软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2022,33(5):1587−1611 [doi: 10.13328/j.cnki.jos.006550]

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

基于软件度量的 Solidity 智能合约缺陷预测方法*

杨慧文[1], 崔展齐[1,4], 陈 翔[2], 贾明华[3], 郑丽伟[1], 刘建宾[1]

[1](北京信息科技大学 计算机学院, 北京 100101)

[2](南通大学 计算机科学与技术学院, 江苏 南通 226019)

[3](中央财经大学 信息学院, 北京 100101)

[4](网络文化与数字传播北京市重点实验室 (北京信息科技大学), 北京 100101)

通信作者: 崔展齐, E-mail: czq@bistu.edu.cn

摘 要: 随着区块链技术的兴起, 智能合约安全问题被越来越多的研究者和企业重视, 目前已有一些针对智能合约缺陷检测技术的研究. 软件缺陷预测技术是软件缺陷检测技术的有效补充, 能够优化测试资源分配, 提高软件测试效率. 然而, 目前还没有针对智能合约的软件缺陷预测研究. 针对这一问题, 提出了面向 Solidity 智能合约的缺陷预测方法. 首先, 设计了一组针对 Solidity 智能合约特有的变量、函数、结构和 Solidity 语言特性的度量元集 (smart contract-Solidity, SC-Sol 度量元集), 并将其与重点考虑面向对象特征的度量元集 (code complexity and features of object-oriented program, COOP 度量元集) 组合为 COOP-SC-Sol 度量元集. 然后, 从 Solidity 智能合约代码中提取相关度量元信息, 并结合缺陷检测结果, 构建 Solidity 智能合约缺陷数据集. 在此基础上, 应用了 7 种回归模型和 6 种分类模型进行 Solidity 智能合约的缺陷预测, 以验证不同度量元集和不同模型在缺陷数量和倾向性预测上的性能差异. 实验结果表明, 相对于 COOP 度量元集, COOP-SC-Sol 能够让缺陷预测模型的 *F*1-*score* 指标提升 8%. 此外, 进一步研究了智能合约缺陷预测中的类不平衡问题, 实验结果表明, 通过采样技术对数据集进行预处理能够提升缺陷预测模型的性能, 其中随机欠采样技术能够使模型的 *F*1-*score* 指标提升 9%. 在特定缺陷倾向性预测问题上, 模型的预测性能受到数据集类不平衡的影响, 在缺陷模块百分比大于 10% 的数据集中能取得较好的预测性能.

关键词: 软件缺陷预测; 缺陷数量预测; 缺陷倾向性预测; 智能合约; Solidity

中图法分类号: TP311

中文引用格式: 杨慧文, 崔展齐, 陈翔, 贾明华, 郑丽伟, 刘建宾. 基于软件度量的Solidity智能合约缺陷预测方法. 软件学报, 2022, 33(5): 1587–1611. http://www.jos.org.cn/1000-9825/6550.htm

英文引用格式: Yang HW, Cui ZQ, Chen X, Jia MH, Zheng LW, Liu JB. Defect Prediction for Solidity Smart Contracts Based on Software Measurement. Ruan Jian Xue Bao/Journal of Software, 2022, 33(5): 1587–1611 (in Chinese). http://www.jos.org.cn/1000-9825/6550.htm

Defect Prediction for Solidity Smart Contracts Based on Software Measurement

YANG Hui-Wen[1], CUI Zhan-Qi[1,4], CHEN Xiang[2], JIA Ming-Hua[3], ZHENG Li-Wei[1], LIU Jian-Bin[1]

[1](School of Computer, Beijing Information Science and Technology University, Beijing 100101, China)

[2](School of Computer Science and Technology, Nantong University, Nantong 226019, China)

[3](School of Information, Central University of Finance and Economics, Beijing 100101, China)

[4](Beijing Key Laboratory of Internet Culture and Digital Dissemination Research (Beijing Information Science and Technology University), Beijing 100101, China)

* 基金项目: 江苏省前沿引领技术基础研究专项 (BK202002001); 国家自然科学基金 (61702041); 北京信息科技大学“勤信人才”培育计划 (QXTCP C201906)

本文由“领域软件工程”专题特约编辑汤恩义副教授、江贺教授、陈俊洁副教授、李必信教授以及唐滨副教授推荐.

收稿时间: 2021-08-08; 修改时间: 2021-10-09; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

Abstract: With the rise of blockchain technology, more and more researchers and companies pay attention to the security of smart contracts. Currently, there are some studies on smart contract defect detection and testing techniques. Software defect prediction technology is an effective supplement to the defect detection techniques, which can optimize the allocation of testing resources and improve the efficiency of software testing. However, there is no research on software defect prediction for the smart contract. To address this problem, this study proposes a defect prediction method for Solidity smart contracts. First, it designs a metrics suite (smart contract-Solidity, SC-Sol) which considers the variables, functions, structures, and features of Solidity smart contracts, and SC-Sol is combined with the traditional metrics suite (code complexity and features of object-oriented program, COOP), which consider the object-oriented features, into COOP-SC-Sol metrics suite. Then, it extracts relevant metric meta-information from the Solidity code and performs defect detection to obtain the defects information to construct a Solidity smart contracts defect data set. On this basis, seven regression models and six classification models are applied to predict the defects of Solidity smart contracts to verify the performance differences of different metrics suites and different models for predicting the number and tendency of defects. Experimental results show that compared with the COOP, COOP-SC-Sol can improve the performance of the defect prediction model by 8% in terms of the *F*1-*score*. In addition, the problem of class imbalance in smart contract defect prediction is further studied. The result shows that the random under-sampling method can improve the performance of the defect prediction model by 9% in *F*1-*score*. In predicting the tendency of specific types of defects, the performance of the model is affected by the imbalance of data sets. Better performance is achieved in predicting the types of defects which the percentage of defect modules is greater than 10%.

Key words: software defect prediction; defect number prediction; defect tendency prediction; smart contract; Solidity

1 引 言

区块链是以比特币为代表的数字加密货币体系的核心支撑技术. 区块链技术的核心优势是去中心化, 为解决中心化机构存在的高成本、低效率以及数据存储不安全等问题提供了解决方案. 近年来, 区块链技术的研究与应用呈现爆发式增长态势, 政府部门、金融机构、科技企业和资本市场均在关注如何利用区块链技术解决可信价值传输、共享经济等实际问题[1,2].

智能合约的概念最早于 1994 年由 Szabo 提出[3], 定义为“一套数字形式的承诺书, 在这份承诺书上约定了合约的参与方可以执行的协议”[4], 但由于缺乏可信的执行环境, 难以获得参与者的信任, 因此很长一段时间没有得到关注. 区块链去中性化、可信任且不可篡改的基础架构, 为智能合约提供了运行环境, 使得智能合约技术随着区块链技术的发展受到了广泛关注[5], 已成为以太坊等主要区块链平台的核心构成要素. 智能合约使用算法和程序来编制合同条款, 是一种运行在区块链上且可按照规则自动执行的数字化协议[2], 具有状态和条件响应的属性, 可封装、验证或执行分布式节点复杂行为, 能够完成信息交换、价值转移和资产管理等功能[4]. 在触发预先规定好的条件时智能合约会自动执行, 在执行过程中不需要依赖于任何第三方的信任, 执行过程中事务状态公开透明.

与比特币脚本相比, 智能合约可以实现更复杂的循环或控制, 提高了代码的灵活性, 能够构建具有复杂业务逻辑的应用, 拓展了区块链的功能. 例如, 由于区块链的公开透明、不可篡改的特性, 使得智能合约成功应用于股权众筹[1]、金融交易[4]、保险[5]、选举投票[6]或知识产权认证[7]等金融及管理领域; 由于其去中心化、可追溯的特性, 使得智能合约在医疗信息存储[8]、智慧家居[9]等智慧医疗及物联网领域也取得了广泛应用.

智能合约在拓展区块链功能的同时, 其隐藏的缺陷也带来了潜在的安全风险. Luu 等人[10]使用 Oyente 工具对 19 366 个以太坊智能合约进行检测, 发现 8 833 个合约中至少存在一种安全漏洞, 表明智能合约中广泛存在缺陷. 智能合约中的缺陷可能会对财产造成巨大损失, 如: 2016 年 6 月, 大型众筹项目 TheDAO 的 360 万以太币被非法转移[11]等; 2017 年 11 月, Parity 钱包遭到攻击, 导致 15 万以太币被窃取[12]. 与传统软件不同, 对部署后的智能合约进行补丁修复十分困难[13,14]. 因此, 智能合约的质量保证技术受到了工业界和学术界的广泛关注.

目前针对智能合约的缺陷检测技术已有一些相关研究, 如 ContractGuard (https://contract.guardstrike.com) 使用模糊测试和符号执行对智能合约进行检测, 可以检测出 11 种严重缺陷以及 14 种警告; 区块链智能合约安全检测平台 (blockchain smart contract security checking system, BSCSCS)(http://www.sjtubsrc.net) 可以检测出分属 9 个类型的 21 种漏洞, BSCSCS 于 2018 年发布了《区块链智能合约安全审计白皮书》, 对相关漏洞的特征进行了描述; sFuzz[14]利用模糊测试方法, 改进了 AFL (American fuzzy lop)(https://lcamtuf.coredump.cx/afl) 的种子选择策略, 从

而覆盖更多的分支, 能够检测出时间戳依赖、危险外部调用和重入等 7 种常见的 Solidity 智能合约漏洞.

软件缺陷预测 (software defect prediction, SDP) 是缺陷检测技术的有效补充, 软件缺陷预测技术通过分析软件代码或开发过程, 设计与缺陷相关的度量元, 借助机器学习等方法, 预测软件模块的缺陷倾向性或缺陷数量, 根据预测结果优化缺陷检测资源的分配, 评价系统的测试充分程度, 以及作为软件是否可以交付的依据, 是软件质量保障的有效手段[15].

目前, 软件缺陷预测已在度量元选择、缺陷数据集预处理等领域取得了较多成果, 如陈翔等人[16]关注缺陷数量预测中非监督与监督学习方法的比较, Gong 等人[17]关注缺陷预测中数据集的类重叠, Bennin 等人[18]探讨了类不平衡现象对数据集的影响等. 但是, 目前在智能合约领域还没有缺陷预测的相关研究. 将软件缺陷预测技术应用于智能合约领域, 将面临以下挑战:

(1) 现有的软件度量元主要关注代码复杂性以及面向对象程序的特征, 缺少针对智能合约这一特殊软件制品设计的特征, 例如转移或接收货币、外部合约的调用或者特有函数等, 已有关于智能合约检测的相关研究表明这些特征与智能合约缺陷存在较强的相关性[10,19,20].

(2) 目前还没有面向智能合约缺陷预测的数据集, 难以验证已有方法在智能合约缺陷预测问题上是否仍可以取得令人满意的预测性能.

本文以目前应用最广泛的通用区块链平台——以太坊[1]所使用的 Solidity 智能合约为例, 提出了一种将软件缺陷预测技术应用到智能合约领域的方法. 首先, 设计了一组针对 Solidity 智能合约的度量元集 (metrics of smart contract-Solidity, SC-Sol 度量元集), 与关注代码复杂性和面向对象程序特征的度量元集 (metrics of code complexity and features of object oriented program, COOP 度量元集) 结合, 构成 COOP-SC-Sol 度量元集. 然后, 从 Solidity 智能合约代码中提取 COOP-SC-Sol 度量元集, 并使用 BSCSCS 对智能合约进行缺陷检测, 获取相应缺陷信息, 以构建 Solidity 智能合约缺陷数据集. 最后, 我们在该数据集中应用了 7 种回归模型和 6 种分类模型, 分别讨论了 Solidity 智能合约中缺陷数量预测和缺陷倾向性预测的相关问题. 实验结果表明, 相对于 COOP 度量元集, 本文提出的 SC-Sol 和 COOP-SC-Sol 度量元集能够有效提升智能合约缺陷预测模型的性能.

本文的主要贡献可总结如下:

(1) 提出一组针对 Solidity 智能合约的 SC-Sol 度量元集, 该度量元集关注了 Solidity 智能合约特有的变量、函数、结构以及 Solidity 语言特性, 与 COOP 度量元集相比, SC-Sol 能够更好地描述 Solidity 智能合约的特征, 提升缺陷预测结果的准确性.

(2) 根据 COOP-SC-Sol 度量元集, 以 Solidity 智能合约中的 contract 和 library 为粒度提取特征信息, 并结合 BSCSCS 提供的缺陷检测结果, 构建了 Solidity 智能合约缺陷数据集 (COOP-SC-Sol-Dataset, https://github.com/yagol2020/COOP-SC-Sol-Dataset), 可用于后续开展其他有关 Solidity 智能合约缺陷预测的研究工作.

(3) 基于 Solidity 智能合约缺陷数据集, 分别验证了不同模型在缺陷数量预测和缺陷倾向性预测问题中的性能差异. 此外, 对于缺陷倾向性预测问题, 还进一步分析了数据集内存在的类不平衡问题, 并分析了不同采样技术对缺陷预测模型性能的影响.

本文第 2 节为相关背景. 第 3 节介绍 COOP 度量元在 Solidity 智能合约中的应用, 并描述 SC-Sol 度量元集的设计. 第 4 节提出了面向 Solidity 智能合约的缺陷预测方法. 第 5 节针对所提出的研究问题设计实验, 并对实验结果进行分析. 第 6 节为有效性分析. 第 7 节介绍相关工作. 最后在第 8 节总结全文并对未来工作进行展望.

