软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(9):3966−3980 [doi: 10.13328/j.cnki.jos.006874]

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

面向智能计算框架的即时缺陷预测*

葛 建, 虞慧群, 范贵生, 唐铜浩, 黄子杰

(华东理工大学 计算机科学与工程系, 上海 200237)

通信作者: 虞慧群, E-mail: yhq@ecust.edu.cn; 范贵生, E-mail: gsfan@ecust.edu.cn; 黄子杰, E-mail: hzj@mail.ecust.edu.cn

摘 要: 作为人工智能工程化的实现工具, 智能计算框架已在近年来被广泛应用, 其可靠性对于人工智能的有效实现至关重要. 然而, 智能计算框架的可靠性保障具有挑战性, 一方面, 智能计算框架代码迭代迅速、测试困难; 另一方面, 与传统软件不同, 智能计算框架涉及大量张量计算, 其代码规范缺乏软件工程理论指导. 为了解决这一问题, 现有的工作主要使用模糊测试手段实现缺陷定位, 然而, 这类方法只能实现特定类型缺陷的精准定位, 却难以即时地在开发过程中引导开发者关注软件质量. 因此, 将国内外常见的智能计算框架 (TensorFlow, 百度飞桨等) 作为研究对象, 选取多种变更特征构建数据集, 在代码提交级别对智能计算框架进行即时缺陷预测. 另外, 在此基础上使用 LDA 主题建模技术挖掘代码和代码提交信息作为新的特征, 并使用随机森林进行预测. 结果发现 AUC-ROC 平均值为 0.77, 且语义信息可以略微提升预测性能. 最后, 使用可解释机器学习方法 SHAP 分析各特征属性对模型预测输出的影响, 发现: (1) 基本特征对于模型的影响符合传统软件开发规律; (2) 代码和提交信息中的语义特征对模型的预测结果有重要影响; (3) 不同系统中的不同特征对模型预测输出的贡献度排序也存在较大差异.

关键词: 智能计算框架; 即时缺陷预测; 可解释人工智能; 实证软件工程

中图法分类号: TP18

中文引用格式: 葛建, 虞慧群, 范贵生, 唐铜浩, 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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/6874.htm

Just-in-time Defect Prediction for Intelligent Computing Frameworks

GE Jian, YU Hui-Qun, FAN Gui-Sheng, TANG Jian-Hao, HUANG Zi-Jie

(School of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China)

Abstract: In recent years, intelligent computing frameworks have been widely applied as implementation tools in artificial intelligence (AI) engineering, and the reliability of the frameworks is the key to AI implementation effectiveness. However, the reliability assurance of intelligent computing frameworks is challenging. On one hand, the code iteration of frameworks is fast, with difficult code testing. On the other hand, unlike traditional software, intelligent computing frameworks involve a large number of tensor calculations, and the code specification lacks the guidance of software engineering theory. To this end, existing research mostly employs fuzzy testing to localize defects. However, such a method can only accurately discover specific fault types, and it is difficult to guide developers and make them focus on software quality during the development process. Therefore, this study takes the popular intelligent computing frameworks (TensorFlow, Baidu PaddlePaddle, etc.) as the research object, selects multiple change features to build datasets, and conducts just-in-time prediction on the defects of the intelligent computing framework at the code submission level. Additionally, LDA is employed to mine codes and code submission information as new features, and then the random forest is adopted for prediction. Results show that the average AUC-ROC is 0.77, and semantic information can slightly improve the prediction performance. Finally, this study leverages an

* 基金项目: 国家自然科学基金 (61772200); 上海市自然科学基金 (21ZR1416300)
本文由“AI 软件系统工程化技术与规范”专题特约编辑张贺教授、夏鑫博士、蒋振鸣副教授、祝立明教授和李宣东教授推荐.
收稿时间: 2022-09-04; 修改时间: 2022-10-13; 采用时间: 2022-12-14; jos 在线出版时间: 2023-01-13
CNKI 网络首发时间: 2023-07-06

explainable machine learning method called SHAP to analyze the influence of each feature on the prediction output of the model. The findings are as follows. (1) The influence of basic features on the model conforms to traditional software development laws. (2) Code and semantic features in submitted information are important in the prediction result of the model. (3) The contribution of different features in different systems to the output of the prediction model varies a lot.

Key words: intelligent computing framework; just-in-time defect prediction; explainable artificial intelligence; empirical software engineering

自深度学习技术在 ImageNet 图像分类任务取得重大突破以来, 人工智能已走出实验室, 进入医疗、金融、电子商务等社会经济重要领域, 成为决策和知识发现的重要参考. 面对日益增长的需求, 人工智能的应用需要实现工程化和便捷化, 也需要做到自主可控. 因此, 国内外研究者和开发人员实现了大量的智能计算框架. 例如, 谷歌的 TensorFlow[1], 亚马逊的 MXNet[2]以及百度的飞桨[3]等. 使用这些框架, 开发者在面向新的数据或应用领域开发新深度学习程序时, 可以在成熟的库或算子上构建应用程序, 大幅提升了开发效率. 因此, 作为底层实现, 深度学习框架代码的质量也影响了这类应用程序的可靠性和有效性. 深度学习库中的一个缺陷可能会导致许多应用程序中的缺陷, 而这样的缺陷可能会导致灾难性的后果. 例如, Pei 等人[4]在报告称, 一辆谷歌自动驾驶汽车和一辆特斯拉轿车因其深度学习软件中的漏洞而相撞. 因此, 如何有效地预测、定位并排除缺陷, 进而提高框架代码的质量及可靠性, 已成为关键的研究问题.

智能计算框架的缺陷症状和原因与普通的软件系统类似, 但智能计算框架存在一些特殊的错误集中于库的算法和 API 中[5]. 综合考虑智能计算框架业务特点和软件可靠性的一般特点, 研究人员提出了一系列以基于模糊测试的方法为代表[6]的缺陷定位和检测方法, 它们可以精确定位诸如数值精度、张量应用等原因造成的缺陷. 然而, 针对开源软件人员轮转率高、变更即时性强、团队建构和管理困难的特点, 上述方法却难以即时地对缺陷倾向进行预测, 让开源软件的开发者关注所提交代码质量. 因此, 本文从开源软件开发和协作的角度, 引入缺陷预测方法, 验证其是否能对维护其可靠性做出贡献. 在通用的软件工程领域, 不少研究者提出了许多软件预测技术[7−9]. 根据预测粒度不同, 主要包括模块级、文件级和变更级 (change-level) 缺陷预测[10]. 其中, 变更级缺陷预测也称为即时 (just-in-time, JIT) 缺陷预测[11], 其中即时表示进行缺陷预测的单位为一次代码提交. 在该方法的数据集建构过程中, 每个历史代码提交会被标记为有缺陷或无缺陷. 然后, 使用机器学习模型在大量历史提交数据上学习其特征, 用以预测未来的代码提交是否有缺陷. 由于即时缺陷预测的细粒度、即时性、易追溯的特点可以很好地解决传统文件级缺陷预测技术面临的挑战, 它逐渐成为缺陷预测领域的研究热点[10], 并且回应了开发者对软件质量保障工具的期待[12].

最近的研究表明[13−15], 当前的 JIT 缺陷预测方法几乎都为黑箱方法, 只提供预测, 而不提供解释, 所以阻碍了 JIT 缺陷模型在实践中的应用. 另外, 即便模型实现了正确预测, 开发者对于预测的一个提交被预测为有缺陷的理由仍然存在困惑. 因此, 本文使用即时缺陷预测技术实现对智能计算框架的每次提交代码进行缺陷分析, 预测其存在缺陷的可能性, 并利用 SHAP[16]对其缺陷模型进行解释.

