软件学报ISSN 1000-9825, CODEN RUXUEW E-mail: jos@iscas.ac.cn

*Journal of Software*, [doi: 10.13328/j.cnki.jos.000000] http://www.jos.org.cn

©中国科学院软件研究所版权所有. Tel: +86-10-62562563

面向Solidity智能合约的缺陷预测方法[[1]](#footnote-2)\*

杨慧文1,2, 崔展齐2, 陈 翔2, 贾明华2, 郑丽伟2, 刘建宾2

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)

通讯作者: 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,本文提出了面向Solidity智能合约的缺陷预测方法.首先,设计了一组针对Solidity智能合约特有的函数、变量类型和Solidity语言特性的度量元集（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（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指标提升15.17%.此外,本文进一步研究了智能合约缺陷预测中的类不平衡问题,通过SMOTE对数据集进行预处理,能够让缺陷预测模型的F1-score指标平均提升30.5%.

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.面向Solidity智能合约的缺陷预测方法.软件学报.http://www.jos.org.cn/1000-9825/ 0000.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. Ruan Jian Xue Bao/Journal of Software, 2016 (in Chinese).http://www.jos.org.cn/1000-9825/0000.htm

Defect Prediction for Solidity Smart Contracts

YANG Hui-Wen1, CUI Zhang-Qi1,4, CHEN Xiang2, JIA Ming-Hua3, ZHENG Li-Wei1, LIU Jian-Bin1

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)

**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 researches on smart contracts defect detection technologies. Software defect prediction technology is an effective supplement to defect detection technology, which can optimize the allocation of test 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 paper proposes a defect prediction method for Solidity smart contracts. First, a metrics suite (i.e. SC-Sol) which consider the functions, variable types, and limitations of Solidity smart contracts are designed, and it combines with the metrics suite (i.e. COOP) which consider the object-oriented features to COOP-SC-Sol metrics suite. Then, extract relevant metric meta-information from the Solidity code, and perform defect detection to obtain the defects information to construct a Solidity smart contracts defect data set. On this basis, 7 regression models and 6 classification models were applied to predict the defects of Solidity smart contracts to verify the performance differences of different metrics suites and different models in the number and tendency prediction of defects. Experimental results show that, comparing with the COOP, COOP-SC-Sol can improve the performance of the defect prediction model by 15.17% on the . In addition, this paper further studies the problem of class imbalance in smart contracts defect prediction. The result shows that SMOTE can improve the performance of the defect prediction model by 30.5% in F1-score on average.

**Key words**: Software defect prediction; Defect number prediction; Defect tendency prediction; Smart contract; Solidity

# 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年被Szabo提出[37] ，定义为“一套以数字形式指定的承诺，包括合约参与方可以在上面执行这些承诺的协议”[35] ，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[3]，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[35] 。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应用，拓展了区块链的功能。例如可以用智能合约实现一个猜数游戏，参与者可以投入资金并进行猜数，如果猜测的数字符合合约定义的数字，则合约会自动向胜利者的账户发放奖励。目前，智能合约已被广泛应用于资产管理、合同管理等方面，如股权众筹[1]、金融交易[35] 和保险[3]等。

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[7] 等人根据Oyente工具对19366个以太坊智能合约进行检测，发现8833个合约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：2017年11月，Parity钱包遭到攻击，导致3000万美元的资金被盗用[4]；2016年6月，大型众筹项目TheDAO的300多万以太币被非法转移[9]等。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[5][12]。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[[2]](#footnote-3)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[[3]](#footnote-4)（Blockchain Smart Contract Security Checking System, 以下简称BSCSCS）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[5] 利用模糊测试方法，改进了AFL（American Fuzzy Lop）的种子选择策略，从而覆盖更多的分支，能够检测出7种常见的漏洞。

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[25]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[19]关注缺陷预测中数据集的类重叠，Bennin等人[21]探讨了类不平衡现象对数据集的影响等。但据我们所知，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，缺少针对智能合约的特征，例如关注转移或接收货币、外部合约的调用或者特有函数等，这些特征与智能合约缺陷存在相关性。
2. 目前还没有面向智能合约缺陷预测的数据集，难以对智能合约缺陷预测方法的性能进行实验评估。

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（Smart Contract-Solidity，简称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，简称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缺陷预测结果的准确性。

本文的主要贡献如下：

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智能合约特有的函数、变量类型以及Solidity语言的限制，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。
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，还进一步分析了在类不平衡数据集中，不同采样技术对缺陷预测模型性能的影响。

论文剩余内容结构安排如下，第2节为相关工作，第3节介绍COOP度量元，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节为总结与展望。

# 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。

## 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[6]基于预定义的参数值生成测试用例，构建EVM并对进行插装，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[5]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[36] 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[7]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[8]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[11]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[10]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，缺陷预测可以在缺陷检测之前，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、更准确的检测。另外，对模块缺陷数量的预测可用于评估测试的充分性，作为停止测试的度量指标。

## 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[13][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][18]。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[26]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[30]关注程序的控制流复杂度，提出了圈复杂度度量元；Bansiya和Davis[27]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。