2 相关背景

为了更好地设计与描述针对 Solidity 智能合约的度量元, 本节对 Solidity 智能合约的语法及特性, 以及 COOP 度量元集的相关背景进行介绍.

2.1 Solidity 智能合约的语法及特性

Solidity 智能合约运行在以太坊区块链的虚拟机 (ethereum virtua machine, EVM) 上, 具有与面向对象语言 (object-oriented language, OOL) 不同的语法与特性. 为了更好地设计与描述针对 Solidity 智能合约的度量元, 本节

将介绍 Solidity 智能合约的运行机制、语法以及特性.

2.1.1 运行机制与语法

Solidity 智能合约的运行机制如图 1 所示. 以太坊是基于账户的公有区块链平台, Solidity 智能合约可通过 EVM 与状态数据库 (StateDB) 交互, 如查询或修改状态数据等, 其中, EVM 与 JVM (Java virtual machine) 相似, 是一种基于堆栈的沙盒环境. 部署 Solidity 智能合约时, 源代码首先被编译器 (如 solc) 编译为 EVM 可读取的字节码, 然后 EVM 将基于字节码构建相应的 EVM 堆栈. 当合约被调用时, EVM 根据函数签名等信息执行堆栈, 从而实现与 StateDB 的交互.

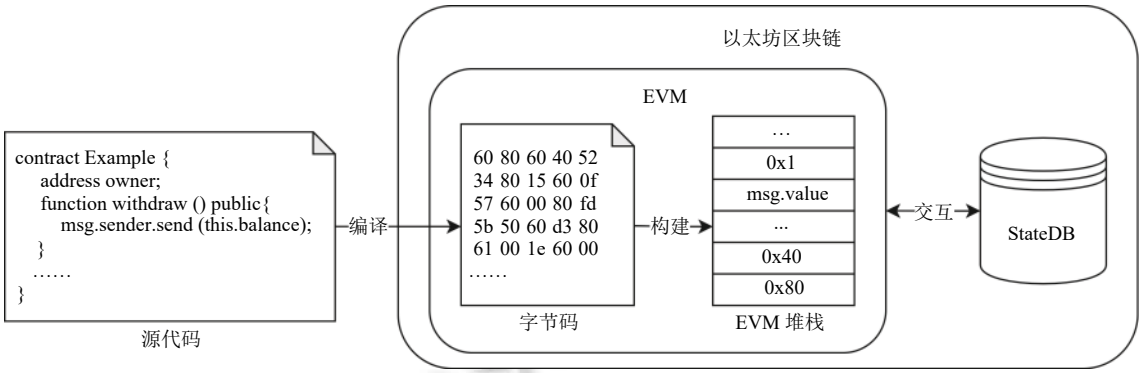


图 1 Solidity 智能合约运行机制

Solidity 智能合约的源代码结构由任意数量的导入指令、编译指令和合约定义组成. 导入指令的关键字为 import, 可用于将其他合约代码导入到当前合约中; 编译指令的关键字为 pragma, 可用于指定编译器的版本, 从而避免因编译器指令集问题导致的错误; 合约定义包括 contract (合约) 和 library (库). 其中, contract 是 Solidity 智能合约的核心部分, 用于存储重要的数据或通过函数实现业务逻辑. library 不能存储数据或接收以太币, 当 contract 调用 library 中的函数时, 该函数将在 contract 的上下文中执行, 从而修改 contract 中的状态变量, 此外, library 还可拓展基础变量类型.

合约由状态变量、结构体、枚举、函数修饰器、事件和函数组成, 其名称和描述如表 1 所示. 状态变量用于永久存储合约中的数据, 通常状态变量存储合约管理者、合约参与者或代币价格等信息. 结构体和枚举与 OOL 相似, 前者组合已有变量类型定义新的类型变量, 后者用于定义有限的数据集合, 并可转换为整型数据. 函数修饰器是 Solidity 智能合约特有的结构, 能够改变函数语义, 带有函数修饰器的函数需要先执行修饰器的内容, 例如在执行函数之前检查某个条件[13]. 事件可以调用 EVM 日志功能的接口, 将合约运行情况反馈给调用者. 函数是 Solidity 智能合约中的可执行单元, 函数定义由 function 关键字、函数名、参数列表、可见标识符、返回值和函数修饰器组成. 其中, function 关键字、函数名和参数列表必须在函数定义中声明, 其余部分如未显式给出, 则使用默认值. 例如可见标识符默认为 public, 返回值和函数修饰器默认为空. 函数按照函数体中代码的先后顺序依次执行, Solidity 智能合约支持 while、do while 和 for 这 3 种循环控制逻辑, 以及 break 和 continue 两种终止条件.

表 1 合约结构

名称	描述
状态变量 (state variables)	永久存储合约中的数据
结构体 (struct)	组合已有变量类型定义新类型
枚举 (enum)	定义有限的数据集合
函数修饰器 (function modifiers)	改变函数语义
事件 (event)	调用EVM日志功能接口
函数 (function)	可执行单元

2.1.2 特 性

Solidity 智能合约部署环境为以太坊区块链, 因此在调用方式、消耗资源方式和存储方式等均与传统软件存在差异, 如具有 gas、回调和不支持浮点类型等特性.

gas 是以太坊执行智能合约时收取的手续费. 智能合约中的每一个操作, 包括合约创建、函数调用以及数据存储等, 均需要向以太坊支付 gas. 如果在合约调用过程中耗光 gas, 会触发 out-of-gas 异常, 导致此次调用所做的状态修改被回滚, 并返还以太币给调用方.

回调 (fallback) 是 Solidity 智能合约中的一种特性. 回调函数是一个没有函数名、参数和返回值的特殊函数. 当外部想要调用的函数在合约中没有给定的函数标识符匹配 (即不存在所调用的函数名和参数列表), 或合约收到以太币且没有任何数据时, 会调用回调函数. 为正常接收以太币, 合约必须实现回调函数, 且标记为 payable.

不支持浮点类型是 Solidity 智能合约开发过程中需要注意的问题, 如果整型之间进行相除运算, 其运算结果会被截断. 在后续版本中, Solidity 编译器将支持浮点类型, 但由于 Solidity 智能合约中可以指定编译版本号, 因此整型相除的截断问题仍会存在风险.

2.2 COOP 度量元集

Marian 等人[21]基于面向对象程序度量等相关工作提出了 20 种关注代码复杂度以及面向对象程序特征的度量元, 本文将这 20 种度量元称为 COOP 度量元集. COOP 度量元集的类别、缩写、全称与描述如表 2 所示[22]. 表 2 中, WMC、DIT、NOC、CBO、RFC、LCOM、IC、CBM 和 AMC 由 Chidamber、Kemerer[23]和 Tang 等人[24]提出, 这组度量元主要关注面向对象特征. NPM、DAM、MOA、MFA 和 CAM 由 Bansiy 和 Davis[25]提出, 这组度量元主要关注聚合、抽象等特征. LCOM3[26]、CA、CE[27]、Max_cc 和 Avg_cc[28]分别由 Sellers、Martin 以及 McCabe 等人提出, 主要关注方法凝聚度、类间耦合度和圈复杂度.

表 2 COOP 度量元集

类别	缩写	全称	描述
复杂性	LOC	Lines of code	代码行数
	WMC	Weighted methods per class	类中加权方法
	NPM	Number of public methods	公有方法数量
	AMC	Average method complexity	方法平均复杂度
	Max_cc	Max McCabe's cyclomatic complexity	最大圈复杂度
	Avg_cc	Avg McCabe's cyclomatic complexity	平均圈复杂度
	MOA	Measure of aggregation	聚合程度
耦合	CBO	Coupling between object classes	类间耦合度
	RFC	Response for a class	类中响应数
	CA	Afferent couplings	传入耦合
	CE	Efferent couplings	传出耦合
	IC	Inheritance coupling	继承耦合
	CBM	Coupling between methods	方法间的耦合
内聚	LCOM	Lack of cohesion in methods	方法凝聚度
	LCOM3	Lack of cohesion in methods	方法凝聚度
	CAM	Cohesion among methods of class	类间凝聚度
抽象	DIT	Depth of inheritance tree	继承树深度
	NOC	Number of children	子类个数
	MFA	Measure of functional abstraction	方法抽象度
封装	DAM	Data access metric	数据访问指标

3 Solidity 智能合约度量元设计

度量元的质量是决定缺陷预测性能最重要的因素之一. 缺陷预测领域经常使用的度量元有 CK 度量元、

McCabe 度量元等. 但这些度量元主要关注代码复杂度以及面向对象特征这两方面, 因此本文针对 Solidity 智能合约设计了 SC-Sol 度量元集.

3.1 COOP 度量元集在 Solidity 智能合约中的应用

COOP 度量元集的模块粒度是面向对象语言中的“类”, 在 Solidity 中, 与“类”相似的是合约 (contract) 和库 (library), 故我们将 COOP 度量元的模块粒度调整为 contract 和 library, 以下统称为合约或合约粒度. 需要注意的是, Solidity 语言中也存在继承、耦合等面向对象语言的特性, 因此 COOP 度量元集也可作为 Solidity 语言的部分度量元.

如图 2(a) 和图 2(b) 分别为用 Java 和 Solidity 编写的具有相同代码结构的代码片段, Solidity 中的合约 (contract) 和库 (library) 与 Java 等面向对象语言体系中的类 (class) 相似, Solidity 中的函数 (function) 可以类比看作 Java 等面向对象语言中的方法 (method). COOP 度量元在 Solidity 智能合约中的映射关系及描述如表 3 所示.

```
class Temp1 {
    int value=10;
    public void method1 () {
    }
}
class Temp2 extents Temp1{
    public void method2 () {
    }
}
```

(a) Java

```
contract Temp1 {
    uint value=10;
    function fun1 () public returns () {
    }
}
contract Tem2 is Temp1{
    function func2 () public returns () {
    }
}
```

(b) Solidity

图 2 使用 Java 语言和 Solidity 语言编写的具有相同结构代码片段

表 3 COOP 度量元在 Solidity 智能合约中的映射关系及描述

COOP度量元集	COOP度量元集映射	描述
WMC (weighted methods per class)	WFC (weighted functions per contract)	合约中加权函数值
NPM (number of public methods)	NPF (number of public functions)	公有函数数量
AMC (average method complexity)	AFC (average function complexity)	函数平均复杂度
CBO (coupling between object classes)	CBC (coupling between contract)	合约间耦合度
RFC (response for a class)	RFC (response for a contract)	合约中响应数
CBM (coupling between methods)	CBF (coupling between functions)	函数间的耦合
LCOF (lack of cohesion in methods)	LCOM (lack of cohesion in functions)	函数凝聚度
LCOF3 (lack of cohesion in methods)	LCOM3 (lack of cohesion in functions)	函数凝聚度
CAM (cohesion among methods of class)	CAF (cohesion among functions of contract)	合约间凝聚度
NOC (number of children)	NOC (number of children)	子合约个数

3.2 SC-Sol 度量元集的设计

为了更好地说明 Solidity 度量元的设计, 本节首先介绍 Solidity 智能合约中用于转移以太币、获取事务信息或区块链信息的特有变量与函数, 以及 Solidity 智能合约中已经被发现的漏洞模式. 然后, 基于 Solidity 智能合约特征和漏洞模式, 提出了针对 Solidity 智能合约的度量元集.

3.2.1 变量和函数

Solidity 智能合约中特有的变量类型及其函数可用于调用 EVM 的底层指令, 或通过全局变量的属性获取所在区块和当前合约的信息. 部分 Solidity 智能合约特有的变量类型、全局变量、函数以及属性的描述如表 4 所示.

表 4 中第 1 行的 address 为地址类型变量, 可以实现 Solidity 智能合约之间的调用、货币查询以及转移等核心功能, address 储存一个 20 字节的值, 与以太坊的地址对应, 底层的 EVM 会将 address 变量类型中的值与实际地

址对应, 从而实现具体的功能. address 变量类型含有 1 个属性和 4 个成员函数, 具体的属性和函数如表 4 所示. 其中, address.balance 属性可以获得指定合约中存储的 wei 的数量, 其中 wei 是以太坊的货币单位. transfer、send、call 和 delegatecall 这 4 个成员函数可用来实现货币转移的功能, 区别在于以下 3 点.

表 4 Solidity 智能合约中部分特有的变量类型、全局变量、函数以及属性

变量类型/全局变量	成员函数/属性	参数列表	描述
address	balance	—	查询指定地址的余额, 单位为wei
	transfer	uint256 amount	向指定地址转移amount数量的wei, 失败则抛出异常并终止
	send	uint256 amount	向指定地址转移amount数量的wei, 失败返回false, 继续执行指令
	call	任意类型, 任意数量	EVM底层调用, 调用环境为外部合约
	delegatecall	任意类型, 任意数量	EVM底层调用, 调用环境为当前合约
block	timestamp	—	获得当前区块的时间戳
	number	—	获得当前区块的块号
now	—	—	获得当前区块的时间戳
tx	origin	—	获得调用链的发起者地址
—	require	bool condition, [string message]	判断condition是否满足, 如果不满足则撤销状态更改, 同时提供message错误信息
—	gasleft	—	获得当前合约剩余的gas数量

(1) 附带的 gas 数量不同

transfer 和 send 函数在转移的同时附带的 gas 数量固定, 为 2 300, call 和 delegatecall 函数可以通过 address.call.gas(uint gasAmount)() 和 address.delegatecall.gas(uint gasAmount)() 指定 gas 数量, 如果没有指定, 则默认发送所有可用 gas.