为了实现可解释的智能计算框架的 JIT 缺陷预测, 首先本文从 GitHub 和 Gitee 开源仓库中获取了 4 个深度学习框架的数据集, 并收集在其变化历史可能导致缺陷的基础特征; 之后, 本文对其代码和提交文本进行主题分析, 并将其作为特征输入; 最后, 本文根据机器学习工程的步骤[17], 删除相关特征[18]、缓解多重共线性[18]、平衡数据集[19]、调整超参数[20], 并应用表现最好的机器学习算法进行了实验, 最终使用 SHAP 对模型进行解释.

本文的主要贡献可总结如下.

(1) 建构了智能计算框架的即时缺陷预测数据集 (https://figshare.com/s/ffca0d8422509b0d8ee3), 便于其他研究者进行后续研究.

(2) 在普遍使用的 14 个度量元的基础上, 添加了 LDA 主题分析来提取代码和提交信息隐藏的语义特征, 提高了基础模型的预测性能.

(3) 通过 SHAP 对于模型的预测结果进行解释, 发现与智能计算框架业务功能有关的代码和提交信息的语义特征可以提高模型的预测性能.

本文第 1 节为研究背景和相关工作. 第 2 节介绍数据集的收集和建构方法. 第 3 节提出研究问题并设计实验. 第 4 节对实验结果进行分析. 第 5 节根据实验结果讨论智能计算框架与常规软件系统缺陷特点. 第 6 节阐明有效性威胁. 第 7 节总结全文并展望未来工作.

1 研究背景和相关工作

本节介绍即时缺陷预测及智能计算框架的相关研究.

1.1 即时软件缺陷预测

软件缺陷预测技术在软件质量保证 (software quality assurance) 中发挥着重要作用, 也是软件工程数据挖掘领域一个活跃的研究课题[21]. 软件缺陷预测技术的提出可以提前预测可能存在缺陷的软件, 从而优化资源分配, 帮助开发人员更早的发现缺陷, 及时采取补救措施. 但是, 过去传统的缺陷预测技术以粗粒度 (包、文件或函数) 执行[9,22−24]. 尽管这些技术在学术研究的实验设置下是有效的, 但是在实际应用中遇到了不少的问题[10]: 第一, 预测的粒度较粗, 开发人员定位缺陷代码需要耗费大量时间和精力; 第二, 软件开发历史中, 代码可能被许多开发者修改, 因此很难找到合适的开发者对容易出现缺陷的模块进行代码检查; 第三, 由于缺陷预测是在软件开发周期的后期进行的, 开发人员可能忘记了开发的细节, 从而降低了缺陷检测和修复的效率.

为了应对以上问题, 研究者提出了一种称为即时软件缺陷预测的新技术[11]. 即时软件缺陷预测是一种细粒度的预测方法, 其预测对象针对开发者的每次提交变更, 具有以下优点[10]: 首先, 由于预测是细粒度的, 因此减少了开发者审查缺陷代码变更的时间和精力; 其次, 由于是针对每次代码的提交变更进行预测, 开发者对于提交的代码有充分的认识, 可以及时修复缺陷; 最后, 针对本次提交, 项目负责人可以清楚地找到开发者, 从而及时分析缺陷原因, 从而进行解决.

Kim 等人在 2008 年首次提出了对每次代码变更进行缺陷预测[25], 在 2013 年中首次将这种缺陷预测技术称为 JIT 缺陷预测技术[11]. 为了持续提升 JIT 缺陷预测技术的性能和稳定性, 需要扩充数据集和特征类型, 使用各种代码更改的有关属性来预测提交级缺陷. 例如, Mockus 等人[26]使用一组主要来自代码变更本身的属性预测了导致修复的提交. 在最近的研究中, 研究者又通过抽象语法树来量化代码的结构, 从而计算代码变更前后的代码结构差异[27]. 还有部分研究人员为了弥补语义和缺陷预测特性之间的差距, 考虑应用自然语言处理 (NLP) 技术即从源代码文件和代码更改中自动学习程序的语义表示, 并报告称, 语义特征可以显著改善缺陷预测任务[28]. 随着即时缺陷预测的发展, JIT 模型已经开始被应用于实践. Lucent[26] 、Blackberry[29]和 Cisco[30] 等许多大型信息技术公司在软件系统开发过程中都在应用 JIT 缺陷预测模型.

1.2 基本度量

Kamei 等人[11]首次关于即时软件缺陷预测进行了大规模的实证研究. 他们提出了 14 个度量元, 这些度量元可以分为 5 个维度即规模、历史、代码分布、目标和经验. 因此, 本文研究中使用的基本度量指标也参照上述研究所用的度量指标. 基本度量的信息如后文表 1 所示.

1.3 LDA 主题模型

LDA 由 Blei 等人于 2003 年提出[31], 是一种主题模型. 它可以将文档集中每篇文档的主题以概率分布的形式给出, 在得到文档的主题之后, 便可以根据主题对文档集进行聚类. 其过程是一种无监督的学习方式, 包含着两个层次的映射: 文档-主题-词汇. 对于一篇文档可以看作有大量词语构成, 各种词语之间不存在顺序关系, 于是文档中的每个词都可以看作一个主题, 而一篇文档可以包含多个主题, 即通过 LDA 会生成两个分布概率中: 一个与文档相关, 另一个与语料库中的术语相关. 第 1 个分布是按主题的文档矩阵 (Θ): 一个 $K\times n$ 矩阵, 其中 K 是主题的数量, n 是文档的数量, $\Theta(i,j)$ 表示第 j 个文档与第 i 个主题相关的概率. 第 2 个分布是主题词矩阵 (Φ): $m\times K$ 矩阵, 其中 m 是语料库中的单词数, K 是主题数, 如 $\Phi(i,j)$ 表示第 i 个单词属于第 j 个主题的概率.

通过建立 LDA 主题模型进行分析, 可以得到任何一篇文档的主题概率以及每个主题相应的词汇, 通过这些词

汇表达的意义便可以归纳出该主题的意义. 因此, LDA 主题模型能够挖掘文档的潜在主题, 进而分析文档的关注点及其相关特征词.

表 1 基本度量的描述

特征维度	特征名	描述
代码分布	ns	变更子系统的数量
	nd	变更修改代码目录的数量
	nf	变更修改文件的数量
	entropy	修改代码在变更相关文件中的分布
规模	la	增加的代码行数
	ld	减少的代码行数
	lt	变更前的代码行数
目标	fix	该变更是否修复缺陷
历史	ndev	对该变更相关文件进行修改过的开发者数量
	age	该变更与上一次变更时间差的平均值
	nuc	该变更文件进行修改过的变更数量
经验	exp	开发者已提交的变更数量
	rexp	开发者近期提交的变更数量
	sexp	开发者已提交的变更中影响到该变更相关子系统的数量

在 LDA 主题模型的应用方面, Nguyen 等人[32]曾提出一种传统软件缺陷预测方法, 即使用 LDA 主题建模关注系统的技术问题以及功能, 用来刻画开发者在源码中对不同技术细节的关切. Chen 等人[33]同样使用 LDA 主题建模研究了软件系统的实际功能对代码质量的影响, 并提出关于这些主题的度量, 以帮助解释实体的缺陷倾向. 在他们的方法中, 软件系统被视为不同技术以及功能的集合, 通过主题建模分析源代码的主题可以反映其中包含的技术概念, 从而利用这些概念与缺陷倾向的联系提升模型性能. 前述工作均在传统缺陷预测的背景下开展, 但是由于其粒度粗、即时性低、不可追溯等问题[10,11], 传统缺陷预测已经难以符合开源社区中智能软件框架开发对软件质量模型的需求. 因此, 即时缺陷预测被提出以应对传统缺陷预测的不足. 在这一任务中, 部分传统缺陷预测的经验已经失效[11], 而部分经验仍然有效 (例如, 结构度量的效果不如开发过程度量[10,34]). 因此, 本文在即时缺陷预测的背景下开展对特定领域程序的研究, 与前述工作既有联系、也有区别. 主要的区别和创新在于: (1) 在内容方面, 本文分析了代码提交中的主题信息是否可以作为新的特征运用于预测缺陷预测, 而这一特征未被前述工作调研; (2) 在结论方面, 本文发现与结构特征的特点[10,34]不同, 主题信息的特征重要性可能与开发过程特征同样强, 体现出在即时缺陷预测中捕获这类特征对建构有效模型的必要性. 同时, 本文针对一类细分的程序类型进行了实证研究, 验证了在一般缺陷预测的工作中“程序功能特征重要”[32]这一结论的泛用性和实用性, 也印证了近期学者[35]在即时软件缺陷预测的可解释应用中对模型捕获程序业务等信息能力不足的关切. 因此, 本文建议在即时缺陷预测的应用中加强对代码提交信息和程序源码主题的建模和关注.