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] [16] 。Pelayo[20]等人在缺陷预测中应用合成少数样本过采样技术[24]（Synthetic Minority Over-sampling Technique，SMOTE）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[21]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[14]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。

然而，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，目前还没有关于智能合约缺陷预测的研究。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，并利用该数据集对Solidity智能合约进行缺陷预测实证研究，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。

# 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，主要关注代码复杂度以及面向对象特征两方面。本文针对Solidity智能合约设计了SC-Sol度量元集。

## COOP度量元集

COOP度量元集由Chidamber、Kemerer、Bansiy、Davis等人提出度量元组合而成[32]，主要关注代码复杂度以及面向对象程序特征两方面。Gong等人[19]和Chen[25]等人将COOP度量元集应用到软件缺陷预测中，分别讨论了类重叠问题以及无监督模型与有监督模型的差异问题，取得了较好的效果。

COOP度量元集的类别、缩写、全称与描述如表1所示。COOP度量元集中，Chidamber、Kemerer[26]和Tang[29]等人提出了一组关注面向对象特征的度量元，即WMC、DIT、NOC、CBO、RFC、LCOM、IC、CBM和AMC。Bansiy和Davis[27]关注聚合、抽象等特征，提出了NPM、DAM、MOA、MFA和CAM度量元。另外，Henderson、Martin以及McCabe等人分别提出了LCOM3[28]、CA、CE[31]、Max\_cc和Avg\_cc[30]度量元。

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。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。

表1 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 |

## SC-Sol度量元集

SC-Sol是本文针对Solidity智能合约特有的函数、变量类型以及Solidity语言的限制所设计的度量元集，SC-Sol度量元集的类别、缩写、全称与描述如表2所示。

表2 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 |
| NOV | Number of Origin Variable | tx.origin变量数量 |
| 不当依赖 | TNGF | Total Number of Gasleft Function | gasleft函数数量 |
| TNTDF | Total Number of Time Dependence Function | 与时间相关函数或变量数量 |
|  | TNBND | Total Numbr of Block Number Dependence | block.number变量数量 |

表2中的SC-Sol度量元集可划分为地址相关、结构相关、异常处理、参数使用和易错操作5个类型，其中各个度量元的粒度均为contract和library，具体分类与各个度量元的解释如下。

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。address是Solidity智能合约中的地址类型变量，是用于实现以太币转移功能的核心变量。Solidity为address提供了3个API用于账户间以太币的转移，即address.transfer(*amount*)、address.send(*amount*)和address.call(*amount*)。

NTF，该度量元统计transfer函数的数量。transfer函数是address类型变量的一个成员函数，接收一个uint256类型的参数*amount*，功能为向地址发送*amount*数量的以太币，失败时抛出异常.

NSF，该度量元统计send函数的数量。send函数与transfer函数相似，同样为address类型变量的成员函数，接受一个uint256类型的参数*amount*，功能为向地址发送数量为*amount*的以太币，与transfer函数的区别在于，send函数发送失败时返回false。send函数转移以太币到目标合约后，则会自动调用目标合约的fallback函数，由于send函数默认的gas为2300，如果被调用合约的fallback函数过于复杂，例如涉及状态变量的修改或者函数调用等，则会抛出ErrOutGas异常，并返还以太币给调用者。如果调用者没有正确处理send函数的返回值或异常，可能导致代码逻辑错误[6] [39] 。

NCF，该度量元统计call函数的数量。call函数是send和transfer函数的底层调用，与send和transfer不同，call函数除了可以用于转移货币，还可以调用外部合约的函数。call函数如果不指定gas的数量，则默认将所有gas用于传输，由此可能导致重入漏洞。重入漏洞是智能合约中经典漏洞之一，导致该漏洞的原因是被攻击合约具有可重入函数[5] [6] [7] 。举例来讲，合约A通过函数fun1中的call函数向外部合约B转移以太币时，如果开发者没有指明B合约中可接受以太币的目标函数（即带有payable标识符的函数），则会自动调用合约B中的fallback函数，恶意攻击者会在合约B中的fallback函数里重新调用合约A中的fun1，如果fun1中没有先进行更改变量的操作，而是先转移以太币再进行变量修改，则再次调用过程中仍会满足转移条件，从而不断地将合约A的以太币转移给合约B。此外，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[38] 。

NDF，该度量元统计delegatecall函数的数量。delegatecall函数与call函数相似，区别在于delegatecall函数会在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fun2，则fun2中的this会指向合约A。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合约相关，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3千万美金的以太币[6] 。

1. 结构相关度量元

NS，该度量元统计结构体的数量。结构体可以自定义数据类型，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modifier的数量。在Solidity语言中，合约类似于面向对象编程语言的类，除了函数、结构体和成员变量等其他面向对象语言常见的结构外，还有函数修饰器（modifier）和事件（event）两种其他语言没有的结构。其中，函数修饰器可以改变函数的行为，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[9]。

NE，该度量元统计event的数量。event接口允许开发者使用以太坊虚拟机中的日志基础设施，开发者可以利用event接口监听合约事件

1. 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。异常处理可以增强代码的健壮性，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。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。