(2) 出现调用错误后的返回值不同

transfer 函数在调用出现错误后没有返回值, 抛出异常并终止合约执行; send、call 和 delegatecall 函数在调用出现错误后返回 false, 继续执行合约.

(3) 调用所处的上下文环境不同

delegatecall 函数与 call 函数相似, 区别在于 delegatecall 函数会在被调用合约的上下文执行, 例如合约 A 通过 delegatecall 函数调用合约 B 的 func, 则 func 中的 this 会指向合约 A.

表 4 中第 2 行的 block 变量由全局命名空间提供, 是一种特殊的全局变量, 通常用于获得区块上的信息, 例如通过 block.timestamp 属性可以获得当前区块的时间戳, block.number 属性可以获得当前区块的块号.

表 4 中第 3 行的 now 是一个全局变量, 与 block.timestamp 属性相似, 均可用于获得当前区块的时间戳.

表 4 中第 4 行的 tx 变量由全局命名空间提供, 是一种特殊的全局变量, 通常用于获得调用链上的信息, 例如通过 tx.origin 属性可以得到调用链的交易发起者的地址, 即整条调用链的发起账户或合约的地址. Solidity 智能合约中, 合约 A 可以调用合约 B 中的函数, 在合约 B 中仍可以继续调用合约 C, 从而形成一条调用链.

表 4 中第 5、6 行的 require 和 gasleft 是 Solidity 智能合约中特有的 2 个函数, 其中:

require 函数可用于检查由输入或外部组件引起的错误. require 函数的参数有两种形式, 接受一个布尔参数, 或一个布尔参数和一个字符串. 当条件不满足时 (即布尔参数为 false) 抛出异常, 同时撤销状态更改. 当 require 接受两个参数时, 会同时提供一个错误信息. Solidity 智能合约中使用状态恢复异常来处理错误, 当发生异常时, 将撤销所有的状态更改, 同时向调用者标记错误.

gasleft 函数可以获得当前合约剩余的 gas 数量.

3.2.2 Solidity 漏洞模式

Solidity 智能合约的漏洞通常会造成合约中以太币受到损失, 本节总结了 Solidity 智能合约中已知的常见漏

洞, 并标记了其中的关键变量和函数.

未进行异常和返回值处理 (send、call、delegatecall): send、call 和 delegatecall 函数的返回值在调用错误时会返回 false, 同时继续执行合约中的指令, 如果开发者没有对这 3 个函数进行异常或返回值的处理, 可能会导致代码逻辑错误[19,29].

重入漏洞 (address、call): 重入漏洞是 2016 年 TheDAO 事件中被攻击者利用的漏洞, 重入漏洞与回调机制相关, 如图 3 所示的合约, 由于使用了 call 函数, 且没有指定 gas 的数量, 因此此次货币的转移会附加所有可用的 gas, 如果恶意攻击者构建一个合约, 在 fallback 函数中回调 pay 函数, 由于 balances[msg.sender] 的值还没有被清零, 因此 Example 合约中的货币会被不断地窃取, 从而导致经济损失.

```
contract Example {
    mapping (address =>uint) private balances;
    function pay () {
        require (msg.sender.call.value (balances [msg.sender])());
        balances [msg.sender]=0;
    }
}
```

图 3 存在重入漏洞的 Solidity 智能合约代码片段

空地址调用 (address): Solidity 编译器不会检查 call 函数的调用对象地址是否存在, 如果地址不存在, 则以太坊会自动创建地址合约, 但由于这个合约不属于任何人, 合约中的以太币无法被转移出来, 导致以太币的丢失[30].

危险的外部调用 (delegatecall): 不受控制的 delegatecall 函数会被恶意攻击者利用, 从而使被攻击合约执行恶意代码. Parity 合约遭受的攻击与 delegatecall 函数相关, 在这次攻击中, 攻击者发现 Parity 合约中的 delegatecall 函数能够任意地被外部控制, 因此攻击者通过编写了一个经过设计的合约, 从而转移了 Parity 中的价值 3 千万美元的以太币[19].

整型相除运算的截断问题 (除号): 如第 2.1.2 节所述如果开发人员不注意截断问题, 在进行乘法运算前或在发送以太币操作前进行除法运算, 可能会造成错误. BSCSCS 发现乘法前进行除法运算 (DivisionBefore Multiply) 和发送以太币前进行除法运算 (DivisionBeforeCallvalue) 两种漏洞与整型相除运算相关.

tx.origin 相关漏洞 (tx.origin): 当开发者使用 tx.origin 作为转移货币的判断条件时, 恶意攻击者可以通过构造一个经过设计的调用链, 从而窃取合约中的以太币[31−33]. BSCSCS 中发现 UseOfOrigin 类型漏洞与该变量有关.

不当依赖漏洞: 不当依赖漏洞指开发者利用区块或合约中提供的信息, 作为货币转移操作的条件, 但这些信息能够被攻击者预测或从外部控制, 从而导致攻击者能够绕过货币转移操作的条件, 窃取合约的货币.

gasleft 函数依赖 (gasleft): 由于 Solidity 难以实现随机数的生成, 因此开发者会使用自创随机数算法和自定义 seed, 例如使用合约剩余的 gas 的数量, 导致生成的随机数可能被攻击者预测和利用.

时间戳依赖 (block.timestamp、now): 在以太坊中, 区块的时间戳为挖出该区块的矿工的本地系统时间, 但以太坊允许矿工在一定范围内修改区块的时间戳. 如果开发人员使用区块的时间戳作为转移以太币或其他关键操作的条件, 那么恶意攻击者 (即挖出该区块的矿工) 可以通过改变时间戳的值使条件满足, 从而获得不当利益, 甚至破坏合约[10].

区块号依赖 (block.number): 与时间戳相似, 区块的块号可被矿工操作. 如果开发者使用区块的块号作为随机数种子生成随机数, 则恶意攻击者可以通过操作块号生成攻击者期望的数字, 导致合约中的以太币被窃取[19].

3.2.3 SC-Sol 度量元集

SC-Sol 是本文针对 Solidity 智能合约特有的变量、函数、结构以及 Solidity 语言特性所设计的度量元集. SC-Sol 度量元集的类别、缩写、全称与描述如表 5 所示. 表 5 中的 SC-Sol 度量元集可划分为地址相关、结构相关、异常处理、参数使用、易错操作和不当依赖 6 个类型, 其中各个度量元的粒度均为合约粒度, 具体分类与各个度量元的解释如下.

表 5 SC-Sol 度量元集

类别	缩写	全称	描述
地址相关	NATV	Number of address type variables	address类型变量数量
	NTF	Number of transfer function	transfer函数数量
	NSF	Number of send function	send函数数量
	NCF	Number of call function	call函数数量
	NDF	Number of delegatecall function	delegatecall函数数量
结构相关	NS	Number of struct	结构体数量
	NM	Number of modifier	函数修饰器数量
	NE	Number of event	event数量
异常处理	NRF	Number of require function	require函数数量
参数使用	TNP	Total number of parameter	参数总数量
	NUAP	Number of using address type parameters	使用address类型参数次数
	TNMSV	Total number of modify state variables	对状态变量修改总次数
易错操作	NDO	Number of division operations	除号数量
	NOP	Number of origin property	tx.origin数量
不当依赖	TNGF	Total number of gasleft function	gasleft函数数量
	TNTD	Total Number of Time Dependence	与时间相关属性或变量数量
	TNBND	Total Number of Block Number Dependence	block.number数量

(1) 地址相关度量元

NATV, 该度量元统计状态变量中 address 类型变量的数量.

NTF, 该度量元统计 transfer 函数的数量.

NSF, 该度量元统计 send 函数的数量.

NCF, 该度量元统计 call 函数的数量.

NDF, 该度量元统计 delegatecall 函数的数量.

(2) 结构相关度量元

NS, 该度量元统计结构体的数量. 合约中使用结构体的数量越多, 代码复杂性和逻辑复杂性越高, 通常出现缺陷的可能性越大.

NM, 该度量元统计函数修饰器 (modifier) 的数量.

NE, 该度量元统计 event 的数量.

(3) 异常处理

NRF, 该度量元统计所有函数调用 require 函数的次数总和.

(4) 参数使用

TNP, 该度量元统计各个函数参数的总数量. Solidity 智能合约中, 各个合约之间通过函数调用实现业务功能, 如以太币的转移或信息交互等, 所以合约中的函数参数应被关注.

NUAP, 该度量元统计各个函数中使用 address 类型参数的次数. 当函数使用 address 类型的参数时, 实现的功能通常与以太币的数量变动有关, 因此 address 类型的参数应被更加关注.

TNMSV, 该度量元统计各个函数对状态变量的修改的次数总和. 合约中的状态变量所存储的值会永久存储在合约中, 状态变量通常存储重要信息, 如: 拥有者地址、参与者地址等, 不恰当的状态变量修改可能会导致缺陷.

如图 4 所示, 本文以地址为 0x0dd1093721997be0421f49597820df3bfd1dc755 (https://etherscan.io/address/0x0dd1093721997be0421f49597820df3bfd1dc755#code) 的 Solidity 合约改写的代码片段为例, 介绍该组度量元的计算方式. 该例中 TNP 的值为 1, 因为函数 func1 的参数列表中只含有 1 个参数; NUAP 的值为 3, 因为 func1 中有 3 个函数调用了 address 类型参数, 即 newOwner; TNMSV 的值为 1, 因为 func1 中最后一行对状态变量进行了修改,

修改次数为 1 次.

(5) 易错操作

NDO, 该度量元统计各个函数使用除号的数量.

NOP, 该度量元统计 tx.origin 属性的数量.

(6) 不当依赖

TNGF, 该度量元统计 gasleft 函数的数量.

TNTD, 该度量元统计合约中与时间相关的属性或变量数量, 即使用 block.timestamp 属性和 now 变量的数量.

TNBND, 该度量元统计使用 block.number 属性的数量.

```
contract Ownable {
  address public owner;
  event event1 (address indexed previousOwner, address indexed newOwner);
  modifier modifier1 () {
    require (msg.sender==owner);
    _;
  }
  function func1 (address newOwner) onlyOwner public {
    require (newOwner!=address (0));
    OwnershipTransferred (owner, newOwner);
    owner=newOwner;
  }
}
```

图 4 Solidity 源代码示例

4 基于软件度量的 Solidity 智能合约缺陷预测方法

基于所设计的 COOP-SC-Sol 度量元集, 本文提出了基于软件度量的 Solidity 智能合约缺陷预测方法. 首先根据 COOP-SC-Sol 度量元集对 Solidity 源代码进行代码度量, 提取代码模块的度量元信息, 然后为每个代码模块标记是否存在缺陷以及缺陷数量, 以构建缺陷数据集. 最后, 分别构建回归模型以及分类模型预测 Solidity 智能合约的缺陷数量和缺陷倾向性, 并评估模型的性能.

本节将详细介绍度量元提取, 模型构建以及评估.

4.1 度量元提取

由于目前还没有针对 Solidity 智能合约的缺陷数据集, 因此首先需要构建缺陷预测数据集. 我们从 XBlock (http://xblock.pro/ethereum/) 获取 Solidity 智能合约的源码, 使用 solidity-parser-antlr (https://github.com/federicobond/solidity-parser-antlr) 获得 Solidity 智能合约中的 AST, 利用 AST 提取 COOP-SC-Sol 度量元集. 具体来讲, solidity-parser-antlr 提供了 visitor 访问器, 并提供了 60 种可供查询访问的 AST Node 类型. 本方法根据 COOP-SC-Sol 度量元的定义, 组合使用不同的 AST Node 类型对 Solidity 源代码文件的 AST 信息进行提取, 整理后得到了每个源代码中 contract 和 library 的度量元信息.

算法 1 描述了度量元提取的过程, 输入为 Solidity 源代码 *C* 和度量元提取方法集合 *MEMS* (metric extraction methods set, 度量元提取方法集合), *MEMS* 中含有预先编写的提取度量元的方法. 首先使用 solidity-parser-antlr 从源代码中获得 contract 和 library 的 AST 信息, 记为 *contractNodes*, 然后, 遍历每个 *contractNode*, 根据集合 *MEMS* 中的提取方法 getMetric 获得度量元信息 *metricInfo*, 所有度量元信息 *metricInfo* 汇总成为该 contract 或 library 的

度量元信息集合 *contractMetricInfo*, 如此遍历获得每个 *contractNode* 的度量元信息, 最终汇总并返回整个 Solidity 源代码 *C* 中的度量元信息集合 *codeInfoResult*.

算法 1. 度量元提取.