1.4 基于测试的深度学习框架缺陷定位

针对智能计算框架的特点, 学者有针对性地提出了一些基于测试的缺陷定位方法.

不同智能计算框架通常会分别实现同一种算法或类似的程序功能. 受到差分测试的启发, 研究者们[36,37]通过对比它们的结果一致性来定位潜在的缺陷. 类似地, Wang 等人[38]设计了一系列变异测试规则对深度学习模型进行调整, Guo 等人[39]在同种框架在不同平台下生成的模型也有不同的表现, 这些工作都在模型应用的层面检测不同框架的不一致性, 进而推断潜在的缺陷.

深度学习算子是赋能智能计算框架的基本单元, 然而, 由于其输入维度的不确定性和浮点计算误差传播机理的多样性, 算子精度的误差难以评估, 其测试输入的生成也是一个挑战. Nejadgholi 等人[40]通过分析框架测试套件中近似断言的测试预言, 发现相关代码是难以维护和易错的. Zhang 等人[41]通过寻找最大精度误差, 提出了一种检

测 TensorFlow 框架算子精度的测试方法的模糊测试方法, 通过算子输入精度和计算精度的 7 可变性生成测试输入, 以测得精度误差作为反馈, 动态引导测试执行过程. Zhang 等人在后续工作中[6]提出了针对深度学习算子的差分模糊测试框架, 通过两类变异方法集实现算子测试输入生成, 并通过变异方法的选择策略、能量调度算法优化模糊测试执行过程, 最后通过差分测试方法实现对算子的实现缺陷的检测.

Xie 等人[42]还试图通过智能学习框架的软件接口文档来生成输入规约, 并使用规约来引导更为精确的测试用例生成, 有效检测出了 28 个受开发者认可的缺陷.

本文与上述方法不同主要在于: (1) 使用信息不同, 上述工作主要使用不同框架的行为和框架算子的精度误差等信息实现预测, 而本文使用的是代码提交信息; (2) 关注视角不同, 本文主要面向代码提交级的缺陷倾向进行缺陷预测, 建议开发者注重容易引入缺陷的代码提交, 不实现缺陷的精确定位.

1.5 可加性解释模型框架 SHAP

SHAP[16]模型是模型无关的模型可解释性的方法, 它既可作用于全局解释, 也可以对每个本地的预测实例生成局部解释. 另外, SHAP 属于模型事后解释的方法, 它的核心思想是计算特征对模型输出的边际贡献, 再从全局和局部两个层面对“黑盒模型”进行解释. SHAP 构建一个加性的解释模型, 所有的特征都视为“贡献者”, 对于每个预测样本, 模型都产生一个预测值, SHAP 值就是该样本中每个特征所分配到的数值. 假设第 i 个样本为 x_i, 第 i 个样本的第 j 个特征为 x_{ij}, 模型对于第 i 个样本的预测值为 y_i, 整个模型的基线 (通常是所有样本目标变量的均值) 为 y_{base}, 那么 SHAP 值见公式 (1).

$$y_i = y_{\text{base}} + f(x_{i1}) + f(x_{i2}) + f(x_{i3}) + \ldots + f(x_{ik}) \tag{1}$$

2 数据集构建

本文的即时缺陷预测流程如图 1 所示, 主要分为数据收集、数据标注、特征获取和模型的搭建以及评价 4 个阶段, 其中数据收集、数据标注和特征获取可以概括为数据集的构建, 在本节将详细的描述, 而模型的搭建以及评价阶段将在第 3 节详细介绍.

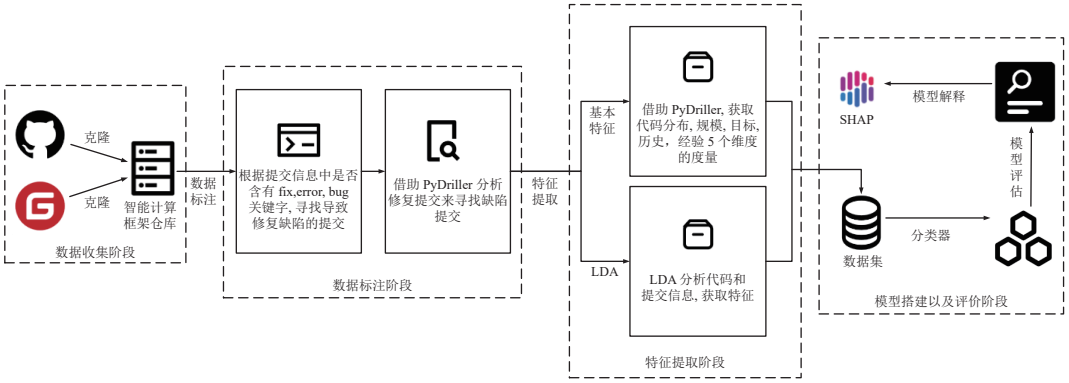

图 1 智能计算框架的即时缺陷预测流程

2.1 数据收集

本文使用的深度学习框架均来自 GitHub 或 Gitee 平台. GitHub 是管理软件开发的关键平台, 它集开源代码库以及版本控制系统于一体, 方便完成后期的数据标注以及特征提取. 另外 Gitee 与 GitHub 类似, 是开源中国社区 2013 年推出的基于 Git 的代码托管服务, 致力于为国内开发者提供优质稳定的托管服务, 部分国内厂商推出的智能计算框架主要在此平台上实现开发和维护.

本文选择了国外 TensorFlow, MXNet 以及国内华为的 Mindspore 和百度飞桨 4 款智能计算框架. 选择 TensorFlow 主要考虑到 Zhang 等人[43]指出在 GitHub 中有 36 000 多个项目调用 TensorFlow 的 API, 导致 TensorFlow

内部的缺陷影响着成千上文的深度学习应用程序. 而 MXNet 是由众多小组织一起合力实现的平台, 排名前 20 的开发者出自不同的公司. 众多小组织的协作可能导致开发过程的管理困难[44], 增加缺陷产生的概率. 之所以引入国内的深度学习框架, 是因为考虑到人工智能基础设施自主可控的客观需求, 选择了市场占有率较高的国产框架. 本文在 GitHub (Gitee) 中克隆选定的 4 款智能计算框架到本地, 为后续数据标注做准备.

2.2 数据标注

识别缺陷引入变更需要考虑代码的动态演进和代码变更之间的关联关系, 借助软件项目的历史变更数据, 识别缺陷引入变更, 从而完成数据标注[10]. 标注数据的方法最早来自 SZZ 方法, 该算法由 Sliwerski 等人[45]在 2005 年首次提出, 并以它们名字的首字母命名该算法. 随后, SZZ 算法成为用于从软件项目代码仓库和缺陷仓库中识别缺陷引入变更的通用框架. 本文根据 SZZ 算法的原理, 借助 PyDriller[46]工具编写代码来完成数据标注工作, PyDriller 是一个用 Python 实现的框架, 可帮助开发人员挖掘软件存储库, 使用 PyDriller, 可以从任何 Git 存储库中提取信息, 包括提交、开发人员、修改、变更差异和源代码信息. 下面将具体描述数据标注过程.