1. 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，如以太币的转账或信息交互等，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。特别是当函数使用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1所示，我们以地址为0x0dd1093721997be0421f49597820df3bfd1dc755[[4]](#footnote-5)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f1的参数列表中只含有1个参数；NUAP的值为3，因为f1中有3个函数调用了address类型参数，即newOwner；TNMSV的值为1，因为f1中最后一行对状态变量进行了修改，修改次数为1次。

contract Ownable {

address public owner;

event event1(address indexed previousOwner, address indexed newOwner);

modifier modifier1() {

require(msg.sender == owner);

\_;

}

function f1(address newOwner) onlyOwner public {

require(newOwner != address(0));

OwnershipTransferred(owner, newOwner);

owner = newOwner;

}

}

图1 Solidity源代码示例

1. 易错操作

NDO，该度量元统计各个函数使用除号的数量。Solidity智能合约中，两个整数相除的结果会被截断（仅被编译为EVM中的DIV操作码），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指出乘法前进行除法运算（DivisionBeforeMultiply）和调用价值前进行除法运算（DivisionBeforeCallvalue）两种漏洞与整数除法相关。此外，被除数如果为0则会导致除零错误[11] 。

NOV，该度量元统计tx.origin变量的数量。Solidity智能合约中，tx.origin代表完整调用链的原始发送方，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[40] [41] 。BSCSCS中检测到的UseOfOrigin类型漏洞与该变量有关。

1. 不当依赖

TNGF，该度量元统计gasleft函数的数量。gasleft用于获得剩余的gas。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。

TNTDF，该度量元统计合约中与时间相关的函数或变量数量，即使用block.timestamp变量和now函数的数量。block.timestamp变量和now函数的作用等价，均是用于在合约中获得当前区块的时间戳。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[7] 。

TNBND，该度量元统计使用block.number变量的数量。block.number变量存储当前合约所在区块的区号。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。

# 面向Solidity智能合约的缺陷预测方法

基于所设计的COOP-SC-Sol度量元集，本文提出了面向Solidity智能合约的缺陷预测方法。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，并为每个代码模块标记各种类型的缺陷数量，以构建缺陷数据集。然后，对Solidity智能合约缺陷数据集中的类不平衡问题采用不同采样方法进行数据预处理。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，类不平衡问题处理以及模型构建与评估。

## 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[[5]](#footnote-6)获取Solidity智能合约的源码，使用solidity-parser-antlr[[6]](#footnote-7)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。我们根据COOP-SC-Sol度量元的定义，组合使用不同的AST Node类型对sol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。

需要注意的是，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。~~在未来工作中可以尝试利用以太坊官方的solc编译器获得AST信息，具体请查看第7节相关工作。~~

## 类不平衡问题处理

在构建数据集的过程中，我们发现Solidity缺陷数据集存在较为严重的类不平衡现象。表3为Solidity智能合约缺陷数据集中，各类型缺陷按照缺陷模块百分比（Percentage of Defect Modules，PDM）降序排序后的结果。其中，PDM的计算公式如公式（1）所示。其中，为含有该类型缺陷的模块数量，为总模块数量。

|  |  |
| --- | --- |
|  | (1) |

表3中，缺陷类型为《白皮书》中提出的21种Solidity智能合约缺陷类型。通常情况下，PDM低于25%的数据集存在类不平衡问题[21]，从表3中可以看出Solidity智能合约的缺陷数据集存在较为严重的类不平衡问题，将会对分类器的预测性能产生影响。为缓解类不平衡问题，本方法参考文献[15] [16] 中使用软件缺陷预测领域中常见的采样方法分别对每种缺陷类型的数据集进行处理。具体来讲，本方法中使用了随机欠采样（Random Under Sampling，RUS）、随机过采样（Random Over Sampling，ROS）和SMOTE方法对数据集进行处理。

表3 Solidity智能合约缺陷类型与缺陷模块百分比

|  |  |  |
| --- | --- | --- |
| 序号 | 缺陷类型 | PDM |
| 1 | UnrestrictedWrite | 16.76 |
| 2 | UnsafeCallTarget | 15.33 |
| 3 | MissingInputValidation | 10.74 |
| 4 | UnrestrictedEtherFlow | 6.73 |
| 5 | TODAmount | 5.67 |
| 6 | UnsafeDependenceOnBlock | 5.25 |
| 7 | TODReceiver | 4.11 |
| 8 | LockedEther | 2.15 |
| 9 | DivisionBeforeCallvalue | 2.05 |
| 10 | DAOConstantGas | 1.70 |
| 11 | DivisionBeforeMultiply | 1.52 |
| 12 | TODTransfer | 1.44 |
| 13 | DAOMethodCall | 0.65 |
| 14 | UnhandledException | 0.51 |
| 15 | WriteOnly | 0.49 |
| 16 | UseOfOrigin | 0.3 |
| 17 | UnsafeDependenceOnGas | 0.21 |
| 18 | DelegateCallWithUserInput | 0.2 |
| 19 | UnprivilegedSuicide | 0.13 |
| 20 | DAO | 0.11 |
| 21 | IntegerOverflow | 0 |

