法在 17 个移动 APP 项目的结果, 以及 B、AG、MA 模型与 RA 模型得分均值的比率, 每行最好的结果以加粗显示 (下同). 图 3 展示了 4 种 SZZ 算法在不同分类器上的 SKESD 统计结果. 基于以上观察, 可以得出以下结论.

表 5　类不平衡情况下 4 种 SZZ 算法的模型得分

指标	分类器	B		AG		MA		RA
		得分	B-RA得分均值比 (%)	得分	AG-RA得分均值比 (%)	得分	MA-RA得分均值比 (%)	得分
AUC	RF	0.8	95	0.8	95	0.8	95	**0.84**
	NB	**0.72**	106	0.68	100	0.68	100	0.68
	LR	0.73	97	0.73	97	0.74	99	**0.75**
	Average	0.75	99	0.74	98	0.74	98	0.76
MCC	RF	0.35	92	0.36	95	0.36	95	**0.38**
	NB	**0.22**	157	0.16	114	0.16	114	0.14
	LR	0.26	90	0.26	90	0.27	93	**0.29**
	Average	0.28	113	0.26	100	0.26	101	0.27
G-mean	RF	**0.68**	105	**0.68**	105	**0.68**	105	0.65
	NB	**0.59**	109	0.54	100	0.54	100	0.54
	LR	0.62	98	0.61	97	0.62	98	**0.63**
	Average	0.63	104	0.61	100	0.61	101	0.61
F-measure@20%	RF	0.27	87	0.30	97	0.3	97	**0.31**
	NB	0.26	93	**0.28**	100	0.28	100	**0.28**
	LR	0.27	90	0.28	93	0.29	97	**0.30**
	Average	0.27	90	0.29	97	0.29	98	0.30
IFA	RF	3	150	2	100	2	100	2
	NB	2	100	2	100	2	100	2
	LR	2	100	2	100	2	100	2
	Average	2	117	2	100	2	100	2

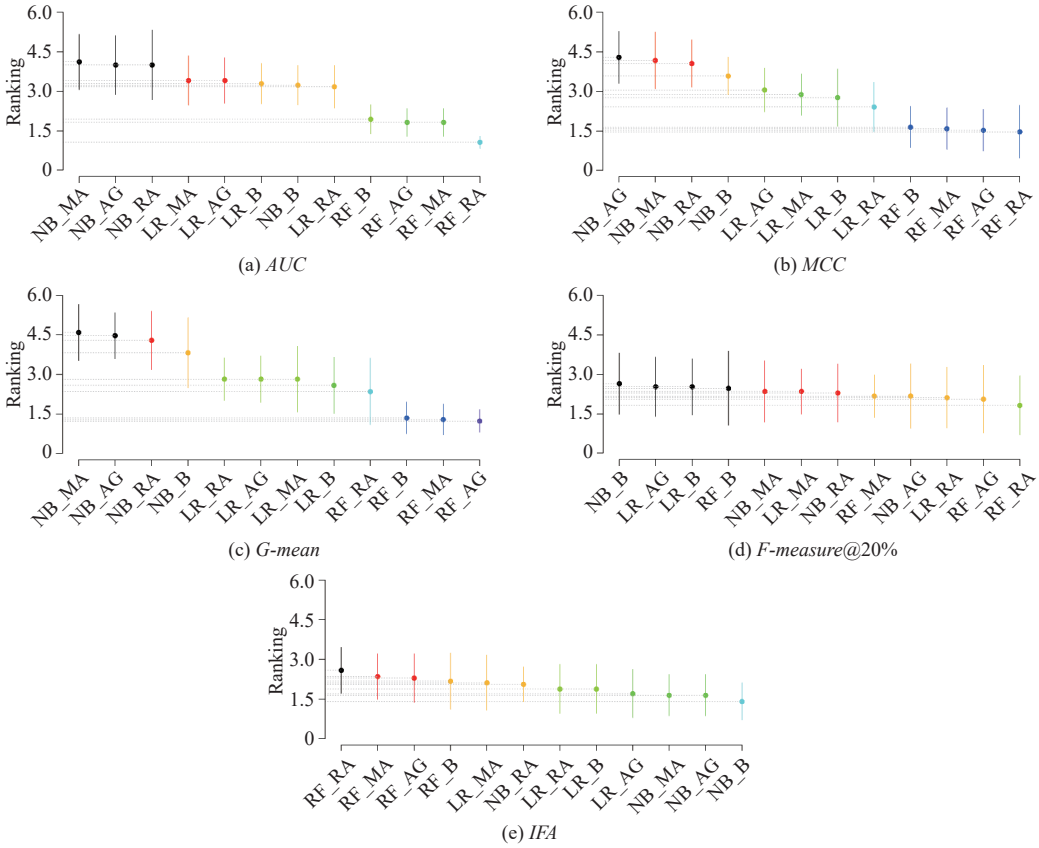


图 3　类不平衡情况下 4 种 SZZ 算法的 SKESD 统计测试结果

(1) B-SZZ: 在传统指标方面, 观察到 B 模型相对于 RA 模型在 *AUC* 方面的得分占比为 95%–97%, 平均占比为 99%, 仅在朴素贝叶斯分类器上优于 RA 模型. 在 *MCC* 指标上, B 模型的得分相对于 RA 模型为 90%–157%, 平均占比为 113%, 仅在朴素贝叶斯分类器上优于 RA 模型. 此外, 在 *G-mean* 指标上, B 模型的得分相对于 RA 模型为 98%–109%, 平均占比为 104%, 仅逻辑回归分类器上低于 RA 模型.

在工作量感知指标方面, 发现 B 模型相对于 RA 模型在 *F-measure*@20% 方面的得分为 87%–90%, 平均占比为 90%, 在所有分类器上不如 RA 模型. 在 *IFA* 方面, B 模型相对于 RA 模型的得分为 100%–150%, 平均占比为 117%, 仅在随机森林分类器上优于 RA 模型.

(2) AG-SZZ: 在传统指标方面, AG 模型的 *AUC* 得分相对于 RA 模型的得分占比为 95%–100%, 平均占比为 98%, 仅在朴素贝叶斯分类器上与 RA 模型相当. 在 *MCC* 方面, AG 模型的得分相对于 RA 模型的得分为 90%–114%, 平均占比为 100%, 仅在朴素贝叶斯分类器上与 RA 模型相当. 此外, 对于 *G-mean* 指标而言, AG 模型的得分相对于 RA 模型的得分占比为 97%–105%, 平均占比为 100%, 仅在逻辑回归分类器上低于 RA 模型.

在工作量感知指标方面, 对于 *F-measure*@20% 而言, AG 模型相对于 RA 模型的得分为 93%–100%, 平均占比为 97%, 仅在朴素贝叶斯分类器上与 RA 模型一致. 在 *IFA* 方面, AG 模型相对于 RA 模型的得分为 100%, 与 RA 模型基本一致.

(3) MA-SZZ: 在传统指标方面, 观察到 MA 模型相对于 RA 模型的 *AUC* 得分占比为 95%–100%, 平均占比为 98%. 值得注意的是, 仅在随机森林分类器上显著低于 RA 模型. 此外, 在 *MCC* 方面, MA 模型相对于 RA 模型的得分为 93%–114%, 平均占比为 101%, 仅在朴素贝叶斯分类器上优于 RA 模型. 在 *G-mean* 指标方面, MA 模型相对于 RA 模型的得分占比为 98%–105%, 平均占比为 101%, 仅在逻辑回归分类器上低于 RA 模型.

在工作量感知指标方面, 对于 *F-measure*@20% 而言, MA 模型相对于 RA 模型的得分为 97%–100%, 平均占比为 98%, 仅在朴素贝叶斯分类器上与 RA 模型一致. 在 *IFA* 方面, MA 模型相对于 RA 模型的得分为 100.0%, 与 RA 模型基本一致.

SKESD 结果表明, 在 *AUC* 上, 随机森林分类器获得了最优的性能, 并且 B、AG、MA 和 RA 模型的性能依次递增, 符合 4 种 SZZ 算法间的递进关系. 而朴素贝叶斯和逻辑回归会造成不同程度的影响. 在 *MCC* 上, 随机森林获得了最优的性能, 且 4 种 SZZ 模型间的性能基本一致. 在 *G-mean* 上, 随机森林仍获得了最优的性能, 但是 RA 模型的表现显著低于其他 3 种 SZZ 模型. 在工作量感知指标方面, 随机森林分类器在 *F-measure*@20% 上整体获得了最优的性能, 同时符合 4 种 SZZ 算法之间的递进关系. 此外, 朴素贝叶斯在 *IFA* 上整体获得了最优的性能, 随机森林性能整体上较差, 结合表 5 中的数据来看, 性能下降的原因可能是局部数据差异较大所造成的.

RQ2: SZZ 错误标注的变更是否影响类平衡下的移动 APP 即时缺陷预测模型的性能?

研究动机: 在类平衡情况下, 目前还未有工作研究不同 SZZ 算法错误标注的数据对移动 APP 即时缺陷预测模型性能的影响. 为此在 RQ2 中, 本文对其进行深入研究. 已有工作在构建即时缺陷预测模型时, 常使用数据采样技术来处理缺陷数据集的类不平衡问题[5,9,74]. 然而现有研究大都采用随机欠采样算法, 这可能会丢失关键数据[16], 进而影响模型的性能. 为此, 本文使用 SMOTE 采样算法对数据进行类重平衡, 不同采样算法的比较结果详见第 4 节 RQ5.

研究方法: 对于每一个基于时间序列划分的训练集, 本文首先使用 SMOTE 采样技术进行类重平衡, 接着采用随机森林、朴素贝叶斯及逻辑回归分类器分别构建 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 模型, 最后使用 *AUC*、*MCC*、*G-mean* 和工作量感知指标 *F-measure*@20%、*IFA* 来评估模型的性能. 随后对比 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 模型的结果, 并应用 SKESD 统计测试方法对所有模型的预测结果进行分析.

实验结果: 表 6 列出了在类平衡情况下, 各分类器对应的 4 种 SZZ 算法在 17 个移动 APP 项目的结果. 图 4 展示了 4 种 SZZ 算法的 SKESD 统计结果. 据此, 可以得出以下结论.

表 6　类重平衡情况下 4 种 SZZ 算法的 AUC 得分

指标	分类器	B		AG		MA		RA
		得分	B-RA得分均值比 (%)	得分	AG-RA得分均值比 (%)	得分	MA-RA得分均值比 (%)	得分
AUC	RF	0.80	94	0.81	95	0.81	95	**0.85**
	NB	**0.71**	106	0.68	101	0.67	100	0.67
	LR	0.73	97	0.73	97	0.73	97	**0.75**
	Average	0.75	99	0.74	98	0.74	98	0.76
MCC	RF	0.35	90	0.36	92	0.36	92	**0.39**
	NB	**0.22**	183	0.16	133	0.15	125	0.12
	LR	0.27	93	0.26	90	0.26	90	**0.29**
	Average	0.28	122	0.26	105	0.26	102	0.27
G-mean	RF	0.68	100	**0.69**	101	**0.69**	101	0.68
	NB	**0.60**	111	0.54	100	0.54	100	0.54
	LR	0.64	97	0.63	95	0.63	95	**0.66**
	Average	0.64	103	0.62	99	0.62	99	0.63
F-measure@20%	RF	0.29	88	0.31	94	0.30	91	**0.33**
	NB	0.26	93	**0.28**	100	**0.28**	100	**0.28**
	LR	0.26	84	0.29	94	0.29	94	0.31
	Average	0.27	88	0.29	96	0.29	95	0.31
IFA	RF	2	100	2	100	2	100	2
	NB	2	100	3	150	2	100	2
	LR	2	100	2	100	2	100	2
	Average	2	100	2	117	2	100	2

(1) B-SZZ: 在传统指标方面, B 模型在 *AUC* 方面的得分占比 RA 模型的 94%–106%, 平均为 99%, 仅在朴素贝叶斯分类器上优于 RA 模型. 在 *MCC* 方面, B 模型的得分为 RA 模型的 90%–183%, 平均占比为 122%, 仅在朴素

贝叶斯分类器上优于 RA 模型. 在 *G-mean* 方面, B 模型的得分占比 RA 模型的 97%–111%, 平均为 103%, 同样仅在朴素贝叶斯分类器上优于 RA 模型.