2.2.1 识别缺陷修复的变更提交

首先, 需要判断一个提交是否是修复缺陷提交. 本文根据每次提交的提交信息, 按照关键字来确定是否为缺陷修复的变更提交. 将提交信息所有字母转换为小写字母, 并提取单词的词干, 如果提交信息中存在诸如“bug”“fix”等关键字 (见表 2 额外特征 fix), 则判定为修复缺陷的提交. 需要注意的是, 为了减少标注的误差, 在缺陷修复中, 本文使用“merge”和“doc”关键字来排除假修复提交, 这些提交可能修复了代码以外的软件制品, 也可能并非由开发者主动发起的代码变更.

表 2 额外特征的基本信息

额外特征	关键字
fix	bug, fix, error, issue
test	test
doc	doc
build	build
merge	merge
comment	comment
refactor	refactor, lint, restructure

2.2.2 根据缺陷修复的变更提交去识别引入缺陷的代码

在获取到修复缺陷的提交后, 需要根据修复代码去识别可能引入缺陷的代码. 本文主要借助 PyDriller 来完成, 对于每个代码提交, PyDriller 可以应用 SZZ 算法, 返回其最近关联提交中包含的文件中修改的行的提交集. 所以, 本文将缺陷修复提交对象输入, 借助 PyDriller 获得返回提交集合, 即为导致缺陷的提交.

2.3 特征提取

在获取到原始的数据集后, 便可以在仓库中挖掘每次提交相关的特征. 本文从收集的代码提交中提取了先前关于即时缺陷预测研究中使用的过程指标 (见表 1). 本文称这些过程指标为“基本特征”, 因为这些指标可直接根据代码仓库的历史提交数据计算.

之后, 本文在该集合中添加了通过 LDA 主题建模分析提交信息和代码的潜在特征和语义特征, 以丰富数据集的信息内容. 在对提交消息运行 LDA 算法之前, 应用标准自然语言预处理步骤来提取更有意义的主题: 首先将从代码和提交信息中清除停用词和非字母数字符号, 并将所有字母转换为小写字母, 之后将提取单词的词干. 考虑到提交信息中 fix, bug, error, refactor 等关键字对于提交信息主题分布的影响, 本文将提交信息出现上述关键字设置为额外特征, 如表 2 所示.

稍后, 本文使用 LDA 主题建模实现为每个提交消息以及代码收集主题的概率分布, 然后将其作为语义特征输入到模型中. 本文对 LDA 模型进行了参数调教, 首先寻找较好的主题数, 在 2 至 10 步长为 1 的值域下根据 u_mass

一致性 (coherence) 指标[47]获得最优的主题数. 随后, 设置 Alpha 和 Beta 值. Alpha 参数与主题稀疏性密切相关. 较高的 Alpha 参数对于主题稀疏性影响较小, 即一篇文档可以包含多数主题, 反之较低的 Alpha 参数意味着一篇文档仅可以涵盖少量主题. Beta 参数与主题词的稀疏性密切相关. 较高的 Beta 值对主题词的稀疏性影响较小, 即一个主题可以包含语料库的多个词语, 反之较低的 Beta 意味着一个主题与多个单词关联的概率很低. 根据经验[48], 本文将 Alpha 设置为 50 与主题数的商, 并将 Beta 设置为 0.01.

在完成数据集的收集后, 本文对数据集做预处理, 以删除异常数据, 忽略更改超过 1 000 行和超过 100 个文件的大型提交, 因为这些提交可能是由日常维护 (例如版权信息更新) 引起的噪音数据. 本文还忽略了不添加任何新行的更改, 因为根据 SZZ 算法的原理, 它只能将引入新行的提交标记为缺陷提交[49]. 最后, 获取数据集的基本信息如表 3 所示, 括号中百分数代表含缺陷的提交数目占总提交数目的比例.

表 3　数据集的基本信息

项目	总提交数目	含缺陷的提交数目	来源平台
TensorFlow	111 272	31 355 (28%)	GitHub
MXNet	11 826	5 020 (42%)	GitHub
PaddlePaddle (百度飞桨)	25 555	11 937 (46%)	GitHub
MindSpore (华为)	20 571	766 (3.7%)	Gitee

3　实验设计

为验证即时智能计算框架的缺陷预测的有效性, 本文提出了 3 个研究问题.

RQ1: 代码和提交信息的关注点是否可以用于智能计算框架的即时缺陷预测?

RQ2: 即时缺陷预测在智能计算框架中的表现如何?

RQ3: 即时缺陷预测模型中各特征属性对模型预测输出的影响如何?

本文设计的所有实验均运行在 CPU 为 i7-11800H, 内存为 16 GB 的 64 位 Windows 10 操作系统的计算机上. 开发和运行环境为 Python 3.7, 对于 GitHub 仓库的分析采用开源工具 PyDriller, LDA 建模采用 sklearn 的 LatentDirichletAllocation 库, 模型的实现使用 Python 开源机器学习库 scikit-learn, 采样方法的实现使用 Python 第三方库 imbalanced-learn.

本节中将详细的介绍实验设计的细节, 包括数据集的划分、数据预处理以及模型的搭建和性能的评估.

3.1　数据集划分

在即时缺陷预测任务中, 对于数据集的划分存在许多关注和争议[30,49], 因为变更数据具有时间顺序, 传统的交叉验证会高估实验性能, 所以在实验中采取 Yang 等人[50]针对即时缺陷预测提出了一种时间感知的验证方法, 这种验证方法对所有变更按照提交时间来排序, 然后将同月份变更的提交分为 1 组. 之后使用 n 月和 n+1 月创建的代码变更来预测 n+4 月和 n+5 月创建的代码变更. 在这里将测试集和训练集的周期设置为 2 个月, 主要考虑到一下因素: (1) 大多数项目以 2 个月为开发周期; (2) 训练集和测试集之间存在 2 个月的间隔, 可以保证项目的变更缺陷充分地被开发者所发现; (3) 2 个月的时间间隔可以保证训练集中有足够多的数据用于训练模型; (4) 使用近期产生的数据来对新变更进行预测保证计算结果的可靠性.

3.2　数据预处理

为了保证后期解释即时缺陷模型的可靠性以及准确性, 对于每个划分的训练集, 需要缓解相关性和共线性, 本文使用了一种自动特征选择的方法 AutoSpearman[51], 该函数使用根据 Spearman 秩相关方法计算的相关性值消除相关性, 使用 VIF 计算的值处理多重共线性. 根据 Kraemer 等人[52]的建议, 对于 Spearman 相关性值使用 0.7 的阈值来表示特征之间的强相关性, 而对于 VIF 多重共线性, 使用 5 作为阈值.

之后, 使用 SMOTE 处理类不平衡. 为了确保 JIT 缺陷模型的预测高度准确, 本文参考先前工作[53]的建议, 应

用了一种类再平衡技术. 由于研究的 JIT 缺陷数据集的缺陷提交和非缺陷提交相比是不平衡的, 因此仅在训练数据集上应用 SMOTE[54]来处理类不平衡. SMOTE 执行以下步骤: 首先, SMOTE 计算一组少数类的 k 近邻; 然后, SMOTE 随机选择邻居并围绕这些邻居生成合成实例; 最后, SMOTE 将合成实例与多数类的欠采样相结合, 以生成最终的平衡实例集. 本文使用 SMOTE 的默认设置 5. 因为对各种 k 设置的实验表明, 改变 k 设置对 JIT 缺陷模型的性能影响很小[55].

3.3 模型搭建

本文试验了多种常用的分类器, 包括逻辑回归、决策树、随机森林等, 结果发现随机森林的效果更好, 与大多数的工作结论类似[8,55], 所以本文选择随机森林作为分类器. 另外 Yang 等人[56]通过基于 6 个开源项目的大规模实证研究来比较本地和全局模型的性能, 局部模型的分类性能比全局模型差, 所以本文使用每个项目的训练数据建立全局 JIT 缺陷模型.