## 预测模型构建与评估

本文参考文献[15] [19] [21] [22] [23] ，使用软件缺陷预测领域中常用的7种回归模型和6种二分类模型分别作为Solidity智能合约的缺陷数量和倾向性预测模型。

使用的缺陷数量预测模型有：线性回归（LR）、贝叶斯岭（BR）、决策树回归（DTR）、随机森林回归（RFR）、K邻近回归（KNNR）、梯度加速回归（GBR）和支持向量机回归（SVR）。

使用的缺陷倾向性预测模型有：伯努利贝叶斯分类器（BNB）、高斯贝叶斯分类器（GNB）、K邻近分类器（KNNC）、决策树分类器（DTC）、随机森林分类器（RFC）和支持向量机分类器（SVC）。

对实验结果使用平均绝对误差[33] （Mean Absolute Error，简称MAE）、平均缺陷百分比[16] [17] （Fault Percentile Average，简称FPA）、宏F1-score（Macro F1score，）、F1-score[13][33]、AUC作为预测模型的评估指标。其中，MAE和FPA作为缺陷数量预测模型的评估指标，、F1-score和AUC作为缺陷倾向性预测模型的评估指标。

MAE计算方式如公式（2）所示。其中，代表样本总数量，代表预测的缺陷数量，代表真实含有的缺陷数量，MAE的值越低，代表预测性能越好。

|  |  |
| --- | --- |
|  | (2) |

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（3）所示。其中，为样本总数量，为缺陷总数量。将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，使用代表排序后第个样本的真实缺陷个数。FPA的值越高，代表预测性能越好。

|  |  |
| --- | --- |
|  | (3) |

为了计算F1-score，需要使用如表4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。

表4 二分类问题的混淆矩阵

|  |  |  |
| --- | --- | --- |
| 预测样本  真实样本 | 正 | 负 |
| 正 | TP | FN |
| 负 | FP | TN |

根据表4中的混淆矩阵，召回率Recall的计算公式如公式（4）所示。

|  |  |
| --- | --- |
|  | (4) |

精确率Precision的计算公式如公式（5）所示。

|  |  |
| --- | --- |
|  | (5) |

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（6）计算F-score。其中，时，称为F1-score。F1-score的值越高，代表预测性能越好。

|  |  |
| --- | --- |
|  | (6) |

当存在多个类标签时，需要合并考虑所有类标签的F1-score。本文使用作为评估存在多个类标时缺陷倾向性预测模型的指标。的计算公式如公式（7）所示。公式（7）中，代表类标的个数，表示第个类标的F1-score值。的值越高，代表预测性能越好。

|  |  |
| --- | --- |
|  | (7) |

本文采用AUC评估指标讨论类不平衡数据集对缺陷预测模型性能的影响。AUC(Area Under a Receiver Operating Characteristic Curve)是接受者操作特性曲线的下面积，ROC(Receiver Operating Characteristic)是一条横轴为FPRate，纵轴为TPRate的曲线，曲线上每个点是由模型中取不同的阈值所构成，AUC可用于评估了基于类不平衡数据集构建的模型[19] 。其中，FPRate和TPRate的计算公式如公式（8）和（9）所示。

|  |  |
| --- | --- |
|  | (8) |

|  |  |
| --- | --- |
|  | (9) |

# 实验设计与结果分析

## 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3个研究问题。

RQ1：对于缺陷数量预测，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？

RQ2：对于缺陷倾向性预测，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？

RQ3：类不平衡情况下，不同采样技术对Solidity智能合约缺陷预测性能的提升效果如何？

本文设计的实验框架如图2所示。

图形用户界面

描述已自动生成

图2 实验框架

### 爬取Solidity源代码和检测结果，利用AST工具获取度量元信息，结合形成缺陷数据集（智能合约缺陷数据集构建）

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。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数量检测结果，具体来讲，在149363个源代码文件中，共有6519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。

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[33] 的方法，将各个类型的缺陷数量二值化，即如果该类型缺陷的数量大于等于1，则标记为含有该类型缺陷，否则标记为不含有该类型缺陷。

接下来，使用solidity-parser-anltr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-anltr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6519个可以获得检测报告的源代码文件中，共有4203个源代码文件可以获得度量元信息,其中包括21138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和缺陷倾向性标签组合，分别得到缺陷数量预测数据集和缺陷倾向性预测数据集。

### 训练缺陷预测模型，评估方法的有效性（模型构建及有效性评价）

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（RQ1）。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、F1-score和AUC进行度量（RQ2）。

对于类不平衡问题，我们采用ROS、RUS和SMOTE共3种采样技术对缺陷倾向性预测数据集的训练集进行预处理，使用F1-score和AUC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（RQ3）。

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，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10折交叉验证随机重复10次。

对于类不平衡问题的处理，实验中参考文献[21] 的方法，在划分训练集和测试集时采用按PDM进行分层切割，即训练集和测试集的PDM相同。

（开头）所有实验均运行在CPU为i7-6700H，内存为8GB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[[7]](#footnote-8)，过采样方法的实现使用python第三方库imbalanced-learn[[8]](#footnote-9)，模型和过采样方法的超参数均为缺省值。