输入: Solidity 源代码 *C*、度量元提取方法集合 *MEMS*;
输出: *C* 中的度量元信息集合 *codeInfoResult*.

```
1 contractNodes = parser(C)
2 codeInfoResult = NullList
3 foreach contractNode in contractNodes:
4    contractMetricInfo = Null
5    foreach extractMethod in MEMS:
6       metricInfo = extractMehod.getMetric(contractNode, contractNodes)
7       contractMetricInfo += metricInfo
8    codeInfoResult += contractMetricInfo
9 return codeInfoResult
```

4.2 预测模型构建与评估

缺陷预测领域根据预测的粒度不同可分为两类, 一类为缺陷倾向性预测, 另一类为缺陷数量预测. 本文研究的对象是一种新型软件制品, 还没有有效的智能合约缺陷预测的相关工作, 为了寻找最优预测模型, 本文使用软件缺陷预测领域中综述或实证研究总结的 6 种二分类模型和 7 种回归模型分别作为 Solidity 智能合约的倾向性预测和缺陷数量预测模型[15,17,18,34,35].

4.2.1 缺陷倾向性预测模型及评估

本文使用的缺陷倾向性预测模型有: 伯努利贝叶斯分类器 (Bernoulli Native Bayes, BNB)、高斯贝叶斯分类器 (Gaussian Native Bayes, GNB)、K 邻近分类器 (K neighbors classifier, KNNC)、决策树分类器 (decision tree classifier, DTC)、随机森林分类器 (random forest classifier, RFC) 和支持向量机分类器 (support vectors machines classifier, SVC).

对缺陷倾向性预测的实验结果使用 *F*1-*score* 和 *g-mean*[36]作为预测模型的评估指标.

为了计算 *F*1-*score*, 需要使用如表 6 所示的混淆矩阵对样本的预测结果进行评价, 其中正样本被正确预测称为 *TP*, 负样本被错误预测称为 *FP*, 负样本被正确预测称为 *TN*, 正样本被错误预测称为 *FN*.

根据表 5 中的混淆矩阵, 召回率 *Recall* 的计算公式如公式 (1) 所示.

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

表 6 二分类问题的混淆矩阵

真实样本	预测样本	
	正	负
正	*TP*	*FN*
负	*FP*	*TN*

精确率 *Precision* 的计算公式如公式 (2) 所示.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

二分类模型应达到较高的召回率和精确率, 当同时考虑召回率和精确率时, 可以通过公式 (3) 计算 *F*-*score*. 其中, $\beta = 1$时, 称为 *F*1-*score*. *F*1-*score* 的值越高, 代表预测性能越好.

$$F\text{-}score=\frac{\left(1+\beta^2\right)\times Precision\times Recall}{\left(\beta^2\times Precision\right)+Recall} \tag{3}$$

除此之外, 本文还使用 *g-mean* 评估类不平衡数据集对模型性能的影响, *g-mean* 是非线性的评估指标, 综合考虑了正样本和负样本分类正确的概率, 对类不平衡问题具有鲁棒性, 常用于评估类不平衡数据集对模型的影响[36]. *g-mean* 的计算公式如 (4) 所示. *g-mean* 的值越高, 代表预测性能越好.

$$g\text{-}mean=\sqrt{\frac{TP}{FN+TP}\times\frac{TN}{TN+FP}} \tag{4}$$

4.2.2 缺陷数量预测模型及评估

本文使用的缺陷数量预测模型有: 线性回归 (linear regression, LR)、贝叶斯岭 (Bayesian ridge, BR)、决策树回归 (decision tree regressor, DTR)、随机森林回归 (random forest regressor, RFR)、K 邻近回归 (K neighbors regressor, KNNR)、梯度加速回归 (gradient boosting regressor, GBR) 和支持向量机回归 (support vector machines regressor, SVR).

对缺陷数量预测的实验结果使用平均绝对误差 (mean absolute error, *MAE*)[37]和平均缺陷百分比 (fault percentile average, *FPA*)[34,38]作为预测模型的评估指标.

MAE 计算方式如公式 (5) 所示.

$$MAE=\left(\frac{1}{n}\right)\sum_{i=1}^{n}\left|y_{\text{pred}}-y_{\text{true}}\right| \tag{5}$$

其中, n 代表样本总数量, y_{pred} 代表预测的缺陷数量, y_{true} 代表真实含有的缺陷数量, *MAE* 的值越低, 代表预测性能越好.

FPA 关注预测缺陷数量的排序是否与真实缺陷数量的排序相同或相似, 并以加权的形式计算准确程度. *FPA* 的计算公式如公式 (6) 所示.

$$FPA=\frac{1}{K\times N}\sum_{m=1}^{K}\sum_{i=K-m+1}^{K}n_i \tag{6}$$

其中, K 为样本总数量, N 为缺陷总数量. 将 K 个样本的预测结果按照缺陷数量升序排列, 将排序后的模块次序记为 $1,2,3,\ldots,n$, 使用 n_i 代表排序后第 i 个样本的真实缺陷个数. *FPA* 的值越高, 代表预测性能越好.

5 实验设计与结果分析

5.1 实验设计

为验证所提出 Solidity 智能合约缺陷预测方法的有效性, 本文提出了 4 个研究问题.

RQ1: 若考虑智能合约缺陷倾向性预测, COOP 度量元集、SC-Sol 度量元集与 COOP-SC-Sol 度量元集之间的性能差异如何?

RQ2: 针对数据集内存在的类不平衡问题, 不同采样技术对 Solidity 智能合约缺陷倾向性预测性能的提升效果如何?

RQ3: Solidity 智能合约缺陷倾向性预测在针对特定缺陷预测时的性能如何?

RQ4: 若考虑智能合约缺陷数量预测, COOP 度量元集、SC-Sol 度量元集与 COOP-SC-Sol 度量元集之间的性能差异如何?

本文设计的实验框架如图 5 所示. 所有实验均运行在 CPU 为 i7-6700H, 内存为 8 GB 的 64 位 Windows 10 操作系统的计算机上. 开发和运行环境为 Python 3.8, 模型的实现使用 Python 开源机器学习库 scikit-learn (https://scikit-learn.org/), 采样方法的实现使用 Python 第三方库 imbalanced-learn (https://github.com/scikit-learn-contrib/imbalanced-learn.git), 模型和采样方法的超参数均为缺省值.

5.1.1 智能合约缺陷数据集构建

由于目前还没有针对 Solidity 智能合约的缺陷数据集, 因此合约的缺陷信息需要从其他途径 (例如: 检测平台

或检测工具) 获得. 本文从 XBlock 中获得 Solidity 源代码文件, 文件名为该 Solidity 合约在以太坊中的地址. 实验中使用 BSCSCS 作为 Solidity 智能合约缺陷信息的来源. 需要注意的是 BSCSCS 是在线检测平台, 用户提交合约地址或者源代码至后台服务器, 服务器对源代码进行检测后将检测结果通过网页返回给用户, 由于性能、资源以及其他原因, BSCSCS 无法提供所有合约的检测结果, 因此部分合约无法得到缺陷检测结果. 在 149 363 个源代码文件中, 共有 6 519 个源代码文件可以获得 BSCSCS 检测结果. 经过处理后, 将这部分检测结果作为缺陷预测数据集的标签.

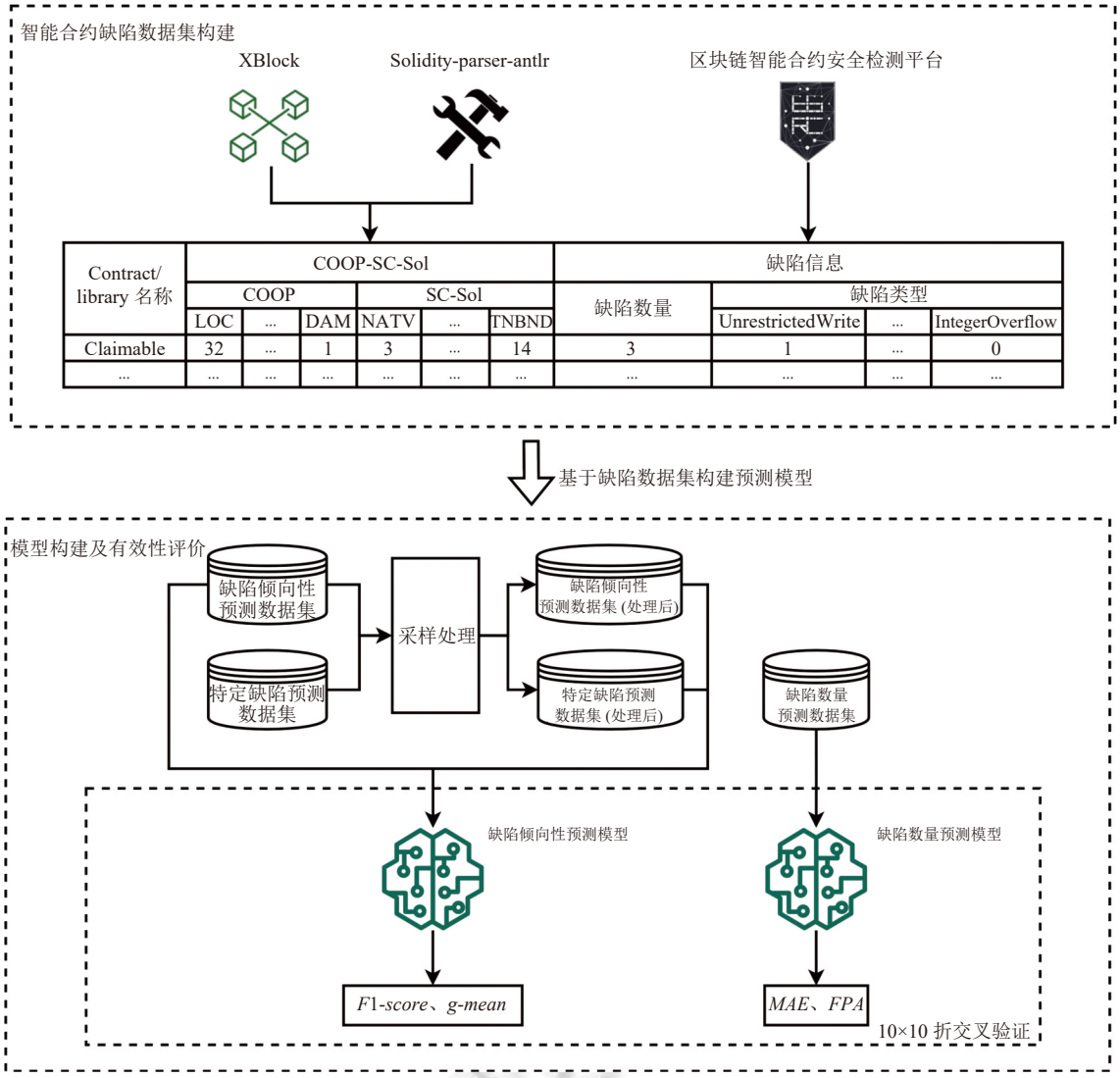


图 5 实验框架

具体来讲, 首先提取 XBlock 提供的源代码文件名作为以太坊智能合约地址, 将地址输入至 BSCSCS, BSCSCS 返回检测结果, 根据检测结果中缺陷所在行号确定缺陷所属的 contract 或 library. 然后, 将检测报告中“违背项”和“警告项”作为缺陷, 将以上 2 项出现的次数作为缺陷数量. 对于缺陷倾向性标签, 本实验参考文献 [37] 的方法, 将缺陷数量二值化, 即如果缺陷的数量大于等于 1, 则标记为存在缺陷, 否则标记为不存在缺陷. 表 7 为 BCSCSC 支持检测的缺陷名称以及分类, 根据检测报告中提供的缺陷名称, 即 call 安全漏洞、条件竞争漏洞、重入漏洞、权

限控制漏洞、事务顺序依赖漏洞、冻结账户漏洞、逻辑设计缺陷漏洞、随机数生成漏洞和数值溢出漏洞, 在每种类别的标签中标记是否存在该类型的缺陷, 将此标签称为特定缺陷标签.

表 7 BCSCSC 支持检测的缺陷名称以及分类

缺陷分类	缺陷名称
call安全漏洞	DAOMethodCall DelegateCallWithUserInput UnsafeCallTarget
条件竞争漏洞	TODAmount TODReceiver TODTransfer
重入漏洞	DAO DAOConstantGas
权限控制漏洞	UnprivilegedSuicide UnrestrictedEtherFlow UnrestrictedWrite UseOfOrigin
数值溢出漏洞	IntegerOverflow
事务顺序依赖漏洞	DivisionBeforeCallvalue DivisionBeforeMultiply
冻结账户漏洞	LockedEther
逻辑设计缺陷漏洞	MissingInputValidation UnhandledException WriteOnly
随机数生成漏洞	UnsafeDependenceOnBlock UnsafeDependenceOnGas

接下来, 使用 solidity-parser-antlr 工具从源代码中提取 COOP 和 SC-Sol 两组度量元的特征信息, 并将两组度量元信息组合为 COOP-SC-Sol 度量元的特征信息. 由于 Solidity 编译器版本不同, solidity-parser-antlr 无法分析所有版本的源代码, 因此部分源代码无法得到度量元信息. 具体来讲, 在 6 519 个可以获得检测报告的源代码文件中, 共有 4 203 个源代码文件可以获得度量元信息, 其中包括 21 138 条 contract 或 library 度量元信息, 由于 Solidity 智能合约存在代码克隆现象, 因此需要对数据集中的数据进行去重, 经过去重后得到 7 964 条 contract 或 library 度量元信息. 将度量元信息与第一步中获得的缺陷数量标签、缺陷倾向性标签和特定缺陷标签组合, 分别得到缺陷数量预测数据集、缺陷倾向性预测数据集和特定缺陷预测数据集.