3.4 性能指标

性能评价指标能够判断一个模型的好坏. 由于本文构建的模型是用于检测代码提交中是否含有缺陷, 属于二分类问题. 为了更直观评估本方法的性能, 考虑其中 2 种指标: *F-measure* 和 AUC-ROC, *Precision* 和 *Recall* 是一对矛盾的度量, 一般来说, *Precision* 高时, *Recall* 值往往偏低; 而 *Precision* 值低时, *Recall* 值往往偏高. 当分类置信度高时, *Precision* 偏高; 分类置信度低时, *Recall* 偏高. 为了能够综合考虑这两个指标, *F-measure* 被提出, 对 *Precision* 和 *Recall* 做加权调和平均, 如公式 (2).

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

AUC-ROC 的值即为 ROC 曲线下的面积, 是分类器区分类的能力的度量, 用作 ROC 曲线的总结. 因为 AUC 的计算中自动考虑了数据中存在的不平衡, 所以 AUC 对于不平衡的数据是鲁棒的[7]. AUC-ROC 的值越高, 模型在区分正类和负类方面的性能越好, 在即时缺陷预测通常认为 AUC-ROC 值大于 0.7 时分类预测模型的性能可评估为优良[57]. 在实验最后, 本文利用 SHAP 解释模型, 计算每个特征对预测结果的贡献度, 通过 SHAP 考虑各个特征对模型结果的影响, 以进一步分析代码和提交信息对即时缺陷预测模型预测的影响.

4 实验结果

4.1 RQ1: 代码和提交信息的语义是否可以用于智能计算框架的即时缺陷预测

这一 RQ 探究代码和提交信息的主题建模结果是否包含与缺陷或框架功能、开发流程有关的信息, 是否可进一步用于智能计算框架的即时缺陷预测. 首先, 用 LDA 算法分别生成代码和提交信息的 5 个主题并输出相应的 5 个主题词, 具体信息如表 4 和表 5 所示, 其中 ct (code theme) 是根据代码文本生成的 5 个主题; mt (message theme) 是根据代码提交信息生成的 5 个主题. 以提交信息为例, 从表 5 可以看出主题 1 涉及 GPU 和 XLA 的操作, 主题 2 涉及代码提交和业务的实现和回滚, 主题 3 和主题 4 的主要内容涵盖程序版本、功能和构建, 其中主题 3 更加侧重实现, 主题 4 更侧重构建, 主题 5 涉及张量、IO 等功能实现的技术名词. 由于表 4 和表 5 的主题词与智能框架的业务和日常维护相关, 因此可以尝试将代码和提交信息用于即时缺陷预测.

表 4 代码产生 5 个主题的前 5 个单词

序号	ct1	ct2	ct3	ct4	ct5
1	shape	inputarg	tensor	name	arg
2	hloinstruct	attr	tfoutput	tensorflowcor	membermethod
3	hloopcod	dtint	option	dep	argspec
4	tensorflowcompilerxla	typettr	shape	node	varargsnon
5	shapeutil	outputarg	func	graph	keywordsnon

表 5 提交信息产生 5 个主题的前 5 个单词

序号	mt1	mt2	mt3	mt4	mt5
1	xla	op	function	file	function
2	oper	compat	gener	disabl	op
3	gpu	commit	op	version	tensor
4	intern	forward	usag	make	call
5	hlo	roolback	llvm	target	input

4.2 RQ2: 即时缺陷预测模型的性能

为了回答 RQ2, 本文将实验分为 3 组, 第 1 组实验采用最原始的 14 个基本度量进行缺陷预测, 记作 Base; 第 2 组实验仅使用 LDA 主题建模的语义潜在特征进行缺陷预测 (包括表 3 的额外特征), 记作 LDA; 第 3 组在基本度量的基础上添加上 LDA 主题建模的语义潜在特征进行缺陷预测, 记作 Base+LDA. 将本文通过对比上述经典评价指标 *F-measure* 值和 AUC-ROC 的值来评估测试异味对即时缺陷预测模型的有效性.

使用随机森林构建即时缺陷预测模型, 对于在上文构建的 4 个数据集进行了验证. 另外, 为了保证实验的可靠性分别重复进行了 10 次实验, 结果取其平均值, 最终结果如表 6 所示.

表 6 模型性能对比

框架	分组	*F-measure*	AUC-ROC
TensorFlow	Base	0.705 0	0.746 0
	LDA	0.623 1	0.650 0
	Base+LDA	**0.716 1**	**0.761 0**
MXNet	Base	0.730 1	0.806 3
	LDA	0.704 6	0.765 0
	Base+LDA	**0.731 6**	**0.807 4**
PaddlePaddle	Base	0.713 9	0.776 0
	LDA	0.627 5	0.664 7
	Base+LDA	**0.718 0**	**0.783 0**
Mindspore	Base	0.746 9	0.706 3
	LDA	0.762 2	0.662 9
	Base+LDA	**0.773 5**	**0.727 7**

根据表 6 中的数据结果, 可以看出基本度量的即时缺陷预测的效果较好, AUC-ROC 值均大于 0.70, *F-measure* 的值为 0.70 以上, 而采用单独的 LDA 度量进行预测表现一般, 模型的 AUC-ROC 均值只有 0.68. 但是, 在 14 个基本特征的基础上加入 LDA 主题建模的特征后, 发现对即时缺陷预测中 *F-measure* 的性能以及 AUC-ROC 的表现均有明显提升. 在所研究的智能计算框架中, 添加 LDA 主题分析后, AUC-ROC 总体性能可以达到 0.77 左右, 所以代码和提交信息的主题分析可以提高智能计算框架的即时缺陷预测的性能, 并且良好的性能表现保证了后续模型解释可靠性.

4.3 RQ3: 模型中各特征属性对于模型预测的影响

为了回答该问题, 本文选取 TensorFlow 数据集中的一个实例, 使用 SHAP 对该实例进行解释. 图 2 显示了由 SHAP 生成对该实例影响最大的 10 个特征的可视化解释. 对于这个特定的实例, 模型最终输出结果为 0.385 (0 表示无缺陷, 1 表示缺陷), 所以该提交的预测结果为无缺陷.

首先, 本文分析 nd, age, nedv, lt, la 等基本特征对于模型的影响, 从图 2 中可以发现 ndev, la 对于“有缺陷”的预测有正贡献, 而 nd, age, lt 对于预测有负贡献. 具体来看, 如 nd 表示本次提交修改的目录数量, 修改目录的数量越多产生缺陷的几率越多, 在这个实例中 nd=1 说明本次提交只修改了一个目录, 所以对于缺陷的产生有负贡献; la 表示本次提交开发者提交代码的行数, 提交代码的行数越多产生缺陷的可能性越高, 在图 2 中 la=83, 本次提交的代码数量比较多, 所以对于缺陷的产生有正贡献. 总体来说, 基本特征对于模型的影响还是符合软件开发规律的.

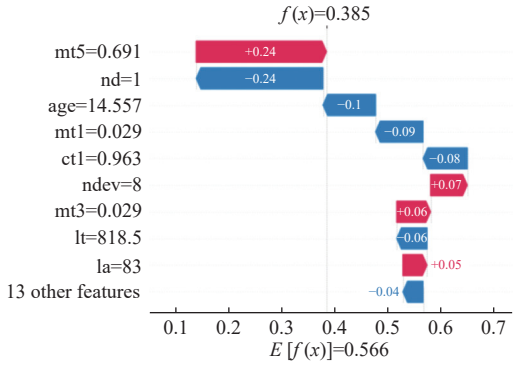

图 2 使用 SHAP 生成的 TensorFlow 的一个 JIT 缺陷预测实例

之后探究 LDA 主题提取的潜在特征对于模型预测的贡献度. 通过图 2 可以发现: 高 mt5 概率、低 mt3 概率与缺陷的产生有正贡献, 而低 mt1 概率对缺陷的产生有负贡献. 这表明, 出现上述主题 5、主题 1、不出现主题 3 的代码提交更容易出现缺陷, 即分类器的行为表明, 涉及功能技术名词和 GPU 操作的代码提交有更高的错误倾向, 这一发现与文献 [5] 的实证研究相吻合, 即核心组件的功能实现缺陷多于建构、安装等部署阶段的缺陷.