在工作量感知指标方面, B 模型在 *F-measure*@20% 方面的得分为 RA 模型的 84%–93%, 平均占比 88%, 在 3 种分类器上均低于 RA 模型. 在 *IFA* 方面, 3 种分类器的 B 模型得分与 RA 模型基本一致.

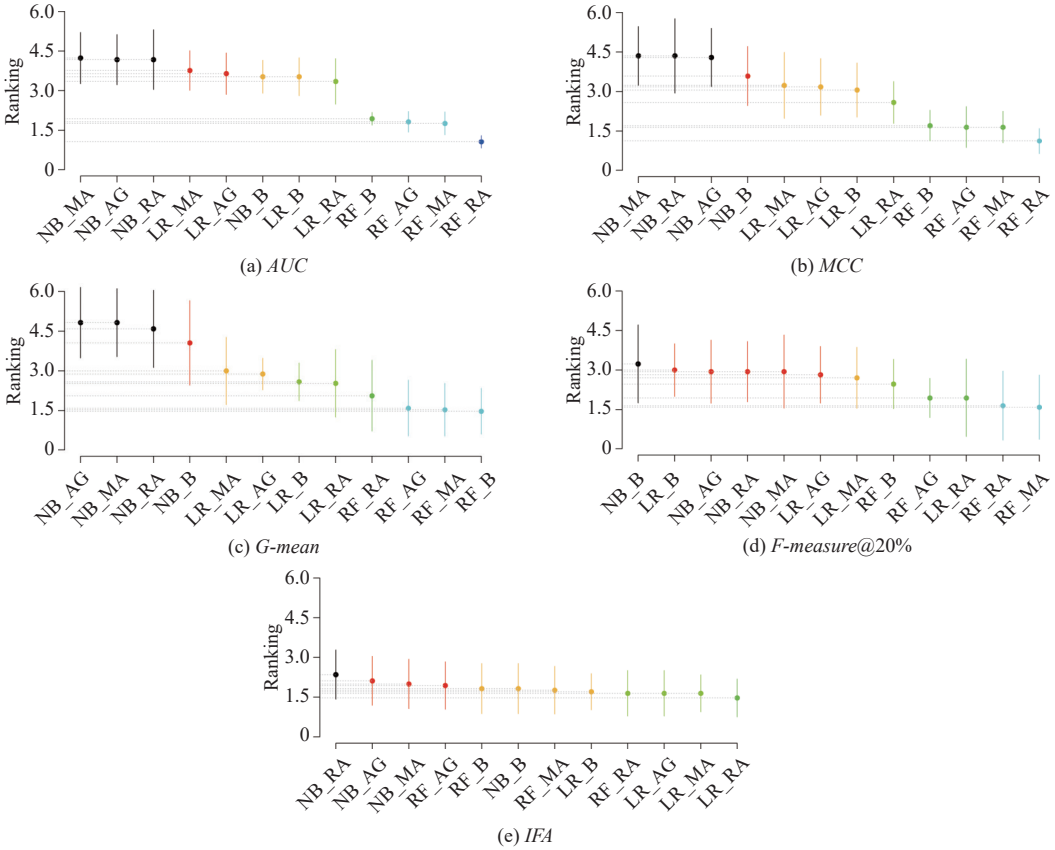


图 4　类重平衡情况下 4 种 SZZ 算法的 SKESD 统计测试结果

(2) AG-SZZ: 在传统指标方面, AG 模型在 *AUC* 方面的得分占比 RA 模型的 95%–97%, 平均为 98%. 仅在朴素贝叶斯分类器上优于 RA 模型. 在 *MCC* 方面, AG 模型的得分为 RA 的 90%–133%, 平均占比为 105%, 同样仅在朴素贝叶斯分类器上优于 RA 模型. 在 *G-mean* 指标上, AG 模型的得分占 RA 模型的 95%–101%, 平均为 99%, 仅在逻辑回归分类器上低于 RA 模型.

在工作量感知指标方面, AG 模型在 *F-measure*@20% 方面的得分为 RA 模型的 94%–100%, 平均占比 96%, 仅在朴素贝叶斯分类器上与 RA 模型相当. 在 *IFA* 方面, AG 模型的得分为 RA 模型的 100%–150%, 平均占比 117%, 3 种分类器的 AG 模型得分不低于 RA 模型.

(3) MA-SZZ: 在传统指标方面, MA 模型在 *AUC* 方面的得分占比 RA 模型的 95–100%, 平均为 98%, 仅在朴素贝叶斯分类器上与 RA 模型一致. 在 *MCC* 方面, MA 模型的得分为 RA 模型的 90%–125%, 平均占比为 102%, 仅在朴素贝叶斯分类器上优于 RA 模型. 在 *G-mean* 方面, MA 模型的得分占比 RA 模型的 95%–101%, 平均为 99%, 仅在逻辑回归分类器上低于 RA 模型.

在工作量感知指标方面, MA 模型在 *F-measure*@20% 方面的得分为 RA 模型的 91%–100%, 平均占比 95%, 仅在朴素贝叶斯分类器上与 RA 模型一致. 在 *IFA* 方面, MA 模型的得分为 RA 模型的 100%, 3 种分类器的 MA 模

型得分与 RA 模型一致.

SKESD 统计测试结果表明, 在 *AUC* 方面, 随机森林分类器获得了最优的性能, 并且 B、AG、MA 和 RA 模型的性能依次递增, 符合 4 种 SZZ 算法间的递进关系. 结合表 6 中的数据来看, 随机森林上的结果普遍高于其他 2 种分类器, 使用朴素贝叶斯和逻辑回归分类器会造成得分显著降低. 在 *MCC* 指标上, 随机森林获得了最优的性能, 除 RA-SZZ 模型之外, 其他 3 种 SZZ 模型的性能基本一致. 在 *G-mean* 指标上, 随机森林分类器仍获得了最优的性能, 但是 RA 模型的表现显著低于其他 3 种 SZZ 模型. 在工作量感知指标方面, 随机森林分类器在 *F-measure*@20% 指标上整体获得了最优的性能, 同时符合 4 种 SZZ 算法之间的递进关系. 此外, 朴素贝叶斯在 *IFA* 指标上整体获得了最优的性能, 虽然在随机森林模型上的性能较差, 但结合图 4 来看, 各模型的性能差距并不显著.

RQ3: SZZ 错误标注的变更是否影响类不平衡下的移动 APP 即时缺陷预测模型的解释?

研究动机: 先前的工作在即时缺陷预测模型可解释性方面进行了探究[50−53], 通过评估软件度量元对于模型预测的贡献度, 帮助人们理解模型的决策过程, 并解释模型的预测结果, 从而更好地指导开发人员修复已有的缺陷或避免引入新的缺陷. 然而, 不同 SZZ 算法标注的数据对移动 APP 即时缺陷预测模型解释的影响尚未可知. 因此在 RQ3 中, 本文研究 SZZ 错误标注的变更在类不平衡情况下对移动 APP 即时缺陷预测模型解释的影响.

研究方法: 先前的研究[7,31,50]表明随机森林分类器在即时缺陷预测模型中效果表现最为出色, 结合本文的实验结果与该结论基本一致, 因此在 RQ3 与 RQ4 中, 本文对使用随机森林分类器构建的模型进行解释, 在类不平衡场景下, 对于每一个移动 APP 项目, 本文首先采用 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 标注的数据构建随机森林模型, 接着使用 SHAP 框架对预测结果进行解释, 每种数据标注算法对应一组 SHAP 解释结果, 再整合每种标注算法对应的所有项目的解释结果, 最后对比 B-SZZ、AG-SZZ、MA-SZZ 与 RA-SZZ 在所有项目上的解释差异, 并重点关注对模型预测结果的影响程度排名前 3 的度量元. 本文将分别从全局和局部的角度对模型预测结果进行可解释分析, 全局解释可以对整个模型的预测结果进行统计, 给出全局上的结果, 揭示每个度量元的作用和重要性排名, 从而可以将有重要影响的度量元反馈给开发者, 有针对性的优化代码质量. 通过局部解释, 可以揭示在单个代码变更上的模型决策, 得到更加细节的度量元贡献方向与大小, 方便开发者更加清晰地理解分类模型在单个代码变更上的决策. 下面介绍具体实现方法.

(1) 如图 2 所示, 对于所有项目的每一个 SZZ 标注算法, 使用基于时间序列划分的训练集构建模型, 并使用 SHAP 框架计算每个度量元对应的 SHAP 值, 得到一次划分训练集的 SHAP 值.

(2) 整合每个项目的每一次划分的训练集所对应的 SHAP 值, 并对整个模型进行解释.

(3) 重复 (1)、(2) 步骤, 使用 SHAP 框架对每个项目的所有 SZZ 模型进行解释.

实验结果: 在全局解释方面, 图 5 展示了使用 SHAP 框架对所有项目的 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 即时缺陷预测模型的全局解释, 左侧纵坐标表示度量元的重要性程度, 排名越高表示其影响越大; 右侧纵坐标表示度量元的数值大小, 其值越大越偏向红色, 其值越小越偏向蓝色; 横坐标表示度量元对模型预测的影响方向和程度, 若度量元的 SHAP 值为正数, 则表示对模型的预测有正向影响, 正向即预测为有缺陷, 值越大则影响程度越大, 反之亦然. 基于以上观察, 可以得出以下结论.

(1) 如图 5 所示, la 的数值越大, 模型将变更预测为有缺陷的概率越大, 其数值越小, 模型将变更预测为无缺陷的概率越大, 这表明 la 在缺陷预测中有重要影响, 增加的代码行数越多, 越容易产生缺陷, 与实际软件开发情况比较相符.

(2) 从图 5 可以看出, 对于 RA-SZZ, 影响程度最大的 3 个度量元为 nuc、la 和 sexp, 其分别表示修改过的变更数量、增加的代码行数和开发者已提交变更中影响到该变更相关子系统的数量. 对于 AG-SZZ 和 MA-SZZ, la、sexp 和 lt 是最具影响力的度量元, lt 表示变更前的代码行数. 对于 B-SZZ, la、sexp 和 ld 是最具影响力的度量元, ld 表示变更所删除的代码行数.

(3) 相较于 RA-SZZ 模型, nuc 在 B-SZZ、AG-SZZ 和 MA-SZZ 模型的预测结果上的影响程度显著下降, 而 la

对 B-SZZ、AG-SZZ 和 MA-SZZ 预测结果的影响程度最大. 这进一步表明 B-SZZ、AG-SZZ 和 MA-SZZ 对预测结果的解释具有一致性, 相比 RA-SZZ 对预测结果的解释具有较低的一致性.