5.1.2 模型构建及有效性评价

对于 Solidity 智能合约的缺陷倾向性预测问题, 使用缺陷倾向性预测数据集, 应用 6 种二分类模型进行缺陷倾向性预测. 对预测结果使用评估指标 *F*1-*score* 和 *g-mean* 进行度量 (RQ1–RQ3). 对于 Solidity 智能合约的缺陷数量预测问题, 使用缺陷数量预测数据集, 应用 7 种回归模型进行缺陷数量预测. 对预测结果使用评估指标 *MAE* 和 *FPA* 进行度量 (RQ4).

实验在构建数据集的过程中, 发现 Solidity 缺陷倾向性数据集存在类不平衡现象. 衡量数据集中的类不平衡状况通常使用缺陷模块百分比 (percentage of defect modules, *PDM*) 指标, *PDM* 的计算公式如公式 (7) 所示. 其中, N_{defect} 为含有该类型缺陷的模块数量, N_{modelues} 为总模块数量.

$$PDM = \frac{N_{\text{defect}}}{N_{\text{modules}}} \tag{7}$$

通常情况下, *PDM* 低于 25% 的数据集存在类不平衡问题[18,39]. 在 Solidity 倾向性预测数据集中, 7 964 条 contract 或 library 度量元信息中共有 2 508 条标记为含有缺陷, *PDM* 为 31.5%, 因此数据集中可能存在类不平衡问

题, 将会对分类器的预测性能产生影响. 为缓解类不平衡问题, 实验参考文献 [15,34] 中总结的软件缺陷预测领域中常见的采样方法即随机欠采样 (random under sampling, RUS)、随机过采样 (random over sampling, ROS) 和 SMOTE 方法, 分别对数据集进行处理. 使用 *F*1-*score* 和 *g*-*mean* 进行度量, 比较 3 种采样技术对 Solidity 智能合约缺陷倾向性预测模型性能的影响 (RQ2). 实验中参考文献 [18] 的方法, 在划分训练集和测试集时按照 *PDM* 进行分层切割, 即训练集和测试集的 *PDM* 相同.

以上实验均采用十折交叉验证. 具体而言, 将原始的数据集等分为 10 份, 使用其中 9 份作为该轮模型的训练集, 剩余的 1 份作为测试集, 记录本轮模型的预测结果. 如此重复 10 次, 每次选择不同的测试集, 以此保证每份数据被作为过测试集 1 次. 为了尽可能避免因数据集划分而引起的误差, 将十折交叉验证随机重复 10 次.

5.2 实验结果与分析

5.2.1 针对 RQ1 的结果分析

为了研究在 Solidity 智能合约缺陷倾向性预测中, 不同度量元集和不同模型之间的性能差异, 本文将由 3 种度量元集构成的缺陷倾向性预测数据集, 分别应用 6 种二分类模型进行实验, 最终得到 18 种不同的预测结果, 并使用 *F*1-*score* 和 *g*-*mean* 两种指标进行度量.

实验结果如图 6、图 7 和表 8 所示. 图 6 和图 7 分别是 Solidity 智能合约缺陷倾向性预测的 *F*1-*score* 和 *g*-*mean* 评估结果, 根据二分类模型将箱图分为 6 组, 按照 3 种度量元集的 *F*1-*score*和*g*-*mean* 降序排列. 每组内分别使用蓝色、橙色和绿色代表 COOP-SC-Sol、SC-Sol 和 COOP 度量元集. 表 8 为 *F*1-*score* 和 *g*-*mean* 的平均值, 使用粗体标记不同模型和不同度量元集的最优结果.

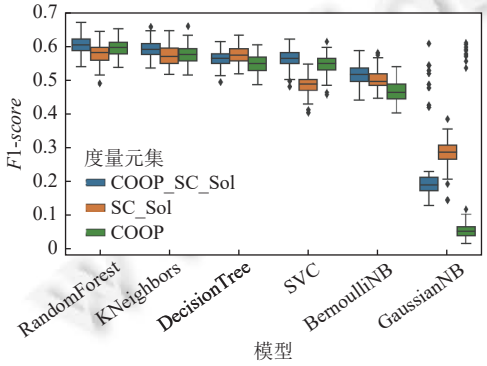


图 6 缺陷倾向性预测模型性能比较箱线图 (*F*1-*score*)

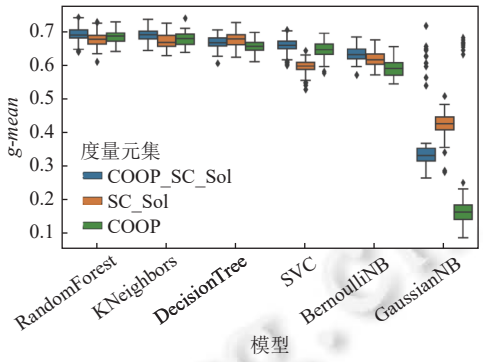


图 7 缺陷倾向性预测模型性能比较箱线图 (*g*-*mean*)

由图 6 可以看出, RFC 模型使用 3 种度量元集的 *F*1-*score* 平均值最高, 为 0.590, 其余依次为 KNNC (0.578)、DTC (0.559)、SVC (0.530)、BNB (0.493) 以及 GNB (0.200). 另外, 结合表 8 可以看出, 使用 COOP-SC-Sol 度量元集时, 6 种模型的 *F*1-*score* 平均值最高, 为 0.507, 其余依次为 SC-Sol (0.497) 和 COOP (0.470). 在 6 种分类模型中, COOP-SC-Sol 在 4 种分类模型 (即 RFC、KNNC、SVC 和 BNB) 中预测性能优于 SC-Sol 和 COOP, 在其余 2 种分类模型 (即 DTC 和 GNB) 中, 基于 SC-Sol 度量元集所构建的预测模型优于另外 2 种度量元集所构建的预测模型. 由此可知, 基于 Solidity 智能合约特征的度量元集所构建的预测模型的 *F*1-*score* 指标优于仅考虑面向对象特征的度量元所构建的预测模型. 使用 COOP-SC-Sol 度量元集时, *F*1-*score* 相对于 SC-Sol 和 COOP 度量元集的分别提升了 2% 和 8%.

由图 7 可以看出, RFC 模型使用 3 种度量元集的 *g*-*mean* 平均值最高, 为 0.684, 其余依次为 KNNC (0.681)、DTC (0.667)、SVC (0.633)、BNB (0.615) 以及 GNB (0.329). 另外, 结合表 8 可以看出, 使用 COOP-SC-Sol 度量元集时, 6 种模型的 *g*-*mean* 平均值最高, 为 0.617, 其余依次为 SC-Sol (0.611) 和 COOP (0.577). 与 *F*1-*score* 评估指标的实验结果相似, COOP-SC-Sol 在 4 种分类模型 (即 RFC、KNNC、SVC 和 BNB) 中预测性能优于 SC-Sol 和 COOP, 在其余 2 种分类模型 (即 DTC 和 GNB) 中, 基于 SC-Sol 度量元集所构建的预测模型优于另外 2 种度量元

集所构建的预测模型. 由此可知, 在考虑了数据集类不平衡的情况下, 基于 Solidity 智能合约特征的度量元集所构建的预测模型的 *g-mean* 指标优于仅考虑面向对象特征的度量元所构建的预测模型.

表 8 缺陷倾向性预测模型的 *F*1-*score* 和 *g-mean* 值

度量元集	模型	*F*1-*score*		*g-mean*	
		单值	均值	单值	均值
COOP-SC-Sol	RFC	**0.601**	**0.507**	**0.692**	**0.617**
	KNNC	0.591		**0.692**	
	DTC	0.560		0.668	
	BNB	0.515		0.634	
	GNB	0.216		0.357	
	SVC	0.56		0.657	
SC-Sol	RFC	**0.577**	0.497	**0.676**	0.611
	KNNC	0.569		0.673	
	DTC	0.571		0.677	
	BNB	0.499		0.619	
	GNB	0.283		0.423	
	SVC	0.484		0.597	
COOP	RFC	**0.591**	0.47	**0.685**	0.577
	KNNC	0.575		0.678	
	DTC	0.546		0.657	
	BNB	0.463		0.591	
	GNB	0.102		0.208	
	SVC	0.545		0.646	

综合以上分析, 可以对 RQ1 得出以下结论: 对于 Solidity 缺陷倾向性预测, 使用 *F*1-*score* 和 *g-mean* 对预测模型进行评估时, 考虑了针对 Solidity 智能合约特征的 COOP-SC-Sol 或 SC-Sol 度量元在 2 种指标上的性能均优于仅考虑面向对象特征的 COOP 度量元, 其中, 相较于其他模型, RFC 模型的性能较好; 在综合比较各模型使用不同度量元集的预测性能时, 结合针对 Solidity 智能合约特征度量元的 COOP-SC-Sol 度量元集的平均性能较好.

5.2.2 针对 RQ2 的结果分析

如第 5.1.2 节所述, Solidity 智能合约缺陷数据集中存在类不平衡现象, 可能会影响预测模型的性能. 为了研究类不平衡问题对 Solidity 智能合约缺陷倾向性预测性能的影响, 我们在 RQ1 实验的基础上, 先根据 *PDM* 划分训练集和测试集, 保证训练集和测试集的 *PDM* 相同; 然后对训练集进行 3 种采样技术预处理 (即 SMOTE、ROS 和 RUS), 将处理后的训练集应用 RFC 构建缺陷倾向性预测模型; 最后将经过预处理的预测结果和未经过预处理的预测结果进行比较.

实验的结果如图 8 和表 9 所示, 其中 None、SMOTE、ROS 和 RUS 分别表示不做数据预处理、经过 SMOTE、ROS 和 RUS 预处理的预测结果. 从图 8 中可以看出, 在 *F*1-*score* 和 *g-mean* 指标中, 3 种经过采样预处理后的缺陷预测模型性能均优于不做数据预处理的模型, 其中, RUS 对模型的提升效果最大. 为进一步分析实验结果, 本文采用 *t* 检验对结果进行分析, 验证经过预处理后的模型与未经过预处理的模型是否存在显著差异, 其中 *t* 检验的原假设为经过预处理和未经过预处理的预测结果之间没有显著性差异, 检验水准 *p* 取 0.05, 即 *p* 值小于 0.05 时否定原假设. 如表 9 所示, 3 种经过采样处理后的缺陷预测模型均与不做数据预处理的模型存在显著性差异.

综合以上分析, 可以对 RQ3 得出如下结论: 使用采样技术对数据集进行数据预处理后能够提升 Solidity 智能合约缺陷倾向性预测模型的性能. 其中, RUS 方法提升性能的效果相较其他采样方法更好.

5.2.3 针对 RQ3 的结果分析

在缺陷倾向性预测的基础上, 我们将 RQ1 和 RQ2 中得出的最优模型和最优采样算法应用于 RQ3, 即先采用 RUS 对数据集进行预处理, 然后基于随机森林分类器构建特定缺陷预测分类模型, 并使用 *F*1-*score* 进行度量, 以

验证所提出的度量元集在预测特定类型缺陷时的性能表现. 需要注意的是, 第 5.1.1 节所述的 9 种缺陷类型中, 数值溢出漏洞的缺陷标签中不含有正样本 (即未检测到该类缺陷), 因此在 RQ3 实验中不考虑数值溢出漏洞.