为了在全局验证这一案例的解释, 本文中输出了模型中各特征的贡献度排序.

图 3 展示了本次实验中对于 TensorFlow 缺陷影响最大的 20 个特征属性, 从图中可以看出对于模型影响最大的特征属性是 la 和 lt, 而 LDA 主题建模提取的潜在语义特征中 mt5, mt2, mt4, ct1, ct3, mt1, ct5 均在前 20 的序列中, 其中 mt5 对于模型的预测结果有重要影响.

为了验证上述发现的泛化性, 本文又生成其他几个模型的贡献度进行分析, 各项目中特征属性的贡献度排序如图 4–图 6 所示. 为了在统计的角度上检验特征贡献、体现特征贡献的显著程度, 本文使用 Scott-Knott ESD (SK-ESD[58]) 对特征的重要性进行聚类和排序. SK-ESD 是对 Scott-Knott 层次聚类算法的改进, Scott-Knott 层次聚类算法可以将多种等长的数据按值分布进行分组聚类, 最终给出每种数据的排名, 同一组内的多种数据统计差异不显著, SK-ESD 在此基础上进一步合并了效应值不显著的组, 保证了不同排名的组间的统计差异也是显著的. 本文在图 4–图 6 中将特征重要性的 SK-ESD 排名标注在纵轴特征名后的括号内.

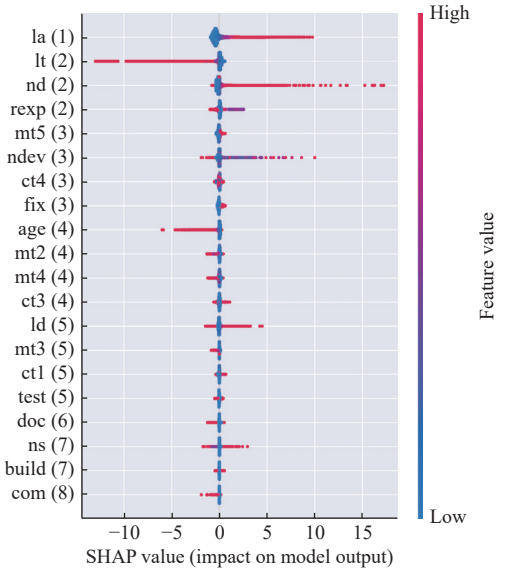

图 3 TensorFlow 框架的 SHAP 散点图

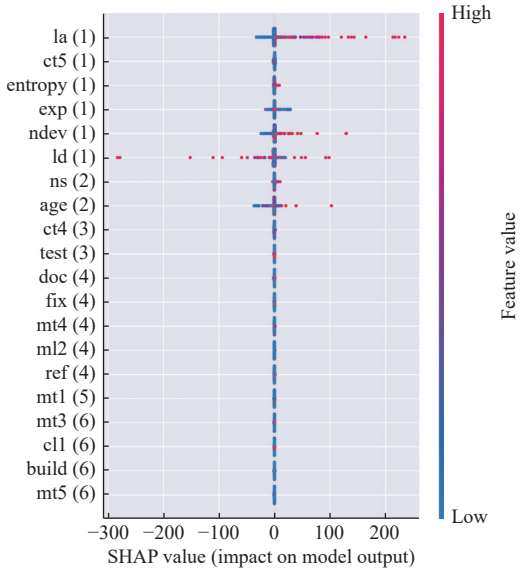

图 4 MXNet 框架的 SHAP 散点图

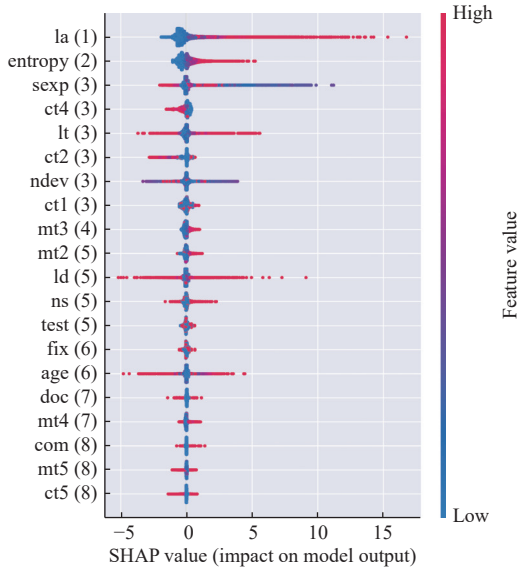

图 5 PaddlePaddle 框架的 SHAP 散点图

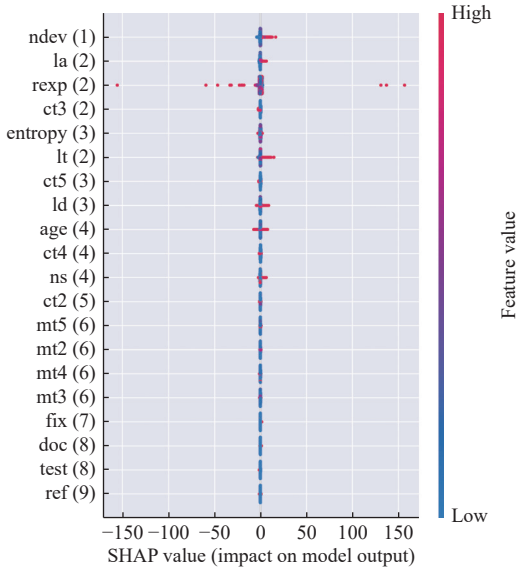

图 6 Mindspore 框架的 SHAP 散点图

通过观察可以发现在不同的智能框架中, 不同的特征属性对模型预测结果的影响也是不同的, 但 la 在每个模型的都具有重要影响, 且 SK-ESD 排名一直为 1, 这说明代码提交行数在缺陷的预测有重要影响. 随着代码添加行数的增多, 发生缺陷的可能性越大. 对于开发者来说, 在后续的代码提交中, 需要重点关注代码提交量大的提交来减少代码缺陷的产生, 也需要避免在一次提交中实现过多的新特性和新需求. 另外, 通过观察代码和提交信息语义特征的分布, 可以看出代码和提交信息的语义特征对于软件缺陷预测的输出具有不可忽视的影响 (在 4 个项目中, 均在排名前 5 的特征中出现, 其 SK-ESD 排名位于 1 至 3). 因此, 在以后使用过程相关指标构建的缺陷预测模型时, 可以结合主题分析模型提取隐藏和语义特征, 来进一步提高基础模型的预测性能.

5 讨 论

文献 [5] 提及一个典型智能计算框架 (TensorFlow) 的缺陷特点与常规软件系统具有类似的根因模式, 即语义类问题占多数, 而算法类问题占少数. 因此, 本文假设即时缺陷预测方法能够应用到智能计算框架, 以在实际应用中弥补现有缺陷定位手段更倾向于算法类的错误定位、及时性不足和对缺陷预防能力不足的问题. 本文 RQ2 的实验结果证实了对于智能计算框架的缺陷处理、预防和质量保障任务, 除了使用以模糊测试的方法为代表[6]的缺陷定位和检测方式外, 也可以使用包含开发过程度量指标的即时缺陷预测的方法实现效果较好的预测.