(4) 通过以上分析可以发现, la 度量元影响最大且倾向于对预测为有缺陷的贡献突出, 因此开发人员在提交代码时, 对于添加行数较多的提交应该重点检查是否引入了缺陷.

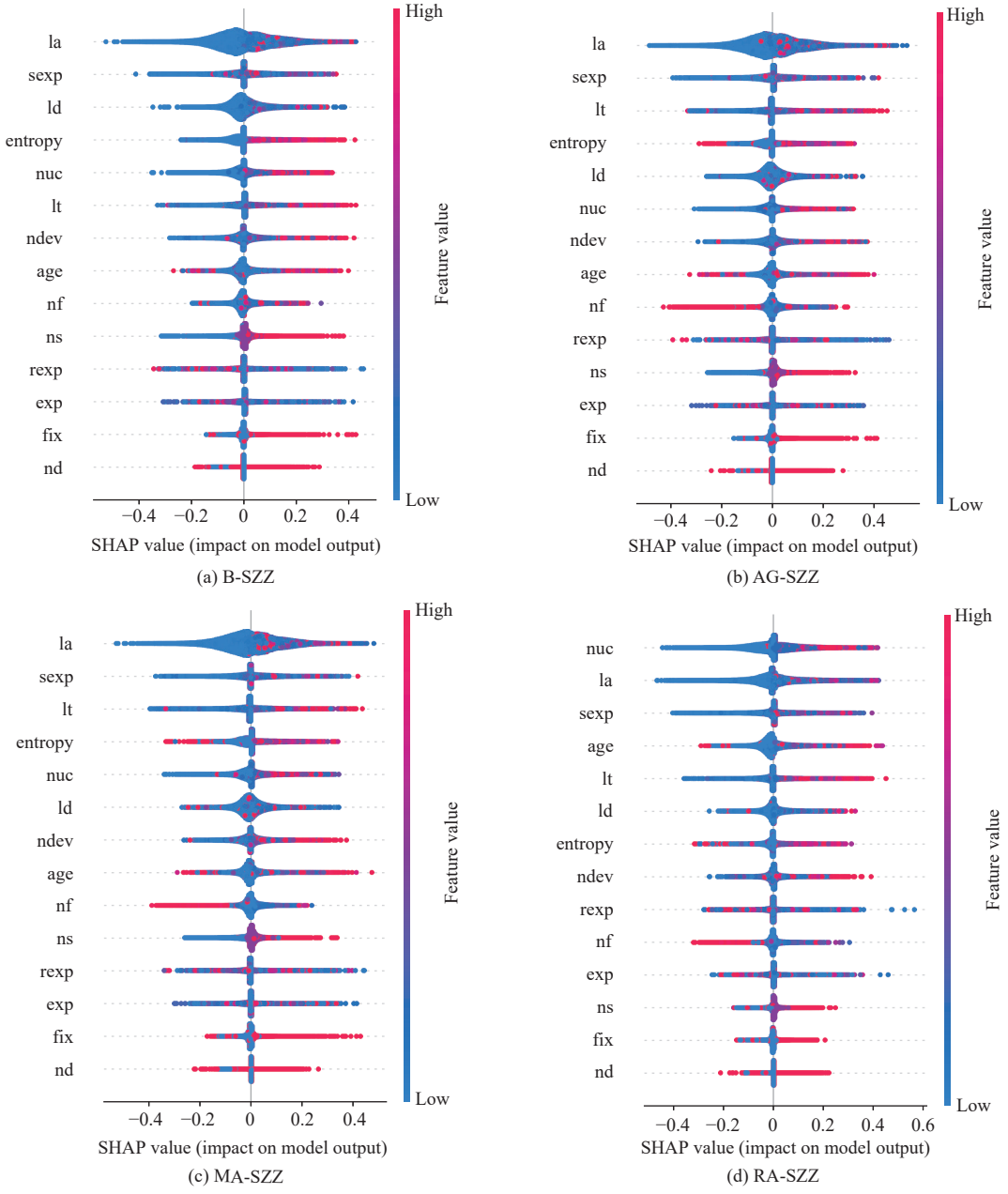


图 5　类不平衡情况下所有项目的 4 种 SZZ 模型的全局解释

在局部解释方面, 为了探究对不同 SZZ 算法标注的数据对某个变更预测结果的影响, 本文使用 SHAP 框架对 AntennaPod 项目中的 1 个变更 (Commit ID: eeeff6203) 进行解释分析. 图 6 展示了 SHAP 对 B-SZZ、AG-SZZ、

MA-SZZ 和 RA-SZZ 模型预测结果中各度量元的解释. 在使用的数据集中, 正类表示有缺陷, 负类表示无缺陷. 图中红色表示度量元对预测结果为有缺陷的贡献程度, 蓝色表示度量元对预测为无缺陷的贡献程度. 基于以上观察, 可以得出以下结论.

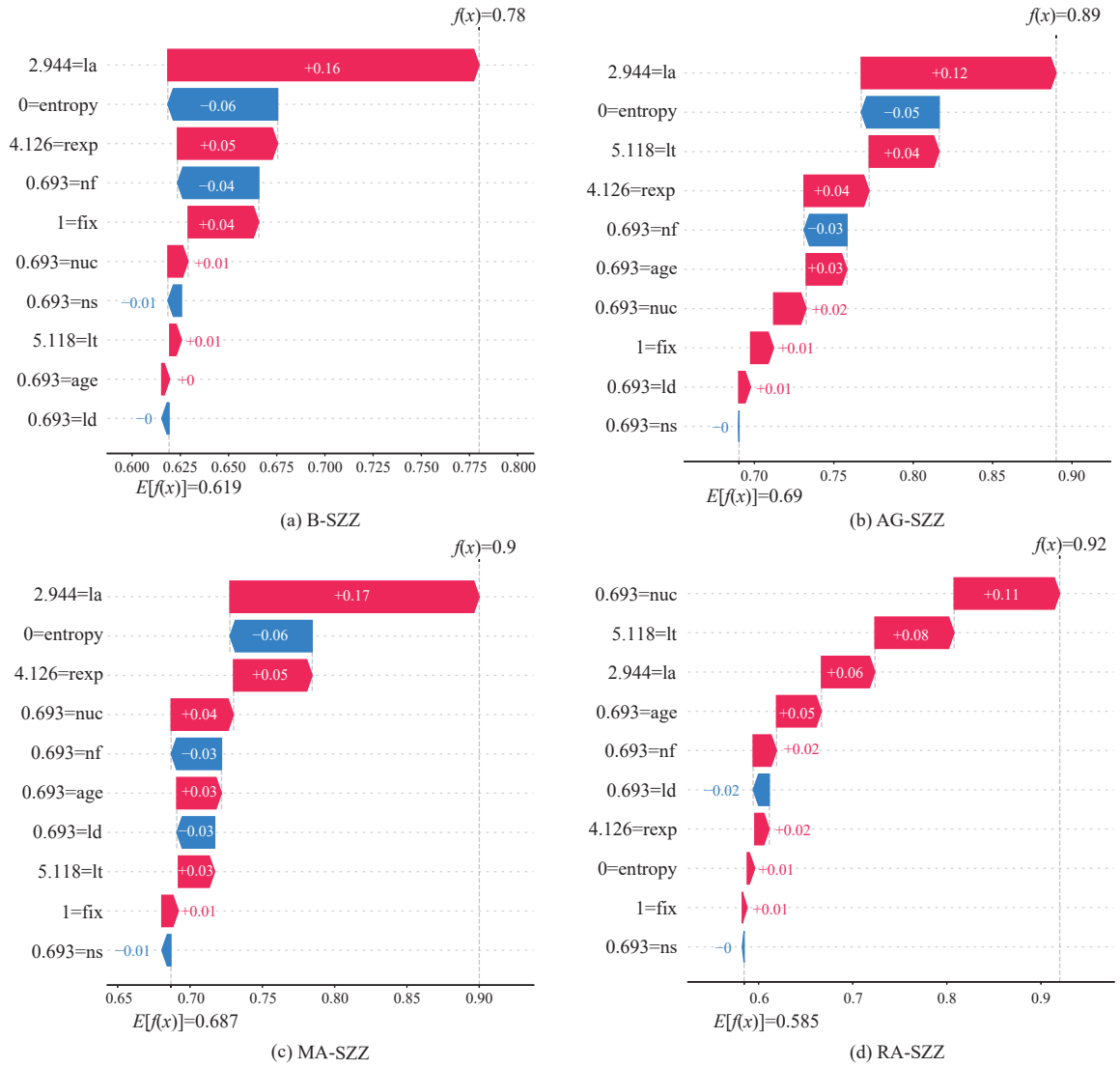


图 6　类不平衡情况下 AntennaPod 项目上其中一个变更 (eeeff6203) 的局部解释

(1) 如图 6 所示, 对于 RA-SZZ 本次预测结果 $f(x)=0.92>0$, 表明该变更被预测为有缺陷, 且预测概率接近 1, 因此可以认为该变更的缺陷程度较高. 由于该变更的预测结果为有缺陷, 本文重点关注对正类预测贡献较大的度量元.

(2) 对于 RA-SZZ, nuc、lt 和 la 等度量元对模型的预测结果有正向贡献, 说明修改过的变更数量、变更前的代码行数与增加的代码行数这些度量元取值越大更倾向于引入缺陷, 而 ld 度量元则对预测结果有负向贡献, 说明减少的代码行数越多越倾向于无缺陷. 为进一步分析, 观察到该变更 la 实际的度量元值为 18, 这表明该变更引入了 18 行代码, 一次修改的代码较多.

(3) 对于 B-SZZ、AG-SZZ 和 MA-SZZ 模型, 它们对该变更的预测结果均为有缺陷, 并且预测概率较高, 说明在预测结果上它们与 RA-SZZ 一致. 具体而言, RA-SZZ 对预测结果产生最显著正向贡献的前 3 名度量元分别是 nuc、lt 和 la. 对于 B-SZZ, 贡献最显著的前 3 名度量元分别是 la、rexp 和 fix. 与 RA-SZZ 相比, B-SZZ 的错误标注会影响 nuc、lt 和 la 的贡献程度. 对于 AG-SZZ, 贡献最为显著的前 3 名度量元分别为 la、lt 和 rexp. 与 RA-SZZ 相比, AG-SZZ 的错误标注会影响 nuc 和 la 的贡献程度. 至于 MA-SZZ, 贡献最为显著的前 3 名度量元分别为 la、rexp 和 nuc. 与 RA-SZZ 相比, MA-SZZ 的错误标注会影响 nuc、lt 和 la 的贡献程度.

(4) 通过以上分析发现, 基于贡献程度最高的前 3 名度量元, B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 对预测结果的解释并不完全一致. 然而它们都包含 la 度量元且都对预测为有缺陷的贡献大, 说明增加的代码行数这一度量元是需要开发人员在代码提交中重点关注的.

上述解释结果仅涵盖了单个变更的局部影响. 为了进一步研究多次代码变更的解释结果, 本文汇报了所有项目中所有变更的解释结果占比情况, 具体如下.

(1) 在类不平衡情况下, 针对每种 SZZ 模型, 计算所有项目中每个变更在 4 种 SZZ 模型上的 SHAP 值, 即每个变更的每个度量元的 SHAP 值.

(2) 对于每个项目的每个变更, 计算每个度量元的 SHAP 值占比, 即该度量元的 SHAP 值除以当前变更所有度量元的 SHAP 值之和.