## 实验结果与分析

### 针对RQ1的结果分析

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，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3、图4和表5所示。其中，表5中使用粗体标记不同模型和不同度量元集的最优结果。

图3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，每组内使用3种颜色代表3种度量元集，并按照3种度量元集的MAE平均值升序排列。

从图3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5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）。

图4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，每组内使用3种颜色代表3种度量元集，并按照3种度量元集的FPA平均值降序排列。

从图4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5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）。

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，我们将各个模型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）。

图表, 箱线图

描述已自动生成

图3 缺陷数量预测模型性能比较（MAE）

图表, 箱线图

描述已自动生成

图4 缺陷数量预测模型性能比较（FPA）

综合以上分析，我们对RQ1得出以下结论：对于Solidity缺陷数量预测，SC-Sol在MAE评估指标中性能最好，COOP-SC-Sol在FPA评估指标中性能最好。此外，综合考虑MAE和FPA指标的排名，RFR模型的平均性能最好，使用RFR模型时，COOP-SC-Sol的预测性能优于SC-Sol。

### 针对RQ2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，与RQ1中的实验类似，我们将由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，并使用和AUC两种指标进行度量。

需要注意的是，如4.2节所述，Solidity缺陷数据集存在较为严重的不平衡现象，因此，为避免类不平衡问题对缺陷预测模型性能产生过大影响，此次实验中我们仅分析PDM大于10%的缺陷类型，即UnrestrictedWrite、UnsafeCallTarget和MissingInputValidation。

分别对上述3种缺陷构建缺陷倾向性预测模型，最终得到3组预测结果（即3个类标签的预测结果），并使用和AUC指标进行评估。实验结果如图5和表6所示，分别是Solidity智能合约缺陷倾向性预测的和AUC评估结果，根据二分类模型将箱图分为6组，按照3种度量元集的和AUC值降序排列。每组内使用3种颜色代表3种度量元集。

表5 使用3种度量元集构建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 | 2.69 | **0.83** |
| GBR | 2.809 | 0.819 |
| BR | 3.2 | 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 | 0.78 |
| 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 |

图表, 箱线图

描述已自动生成

图5 缺陷倾向性预测模型性能比较（）

图表, 箱线图

描述已自动生成

图6 缺陷倾向性预测模型性能比较（AUC）

由图5可以看出，RFC模型在3种度量元集中的平均值最高，为0.518，其余依次为KNNC（0.499）、DTC（0.48）、BNB（0.347）、SVC（0.259）以及GNB（0.219）。另外，结合表6可以看出，使用COOP-SC-Sol度量元集时，不同模型的平均值最高，为0.424，其余依次为SC-Sol（0.385）和COOP（0.353）。使用COOP-SC-Sol度量元集时，相对于SC-Sol和COOP度量元集的分别提升了10.12%和20.11%。

由图6可以看出，RFC模型在3种度量元集中AUC的平均值最高，为0.889，其余依次为KNNC（0.826）、SVC（0.813）、GNB（0.759）、BNB（0.759）以及DTC（0.712）。另外，结合表7可以看出，使用COOP-SC-Sol度量元集时，不同模型的AUC平均值最高，为0.809，其余依次为COOP（0.792）和SC-Sol（0.778）。

综合以上分析，我们对RQ2得出以下结论：对于Solidity缺陷倾向性预测，使用和AUC进行评价时，RFC模型的性能最好，不同模型使用COOP-SC-Sol度量元集的平均性能最好。

表6使用3种度量元集构建缺陷倾向性预测模型的值

|  |  |  |  |
| --- | --- | --- | --- |
| 度量元集 | 模型 | F1-scoremacro | |
| 单值 | 均值 |
| COOP-SC-Sol | RFC | **0.527** | **0.424** |
| KNNC | 0.516 |
| DTC | 0.483 |
| BNB | 0.433 |
| GNB | 0.227 |
| SVC | 0.357 |
| SC-Sol | RFC | **0.527** | 0.385 |
| KNNC | 0.492 |
| DTC | 0.5 |
| BNB | 0.272 |
| GNB | 0.236 |
| SVC | 0.28 |
| COOP | RFC | **0.5** | 0.353 |
| KNNC | 0.489 |
| DTC | 0.465 |
| BNB | 0.337 |
| GNB | 0.193 |
| SVC | 0.14 |

表7使用3种度量元集构建缺陷倾向性预测模型的AUC值

|  |  |  |  |
| --- | --- | --- | --- |
| 度量元集 | 模型 | AUC | |
| 单值 | 均值 |
| COOP-SC-Sol | RFC | **0.9** | **0.809** |
| KNNC | 0.838 |
| DTC | 0.714 |
| BNB | 0.78 |
| GNB | 0.774 |
| SVC | 0.847 |
| SC-Sol | RFC | **0.881** | 0.778 |
| KNNC | 0.822 |
| DTC | 0.724 |
| BNB | 0.738 |
| GNB | 0.733 |
| SVC | 0.771 |
| COOP | RFC | **0.885** | 0.792 |
| KNNC | 0.82 |
| DTC | 0.698 |
| BNB | 0.759 |
| GNB | 0.77 |
| SVC | 0.82 |