然而, 即便智能计算框架与更普遍的传统软件在缺陷方面有较多相似之处, 它们也具有一定的独特性. 例如, 在引发缺陷的根因方面, TensorFlow 的接口错误经常由算子的张量对齐问题和类型问题引发[5], 而这类信息是难以被衡量代码结构特征的基础度量所捕获到的. 因此, 本文引入了 LDA 主题建模, 并试图通过代码片段和代码提交的主题概率特征来提升即时缺陷预测方法对业务特点的捕获能力. 在 RQ1 的建模结果中, 有多个主题特征 (例如 ct3, mt5) 在代码片段和提交两方面捕获了“张量 (tensor)”的信息, 也在代码片段的有关特征中 (例如 ct2) 捕获了“类型 (type)”信息. 在 RQ3 的特征重要性分析中, TensorFlow 项目的 ct3 和 mt5 的特征重要性较高, 智能计算框架软件缺陷常见模式在模型行为中有所体现. 在后续工作中, 可引入程序测试频谱、测例执行结果、软件日志等更多的软件制品信息, 对智能计算框架的缺陷预测模型进行进一步的设计和优化.

通用和智能计算框架在软件缺陷上的相似和不同特性也反映在了它们缺陷预测模型的特征重要性之上. 在先前对大量通用软件系统进行实证调研的工作中[11], 学者发现尽管开源项目和商业项目的预测模型有不同的特征重要性模式, 但它们具备一定的共性, 即变更文件数量 (nf)、相对代码搅动指标 (la/lt 和 lt/nf) 是预测缺陷提交的重

要特征. 作为开源软件, 智能计算框架的缺陷也在很大程度上可以由 la 和 lt 度量指标预测, 前者和后者的特征重要性分别在 4 个 (全部) 和 3 个框架中排名前 3. 由于被特征选择算法筛出, 因此 nf 指标没有参与本文的预测, 但衡量变更规模的相关指标 (例如 la、nd) 仍然是排名较高的特征. 同时, 本文也发现开发者数量 (NDEV) 特征在智能框架缺陷预测中的重要性较高, 在 4 个项目中的 3 个中排名前 3, 但在文献 [11] 对于开源软件的观察中, 它们均不是重要的特征. 对于 NDEV, 文献 [11] 在商业软件中发现较高的值会增加风险, 他们还进一步推测商业项目和开源项目中代码拥有度的不同是导致这一因素贡献不同的理由. 在本文的预测中, 即便 NDEV 对模型的贡献较高, 它是否导致更高缺陷倾向的结果也因项目而异, 近期的工作发现赞助等活动会在开发者动机、合作倾向等方面对 GitHub 开发者的行为造成影响[59], 智能计算框架由大型科技公司支持甚至主导研发, 但也其代码也同时由普通开发者贡献, 这可能是造成这一区别的原因. 在过程度量的基础上, 本文增加的代码提交和源码主题度量也获得了较高的特征重要性排名, 体现出在即时缺陷预测任务中引入代码业务有关信息的可行性和必要性[35](见第 1.3 节).

6 有效性威胁

在数据标注阶段, 本文通过关键词判断和标记修复的代码变更, 并通过 SZZ 算法判断引入缺陷的变更, 由于标记修复的代码变更. 然后, SZZ 在标记引入修复的代码变更时可能存在噪音, 导致缺陷的标注数据存在误差, 但本文的第 1、第 5 作者对 100 个样本进行了抽样检测, 发现它们均为缺陷提交, 这一结果可在一定程度上作为数据可靠性的参考. 后续工作可能需要人工介入来验证和去除所构建缺陷数据集存在噪音, 以进一步提升数据集的可靠性.

对于特征的提取, 本文借助 PyDriller 工具计算度量元, 该工具可以帮助开发人员挖掘软件存储库; 对于主题建模, 本文采用 sklearn 实现的 LatentDirichletAllocation, 来进一步保障获取主题概率特征的可靠性, 这些工具可能存在一定的实现误差, 但它们都是被科研社区广泛使用的工具, 其可靠性有一定保证.

在模型定义和验证过程中, 本文对数据进行预处理并处理了数据不平衡问题, 利用了机器学习中的逻辑回归算法进行缺陷分类, 并多次重复实验取均值来进行模型评估, 因此, 性能数据是相对可靠的.

然而, 本次实验只在 4 个项目上进行探究, 选取了使用较为广泛的 TensorFlow、百度飞桨等智能计算框架, 本文的所有项目均从 GitHub 或者 Gitee 的官方代码仓库获取, 但并没有涵盖全部的智能计算框架, 还需要在一些用于特定目的或不活跃的智能计算框架中尝试复现本文的性能.

7 总 结

本文通过挖掘智能计算框架的代码仓库, 基于通用的 14 个基本度量元构建了数据集, 探究即时缺陷预测是否适用于智能计算框架; 另外, 在初始数据集的基础上通过 LDA 主题建模提取了潜在语义特征, 探究代码和提交信息对于即时缺陷预测性能的影响. 实验结果表明, 通过引入对代码和提交信息的主题建模信息, 可以略微提升性能表现; 另外, 根据 SHAP 分析的各特征属性对模型预测输出的贡献度的排序, 确定了以提交变更行数为代表的基本特征具有最大的贡献, 代码和提交信息的潜在语义也对缺陷预测的具有一定的作用, 这表明代码的结构信息和业务特征都是有价值的缺陷预测信息源.

在未来的工作中, 首先考虑扩充智能计算框架的数据集, 以更好的探究代码和提交信息对即时缺陷预测模型有效性的普遍适用性. 其次结合深度学习技术, 在代码结构和业务特性两方面, 更好地捕捉程序源代码的关键特征信息, 进而增强现有即时缺陷预测模型的预测效果.

References:

[1] TensorFlow. 2022. https://github.com/tensorflow/tensorflow

[2] MXNet. 2022. https://github.com/apache/incubator-mxnet

[3] PaddlePaddle. 2022. https://github.com/PaddlePaddle/Paddle

[4] Pei KX, Cao YZ, Yang JF, Jana S. DeepXplore: Automated whitebox testing of deep learning systems. In: Proc. of the 26th Symp. on Operating Systems Principles. Shanghai: ACM, 2017. 1–18. [doi: 10.1145/3132747.3132785]

[5] Jia L, Zhong H, Wang XY, Huang LP, Lu XS. The symptoms, causes, and repairs of bugs inside a deep learning library. Journal of Systems and Software, 2021, 177: 110935. [doi: 10.1016/j.jss.2021.110935]

[6] Zhang XF, Liu JW, Sun N, Fang CR, Liu J, Wang J, Chai D, Chen ZY. Duo: Differential fuzzing for deep learning operators. IEEE Trans. on Reliability, 2021, 70(4): 1671–1685. [doi: 10.1109/TR.2021.3107165]

[7] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. on Software Engineering, 2008, 34(4): 485–496. [doi: 10.1109/TSE.2008.35]

[8] Tabassum S, Minku LL, Feng D, Cabral GG, Song LY. An investigation of cross-project learning in online just-in-time software defect prediction. Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. 2020. 554–565. [doi: 10.1145/3377811.3380403]

[9] 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]

[10] 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.005713]

[11] 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]

[12] Wan ZY, Xia X, Hassan AE, Lo D, Yin JW, Yang XH. Perceptions, expectations, and challenges in defect prediction. IEEE Trans. on Software Engineering, 2020, 46(11): 1241–1266. [doi: 10.1109/TSE.2018.2877678]

[13] Dam HK, Tran T, Ghose A. Explainable software analytics. In: Proc. of the 40th Int'l Conf. on Software Engineering: New Ideas and Emerging Results. Gothenburg: ACM, 2018. 53–56. [doi: 10.1145/3183399.3183424]

[14] Jiarpakdee J, Tantithamthavorn CK, Dam HK, Grundy J. An empirical study of model-agnostic techniques for defect prediction models. IEEE Trans. on Software Engineering, 2022, 48(1): 166–185. [doi: 10.1109/TSE.2020.2982385]

[15] Jiarpakdee J, Tantithamthavorn CK, Grundy J. Practitioners' perceptions of the goals and visual explanations of defect prediction models. In: Proc. of the 18th IEEE/ACM Int'l Conf. on Mining Software Repositories. Madrid: IEEE, 2021. 432–443. [doi: 10.1109/MSR52588.2021.00055]

[16] 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.