(3) 最后, 对于所有项目的所有变更, 计算每个度量元的 SHAP 值占比.

图 7 展示了在类不平衡情况下所有项目中所有变更的 4 种 SZZ 模型的局部解释结果占比. 从图 7 中可以观察到, 对于 B-SZZ 模型, 影响程度最大的 3 个度量元是 la (20%)、sexp (11%) 和 ld (9%); 对于 AG-SZZ 和 MA-SZZ 模型, 影响程度最大的 3 个度量元是 la (20%)、sexp (11%) 和 lt (9%), 而对于 RA-SZZ 模型, 影响程度最大的 3 个度量元是 nuc (18%)、la (16%) 和 sexp (11%). 这表明, 在所有项目的所有变更中, 局部解释结果与全局解释的结果相一致, 从而证实了全局解释对局部解释的泛化性. 这一发现与先前的研究工作[7]的结论一致. 该研究使用 SHAP 对 TensorFlow 项目中的一个缺陷预测变更进行局部解释, 并使用 SHAP 对整个模型进行全局解释, 实验结果验证了局部解释的泛化性.

RQ4: SZZ 错误标注的变更是否影响类平衡下的移动 APP 即时缺陷预测模型的解释?

研究动机: 在 RQ3 中, 本文研究了在类不平衡情况下 SZZ 错误标注的变更对移动 APP 即时缺陷预测模型解释的影响. 考虑到类不平衡可能会影响模型的解释性, 本问题在 RQ3 的基础上, 研究在类平衡情况下, SZZ 错误标注的变更对移动 APP 即时缺陷预测模型解释的影响.

研究方法: 对于每一个移动 APP 项目, 本文采用 SMOTE 算法对训练数据进行类重平衡, 接着分别使用 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 标注的数据构建随机森林模型, 然后利用 SHAP 框架对预测结果进行解释, 每种数据标注算法对应一组 SHAP 解释结果, 再整合每种标注算法对应的所有项目的解释结果, 最后对比 B-SZZ、AG-SZZ、MA-SZZ 与 RA-SZZ 在所有项目上的解释差异, 并重点关注对模型预测结果的影响程度排名前 3 的度量元. 本文将分别从全局和局部的角度分别对模型预测结果进行可解释分析, 其具体实验步骤与 RQ3 描述相同.

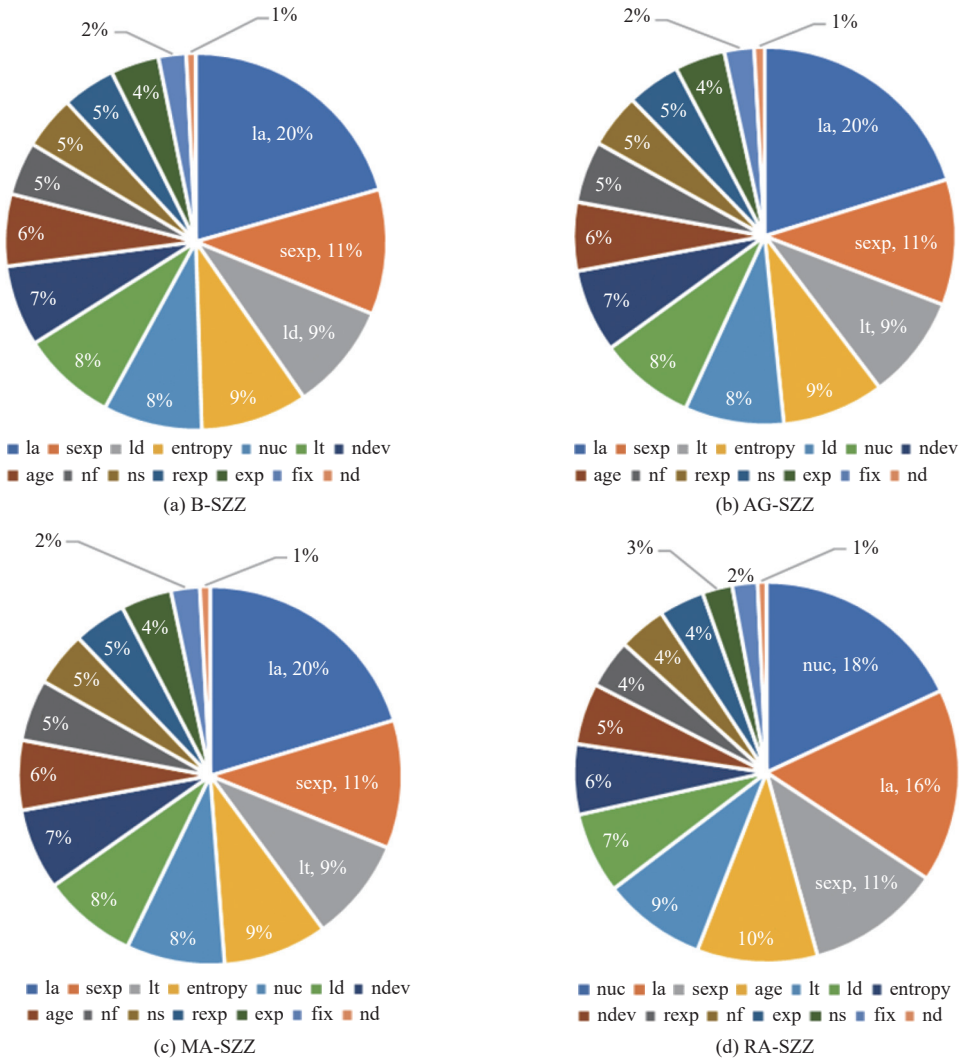
实验结果: 在全局解释方面, 图 8 展示了 SHAP 框架在所有项目上的 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 即时缺陷预测模型的解释结果. 基于以上观察, 可以得出以下结论.

(1) 如图 8 所示, 对 RA-SZZ 模型预测结果的影响程度最大的前 3 名度量元为 nuc、age 和 la. 对于 B-SZZ、AG-SZZ 和 MA-SZZ, 影响程度最大的前 3 名度量元分别为 la、entropy 和 sexp.

(2) 相较于 RA-SZZ, nuc 和 age 度量元在 B-SZZ、AG-SZZ 和 MA-SZZ 模型中的影响程度显著下降. 而 la 在 B-SZZ、AG-SZZ 和 MA-SZZ 模型中对预测结果的影响程度排名均为第 1.

(3) 在类重平衡情况下, B-SZZ、AG-SZZ、MA-SZZ 与 RA-SZZ 在全局解释方面并不一致, 然而它们的前 3 名度量元都包括 la 且倾向于对预测为有缺陷的贡献突出, 说明 la 对模型的预测结果有重要影响.

在局部解释方面, 本文使用 SHAP 对 AntennaPod 项目中 1 个变更 (eeeff6203) 的预测结果进行解释, 如图 9 所示, SHAP 解释了该变更中的所有度量元对 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 模型预测结果的影响. 与 RQ3 相同, 本文重点关注对预测为正类的贡献程度较高的度量元. 基于以上观察, 可以得出以下结论.



(a) B-SZZ (b) AG-SZZ

(c) MA-SZZ (d) RA-SZZ

图 7 类不平衡情况下所有项目中所有变更的 4 种 SZZ 模型的局部解释

(1) 从图 9 可以看出, RA-SZZ 模型的 $f(x)$=0.93>0, 说明将该变更预测为有缺陷, 且 $f(x)$ 大小接近 1, 说明预测为有缺陷的程度更高. 此外, nuc、lt、la 等度量元对模型的预测结果有正向贡献, 而 ld 度量元对模型的预测结果有负向贡献. 对于 B-SZZ、AG-SZZ 和 MA-SZZ, 它们将该变更都预测为有缺陷, 且概率较大, 说明这 3 种 SZZ 与 RA-SZZ 相比在预测结果上并无显著差异.

(2) 对于 RA-SZZ, 对预测结果产生最显著正向贡献的前 3 名度量元分别是 nuc、lt 和 la. 对于 B-SZZ, 贡献最显著的前 3 名度量元分别是 la、rexp 和 fix. 对于 AG-SZZ 和 MA-SZZ 模型, 贡献最显著的前 3 名度量元分别为 la、rexp 和 nuc. 可以看出, B-SZZ、AG-SZZ 和 MA-SZZ 贡献最高的前两名度量元相一致, 与 RA-SZZ 并不一致, 但都包含了 la 度量元.

(3) 通过以上分析发现, 在类重平衡情况下, B-SZZ、AG-SZZ、MA-SZZ 与 RA-SZZ 在局部解释方面不太一致, 其中 la 度量元对模型的预测结果有重要影响且都对预测为有缺陷的贡献大.

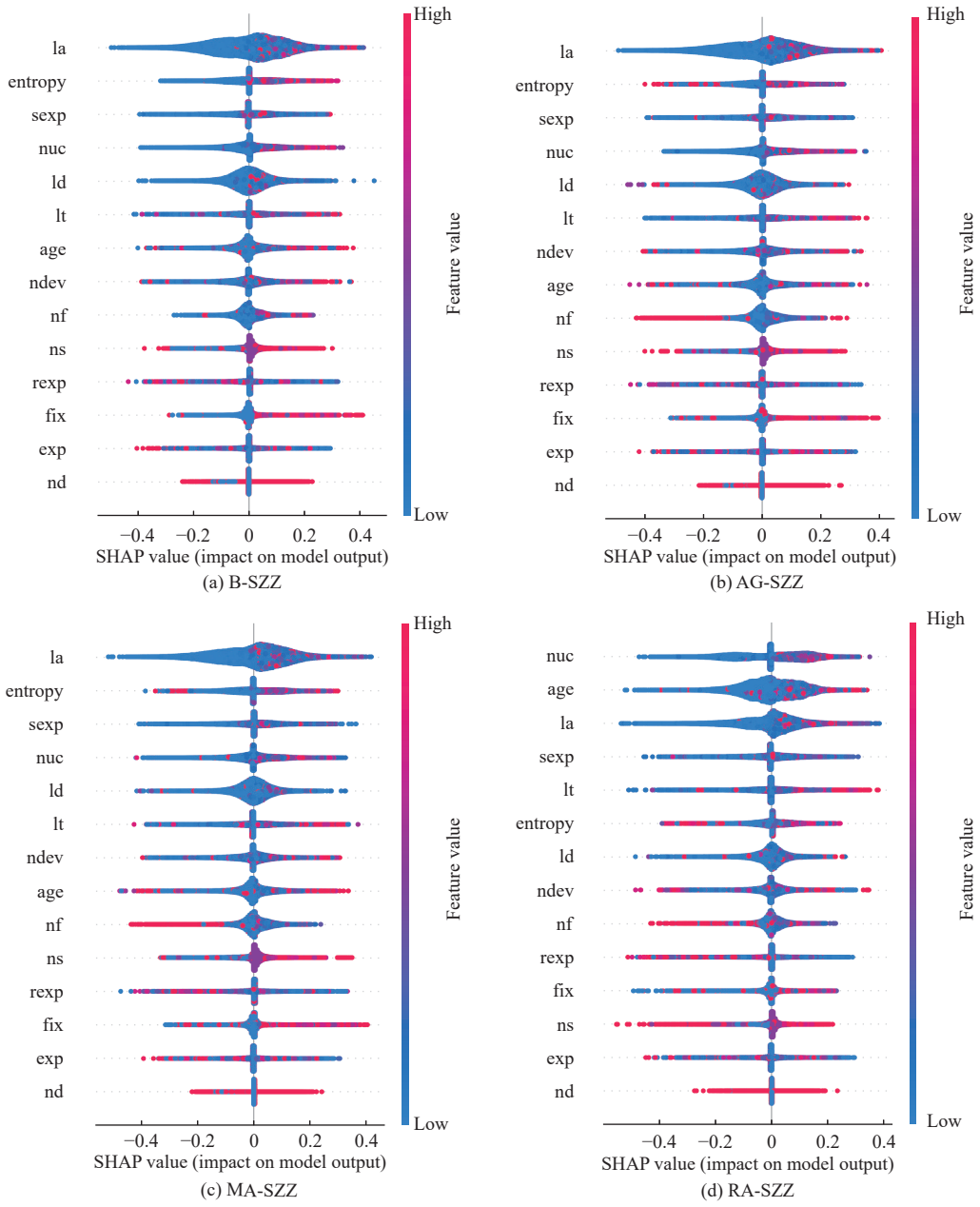


图 8　类重平衡后所有项目的 4 种 SZZ 模型的全局解释

图 10 展示了在类重平衡情况下所有项目中所有变更的 4 种 SZZ 模型的局部解释结果占比. 从图 10 中可以看出, 对于 B-SZZ 模型, 影响程度最大的 3 个度量元为 la (19%)、entropy (10%) 和 sexp (10%), 对于 AG-SZZ 和 MA-SZZ 模型, 影响程度最大的 3 个度量元为 la (18%)、entropy (10%) 和 lt (10%), 而对于 RA-SZZ 模型, 影响程度最大的 3 个度量元是 nuc (19%)、age (14%) 和 la (13%). 这表明, 在所有项目的所有变更中, 局部解释结果与全