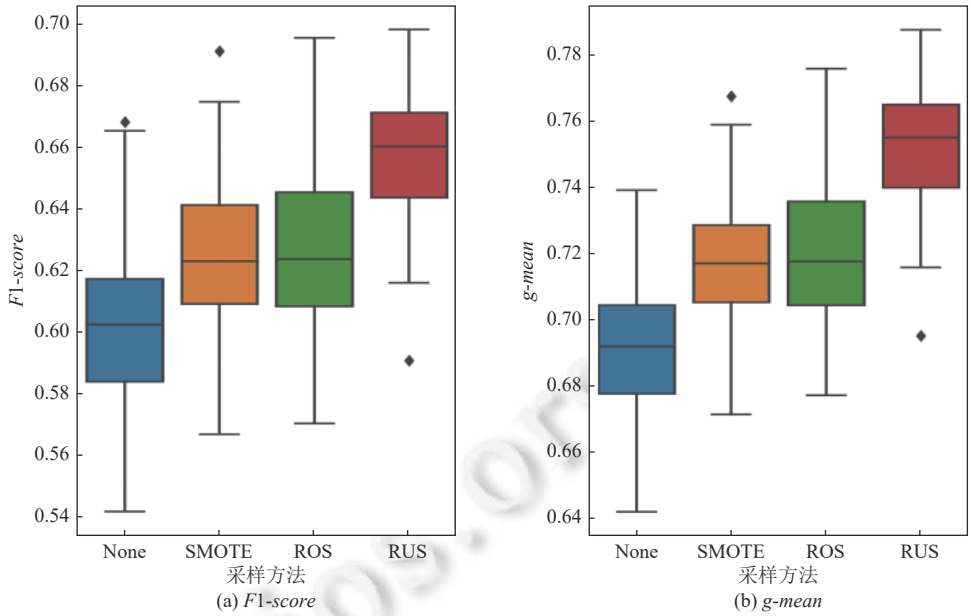


图 8 使用不同采样方法的模型性能对比箱线图

表 9 不同采样方法的模型性能平均值对比

采样方法	*F*1-*score* (*p*-value)	*g*-*mean* (*p*-value)
None	0.602	0.692
SMOTE	0.624 (2.974E–22)	0.718 (2.030E–33)
ROS	0.627 (9.993E–24)	0.720 (1.364E–34)
RUS	0.658 (2.204E–40)	0.754 (2.571E–50)

图 9 为预测 8 种特定类型缺陷的 *F*1-*score* 值按照平均值进行降序排列后的箱线图. 从图 9 中可以看出, 特定缺陷预测模型在不同的缺陷类型上的预测性能差异较为明显, *F*1-*score* 的均值最高可达 0.574 (权限控制漏洞), 其余依次为 call 安全漏洞 0.572、逻辑设计漏洞 0.536、条件竞争漏洞 0.402、随机数生成漏洞 0.397、事务顺序依赖漏洞 0.203、重入漏洞 0.163 和冻结账户漏洞 0.077. 表 10 为 8 种缺陷类型的 *PDM* 和预测模型的 *F*1-*score* 值, 可以看出, 预测特定类型缺陷的模型性能与数据集中的类不平衡严重性存在联系. 图 9 中蓝色垂直虚线分别表示 *PDM*=10% 和 *PDM*=5% 的分割线. 结合图 9 和表 10 可知, 根据数据集中不同类型缺陷的 *PDM* 可划分为 3 个层次, 即 *PDM* 大于 10%, 大于 5% 小于 10% 和小于 5%. 其中, 当 *PDM* 大于 10% 时, 模型的预测性能表现较好, *F*1-*score* 能够达到 0.5 以上; 当 *PDM* 下降到 10% 以下时, 模型的性能显著下降, *F*1-*score* 降低为 0.4; 当 *PDM* 进一步下降到 5% 以下时, *F*1-*score* 的均值小于 0.2, 此时预测模型失效.

综合以上分析, 可以对 RQ3 得出以下结论: 对于 Solidity 特定缺陷倾向性预测, 在 *PDM* 大于 10% 时, 能够取得较好的预测性能, 其中权限控制漏洞、call 安全漏洞和逻辑设计漏洞的 *F*1-*score* 均值在 0.5 以上.

5.2.4 针对 RQ4 的结果分析

软件缺陷数量是更详细的缺陷预测结果, 能够为开发者或测试人员提供更精确的预测信息. 为了研究在 Solidity 智能合约缺陷数量预测中, 不同度量元集和不同模型之间的性能差异, 与 RQ1 中的实验类似, 实验将 3 种度量元集 (即 COOP、SC-Sol 和 COOP-SC-Sol) 构成的缺陷数量预测数据集, 分别应用 7 种回归模型进行训练, 最

终得到 21 种不同的预测结果, 并使用 MAE 和 FPA 两种指标进行度量. 实验的结果如图 10、图 11 和表 11 所示. 其中, 表 11 为 MAE 和 FPA 的平均值, 使用粗体标记不同模型和不同度量元集的最优结果.

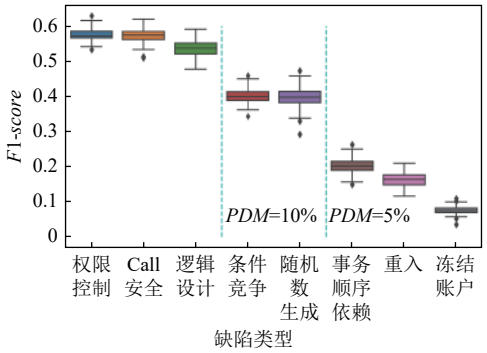


图 9 预测 8 种特定类型缺陷的 *F*1-*score* 值对比箱线图

表 10 各类型缺陷的 *PDM* 值和预测模型 *F*1-*score* 值

缺陷类型	*F*1-*score*	*PDM* (%)
权限控制漏洞	0.574	20.04
call安全漏洞	0.572	15.50
逻辑设计漏洞	0.536	11.63
条件竞争漏洞	0.402	6.52
随机数生成漏洞	0.397	5.39
事务顺序依赖漏洞	0.203	2.89
冻结账户漏洞	0.077	2.15
重入漏洞	0.163	1.78

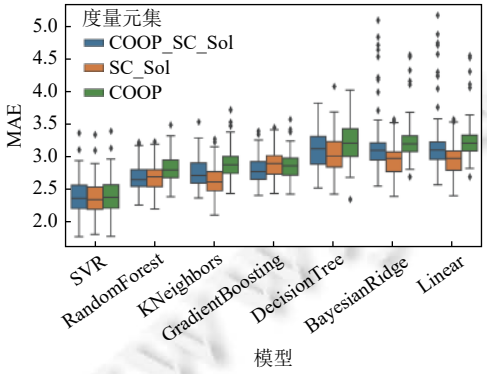


图 10 缺陷数量预测模型性能比较箱线图 (MAE)

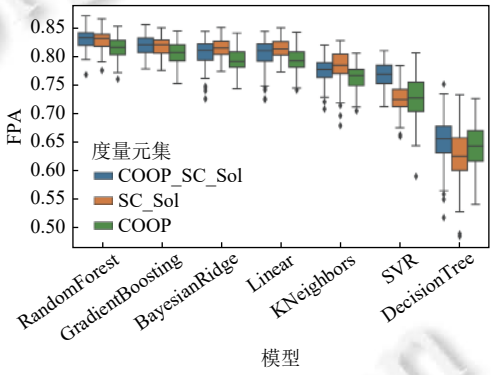


图 11 缺陷数量预测模型性能比较箱线图 (FPA)

图 10 是使用不同模型对 Solidity 智能合约缺陷数量预测情况使用 MAE 进行评估的结果. 根据回归模型将箱图分为 7 组, 每组内分别使用蓝色、橙色和绿色代表 COOP-SC-Sol、SC-Sol 和 COOP 度量元集, 并按照 3 种度量元集的 MAE 平均值升序排列.

从图 10 中可以看出, SVR 在使用 3 种度量元集时的平均 MAE 值最低, 为 2.4, 其余依次为 RFR (2.741)、KNNR (2.769)、GBR (2.866)、DTR (3.135)、BR (3.14) 以及 LR (3.151). 另外, 结合表 11 可以看出, 使用 SC-Sol 度量元集时, 不同模型的 MAE 平均值最低, 为 2.802, 其余依次为 COOP-SC-Sol (2.886) 和 COOP (2.969). 其中, 基于 SC-Sol 度量元集构建的 5 种回归模型 (即 SVR、KNNR、DTR、BR 和 LR) 的 MAE 指标优于 COOP-SC-Sol 和 COOP 度量元集, 基于 COOP-SC-Sol 度量元集构建的 2 种模型 (即 RFR 和 GBR) 的 MAE 指标优于另外 2 种度量元集. 由此可知, 基于 Solidity 智能合约特征的度量元集所构建预测模型的 MAE 指标优于仅考虑面向对象特征的度量元所构建的预测模型.

图 11 是使用不同模型对 Solidity 智能合约缺陷数量预测情况使用 FPA 进行评估的结果. 根据回归模型将箱图分为 7 组, 每组内分别使用蓝色、橙色和绿色代表 COOP-SC-Sol、SC-Sol 和 COOP 度量元集, 并按照 3 种度量元集的 FPA 平均值降序排列.

从图 11 中可以看出, RFR 在使用 3 种度量元集时 FPA 的平均值最高, 为 0.824, 其余依次为 GBR (0.813)、BR (0.804)、LR (0.803)、KNNR (0.772)、SVR (0.739) 以及 DTR (0.639). 另外, 结合表 11 可以看出, 使用 COOP-SC-Sol 度量元集时, 不同模型的 FPA 平均值最高, 为 0.779, 其余依次为 SC-Sol (0.771) 和 COOP (0.762). 其中, 基

于 COOP-SC-Sol 度量元集构建的 4 种回归模型 (即 RFR、GBR、SVR 和 DTR) 的 FPA 指标优于 SC-Sol 和 COOP 度量元集; 基于 SC-Sol 度量元集构建的 3 种模型 (即 BR、LR 和 KNNR) 的 FPA 指标优于另外 2 种度量元集. 由此可知, 基于 Solidity 智能合约特征的度量元集所构建的预测模型的 FPA 指标优于仅考虑面向对象特征的度量元所构建的预测模型.

表 11 缺陷数量预测模型的 MAE 和 FPA 值

度量元集	模型	MAE		FPA	
		单值	均值	单值	均值
COOP-SC-Sol	SVR	**2.404**	2.886	0.767	**0.779**
	KNNR	2.763		0.775	
	RFR	2.690		**0.830**	
	GBR	2.809		0.819	
	BR	3.200		0.805	
	LR	3.218		0.804	
	DTR	3.117		0.651	
SC-Sol	SVR	**2.384**	**2.802**	0.725	0.771
	KNNR	2.637		0.780	
	RFR	2.697		**0.828**	
	GBR	2.902		0.817	
	BR	2.966		0.813	
	LR	2.973		0.813	
	DTR	3.058		0.625	
COOP	SVR	**2.411**	2.969	0.725	0.762
	KNNR	2.907		0.762	
	RFR	2.835		**0.813**	
	GBR	2.888		0.803	
	BR	3.254		0.793	
	LR	3.261		0.793	
	DTR	3.231		0.642	

不同模型在 MAE 和 FPA 指标上的性能表现存在差异, 为综合比较各个回归模型在 MAE 和 FPA 指标上的性能差异, 本文将各个模型 MAE 和 FPA 指标的排名相加, 结果为: RFR 的排名最高, 为 2, 其余依次为 GBR (6)、SVR (7)、KNNR (8)、BR (9)、LR (11) 以及 DTR (12).

综合以上分析, 可以对 RQ4 得出以下结论: 对于 Solidity 缺陷数量预测, 考虑了针对 Solidity 智能合约特征的 COOP-SC-Sol 或 SC-Sol 度量元集在 MAE 和 FPA 指标上的性能均优于仅考虑面向对象特征的 COOP 度量元, 其中, SC-Sol 在 MAE 评估指标中性能较好, COOP-SC-Sol 在 FPA 评估指标中性能较好. 基于本文所提出针对 Solidity 智能合约特性的度量元集所构建的 7 种回归模型的性能均优于 COOP 度量元集. 此外, 综合考虑 MAE 和 FPA 指标的排名时, RFR 模型的平均性能较好, 使用 RFR 模型时, COOP-SC-Sol 的预测性能优于 SC-Sol.

6 有效性分析

内部有效性分析主要与影响实验正确性的因素相关. 在本文的实验中, 内部有效性威胁因素主要体现在代码实现是否正确. 首先, 为保证 Solidity 智能合约度量元提取的正确性, 本文使用开源的 Solidity 智能合约语法分析工具 solidity-parser-antlr. 其次, 为保证模型的训练与评估指标计算的有效性, 本文使用第三方机器学习库 sckit-learn 中的模型与评估指标, 采样算法的实现由第三方 Python 库 imbalanced-learn 提供. 此外, 在实验的过程中对编写的代码进行了交叉检查, 尽可能减少内部有效性威胁.

外部有效性分析主要关注实验结果的一般性. 本文使用的缺陷数据来源于区块链智能合约安全检测平台, 该平台利用基于事实推理的符号执行方法, 截至 2018 年已经对 31 276 份智能合约进行自动化检测, 可以在一定程度

上保障实验所用数据集的正确性与可靠性. 此外, 本文使用 solidity-parser-antlr 提取 AST 信息, 由于 Solidity 编译器仍处于开发更新过程中, 且更新频繁 (如 2021 年 1 月至 2021 年 4 月共更新了 4 次, 其中 3 月更新了 2 次), 而 solidity-parser-antlr 工具已于 2019 年 10 月停止更新, 因此部分 Solidity 智能合约的源代码无法通过 solidity-parser-antlr 工具解析, 即无法获得度量元信息. 具体地, 149 363 份 Solidity 智能合约中 141 752 份代码的 AST 信息可通过该工具解析获得, 解析成功率达 95%. 此外, solidity-parser-antlr 曾被用于在智能合约克隆检测的相关工作[40]中提取代码结构信息. 因此在实验中选用该工具提取 AST 信息.

另外, 本文构建的 Solidity 智能合约缺陷数据集中的缺陷数量与 Solidity 智能合约中含有的真实缺陷数量可能存在差异, 即由 BSCSCS 提供的缺陷信息存在误报或漏报的可能. BSCSCS 利用符号执行的方法对智能合约进行检测, 符号执行是软件分析和安全漏洞检测的常用手段[41,42], 在检测到安全漏洞后, 能够为触发漏洞的路径生成相应的测试用例, 从而有效降低误报率. 同时, 由于 BSCSCS 需要根据预先定义的漏洞逻辑约束集进行漏洞检测, 因此可能存在漏报. 在之前的工作中, Yatish 等人[37]发现: 尽管数据集中的缺陷数量存在误差, 但使用不同准确程度的两种数据集所构建回归模型的预测结果之间不存在显著性差异; 在缺陷倾向性预测中, 使用不同准确程度的两种数据集所构建分类模型的预测结果中, *F*1-*score* 差异值的中位数小于 0.1. 因此, 即使存在漏报, 本文所构建的 Solidity 缺陷数据集可在一定程度上作为参考.