### 针对RQ3的结果分析

如4.2 节所述，Solidity智能合约缺陷数据集中存在较为严重的不平衡现象，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RQ2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；最后将进行预处理的预测结果和不进行预处理的预测结果进行比较。

实验的结果如图7、图8、表8和表9所示，其中，图7和图8中序号与表3中的序号对应，None、SMOTE、ROS和RUS分别表示不做数据预处理、进行SMOTE、ROS和RUS预处理的预测结果。需要注意的是，表3中的21号缺陷类型（即IntegerOverflow类型缺陷）的PDM为0，即缺陷数据集中没有正样本，无法构建二分类模型和采样处理，因此实验中不对21号缺陷构建二分类模型。表8和表9分别是20种缺陷倾向性预测结果F1-score和AUC的平均值比较。

结合图7和表8可以看出，在F1-score指标中，SMOTE和ROS在所有20种缺陷预测模型中均优于不做数据预处理的模型，其中，SMOTE和ROS在16号缺陷中对模型的提升效果最大，相对于不进行预处理分别提升了79.8%%和39.4%%。在平均20种缺陷预测模型的F1-score后，相对于不进行数据预处理提升了27.4%。RUS在部分缺陷模型中对预测性能有一定提升，在6号缺陷中对模型提升效果最大，相对于不进行预处理提升了59.8%%，但是，RUS在PDM下降到2.05%后会拉低预测性能。造成这种结果的原因是RUS会将随机去除负样本，使负样本数量与正样本数量相等，当PDM较低时，正样本数量较少，经过RUS处理后整体样本数量相对于None减少较多，导致模型无法学习到足够的知识，从而降低了模型的预测性能。

从图8和表9中可以看出，AUC指标下的3种预处理方法和不进行预处理的结果相近。从平均值看出，SMOTE相对于不进行预处理提升了2.1%，RUS提升了1.6%。

综合以上分析，我们对RQ3得出结论：使用过采样技术能够提升Solidity智能合约缺陷倾向性预测的性能。其中，SMOTE采样方法提升性能的效果较为明显。

表8 使用采样技术预处理与不使用采样技术预处理的预测模型性能差异分析（F1-score）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | None | SMOTE | ROS | RUS |
| 1 | 0.528 | +0.027 | +0.03 | +0.017 |
| 2 | 0.489 | +0.066 | +0.056 | +0.087 |
| 3 | 0.573 | +0.03 | +0.029 | -0.027 |
| 4 | 0.338 | +0.1 | +0.071 | +0.076 |
| 5 | 0.308 | +0.12 | +0.06 | +0.073 |
| 6 | 0.249 | +0.143 | +0.093 | +0.149 |
| 7 | 0.264 | +0.068 | +0.046 | +0.021 |
| 8 | 0.033 | +0.029 | +0.069 | +0.045 |
| 9 | 0.216 | +0.024 | +0.031 | -0.048 |
| 10 | 0.202 | +0.061 | +0.047 | -0.044 |
| 11 | 0.162 | +0.089 | +0.03 | -0.046 |
| 12 | 0.239 | +0.06 | +0.065 | -0.109 |
| 13 | 0 | +0.039 | +0.006 | +0.049 |
| 14 | 0.104 | +0.098 | +0.07 | -0.028 |
| 15 | 0.211 | +0.076 | +0.086 | -0.17 |
| 16 | 0.292 | +0.233 | +0.115 | -0.097 |
| 17 | 0.204 | -0.016 | +0 | -0.19 |
| 18 | 0.272 | -0.01 | -0.002 | -0.259 |
| 19 | 0 | +0.05 | +0.097 | +0.006 |
| 20 | 0 | 0 | 0 | +0.005 |
| 平均值 | 0.234 | 0.299(+0.064) | 0.284(+0.05) | 0.209(-0.025) |

表9 使用采样技术预处理与不使用采样技术预处理的预测模型性能差异分析（AUC）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | None | SMOTE | ROS | RUS |
| 1 | 0.871 | -0.001（pvalue） | 0 | 0 |
| 2 | 0.906 | 0 | -0.001 | -0.003 |
| 3 | 0.926 | 0.002 | 0.003 | -0.001 |
| 4 | 0.926 | 0.002 | 0.001 | -0.003 |
| 5 | 0.929 | 0.002 | 0 | -0.002 |
| 6 | 0.919 | 0.004 | 0.002 | -0.003 |
| 7 | 0.918 | 0.006 | 0.001 | 0.001 |
| 8 | 0.768 | 0.014 | 0.001 | -0.018 |
| 9 | 0.906 | 0.016 | -0.004 | 0.012 |
| 10 | 0.926 | 0.016 | 0.005 | 0.008 |
| 11 | 0.89 | 0.008 | -0.009 | 0.001 |
| 12 | 0.908 | 0.03 | 0.015 | 0.017 |
| 13 | 0.837 | 0.028 | -0.031 | 0.042 |
| 14 | 0.941 | 0.011 | -0.008 | -0.012 |
| 15 | 0.836 | 0.034 | 0.015 | 0.069 |
| 16 | 0.999 | 0 | 0 | -0.003 |
| 17 | 0.872 | 0.054 | -0.034 | -0.014 |
| 18 | 0.84 | 0.024 | 0.008 | -0.021 |
| 19 | 0.74 | 0.047 | -0.001 | 0.06 |
| 20 | 0.524 | 0.059 | -0.017 | 0.156 |
| 平均值 | 0.869 | 0.887(+0.018) | 0.866(-0.003) | 0.883(+0.014) |