局解释的结果相一致, 从而证实了全局解释对局部解释的泛化性.

RQ5: 不同数据采样算法间的性能对比如何?

为了验证不同数据采样算法对移动 APP 即时缺陷预测性能的影响, 本节对比随机欠采样 (random under-sampling, RUS)[16]、随机过采样 (random over-sampling, ROS)[16]、合成少数类过采样 (synthetic minority over-sampling technique, SMOTE)[82]、随机过采样示例 (random over sampling examples, ROSE)[83]这 4 种常用的数据采样技术用于数据重平衡的效果, 并使用 SKESD 统计测试对结果进行对比分析 (相同颜色表示性能相近, 排名越低则表示性能越好). 根据 RQ1 与 RQ2 的结果, 相比于朴素贝叶斯与逻辑回归分类器, 随机森林分类器在 5 个指标上表现最优, 因此本节使用随机森林构建模型.

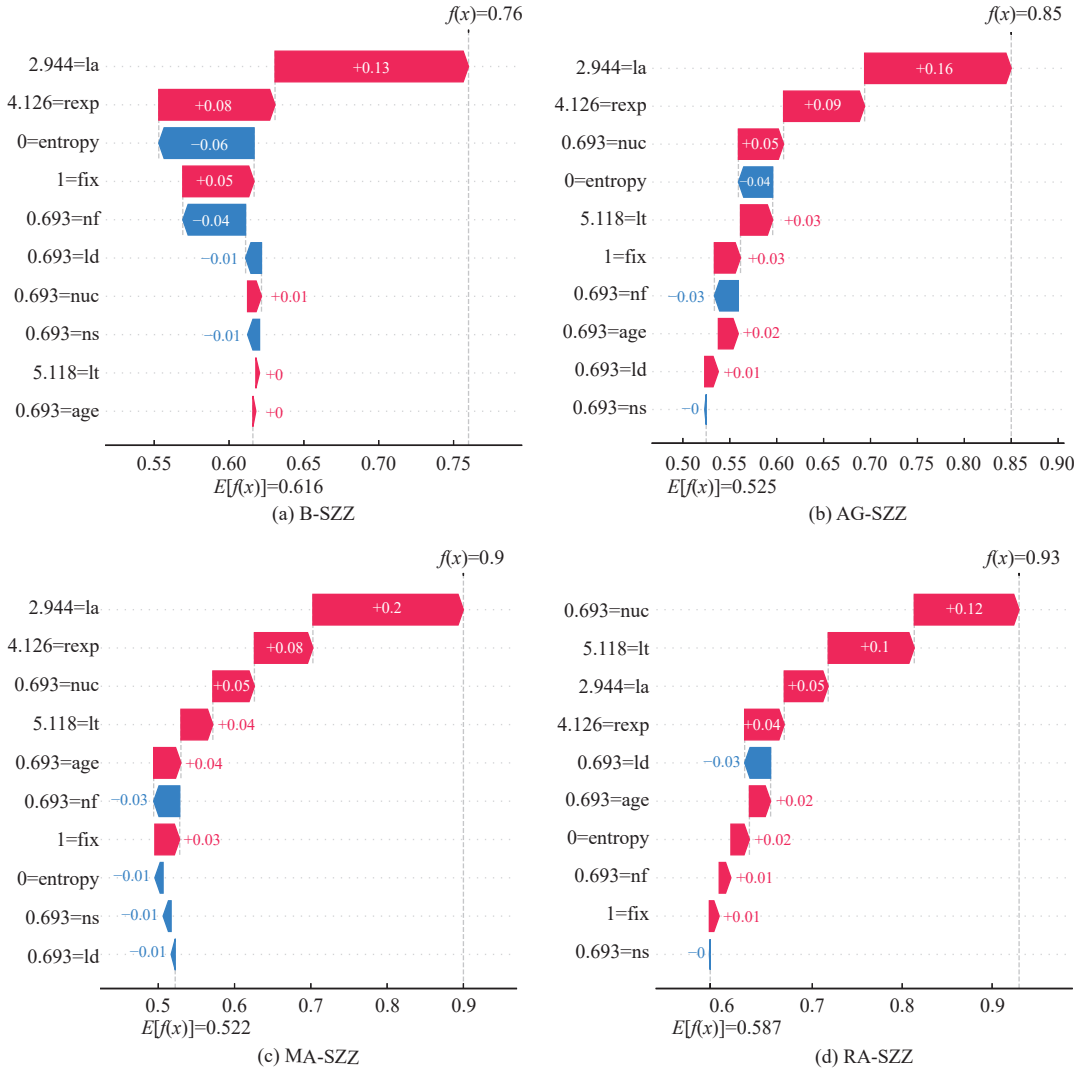


图 9　类重平衡后 AntennaPod 项目上其中一个变更 (eeeff6203) 的局部解释

图 11 分别展示了在 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ (图中使用 B、AG、MA 和 RA 表示) 这 4 种数据采样算法在 *AUC*、*MCC*、*G-mean*、*F-measure*@20% 及 *IFA* 指标上的性能表现. 由于 *IFA* 指标数值间的差值较大, 为便于比较, 本文对其进行了 log 归一化处理. 从图 11 中可以看出, 对于 B-SZZ 模型, SMOTE 在 *AUC*、*MCC*、

IFA 指标上优于其他采样算法. 对于 AG-SZZ 模型, 除 *IFA* 指标之外, SMOTE 优于其他采样算法. 对于 MA-SZZ 模型, SMOTE 均优于其他采样算法. 而对于 RA-SZZ 模型, SMOTE 仅在 *AUC* 和 *F-measure*@20% 指标上优于其他采样算法. 通过以上分析发现, 相比于其他采样算法, SMOTE 总体上表现最优. 因此, 在能够保留原始数据特征的情况下, 本文建议未来的移动 APP 即时缺陷预测研究可使用 SMOTE 采样算法进行数据重平衡.

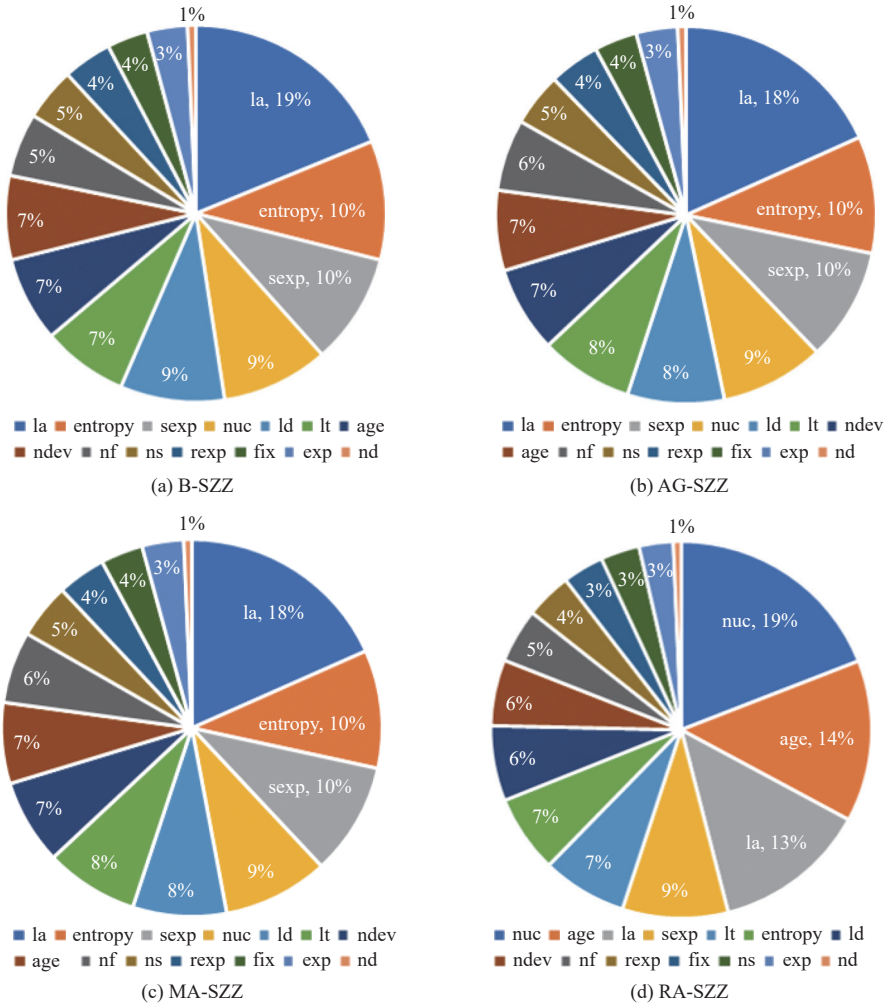


图 10　类重平衡情况下所有项目中所有变更的 4 种 SZZ 模型的局部解释

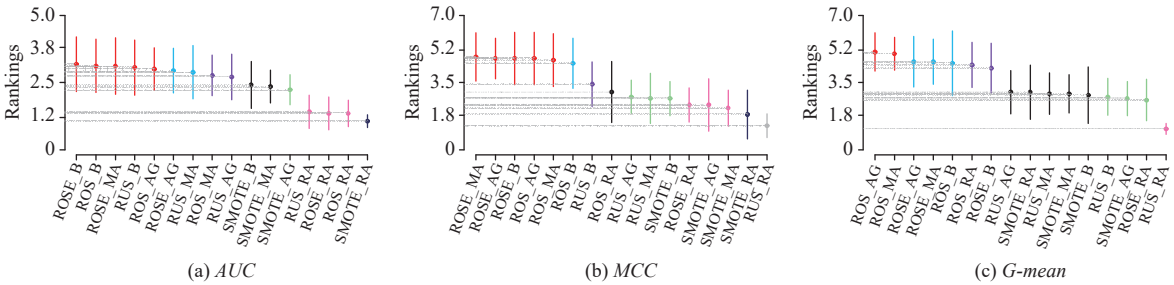
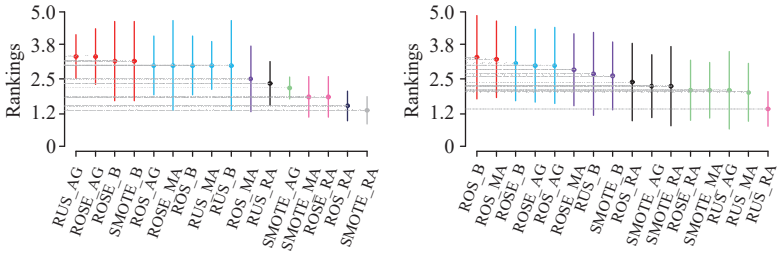


图 11　4 种采样算法的对比结果



(d) *F-measure*@20% (e) *IFA*

图 11 4 种采样算法的对比结果 (续)

5 讨 论

5.1 本文与文献 [5] 研究工作的结论对比

为了比较本文与 Fan 等人[5]研究工作的结论是否具有一致性, 特从以下几个方面进行详细分析和讨论.