构造有效性分析主要关注实验使用的评估指标. 本文使用 MAE 和 FPA 作为缺陷数量预测的评估指标, 使用 *F*1-*score* 和 *g*-*mean* 作为缺陷倾向性的评估指标, 这些指标在软件缺陷预测领域被广泛使用[15,34,36,37,43], 可以有效评估评估缺陷预测模型性能.

7 相关工作

本节主要介绍智能合约缺陷检测和软件缺陷预测的相关工作.

7.1 智能合约缺陷检测技术

目前研究人员针对智能合约的缺陷检测技术开展了大量研究, 主要基于模糊测试以及符号执行技术. 如: ContractFuzzer[19]基于预定义的参数值生成测试用例, 构建 EVM 并对其进行插装, 在模糊测试过程中收集与漏洞相关的信息, ContractFuzzer 可以识别 7 种类型的漏洞. sFuzz[14]利用轻量级自适应策略优化种子选择, 能够覆盖条件难以满足的路径, 从而提升模糊测试的性能, 相较于 ContractFuzzer, sFuzz 能够覆盖更多的路径从而发现更多的缺陷, 且 sFuzz 的检测速度比 ContractFuzzer 快两个数量级. ReGuard[44]针对重入漏洞设计了一个基于翻译的框架, 能够将特定的智能合约语言转换为 C++, 在测试时 ReGuard 记录关键执行路径信息, 通过重入自动机识别潜在的重入漏洞. Oyente[10]利用符号执行技术检测智能合约, 通过约束求解生成测试用例, 以涵盖单个函数中的不同程序路径, Oyente 可识别 4 种类型的漏洞. 与 Oyente 类似, teEhter[45]同样利用符号执行技术覆盖程序路径, 通过分析字节码中的指令识别关键路径, 定义了 2 种可能存在漏洞的模式. Osiris[46]结合了符号执行和污点分析, 能够发现 3 种关于整数的缺陷. MAIAN[20]针对以太币的流动总结了 3 种漏洞以及判断条件, 利用符号执行技术并在检测时考虑连续合约调用来检测 3 种漏洞.

除此之外, 已有一些平台支持智能合约缺陷的在线检测, 如 ContractGuard 和区块链智能合约安全检测平台等. 其中, ContractGuard 可自动配置区块链网络, 部署智能合约并利用模糊测试技术生成多个测试用例, 根据已知常见漏洞定义关键属性, 并结合符号执行技术进行缺陷检测. BSCSCS 利用基于事实推理的符号执行技术, 根据常见漏洞制定自定义约束条件, 通过对智能合约代码进行约束转换、约束校验等步骤进行缺陷检测.

与缺陷检测的应用场景不同, 本文所关注的缺陷预测技术可以在较短的时间内, 对大量软件模块进行预测, 识别有可能含有缺陷的模块, 使测试人员更加合理地分配有限的测试资源. 换言之, 缺陷预测可以应用于缺陷检测之前, 用于筛选出最有可能含有缺陷的模块, 然后使用缺陷检测技术对这些模块进行更快、更准确地检测. 另外, 模块缺陷数量的预测结果可用于评估测试的充分性, 作为停止测试的度量指标.

7.2 软件缺陷预测技术

软件缺陷预测技术[15]通过挖掘软件历史仓库, 预测软件模块出现缺陷的倾向性或数量, 其中缺陷倾向性指软

件模块是否可能存在缺陷, 缺陷数量指软件模块内可能含有的缺陷个数. 测试人员根据预测结果可以优化资源分配, 或将预测结果作为评估一个系统是否可以交付使用的重要指标[15,43]. 影响软件缺陷预测性能的 3 个重要因素为度量元的设计、缺陷数据集的质量和缺陷预测模型的构建[15].

度量元的设计是软件缺陷预测研究中的一个核心问题, 度量元的质量对缺陷预测的性能有很大的影响[15,47]. 度量元指软件历史仓库的代码或其他数据中, 与软件缺陷存在强相关性的特征. Chidamber 和 Kemerer[23]关注面向对象程序中的继承、耦合与内聚等特性, 提出了 CK 度量元; McCabe[28]关注程序的控制流复杂度, 提出了圈复杂度度量元; Bansiya 和 Davis[25]提出了一组关注面向对象语言中封装、多态以及信息传递等特性的度量元等.

Marian 等人[21]基于面向对象程序度量等相关工作, 拓展了 Ckjm 工具, 能够自动度量程序代码中的 20 种度量元, 同时收集源代码存储库或代码版本仓库的日志信息, 根据代码变动提交信息识别提交的代码是否用于修复缺陷, 从而获得缺陷信息, 他们将度量元信息和缺陷信息组合, 构建了 MetricsRepo 缺陷预测数据集 (http://purl.org/MarianJureczko/MetricsRepo), 该数据集包含了 11 个实际开发项目的度量元信息和缺陷标签, 为缺陷预测领域的后续研究提供了公开的缺陷预测数据集. He 等人[48]基于该数据集讨论了跨项目缺陷预测的数据集选择问题, 发现使用其他项目的缺陷数据集比使用同一项目的训练数据集能提升预测性能, 同时提出了一种能够为没有缺陷数据集的项目提供跨项目数据集选择的方法. Chen 等人[16]基于该数据集对无监督模型和监督模型之间的差异进行了实证研究和分析, 发现无监督模型在预测缺陷数量排名问题上优于监督模型.

软件缺陷数据集的处理是缺陷预测研究的另一个重点, 类不平衡是软件缺陷数据集中普遍存在的问题, 缺陷预测模型受到数据不平衡的影响, 会降低预测的模型性能. 在软件缺陷预测领域中, 通常使用采样技术缓解类不平衡问题[15,34]. Pelayo 等人[49]在缺陷预测中应用合成少数样本过采样技术 (synthetic minority over-sampling technique, SMOTE)[50]对数据进行预处理, 发现可以将预测模型性能提高 23%; Kwabena 等人[18]发现 SMOTE 算法会过度概括, 导致预测结果误报率较高, 提出了基于遗传继承理论的 MAHAKIL 算法, 相较于其他过采样算法 (如 SMOTE、Borderline-SMOTE 和 ADASYN 等) 拥有更低的误报率. 于巧等人[51]针对类不平衡数据集对不同分类模型性能的影响做了实证研究, 结果表明代价敏感学习和集成学习在类不平衡数据集中的稳定性更好.

缺陷预测模型的构建是影响预测性能的关键因素. 针对软件缺陷预测, 研究者们提出了基于机器学习算法的预测模型, Zhang 等人[52]探究了通用缺陷预测模型的可行性, 提出了一种上下文感知的变换方法处理预测模型, 将度量元转换为相同的尺度, Zhang 等人构建了朴素贝叶斯缺陷预测模型, 实验表明在 5 个实际项目中通用预测模型与项目内预测模型的性能相同, 证明了通用预测模型的可行性. Shivaji 等人[53]对软件缺陷预测中特征选择的问题进行了实证研究, 通过构建了朴素贝叶斯和支持向量机 2 种模型, 在 11 个实际项目中对 5 种特征选择技术进行实验, 结果表明特征选择技术能够减少构建模型所需的时间, 并且经过选择后的特征能达到更好的分类效果. 随着近年来深度学习方法的兴起, 部分研究者也尝试将深度学习模型应用到软件缺陷预测领域, 并取得了较好的成果. 如张献等人[54]构建语言模型, 将可用于描述代码自然性的交叉熵作为一类新的度量元, 并基于此度量元提出了一种针对切片粒度的缺陷预测方法; Dam 等人[55]提出了一种用于自动学习代码抽象语法树表示的树结构的 LSTM 网络, 根据学习出的特征进行预测. 虽然以上研究成果表明应用缺陷预测模型能够取得较好的预测性能, 但不同的项目、应用场景和数据集对应的最优模型不同, 不存在一种机器学习或深度学习模型能够在所有项目中均取得最优性能[15]. 本文提出的是一种基于软件度量的缺陷预测方法, 不同于深度学习方法中通过模型自动提取语义特征或通过程序表示技术将程序表示为控制流图或抽象语法树的形式, 基于软件度量的缺陷预测方法是通过先验知识提取软件历史仓库中与缺陷强相关的度量元作为程序特征. 相比于深度学习方法, 基于软件度量的预测方法具有更好的可解释性.

以上工作的研究对象均为传统的结构化程序或面向对象程序, 目前研究人员针对智能合约这一特殊软件制品展开缺陷预测的研究较少, 缺少有效的方法和数据集. 本文基于软件缺陷预测技术的基本原理, 设计了一组针对 Solidity 智能合约的度量元集, 构建 Solidity 智能合约缺陷数据集, 利用该数据集对 Solidity 智能合约进行缺陷预测, 并讨论度量元选择、模型性能以及类不平衡数据集预处理等问题.

8 总结与展望

本文首先提出了一组针对 Solidity 智能合约特有的变量、函数、结构以及 Solidity 语言特性的度量元集, 即 SC-Sol 度量元集, 并将其与 COOP 度量元集在 6 种缺陷倾向性预测模型和 7 种缺陷数量预测模型上的性能差异进行比较. 实验结果表明, 结合 COOP 和 SC-Sol 的 COOP-SC-Sol 度量元集具有较好的缺陷预测性能, 随机森林分类模型和随机森林回归模型分别在缺陷倾向性预测和缺陷数量预测中优于其他模型. 另外, 本文还讨论了在类不平衡的情况下, 3 种经典采样技术对缺陷倾向性预测模型性能的影响. 结果表明使用采样技术会对缺陷倾向性预测模型的性能有一定的提升, 其中随机欠采样方法提升效果更好. 在针对特定缺陷类型的倾向性预测上, 基于所提出度量元集构建的预测模型在 *PDM* 大于 10% 的数据集中取得了较好的效果.

在未来工作中, 将考虑其他 Solidity 智能合约缺陷检测平台或工具, 如 ContractGuard、Oyente 和 sFuzz 等, 以更准确地获得缺陷信息, 还将尝试利用基于信息检索等[56]静态分析技术, 如分析智能合约历史修改记录[57]等方式获取缺陷信息, 减少数据集中的误报和漏报缺陷数量, 以进一步完善 Solidity 智能合约缺陷数据集. 此外, 还将尝试利用深度学习等方法构建更加准确的缺陷预测模型, 以及尝试将本方法应用到其他语言编写的智能合约中, 例如 Vyper 等.

References:

[1] Yuan Y, Wang FY. Blockchain: The state of the art and future trends. Acta Automatica Sinica, 2016, 42(4): 481–494 (in Chinese with English abstract). [doi: 10.16383/j.aas.2016.c160158]

[2] Shao QF, Jin CQ, Zhang Z, Qian WN, Zhou AY. Blockchain: Architecture and research progress. Chinese Journal of Computers, 2018, 41(5): 969–988 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00969]

[3] Szabo N. Formalizing and securing relationships on public networks. First Monday, 1997, 2(9): 1–21. [doi: 10.5210/fm.v2i9.548]

[4] Ouyang LW, Wang S, Yuan Y, Ni XC, Wang FY. Smart contracts: Architecture and research progresses. Acta Automatica Sinica, 2019, 45(3): 445–457 (in Chinese with English abstract). [doi: 10.16383/j.aas.c180586]

[5] He HW, Yan A, Chen ZH. Survey of smart contract technology and application based on blockchain. Journal of Computer Research and Development, 2018, 55(11): 2452–2466 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2018.20170658]

[6] Wang S, Ni XC, Yuan Y, Wang FY, Wang X, Ouyang LW. A preliminary research of prediction markets based on blockchain powered smart contracts. In: Proc. of the 2018 IEEE Int'l Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). Halifax: IEEE, 2018. 1287–1293. [doi: 10.1109/Cybermatics_2018.2018.00224]

[7] Maicher L, de la Rosa JL, Gibovic D, Torres-Padrosa V. On intellectual property in online open innovation for SME by means of blockchain and smart contracts. In: Proc. of the World Open Innovation Conf. 2016. Barcelona, 2016.