图表, 条形图

描述已自动生成

图7 使用不同采样技术进行数据预处理的缺陷倾向性预测性能对比（F1-score）

图表

描述已自动生成

图8 使用不同采样技术进行数据预处理的缺陷倾向性预测性能对比（AUC）

# 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，我们使用开源的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，我们使用第三方机器学习库sckit-learn中的模型与评估指标，SMOTE算法的实现由第三方python库imbalanced-learn提供。此外，我们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与实验结果是否存在一般性有关，用于描述在多大程度上可以将研究结果推广到其他情景。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，可以保证实验所用数据集的正确性与可靠性。

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。文献[33]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，使用两种数据集构建的分类模型的预测结果中，F1-score差异值的中位数小于0.1。因此，本文所构建的Solidity缺陷数据集可在一定程度上作为参考。在未来工作中，我们将考虑其他Solidity智能合约缺陷检测平台，如ContractGuard等，以完善Solidity缺陷数据集，进一步缓解外部有效性威胁。

构造有效性分析主要与实验使用的评估指标有关。本文使用MAE和FPA作为缺陷数量预测的评估指标，使用F1-score、和AUC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[13] [15] [16] [33] ，可以有效评估评估缺陷预测模型性能。

# 总结与展望

本文首先提出了一组针对Solidity智能合约特有的函数、变量类型以及智能合约限制的度量元集，即SC-Sol度量元集，并将其与COOP度量元集在7种缺陷数量预测模型和6种缺陷倾向性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度量元集具有更好的缺陷预测性能，随机森林回归模型和随机森林分类模型分别在缺陷数量预测和缺陷倾向性预测中性能最好。另外，我们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SMOTE过采样技术会对缺陷倾向性预测模型的性能有一定的提升。

在未来的工作中，我们将采用更多智能合约缺陷检测平台获取缺陷信息，以进一步完善智能合约缺陷数据集，此外，我们还将尝试对Serpent等其他智能合约进行缺陷预测。

References:

1. Yuan Yong, Wang Fei-Yue. Blockchain: The State of the Art and Future Trends. Acta Automatica Sinica, 2016, 42(4): 581-589 (in Chinese)
2. Shao Qi-Feng, Jin Che-Qing, Zhang Zhao, Qian Wei-Ning, Zhou Ao-Ying. Blockchain: Architecture and Research Progress. Chinese Journal of Computers, 2018, 41(05): 969-988(in Chinese)
3. He Hai-Wu, Yan An, Chen Ze-Hua. Survey of Smart Contract Technology and Application Based on Blockchain. Journal of Computer Research and Development, 2018, 55(11): 2452-2466(in Chinese).
4. Liu J, Liu Z. Survey on Security Verification of Blockchain Smart Contracts. IEEE Access, 2019, 7:77894-77904.
5. Tai D, Long H, Jun S, Yun L, Quang T. 2020. SFuzz: an efficient adaptive fuzzer for solidity smart contracts//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. New York, USA, 2020: 778–788.
6. Jiang B, Liu Y, Chan WK. ContractFuzzer: fuzzing smart contracts for vulnerability detection//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Montpellier, France, 2018: 259–269.
7. Luu L, Chu DH, Olickel H, Saxena P, Hobor A. Making smart contracts smarter//Proceedings of the ACM Conference on Computer and Communications Security. New York, USA, 2016: 254–269.
8. Krupp J, Rossow C. teEhter: gnawing at ethereum to automatically exploit smart contracts//Proceedings of the 27th USENIX Security Symposium. Baltimore, USA, 2018: 1317–1333.
9. Chen J, Xia X, Lo D, Grundy J, Luo X, Chen T. Defining smart contract defects on Ethereum. IEEE Transactions on Software Engineering, 2020: ( Early Access, DOI: 10.1109/TSE.2020.2989002)
10. Nikolić I, Kolluri A, Sergey I, Saxena P, Hobor A. Finding the greedy, prodigal, and suicidal contracts at scale//Proceedings of the 34th Annual Computer Security Applications Conference. New York, USA, 2018: 653–663.
11. Christof F, Julian S, Radu S. Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts//Proceedings of the 34th Annual Computer Security Applications Conference. New York, USA, 2018: 664–676.
12. Marino B, Juels A. Setting standards for altering and undoing smart contracts//International Symposium on Rules and Rule Markup Languages for the Semantic Web. Springer, Cham, 2016: 151-166.
13. Chen Xiang, Wang Shu-Ping, Gu Qing, Wang Zan, Ni Chao, Liu Wang-Shu, Wang Qiu-ping. A Survey on Cross-Project Software Defect Prediction Methods. Chinese Journal of Computers, 2018, 41(1): 254-274(in Chinese).
14. Yu Qiao, Jiang Shu-Juan, Zhang Yan-Mei, Wang Xing-Ya, Gao Peng-Fei, Qian Jun-Yan. 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).
15. Chen Xiang, Gu Qing, Liu Wang-Shu, Liu Shu-Long, Ni Chao. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016,27(1):1−25 (in Chinese).
16. Gong Li-Na, Jiang Shu-Juan, Jiang Li. Research progress of software defect prediction. Ruan Jia Xue Bao/Journal of Software, 2019,30(10):3090-3114 (in Chinese).
17. Weyuker E J, Ostrand T J, Bell R M. Comparing the effectiveness of several modeling methods for fault prediction. Empirical Software Engineering, 2010, 15(3): 277-295.
18. 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.
19. Gong L, Jiang S, Wang R, Jiang L. Empirical evaluation of the impact of class overlap on software defect prediction//Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering. San Diego, USA, 2019: 698–709.
20. Pelayo L, Dick S. Applying novel resampling strategies to software defect prediction// Proceedings of the Annual Conference of the North American Fuzzy Information Processing Society. San Diego, USA, 2007: 69–72.
21. Bennin K, Keung J, Phannachitta P, Monden A, Mensah S. MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. IEEE Transactions on Software Engineering, 2017, 44(6): 534-550.
22. Chen L, Fang B, Shang Z, Tang Y. Tackling class overlap and imbalance problems in software defect prediction. Software Quality Journal, 2018, 26(1): 97–125.
23. Cabral GG, Minku LL, Shihab E, Mujahid S. Class imbalance evolution and verification latency in just-in-time software defect prediction//Proceedings of the International Conference on Software Engineering, Montreal, Canada, 2019: 666–676.
24. Chawla N V, Bowyer K W, Hall L O, Kegelmeyer W P. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 2002, 16: 321-357.
25. Chen X, Zhang D, Zhao Y, Cui Z, Ni C. Software defect number prediction: unsupervised vs supervised methods. Information and Software Technology, 2019, 106: 161–181.
26. Chidamber S R, Kemerer C F. A metrics suite for object oriented design. IEEE Transactions on software engineering, 1994, 20(6): 476-493.
27. Bansiya J, Davis CG. A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 2002, 28(1): 4–17.
28. Henderson, Sellers. Object-oriented metrics: measures of complexity. Prentice Hall, Prentice-Hall, Inc., 1996.
29. Tang M-H, Kao M-H, Chen M-H. An empirical study on object-oriented metrics//Proceedings of the sixth international software metrics symposium, Boca Raton, USA, 1999: 242–249.
30. McCabe T J. A complexity measure. IEEE Transactions on software Engineering, 1976 (4): 308-320.
31. Martin R. OO design quality metrics. An analysis of dependencies, 1994, 12(1): 151-170.
32. Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction//Proceedings of the 6th international conference on predictive models in software engineering, New York, USA, 2010: 1-10.
33. Yatish S, Jiarpakdee J, Thongtanunam P, Tantithamthavorn C. Mining software defects: should we consider affected releases?// Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering, Montreal, Canada, 2019: 654-665.
34. Machine Learning for the Detection of Oil Sills in Satellite Radar Images
35. Smart Contract: Architecture and Resuarch Progresses(in Chinese)
36. ReGuard
37. Formalizing and securing relationships on public networks
38. sCompile
39. A survey of attacks on Ethereum smart contract
40. SmartCheck: Static Analysis of Ethereum Smart Contracts
41. Security Assurance for Smart Contract

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

[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(05): 969-988

[3] 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.

[13] 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

[14] 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.

[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

[16] 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

[35] 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

1. \* 基金项目: 江苏省前沿引领技术基础研究专项(BK202002001);国家自然科学基金项目(No. 61702041);江苏省自然科学基金面上项目(No. BK20181353);北京信息科技大学“勤信人才”培育计划项目(No. QXTCP C201906)

   Jiangsu Provincial Frontier Leading Technology Fundamental Research Project (BK20202001), the National Natural Science Foundation of China (No. 61702041), the Jiangsu Natural Science Foundation of General Program (No. BK20181353), and the Beijing Information Science and Technology University "Qin-Xin Talent" Cultivation Project (No. QXTCP C201906).

   收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos在线出版时间: 0000-00-00

   CNKI在线出版时间: 0000-00-00 [↑](#footnote-ref-2)
2. ContractGuard, https://contract.guardstrike.com [↑](#footnote-ref-3)
3. 区块链智能合约安全检测平台, http://www.sjtubsrc.net [↑](#footnote-ref-4)
4. https://etherscan.io/address/0x0dd1093721997be0421f49597820df3bfd1dc755#code [↑](#footnote-ref-5)
5. Xblock, http://xblock.pro/ethereum/ [↑](#footnote-ref-6)
6. solidity-parser-antlr, https://github.com/federicobond/solidity-parser-antlr [↑](#footnote-ref-7)
7. scikit-learn, https://scikit-learn.org/ [↑](#footnote-ref-8)
8. imbalanced-learn, https://github.com/scikit-learn-contrib/imbalanced-learn.git [↑](#footnote-ref-9)