(1) 数据标注质量方面

本文: 数据的标注质量符合 SZZ 变体之间的递进关系.

文献 [5]: B-SZZ、MA-SZZ 存在递进关系, 但 AG-SZZ 在某些项目上比 B-SZZ 产生更多的误标数量.

分析: AG-SZZ 算法使用注释分析和代码格式检测的方法, 来过滤掉不相关的代码行, 如空行、注释行和代码风格的修改, 基于图的方法, 来构建代码行之间的依赖关系, 从而更准确地追溯缺陷导致的变更. 然而, 如果没有及时删除无用的代码, 没有写清楚的提交信息, 没有及时合并分支等, 造成项目的代码复杂度较高, 代码行之间的依赖关系较多, 会增加 AG-SZZ 算法的计算开销. 这些情况会导致 AG-SZZ 算法的追溯效果受到干扰, 从而影响 AG-SZZ 算法的识别效果, 最终导致在某些项目上出现 AG-SZZ 比 B-SZZ 误标数量更多的情况.

(2) 模型的性能方面

本文: B-SZZ、AG-SZZ 和 MA-SZZ 错误标注的变更会造成 *AUC*、*MCC* 得分不同程度的下降, 但不会造成 *G-mean* 得分下降.

文献 [5]: 在类不平衡情况下, B-SZZ 和 MA-SZZ 的错误标注更改不会导致 *AUC* 方面的显著性能降低, 而 AG-SZZ 会造成显著的性能下降. 在类平衡情况下, B-SZZ 和 MA-SZZ 的错误标注变化不会导致 *AUC*, *F*1 得分和 *G-mean* 得分方面的显著性能降低. AG-SZZ 错误标注会造成显著的性能下降.

分析: 本文的结论在类不平衡和类平衡情况下一致, 结论为 B-SZZ、AG-SZZ、MA-SZZ 都会造成 *AUC* 得分不同程度的下降, 但不会造成 *G-mean* 的得分下降. 文献 [5] 发现只有 AG-SZZ 会显著降低 *AUC*、*F*1 和 *G-mean* 得分. 但总体来说, 不论是本文还是 Fan 等人[5]的研究, RA-SZZ 的性能是仍然是最优的, 这点在本文和 Fan 等人的工作中都有所体现.

(3) 模型的解释方面

本文: 在模型解释方面, 不同 SZZ 算法会影响预测过程中贡献程度排名前 3 的度量元, 并且 la (变更增加的代码行数) 度量元对预测结果有重要影响.

文献 [5]: SZZ 错误标注的变更会不会影响最重要的度量元 (即 nf, 文件数量), 然而在随机森林分类器上, SZZ 错误标注的变更会影响第 2、3 名最重要的度量元.

分析: 由于本文的实验数据、相关设置等与文献 [5] 的不同, 度量元在预测过程所产生的效果可能不同, 因此度量元对模型的解释方面有着不同影响, 但总体来说, 在使用随机森林作为分类器时, SZZ 错误标注的变更会影响第 2、3 名最重要的度量元.

5.2 经验发现

通过深入比较 4 种 SZZ 算法对移动 APP 即时缺陷预测性能与解释的实证研究, 本文获得了一些经验发现, 对

未来的即时缺陷预测研究具有一定的指导意义.

在数据预处理方面: 根据 RQ5 的实验结果, SMOTE 数据采样算法在 5 个评价指标上总体性能优于 RUS、ROS 及 ROSE 采样算法. 因此本文建议未来的移动 APP 即时缺陷预测研究可使用 SMOTE 采样算法对缺陷数据进行重平衡.

在模型构建方面: 根据 RQ1 与 RQ2 的实验结果, 相比朴素贝叶斯与逻辑回归分类器, 随机森林分类器能够获得较优的 *AUC*、*MCC*、*G-mean*、*F-measure*@20% 和 *IFA* 性能. 因此, 未来的移动 APP 即时缺陷预测研究可使用随机森林分类器构建模型.

在模型性能方面: 根据 RQ1 与 RQ2 的实验结果, 相比 RA-SZZ 算法, B-SZZ 算法在大部分指标上的得分显著下降, 而 AG-SZZ 和 MA-SZZ 算法仅会造成少数指标得分下降, 大多数情况下与 RA-SZZ 算法的性能基本一致或差距较小. 总的来说, RA-SZZ 算法具有较好的数据标注质量.

在模型解释方面: 根据 RQ3 与 RQ4 的实验结果, 不同 SZZ 算法对预测结果的解释会一定程度的影响. 特别是当预测结果为有缺陷时, 度量元 nuc 和 la 与结果高度相关. 因此, 当开发人员提交的变更中涉及历史修改次数较多的文件或新增了大量代码行时, 有必要重点审查该变更是否引入了缺陷.

此外, 类是否平衡对模型的性能有影响, 但对模型的解释性影响较小, 因为分类模型的性能往往依赖于数据的分布和数量, 而模型的解释通常关注不同度量元的依赖关系, 解释性方法如 SHAP、LIME 等通常关注单个样本或特定预测的解释, 而不是整体的类别不平衡问题. 度量元 nuc、la 和 sexp 往往伴随多个模块的修改, 从而影响模块之间的耦合程度和依赖性, 涉及的模块越多则越可能出现缺陷. 因此无论是类平衡与否, 这些度量元均表现出了较高的排名.

在 SZZ 算法方面: 在今后构建移动 APP 即时缺陷预测模型时, 应避免使用 B-SZZ 算法进行数据标注. 由于 RA-SZZ 算法所标注的数据具有较低的噪音水平, 又可获得较好的预测性能与解释, 因此本文推荐使用 RA-SZZ 算法进行数据标注, 特别是在计算资源充足的情况下.

5.3 有效性威胁

构建有效性: 在软件度量元的提取阶段, 本文使用开源工具 PyDriller[71]来提取 Git 仓库中的代码变更数据, PyDriller 已被广泛应用于软件缺陷预测方面的研究, 可靠性较强. 在数据标注阶段, 本文使用 Rosa 等人[84]提供的 4 种 SZZ 实现算法进行数据标注. 在类平衡场景下, 本文对比了随机欠采样、随机过采样、ROSE 和 SMOTE 这 4 种采样算法的性能差异, 发现 SMOTE 在总体上具有最优的性能. 最后采用 *AUC*、*MCC*、*G-mean* 以及工作量感知指标 *F-measure*@20%、*IFA* 来评价模型, 这些指标被广泛应用于已有的研究中[37,41,85,86]. 此外, 本文应用了 SKESD 统计测试方法来保证实验结果的可靠性. 由于 SZZ 算法依赖于修复缺陷的变更, 对于修复缺陷或关联 issue 的变更 (例如#123-bug fix for android 12), 如果开发人员对修复变更的描述不正确, 那么可能会带来噪音. 其次, 如果提交信息中不包含缺陷关键字, 仅包含 issue ID (例如 #123 xxx), 则需要进一步判断该 issue 是否属于缺陷类型. 本文对所有缺陷类型的 issue 进行了人工筛查, 发现这些 issue 对本文数据标注的影响可以忽略不计. 然而, 在未来的数据挖掘阶段的实验中, 应当考虑到这些 issue. 最后, 本文使用 RA-SZZ 作为基准算法, 但该算法仍然可能存在错误标记的变更. 为了尽量减少 RA-SZZ 算法对本实验的影响, 本文将开发过程较为规范的大型 APP 项目作为实验对象, 以降低开发人员提交的不规范的缺陷修复变更对 SZZ 标注算法的影响.

内部有效性: 首先, 为验证本文所使用的 4 种 SZZ 算法所标记的数据是否准确, 本文将 B-SZZ 所标注的数据集与 Commit guru[87]所生成的数据集进行了对比, 发现数据基本一致. 其次, 对于每一个项目, 确保了各对比模型使用相同的数据进行 SKESD 统计测试. 此外, 本文使用 Python 库中的 SHAP 算法包进行模型解释, 并且在局部解释方面, 确保了类不平衡与类重平衡情况下对相同的变更进行解释.

外部有效性: 本文使用的 17 个移动 APP 项目均来源于 GitHub 社区, 其中 12 个项目在先前的研究[44]已被使用过, 其余 5 个项目来自 GitHub 社区中开发者较多, 项目较为完善, 且发布于应用市场的大型移动项目, 一方面, 这些项目的开发流程较为规范, 不仅保证了数据集的可靠性, 而且能够保证 SZZ 算法标注出尽可能多的缺陷. 另

一方面代码提交数量较多, 以确保实验中有足够的实例.

6 总结与展望

本文选取 GitHub 库中 17 个大型开源移动 APP 项目, 抽取了 Kamei 等人[9]提出的 14 个变更度量元, 并使用 B-SZZ、AG-SZZ、MA-SZZ 和 RA-SZZ 算法进行数据标注, 构造了 17 个移动 APP 数据集. 为探究不同 SZZ 算法错误标注的变更对移动 APP 即时缺陷预测性能与解释的影响, 本文基于 4 种 SZZ 算法标注的数据集利用随机森林、朴素贝叶斯和逻辑回归分类器分别建立即时缺陷预测模型, 采用 *AUC*、*MCC*、*G-mean*、*F-measure*@20% 及 *IFA* 这 5 个指标进行评估, 并使用 SKESD 和 SHAP 算法对结果进行排序比较与解释分析. 在模型性能方面, B-SZZ、AG-SZZ、MA-SZZ 算法会在不同程度上导致即时缺陷预测模型性能下降. 在模型解释方面, B-SZZ、AG-SZZ、MA-SZZ 算法会影响模型预测过程中最重要的前 3 名度量元.

在未来的即时缺陷预测研究中, 本文推荐使用 RA-SZZ 算法对数据进行标注以构建缺陷预测数据集. 此外, 本文的预测粒度为变更级, 相比于代码行, 粒度仍然较大, 因此后续工作拟研究基于代码行级的移动 APP 即时缺陷预测, 这将大幅度缩小开发人员需要人工审查的代码范围, 有助于进一步提升软件开发效率.