[8] Azaria A, Ekblaw A, Vieira A, Lippman A. MedRec: Using blockchain for medical data access and permission management. In: Proc. of the 2nd Int'l Conf. on Open and Big Data (OBD). Vienna: IEEE, 2016. 25–30. [doi: 10.1109/OBD.2016.11]

[9] Dorri A, Kanhere SS, Jurdak R. Towards an optimized blockchain for IoT. In: Proc. of the 2nd IEEE/ACM Int'l Conf. on Internet-of-Things Design and Implementation (IoTDI). Pittsburgh: IEEE, 2017. 173–178. [doi: 10.1145/3054977.3055003]

[10] Luu L, Chu DH, Olickel H, Saxena P, Hobor A. Making smart contracts smarter. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 254–269. [doi: 10.1145/2976749.2978309]

[11] Chen JC, Xia X, Lo D, Grundy J, Luo XP, Chen T. Defining smart contract defects on Ethereum. IEEE Trans. on Software Engineering, 2022, 48(1): 327–345. [doi: 10.1109/TSE.2020.2989002]

[12] Liu J, Liu ZT. A survey on security verification of blockchain smart contracts. IEEE Access, 2019, 7: 77894–77904. [doi: 10.1109/ACCESS.2019.2921624]

[13] Marino B, Juels A. Setting standards for altering and undoing smart contracts. In: Proc. of the 10th Int'l Symp. on Rules and Rule Markup Languages for the Semantic Web. Stony Brook: Springer, 2016. 151–166. [doi: 10.1007/978-3-319-42019-6_10]

[14] Nguyen TD, Pham LH, Sun J, Lin Y, Minh QT. sFuzz: An efficient adaptive fuzzer for solidity smart contracts. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering (ICSE). Seoul: ACM, 2020. 778–788. [doi: 10.1145/3377811.3380334]

[15] 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]

[16] Chen X, Zhang D, Zhao YQ, Cui ZQ, Ni C. Software defect number prediction: Unsupervised vs supervised methods. Information and Software Technology, 2019, 106: 161–181. [doi: 10.1016/j.infsof.2018.10.003]

[17] Gong LN, Jiang SJ, Wang RC, Jiang L. Empirical evaluation of the impact of class overlap on software defect prediction. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 698–709. [doi: 10.1109/ASE.2019.00071]

[18] Bennin KE, Keung J, Phannachitta P, Monden A, Mensah S. MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. IEEE Trans. on Software Engineering, 2018, 44(6): 534–550. [doi: 10.1109/TSE.2017.2731766]

[19] Jiang B, Liu Y, Chan WK. ContractFuzzer: Fuzzing smart contracts for vulnerability detection. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Montpellier: IEEE, 2018. 259–269. [doi: 10.1145/3238147.3238177]

[20] Nikolić I, Kolluri A, Sergey I, Saxena P, Hobor A. Finding the greedy, prodigal, and suicidal contracts at scale. In: Proc. of the 34th Annual Computer Security Applications Conf. San Juan: ACM, 2018. 653–663. [doi: 10.1145/3274694.3274743]

[21] Jureczko M, Spinellis DD. Using object-oriented design metrics to predict software defects. In: Proc. of the 5th Int'l Conf. on Dependability of Computer Systems DepCoS. Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, 2010. 69–81.

[22] Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction. In: Proc. of the 6th Int'l Conf. on Predictive Models in Software Engineering. Timişoara: ACM, 2010. 9. [doi: 10.1145/1868328.1868342]

[23] Chidamber SR, Kemerer CF. A metrics suite for object oriented design. IEEE Trans. on Software Engineering, 1994, 20(6): 476–493. [doi: 10.1109/32.295895]

[24] Tang MH, Kao MH, Chen MH. An empirical study on object-oriented metrics. In: Proc. of the 6th Int'l Software Metrics Symposium. Boca Raton: IEEE, 1999. 242–249. [doi: 10.1109/METRIC.1999.809745]

[25] Bansiya J, Davis CG. A hierarchical model for object-oriented design quality assessment. IEEE Trans. on Software Engineering, 2002, 28(1): 4–17. [doi: 10.1109/32.979986]

[26] Sellers H. Object-oriented Metrics: Measures of Complexity. Upper Saddle River: Prentice Hall, 1996.

[27] Martin R. OO design quality metrics. An Analysis of Dependencies, 1994, 12(1): 151–170.

[28] McCabe TJ. A complexity measure. IEEE Trans. on Software Engineering, 1976, SE-2(4): 308–320. [doi: 10.1109/TSE.1976.233837]

[29] Atzei N, Bartoletti M, Cimoli T. A survey of attacks on Ethereum smart contracts (SoK). In: Proc. of the 6th Int'l Conf. on Principles of Security and Trust. Uppsala: Springer, 2017. 164–186. [doi: 10.1007/978-3-662-54455-6_8]

[30] Chang JL, Gao B, Xiao H, Sun J, Cai Y, Yang ZJ. sCompile: Critical path identification and analysis for smart contracts. In: Proc. of the 21st Int'l Conf. on Formal Engineering Methods. Shenzhen: Springer, 2019. 286–304. [doi: 10.1007/978-3-030-32409-4_18]

[31] Qian P, Liu ZG, He QM, Huang BT, Tian DZ, Wang X. Smart contract vulnerability detection technique: A survey. Ruan Jian Xue Bao/Journal of Software, 2021 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6375.htm [doi: 10.13328/j.cnki.jos.006375]

[32] Tikhomirov S, Voskresenskaya E, Ivanitskiy I, Takhaviev R, Marchenko E, Alexandrov Y. SmartCheck: Static analysis of Ethereum smart contracts. In: Proc. of the 1st IEEE/ACM Int'l Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Gothenburg: IEEE, 2018. 9–16.

[33] Zhou EC, Hua S, Pi BF, Sun J, Nomura Y, Yamashita K, Kurihara H. Security assurance for smart contract. In: Proc. of the 9th IFIP Int'l Conf. on New Technologies, Mobility and Security (NTMS). Paris: IEEE, 2018. 1–5. [doi: 10.1109/NTMS.2018.8328743]

[34] 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]

[35] Chen L, Fang B, Shang ZW, Tang YY. Tackling class overlap and imbalance problems in software defect prediction. Software Quality Journal, 2018, 26(1): 97–125. [doi: 10.1007/s11219-016-9342-6]

[36] 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]

[37] Yatish S, Jiarpakdee J, Thongtanunam P, Tantithamthavorn C. Mining software defects: Should we consider affected releases? In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 654–665. [doi: 10.1109/ICSE.2019.00075]

[38] Weyuker EJ, Ostrand TJ, Bell RM. Comparing the effectiveness of several modeling methods for fault prediction. Empirical Software Engineering, 2010, 15(3): 277–295. [doi: 10.1007/s10664-009-9111-2]

[39] Van Hulse J, Khoshgoftaar TM, Napolitano A. Experimental perspectives on learning from imbalanced data. In: Proc. of the 24th Int'l Conf. on Machine Learning. Corvalis: ACM, 2007. 935–942. [doi: 10.1145/1273496.1273614]

[40] Kondo M, Oliva GA, Jiang ZM, Hassan AE, Mizuno O. Code cloning in smart contracts: A case study on verified contracts from the Ethereum blockchain platform. Empirical Software Engineering, 2020, 25(6): 4617–4675. [doi: 10.1007/s10664-020-09852-5]

[41] Zhang J, Zhang C, Xuan JF, Xiong YF, Wang QX, Liang B, Li L, Dou WS, Chen ZB, Chen LQ, Cai Y. Recent progress in program analysis. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 80−109 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]

[42] Li ZJ, Zhang JX, Liao XK, Ma JX. Survey of software vulnerability detection techniques. Chinese Journal of Computers, 2015, 38(4): 717–732 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2015.00717]

[43] Chen X, Wang LP, Gu Q, Wang Z, Ni C, Liu WS, Wang QP. A survey on cross-project software defect prediction methods. Chinese Journal of Computers, 2018, 41(1): 254–274 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00254]

[44] Liu C, Liu H, Cao Z, Chen Z, Chen BD, Roscoe B. ReGuard: Finding reentrancy bugs in smart contracts. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering: Companion (ICSE-Companion). Gothenburg: IEEE, 2018. 65–68.

[45] Krupp J, Rossow C. TeEther: Gnawing at Ethereum to automatically exploit smart contracts. In: Proc. of the 27th USENIX Security Symp. Baltimore: USENIX Association, 2018. 1317–1333.

[46] Torres CF, Schütte J, State R. Osiris: Hunting for integer bugs in Ethereum smart contracts. In: Proc. of the 34th Annual Computer Security Applications Conf. San Juan: ACM, 2018. 664–676. [doi: 10.1145/3274694.3274737]

[47] Radjenović D, Heričko M, Torkar R, Živkovič A. Software fault prediction metrics: A systematic literature review. Information and Software Technology, 2013, 55(8): 1397–1418. [doi: 10.1016/j.infsof.2013.02.009]

[48] He ZM, Shu FD, Yang Y, Li MS, Wang Q. An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering, 2012, 19(2): 167–199. [doi: 10.1007/s10515-011-0090-3]

[49] Pelayo L, Dick S. Applying novel resampling strategies to software defect prediction. In: Proc. of the 2007 Annual Meeting of the North American Fuzzy Information Processing Society. San Diego: IEEE, 2007. 69–72. [doi: 10.1109/NAFIPS.2007.383813]

[50] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 2002, 16(1): 321–357.

[51] Yu Q, Jiang SJ, Zhang YM, Wang XY, Gao PF, Qian JY. The impact study of class imbalance on the performance of software defect prediction models. Chinese Journal of Computers, 2018, 41(4): 809–824 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00809]

[52] Zhang F, Mockus A, Keivanloo I, Zhou Y. Towards building a universal defect prediction model. In: Proc. of the 11th Working Conf. on Mining Software Repositories. Hyderabad: ACM, 2014. 182–191. [doi: 10.1145/2597073.2597078]

[53] 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]

[54] Zhang X, Ben KR, Zeng J. Code naturalness based defect prediction method at slice level. Ruan Jian Xue Bao/Journal of Software, 2021, 32(7): 2219–2241 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6261.htm [doi: 10.13328/j.cnki.jos.006261]

[55] Dam HK, Pham T, Ng SW, Tran T, Grundy J, Ghose A, Kim T, Kim CJ. Lessons learned from using a deep tree-based model for software defect prediction in practice. In: Proc. of the 16th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Montreal: IEEE, 2019. 46–57. [doi: 10.1109/MSR.2019.00017]

[56] Li ZL, Chen X, Jiang ZW, Gu Q. Survey on information retrieval-based software bug localization methods. Ruan Jian Xue Bao/Journal of Software, 2021, 32(2): 247−276 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6130.htm [doi: 10.13328/j.cnki.jos.006130]

[57] Wen M, Wu RX, Cheung SC. Locus: Locating bugs from software changes. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: IEEE, 2016. 262−273.

附中文参考文献:

[1] 袁勇, 王飞跃. 区块链技术发展现状与展望. 自动化学报, 2016, 42(4): 481–494. [doi: 10.16383/j.aas.2016.c160158]

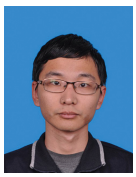
[2] 邵奇峰, 金澈清, 张召, 钱卫宁, 周傲英. 区块链技术: 架构及进展. 计算机学报, 2018, 41(5): 969–988. [doi: 10.11897/SP.J.1016.2018.00969]

[4] 欧阳丽炜, 王帅, 袁勇, 倪晓春, 王飞跃. 智能合约: 架构及进展. 自动化学报, 2019, 45(3): 445–457. [doi: 10.16383/j.aas.c180586]

[5] 贺海武, 延安, 陈泽华. 基于区块链的智能合约技术与应用综述. 计算机研究与发展, 2018, 55(11): 2452–2466. [doi: 10.7544/issn1000-1239.2018.20170658]

[15] 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]
[31] 钱鹏, 刘振广, 何钦铭, 黄步添, 田端正, 王勋. 智能合约安全漏洞检测技术研究综述. 软件学报, 2021. http://www.jos.org.cn/1000-9825/6375.htm [doi: 10.13328/j.cnki.jos.006375]
[34] 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]
[41] 张健, 张超, 玄跻峰, 熊英飞, 王千祥, 梁彬, 李炼, 窦文生, 陈振邦, 陈立前, 蔡彦. 程序分析研究进展. 软件学报, 2019, 30(1): 80–109. http://www.jos.org.cn/1000-9825/5651.htm [doi: 10.13328/j.cnki.jos.005651]
[42] 李舟军, 张俊贤, 廖湘科, 马金鑫. 软件安全漏洞检测技术. 计算机学报, 2015, 38(4): 717–732. [doi: 10.3724/SP.J.1016.2015.00717]
[43] 陈翔, 王莉萍, 顾庆, 王赞, 倪超, 刘望舒, 王秋萍. 跨项目软件缺陷预测方法研究综述. 计算机学报, 2018, 41(1): 254–274. [doi: 10.11897/SP.J.1016.2018.00254]
[51] 于巧, 姜淑娟, 张艳梅, 王兴亚, 高鹏飞, 钱俊彦. 分类不平衡对软件缺陷预测模型性能的影响研究. 计算机学报, 2018, 41(4): 809–824. [doi: 10.11897/SP.J.1016.2018.00809]
[54] 张献, 贲可荣, 曾杰. 基于代码自然性的切片粒度缺陷预测方法. 软件学报, 2021, 32(7): 2219–2241. http://www.jos.org.cn/1000-9825/6261.htm [doi: 10.13328/j.cnki.jos.006261]
[56] 李政亮, 陈翔, 蒋智威, 顾庆. 基于信息检索的软件缺陷定位方法综述. 软件学报, 2021, 32(2): 247−276. http://www.jos.org.cn/1000-9825/6130.htm [doi: 10.13328/j.cnki.jos.006130]



杨慧文(1997－), 男, 硕士生, CCF 学生会员, 主要研究领域为智能合约测试技术.

贾明华(1998－), 男, 硕士生, CCF 学生会员, 主要研究领域为金融科技, 软件测试技术.

崔展齐(1984－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件分析与软件测试技术.

郑丽伟(1979－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为需求软件工程, 社交网络, 数据质量增强.

陈翔(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件缺陷预测, 软件缺陷定位, 组合测试.

刘建宾(1963－), 男, 博士, 教授, 主要研究领域为智能软件技术, 模型驱动开发.