[17] Tantithamthavorn C, Hassan AE. An experience report on defect modelling in practice: Pitfalls and challenges. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). Gothenburg: IEEE, 2018. 286–295.

[18] O'Brien RM. A caution regarding rules of thumb for variance inflation factors. Quality and Quantity, 2007, 41(5): 673–690. [doi: 10.1007/s11135-006-9018-6]

[19] Pecorelli F, Di Nucci D, De Roover C, De Lucia A. A large empirical assessment of the role of data balancing in machine-learning-based code smell detection. Journal of Systems and Software, 2020, 169: 110693. [doi: 10.1016/j.jss.2020.110693]

[20] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 2012, 13: 281–305.

[21] Dowd M, McDonald J, Schuh J. The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. London: Pearson Education, 2006.

[22] 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]

[23] Menzies T, Butcher A, Cok D, Marcus A, Layman L, Shull F, Turhan B, Zimmermann T. Local versus global lessons for defect prediction and effort estimation. IEEE Trans. on Software Engineering, 2013, 39(6): 822–834. [doi: 10.1109/TSE.2012.83]

[24] Zhang F, Hassan AE, McIntosh S, Zou Y. The use of summation to aggregate software metrics hinders the performance of defect prediction models. IEEE Trans. on Software Engineering, 2017, 43(5): 476–491. [doi: 10.1109/TSE.2016.2599161]

[25] 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]

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

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

[28] Wang S, Liu TY, Nam J, Tan L. Deep semantic feature learning for software defect prediction. IEEE Trans. on Software Engineering, 2020, 46(12): 1267–1293. [doi: 10.1109/TSE.2018.2877612]

[29] 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]

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

[31] Blei DM, Ng AY, Jordan MI. Latent dirichlet allocation. The Journal of Machine Learning Research, 2003, 3: 993–1022.

[32] Nguyen TT, Nguyen TN, Phuong TM. Topic-based defect prediction (NIER track). In: Proc. of the 33rd Int'l Conf. on Software Engineering. Honolulu: ACM, 2011. 932–935. [doi: 10.1145/1985793.1985950]

[33] Chen TH, Thomas SW, Nagappan M, Hassan AE. Explaining software defects using topic models. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories. Zurich: IEEE, 2012. 189–198. [doi: 10.1109/MSR.2012.6224280]

[34] 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 ACM/IEEE Int'l Conf. on Software Engineering. Leipzig: IEEE, 2008. 181–190. [doi: 10.1145/1368088.1368114]

[35] Aleithan R. Explainable just-in-time bug prediction: Are we there yet? In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 129–131. [doi: 10.1109/ICSE-Companion52605.2021.00056]

[36] Pham HV, Lutellier T, Qi WZ, Tan L. CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 1027–1038. [doi: 10.1109/ICSE.2019.00107]

[37] Xie XF, Ma L, Wang HJ, Li YK, Liu Y, Li XH. DiffChaser: Detecting disagreements for deep neural networks. In: Proc. of the 28th Int'l Joint Conf. on Artificial Intelligence. Macao, 2019. 5772–5778. [doi: 10.24963/ijcai.2019/800]

[38] Wang Z, Yan M, Chen JJ, Liu S, Zhang DD. Deep learning library testing via effective model generation. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Virtual Event: ACM, 2020. 788–799. [doi: 10.1145/3368089.3409761]

[39] Guo QY, Chen S, Xie XF, Ma L, Hu Q, Liu HT, Liu Y, Zhao JJ, Li XH. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 810–822. [doi: 10.1109/ASE.2019.00080]

[40] Nejadgholi M, Yang JQ. A study of oracle approximations in testing deep learning libraries. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 785–796. [doi: 10.1109/ASE.2019.00078]

[41] Zhang XF, Sun N, Fang CR, Liu JW, Liu J, Chai D, Wang J, Chen ZY. Predoo: Precision testing of deep learning operators. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Virtual: ACM, 2021. 400–412.

[42] Xie DM, Li YT, Kim M, Pham HV, Tan L, Zhang XY, Godfrey M. Leveraging documentation to test deep learning library functions. arXiv:2109.01002, 2021.

[43] Zhang YH, Chen YF, Cheung SC, Xiong YF, Zhang L. An empirical study on TensorFlow program bugs. In: Proc. of the 27th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Amsterdam: ACM, 2018. 129–140. [doi: 10.1145/3213846.3213866]

[44] Huang ZJ, Shao ZQ, Fan GS, Yu HQ, Yang XG, Yang K. Community smell occurrence prediction on multi-granularity by developer-oriented features and process metrics. Journal of Computer Science and Technology, 2022, 37(1): 182–206. [doi: 10.1007/s11390-021-1596-1]

[45] 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]

[46] 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: ACM, 2018. 908–911. [doi: 10.1145/3236024.3264598]

[47] Röder M, Both A, Hinneburg A. Exploring the space of topic coherence measures. In: Proc. of the 8th ACM Int'l Conf. on Web Search and Data Mining. Shanghai: ACM, 2015. 399–408. [doi: 10.1145/2684822.2685324]

[48] Wei X, Croft WB. LDA-based document models for ad-hoc retrieval. In: Proc. of the 29th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. Seattle: ACM, 2006. 178–185. [doi: 10.1145/1148170.1148204]

[49] McIntosh S, Kamei Y. Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. IEEE Trans. on Software Engineering, 2018, 44(5): 412–428. [doi: 10.1109/TSE.2017.2693980]

[50] 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]

[51] 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. Madrid: IEEE, 2018. 92–103. [doi: 10.1109/

ICSME.2018.00018]
[52] Kraemer HC, Morgan GA, Leech NL, Gliner JA, Vaske JJ, Harmon RJ. Measures of clinical significance. Journal of the American Academy of Child and Adolescent Psychiatry, 2003, 42(12): 1524–1529. [doi: 10.1097/00004583-200312000-00022]
[53] Agrawal A, Menzies T. Is "better data" better than "better data miners"? In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering. Gothenburg: IEEE, 2018. 1050–1061. [doi: 10.1145/3180155.3180197]
[54] 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]
[55] 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. Melbourne: IEEE, 2021. 407–418. [doi: 10.1109/ASE51524.2021.9678763]
[56] Yang XG, Yu HQ, Fan GS, Shi K, Chen LQ. Local versus global models for just-in-time software defect prediction. Scientific Programming, 2019, 2019: 2384706. [doi: 10.1155/2019/2384706]
[57] Jiarpakdee J, Tantithamthavorn C, Treude C. The impact of automated feature selection techniques on the interpretation of defect models. Empirical Software Engineering, 2020, 25(5): 3590–3638. [doi: 10.1007/s10664-020-09848-1]
[58] 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]
[59] Wang YX, Wang L, Hu H, Jiang J, Kuang HY, Tao XP. The influence of sponsorship on open-source software developers' activities on GitHub. In: Proc. of the 46th IEEE Annual Computers, Software, and Applications Conf. Los Alamitos: IEEE, 2022. 924–933. [doi: 10.1109/COMPSAC54236.2022.00144]

附中文参考文献:

[10] 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.005713]

葛建(1999－), 男, 硕士生, 主要研究领域为代码异味, 软件质量保障.

唐锏浩(1999－), 男, 硕士生, 主要研究领域为软件质量保障.

虞慧群(1967－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件工程, 可信计算, 云计算, 形式化方法.

黄子杰(1994－), 男, 博士生, CCF 学生会员, 主要研究领域为代码异味, 软件质量保障, 程序理解, 实证软件工程.

范贵生(1980－), 男, 博士, 副研究员, 博士生导师, CCF 专业会员, 主要研究领域为复杂软件系统的形式化方法, 服务计算, 软件架构分析技术.