References:

[1] Chen X, Gu Q, Liu WS, Liu SL, Ni C. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016, 27(1): 1–25 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[2] Gong LN, Jiang SJ, Jiang L. Research progress of software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2019, 30(10): 3090–3114 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5790.htm [doi: 10.13328/j.cnki.jos.005790]

[3] Cai L, Fan YR, Yan M, Xia X. Just-in-time software defect prediction: Literature review. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1288–1307 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5713.htm [doi: 10.13328/j.cnki.jos.05713]

[4] Li ZQ, Jing XY, Zhu XK. Progress on approaches to software defect prediction. IET Software, 2018, 12(3): 161–175. [doi: 10.1049/iet-sen.2017.0148]

[5] Fan YR, Xia X, Da Costa DA, Lo D, Hassan AE, Li SP. The impact of mislabeled changes by SZZ on just-in-time defect prediction. IEEE Trans. on Software Engineering, 2021, 47(8): 1559–1586. [doi: 10.1109/TSE.2019.2929761]

[6] Kamei Y, Fukushima T, McIntosh S, Yamashita K, Ubayashi N, Hassan AE. Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering, 2016, 21(5): 2072–2106. [doi: 10.1007/s10664-015-9400-x]

[7] Ge J, Yu HQ, Fan GS, Tang JH, Huang ZJ. Just-in-time defect prediction for intelligent computing frameworks. Ruan Jian Xue Bao/Journal of Software, 2023, 34(9): 3966–3980 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6874.htm [doi: 10.13328/j.cnki.jos.006874]

[8] Jiang T, Tan L, Kim S. Personalized defect prediction. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Silicon Valley: IEEE, 2013. 279–289. [doi: 10.1109/ASE.2013.6693087]

[9] Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N. A large-scale empirical study of just-in-time quality assurance. IEEE Trans. on Software Engineering, 2013, 39(6): 757–773. [doi: 10.1109/TSE.2012.70]

[10] Li ZQ, Zhang HY, Jing XY, Xie JY, Guo M, Ren J. DSSDPP: Data selection and sampling based domain programming predictor for cross-project defect prediction. IEEE Trans. on Software Engineering, 2023, 49(4): 1941–1963. [doi: 10.1109/TSE.2022.3204589]

[11] Xia X, Lo D, Pan SJ, Nagappan N, Wang XY. HYDRA: Massively compositional model for cross-project defect prediction. IEEE Trans. on Software Engineering, 2016, 42(10): 977–998. [doi: 10.1109/TSE.2016.2543218]

[12] Śliwerski J, Zimmermann T, Zeller A. When do changes induce fixes? ACM SIGSOFT Software Engineering Notes, 2005, 30(4): 1–5. [doi: 10.1145/1082983.1083147]

[13] Kim S, Zimmermann T, Pan K Jr, Whitehead E. Automatic identification of bug-introducing changes. In: Proc. of the 21st IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2006). Tokyo: IEEE, 2006. 81–90. [doi: 10.1109/ASE.2006.23]

[14] Da Costa DA, McIntosh S, Shang WY, Kulesza U, Coelho R, Hassan AE. A framework for evaluating the results of the SZZ approach for identifying bug-introducing changes. IEEE Trans. on Software Engineering, 2017, 43(7): 641–657. [doi: 10.1109/TSE.2016.2616306]

[15] Neto EC, Da Costa DA, Kulesza U. The impact of refactoring changes on the SZZ algorithm: An empirical study. In: Proc. of the 25th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Campobasso: IEEE, 2018. 380–390. [doi: 10.1109/SANER.2018.8330225]

[16] Tantithamthavorn C, Hassan AE, Matsumoto K. The impact of class rebalancing techniques on the performance and interpretation of

defect prediction models. IEEE Trans. on Software Engineering, 2020, 46(11): 1200–1219. [doi: 10.1109/TSE.2018.2876537]

[17] Catolino G, Di Nucci D, Ferrucci F. Cross-project just-in-time bug prediction for mobile APPs: An empirical assessment. In: Proc. of the 6th IEEE/ACM Int'l Conf. on Mobile Software Engineering and Systems (MOBILESoft). Montreal: IEEE, 2019: 99–110. [doi: 10.1109/MOBILESoft.2019.00023]

[18] Bird C, Bachmann A, Aune E, Duffy J, Bernstrin A, Filkov V, Devanbu P. Fair and balanced? Bias in bug-fix datasets. In: Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. Amsterdam: ACM, 2009. 121–130. [doi: 10.1145/1595696.1595716]

[19] Bachmann A, Bird C, Rahman F, Devanbu P, Bernstein A. The missing links: Bugs and bug-fix commits. In: Proc. of the 18th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Santa Fe: ACM, 2010. 97–106. [doi: 10.1145/1882291.1882308]

[20] Kim S, Zhang HY, Wu RX, Gong L. Dealing with noise in defect prediction. In: Proc. of the 33rd Int'l Conf. on Software Engineering. Honolulu: ACM, 2011. 481–490. [doi: 10.1145/1985793.1985859]

[21] Antoniol G, Ayari K, Di Penta M, Khomh F, Guéhéneuc YG. Is it a bug or an enhancement? A text-based approach to classify change requests. In: Proc. of the 2008 Conf. of the Center for Advanced Studies on Collaborative Research: Meeting of Minds. Ontario: ACM, 2008. 304–318. [doi: 10.1145/1463788.1463819]

[22] Kochhar PS, Thung F, Lo D. Automatic fine-grained issue report reclassification. In: Proc. of the 19th Int'l Conf. on Engineering of Complex Computer Systems. Tianjin: IEEE, 2014. 126–135. [doi: 10.1109/ICECCS.2014.25]

[23] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). San Francisco: IEEE, 2013. 392–401. [doi: 10.1109/ICSE.2013.6606585]

[24] Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, Mastsumoto K. The impact of mislabelling on the performance and interpretation of defect prediction models. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 812–823. [doi: 10.1109/ICSE.2015.93]

[25] Herzig K, Just S, Zeller A. The impact of tangled code changes on defect prediction models. Empirical Software Engineering, 2016, 21(2): 303–336. [doi: 10.1007/s10664-015-9376-6]

[26] Zimmermann T, Kim S, Zeller A, Whitehead EJ. Mining version archives for co-changed lines. In: Proc. of the 2006 Int'l Workshop on Mining Software Repositories. Shanghai: ACM, 2006. 72–75. [doi: 10.1145/1137983.1138001]

[27] Shivaji S, Whitehead EJ, Akella R, Kim S. Reducing features to improve code change-based bug prediction. IEEE Trans. on Software Engineering, 2013, 39(4): 552–569. [doi: 10.1109/TSE.2012.43]

[28] Koru AG, Zhang DS, El Emam K, Liu HF. An investigation into the functional form of the size-defect relationship for software modules. IEEE Trans. on Software Engineering, 2009, 35(2): 293–304. [doi: 10.1109/TSE.2008.90]

[29] Mockus A, Weiss DM. Predicting risk of software changes. Bell Labs Technical Journal, 2002, 5(2): 169–180. [doi: 10.1002/bltj.2229]

[30] Kim S, Whitehead EJ, Zhang Y. Classifying software changes: Clean or buggy? IEEE Trans. on Software Engineering, 2008, 34(2): 181–196. [doi: 10.1109/TSE.2007.70773]

[31] Shihab E, Hassan AE, Adams B, Jiang ZM. An industrial study on the risk of software changes. In: Proc. of the 20th ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering. Cary: ACM, 2012. 62. [doi: 10.1145/2393596.2393670]

[32] Arisholm E, Briand LC, Fuglerud M. Data mining techniques for building fault-proneness models in telecom Java software. In: Proc. of the 18th IEEE Int'l Symp. on Software Reliability (ISSRE 2007). Trollhattan: IEEE, 2007. 215–224. [doi: 10.1109/ISSRE.2007.22]

[33] Mende T, Koschke R. Revisiting the evaluation of defect prediction models. In: Proc. of the 5th Int'l Conf. on Predictor Models in Software Engineering. Vancouver: ACM, 2009. 7. [doi: 10.1145/1540438.1540448]

[34] Yang YB, Zhou YM, Liu JP, Zhao YY, Lu HM, Xu L, Xu BW, Leung H. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Seattle: ACM, 2016. 157–168. [doi: 10.1145/2950290.2950353]

[35] Liu JP, Zhou YM, Yang YB, Lu HM, Xu BW. Code churn: A neglected metric in effort-aware just-in-time defect prediction. In: Proc. of the 2017 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement (ESEM). Toronto: IEEE, 2017. 11–19. [doi: 10.1109/ESEM.2017.8]

[36] Fu W, Menzies T. Revisiting unsupervised learning for defect prediction. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 72–83. [doi: 10.1145/3106237.3106257]

[37] Huang Q, Xia X, Lo D. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. Empirical Software Engineering, 2019, 24(5): 2823–2862. [doi: 10.1007/s10664-018-9961-2]

[38] Chen X, Zhao YQ, Wang QP, Yuan ZD. MULTI: Multi-objective effort-aware just-in-time software defect prediction. Information and Software Technology, 2018, 93: 1–13. [doi: 10.1016/j.infsof.2017.08.004]

[39] Li WW, Zhang WZ, Jia XY, Huang ZQ. Effort-aware semi-supervised just-in-time defect prediction. Information and Software

Technology, 2020, 126: 106364. [doi: 10.1016/j.infsof.2020.106364]

[40] Yan M, Xia X, Fan YR, Lo D, Hassan AE, Zhang XD. Effort-aware just-in-time defect identification in practice: A case study at Alibaba. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 1308–1319. [doi: 10.1145/3368089.3417048]

[41] Ni C, Xia X, Lo D, Yang XH, Hassan AE. Just-in-time defect prediction on JavaScript projects: A replication study. ACM Trans. on Software Engineering and Methodology (TOSEM), 2022, 31(4): 76. [doi: 10.1145/3508479]

[42] Xu Z, Zhao KS, Zhang T, Fu CL, Yan M, Xie ZW, Zhang XH, Catolino G. Effort-aware just-in-time bug prediction for mobile APPs via cross-triplet deep feature embedding. IEEE Trans. on Reliability, 2022, 71(1): 204–220. [doi: 10.1109/TR.2021.3066170]

[43] Zhao KS, Xu Z, Zhang TZ, Tang YT, Yan M. Simplified deep forest model based just-in-time defect prediction for Android mobile APPs. IEEE Trans. on Reliability, 2021, 70(2): 848–859. [doi: 10.1109/TR.2021.3060937]

[44] Zhao KS, Xu Z, Yan M, Xue L, Li W, Catolino G. A compositional model for effort-aware just-in-time defect prediction on Android APPs. IET Software, 2022, 16(3): 259–278. [doi: 10.1049/sfw2.12040]

[45] Yang XL, Lo D, Xia X, Zhang Y, Sun JL. Deep learning for just-in-time defect prediction. In: Proc. of the 2015 IEEE Int'l Conf. on Software Quality, Reliability and Security. Vancouver: IEEE, 2015. 17–26. [doi: 10.1109/QRS.2015.14]

[46] Hoang T, Dam HK, Kamei Y, Lo D, Ubayashi N. DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction. In: Proc. of the 2019 IEEE/ACM 16th Int'l Conf. on Mining Software Repositories (MSR). Montreal: IEEE, 2019. 34–45. [doi: 10.1109/MSR.2019.00016]

[47] Hoang T, Kang HJ, Lo D, Lawall J. CC2Vec: Distributed representations of code changes. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 518–529. [doi: 10.1145/3377811.3380361]

[48] Zeng ZR, Zhang YQ, Zhang HT, Zhang LM. Deep just-in-time defect prediction: How far are we? In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 427–438. [doi: 10.1145/3460319.3464819]

[49] Pornprasit C, Tantithamthavorn CK. JITLine: A simpler, better, faster, finer-grained just-in-time defect prediction. In: Proc. of the 18th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Madrid: IEEE, 2021. 369–379. [doi: 10.1109/MSR52588.2021.00049]

[50] Pornprasit C, Tantithamthavorn C, Jiarpakdee J, Fu M, Thongtanunam P. PyExplainer: Explaining the predictions of just-in-time defect models. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 407–418. [doi: 10.1109/ASE51524.2021.9678763]

[51] Lin DY, Tantithamthavorn C, Hassan AE. The impact of data merging on the interpretation of cross-project just-in-time defect models. IEEE Trans. on Software Engineering, 2022, 48(8): 2969–2986. [doi: 10.1109/TSE.2021.3073920]

[52] Zheng W, Shen TR, Chen X. Just-in-time defect prediction technology based on interpretability technology. In: Proc. of the 8th Int'l Conf. on Dependable Systems and Their Applications (DSA). Yinchuan: IEEE, 2021. 78–89. [doi: 10.1109/DSA52907.2021.00017]

[53] Chen LQ, Wang C, Song SL. Just-in-time software defect prediction model and its interpretability research. Journal of Chinese Computer Systems, 2022, 43(4): 865–871 (in Chinese with English abstract). [doi: 10.20009/j.cnki.21-1106/TP.2020-1075]

[54] Yang XG, Yu HQ, Fan GS, Huang ZJ, Yang K, Zhou ZY. An empirical study of model-agnostic interpretation technique for just-in-time software defect prediction. In: Proc. of the 17th EAI Int'l Conf. on Collaborative Computing: Networking, Applications and Worksharing. Springer, 2021. 420–438. [doi: 10.1007/978-3-030-92635-9_25]

[55] Pascarella L, Palomba F, Bacchelli A. Fine-grained just-in-time defect prediction. Journal of Systems and Software, 2019, 150: 22–36. [doi: 10.1016/j.jss.2018.12.001]

[56] Cabral GG, Minku LL, Shihab E, Mujahid S. Class imbalance evolution and verification latency in just-in-time software defect prediction. In Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 666–676. [doi: 10.1109/ICSE.2019.00076]

[57] Zhao YH, Damevski K, Chen H. A systematic survey of just-in-time software defect prediction. ACM Computing Surveys, 2023, 55(10): 201. [doi: 10.1145/3567550]

[58] Scandariato R, Walden J. Predicting vulnerable classes in an Android application. In: Proc. of the 4th Int'l Workshop on Security Measurements and Metrics. Lund: ACM, 2012. 11–16. [doi: 10.1145/2372225.2372231]

[59] Kaur A, Kaur K, Kaur H. Application of machine learning on process metrics for defect prediction in mobile application. In: Proc. of the 3rd Int'l Conf. on Information Systems Design and Intelligent Applications. Springer, 2016. 81–98. [doi: 10.1007/978-81-322-2755-7_10]

[60] Malhotra R. An empirical framework for defect prediction using machine learning techniques with Android software. Applied Soft Computing, 2016, 49: 1034–1050. [doi: 10.1016/j.asoc.2016.04.032]

[61] Ricky MY, Purnomo F, Yulianto B. Mobile application software defect prediction. In: Proc. of the 2016 IEEE Symp. on Service-oriented System Engineering (SOSE). Oxford: IEEE, 2016. 307–313. [doi: 10.1109/SOSE.2016.25]

[62] Fan YQ, Cao XY, Xu J, Xu SH, Yang HJ. High-frequency keywords to predict defects for Android applications. In: Proc. of the 42nd IEEE Annual Computer Software and Applications Conf. (COMPSAC). Tokyo: IEEE, 2018. 442–447. [doi: 10.1109/COMPSAC.2018.10273]

[63] Cheng T, Zhao KS, Sun S, Mateen M, Wen JH. Effort-aware cross-project just-in-time defect prediction framework for mobile APPs. Frontiers of Computer Science, 2022, 16(6): 166207. [doi: 10.1007/s11704-021-1013-5]

[64] Hu XY, Chen X, Xia HL, Gu YF. Interpretable method of just-in-time defect prediction model for mobile APP. Application Research of Computers, 2022, 39(7): 2104–2108 (in Chinese with English abstract). [doi: 10.19734/j.issn.1001-3695.2021.12.0679]

[65] McIntosh S, Kamei Y. [Journal First] Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering. Gothenburg: IEEE, 2018. 560. [doi: 10.1145/3180155.3182514]

[66] Hassan AE. Predicting faults using the complexity of code changes. In: Proc. of the 31st IEEE Int'l Conf. on Software Engineering. Vancouver: IEEE, 2009. 78–88. [doi: 10.1109/ICSE.2009.5070510]

[67] Moser R, Pedrycz W, Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proc. of the 30th Int'l Conf. on Software Engineering (ICSE). Leipzig: ACM, 2008. 181–190. [doi: 10.1145/1368088.1368114]

[68] Guo PJ, Zimmermann T, Nagappan N, Murphy B. Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering (ICSE). Cape Town: ACM, 2010. 495–504. [doi: 10.1145/1806799.1806871]

[69] Purushothaman R, Perry DE. Toward understanding the rhetoric of small source code changes. IEEE Trans. on Software Engineering, 2005, 31(6): 511–526. [doi: 10.1109/TSE.2005.74]

[70] Matsumoto S, Kamei Y, Monden A, Matsumoto KI, Nakamura M. An analysis of developer metrics for fault prediction. In: Proc. of the 6th Int'l Conf. on Predictive Models in Software Engineering. Timişoara: ACM, 2010. 18. [doi: 10.1145/1868328.1868356]

[71] Spadini D, Aniche M, Bacchelli A. PyDriller: Python framework for mining software repositories. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 908–911. [doi: 10.1145/3236024.3264598]

[72] Tantithamthavorn C, Hassan AE. An experience report on defect modelling in practice: Pitfalls and challenges. In: Proc. of the 40th Int'l Conf. on Software Engineering: Software Engineering in Practice. Gothenburg: ACM, 2018. 286–295. [doi: 10.1145/3183519.3183547]

[73] Jiarpakdee J, Tantithamthavorn C, Treude C. AutoSpearman: Automatically mitigating correlated software metrics for interpreting defect models. In Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Madrid: IEEE, 2018. 92–103. [doi: 10.1109/ICSME.2018.00018]

[74] Tan M, Tan L, Dara S, Mayeux C. Online defect prediction for imbalanced data. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 99–108. [doi: 10.1109/ICSE.2015.139]

[75] Liaw A, Wiener M. Classification and regression by randomForest. R News, 2002, 2(3): 18–22.

[76] Freund Y, Schapire RE. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 1997, 55(1): 119–139. [doi: 10.1006/jcss.1997.1504]

[77] Liu XT, Guo ZQ, Liu SR, Zhang P, Lu HM, Zhou YM. Comparing software defect prediction models: Research problem, progress, and challenges. Ruan Jian Xue Bao/Journal of Software, 2023, 34(2): 582–624 (in Chinese with English abstract). [doi: 10.13328/j.cnki.jos.006714]

[78] Li ZQ, Jing XY, Zhu XK, Zhang HY, Xu BW, Ying S. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. IEEE Trans. on Software Engineering, 2019, 45(4): 391–411. [doi: 10.1109/TSE.2017.2780222]

[79] Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. on Software Engineering, 2017, 43(1): 1–18. [doi: 10.1109/TSE.2016.2584050]

[80] Li ZQ, Jing XY, Zhu XK, Zhang HY, Xu BW, Ying S. Heterogeneous defect prediction with two-stage ensemble learning. Automated Software Engineering, 2019, 26(3): 599–651. [doi: 10.1007/s10515-019-00259-1]

[81] Lundberg SM, Lee SI. A unified approach to interpreting model predictions. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 4768–4777.

[82] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 2002, 16: 321–357. [doi: 10.1613/jair.953]

[83] Lunardon N, Menardi G, Torelli N. ROSE: A package for binary imbalanced learning. The R Journal, 2014, 6(1): 79–89. [doi: 10.32614/RJ-2014-008]

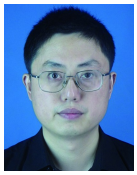
[84] Rosa G, Pascarella L, Scalabrino S, Tufano R, Bavota G, Lanza M, Oliveto R. Evaluating SZZ implementations through a developer-informed oracle. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 436–447. [doi: 10.

1109/ICSE43902.2021.00049]
[85] Li ZQ, Jing XY, Wu F, Zhu XK, Xu BW, Ying S. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. Automated Software Engineering, 2018, 25(2): 201–245. [doi: 10.1007/s10515-017-0220-7]
[86] Li ZQ, Niu JW, Jing XY, Yu WY, Qi C. Cross-project defect prediction via landmark selection-based kernelized discriminant subspace alignment. IEEE Trans. on Reliability, 2021, 70(3): 996–1013. [doi: 10.1109/TR.2021.3074660]
[87] Rosen C, Grawi B, Shihab E. Commit guru: Analytics and risk prediction of software commits. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 966–969. [doi: 10.1145/2786805.2803183]

附中文参考文献:

[1] 陈翔, 顾庆, 刘望舒, 刘树龙, 倪超. 静态软件缺陷预测方法研究. 软件学报, 2016, 27(1): 1–25. http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]
[2] 宫丽娜, 姜淑娟, 姜丽. 软件缺陷预测技术研究进展. 软件学报, 2019, 30(10): 3090–3114. http://www.jos.org.cn/1000-9825/5790.htm [doi: 10.13328/j.cnki.jos.005790]
[3] 蔡亮, 范元瑞, 鄢萌, 夏鑫. 即时软件缺陷预测研究进展. 软件学报, 2019, 30(5): 1288–1307. http://www.jos.org.cn/1000-9825/5713.htm [doi: 10.13328/j.cnki.jos.05713]
[7] 葛建, 虞慧群, 范贵生, 唐锏浩, 黄子杰. 面向智能计算框架的即时缺陷预测. 软件学报, 2023, 34(9): 3966–3980. http://www.jos.org.cn/1000-9825/6874.htm [doi: 10.13328/j.cnki.jos.006874]
[53] 陈丽琼, 王璨, 宋士龙. 一种即时软件缺陷预测模型及其可解释性研究. 小型微型计算机系统, 2022, 43(4): 865–871. [doi: 10.20009/j.cnki.21-1106/TP.2020-1075]
[64] 胡新宇, 陈翔, 夏鸿峻, 顾亚锋. 移动 APP 即时缺陷预测模型的可解释性方法. 计算机应用研究, 2022, 39(7): 2104–2108. [doi: 10.19734/j.issn.1001-3695.2021.12.0679]
[77] 刘旭同, 郭肇强, 刘释然, 张鹏, 卢红敏, 周毓明. 软件缺陷预测模型间的比较实验: 问题、进展与挑战. 软件学报, 2023, 34(2): 582–624. [doi: 10.13328/j.cnki.jos.006714]

李志强(1987－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为智能化软件工程, 软件缺陷预测, 软件数据挖掘.



荆晓远(1971－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为模式识别, 机器学习, 软件工程, 通用人工智能, 信息安全.

马睿(1998－), 男, 硕士, CCF 学生会员, 主要研究领域为软件缺陷预测, 软件仓库挖掘.

任杰(1988－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为程序优化, 移动计算.

张洪宇(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化软件开发和维护, 软件数据挖掘.

刘金会(1989－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为密码算法设计与分析, 数据安全, 隐私计算, 区块链安全.