软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(1):50−102 [doi: 10.13328/j.cnki.jos.006503]

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

面向对象软件度量阈值的确定方法: 问题、进展与挑战*

梅元清[1,2], 郭肇强[1,2], 周慧聪[1,2], 李言辉[1,2], 陈 林[1,2], 卢红敏[1,2], 周毓明[1,2]

[1](计算机软件新技术国家重点实验室 (南京大学), 江苏 南京 210023)

[2](南京大学 计算机科学与技术系, 江苏 南京 210023)

通信作者: 李言辉, E-mail: yanhuili@nju.edu.cn; 周毓明, E-mail: zhouyuming@nju.edu.cn

摘 要: 面向对象软件度量是理解和保证面向对象软件质量的重要手段之一. 通过将面向对象软件的度量值与其阈值比较, 可简单直观评价其是否有可能包含缺陷. 确定度量阈值方法主要有基于数据分布特征的无监督学习方法和基于缺陷相关性的有监督学习方法. 两类方法各有利弊: 无监督学习方法无需标签信息而易于实现, 但所得阈值的缺陷预测性能通常较差; 有监督学习方法通过机器学习算法提升所得阈值的缺陷预测性能, 但标签信息在实际过程中不易获得且度量与缺陷链接技术复杂. 近年来, 两类方法的研究者不断探索并取得较大进展. 同时, 面向对象软件度量阈值确定方法研究仍存在一些亟待解决的挑战. 对近年来国内外学者在该领域的研究成果进行系统性的总结. 首先, 阐述面向对象软件度量阈值确定方法的研究问题. 其次, 分别从无监督学习方法和有监督学习方法总结相关研究进展, 并梳理具体的理论和实现的技术路径. 然后, 简要介绍面向对象软件度量阈值的其他相关技术. 最后, 总结当前该领域研究过程面临的挑战并给出建议的研究方向.

关键词: 面向对象软件; 度量; 阈值; 缺陷预测

中图法分类号: TP311

中文引用格式: 梅元清, 郭肇强, 周慧聪, 李言辉, 陈林, 卢红敏, 周毓明. 面向对象软件度量阈值的确定方法: 问题、进展与挑战. 软件学报, 2023, 34(1): 50–102. http://www.jos.org.cn/1000-9825/6503.htm

英文引用格式: Mei YQ, Guo ZQ, Zhou HC, Li YH, Chen L, Lu HM, Zhou YM. Deriving Object-oriented Metric Thresholds: Research Problems, Progress, and Challenges. Ruan Jian Xue Bao/Journal of Software, 2023, 34(1): 50–102 (in Chinese). http://www.jos.org.cn/1000-9825/6503.htm

Deriving Object-oriented Metric Thresholds: Research Problems, Progress, and Challenges

MEI Yuan-Qing[1,2], GUO Zhao-Qiang[1,2], ZHOU Hui-Cong[1,2], LI Yan-Hui[1,2], CHEN Lin[1,2], LU Hong-Min[1,2], ZHOU Yu-Ming[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

Abstract: Object-oriented software metrics are important for understanding and guaranting the quality of object-oriented software. By comparing object-oriented software metrics with their thresholds, it could be simply and intuitively evaluated whether there is a bug. The methods to deriving metrics thresholds mainly include unsupervised learning methods based on the distribution of metric data and supervised learning methods based on the relationship between the metrics and defect-proneness. The two types of methods have their own advantages and disadvantages: unsupervised methods do not require label information to derive thresholds and are easy to implement, but the resulting thresholds often have a low performance in defect prediction; supervised methods improve the defect prediction performance by machine learning algorithms, but they need label information to derive the thresholds, which is not easy to obtain, and the linking technology between metrics and defect-proneness is complex. In recent years, researchers of the two types of methods have continued to explore and made a great progress. At the same time, it is still challenging to derive the thresholds of object-oriented software metrics.

* 基金项目: 国家自然科学基金 (62172205)
收稿时间: 2021-03-28; 修改时间: 2021-06-06, 2021-09-15; 采用时间: 2021-10-08; jos 在线出版时间: 2021-11-24
CNKI 网络首发时间: 2022-11-15

This paper presents the systematic survey on the recent research achievements in deriving metric thresholds. First, the research problem is introduced in object-oriented software metric threshold derivation. Then, the current main research work is described in detail from two aspects: unsupervised and supervised learning methods. After that, the related techniques are discussed. Finally, the opportunities and challenges are summarized in this field and the research directions in the future are outlined.

Key words: object-oriented software; metrics; threshold; defect prediction

软件度量 (software metric)[1]是理解、评价、预测和管理面向对象软件系统质量的一种重要手段[2]. 它是对软件开发项目、过程和产品进行数据定义、收集以及分析的持续性定量化过程, 主要包括项目度量、过程度量和产品度量 3 方面. 其中, 产品度量侧重于代码和文档等软件产品的质量度量[3]. 通常, 软件产品属性分为外部属性和内部属性[3]. 其中, 外部属性指与软件运行环境相关的质量属性, 如易维护性、易用性和可靠性等, 一般很难直接获得; 内部属性指与软件自身相关的属性, 如规模和结构等通常可以直接度量. 在实践中, 由于软件产品的外部属性不易直接获得, 通常先度量软件产品的内部属性, 然后在此基础上建立外部属性和内部属性之间的联系. 建立这种联系常见做法之一是应用面向对象 (object-oriented) 软件的各种度量 (以下简称 OO 度量) 阈值来对类的缺陷信息进行预测. 软件缺陷 (software defect) 是在软件产品整个生命周期上的产物. 在大型软件中, 由于复杂的结构和众多的开发人员, 缺陷不可避免的在开发过程中被引入. 软件缺陷的存在降低了软件质量且增加了软件维护成本, 严重的缺陷可能会使企业破产甚至对人的生命安全构成威胁. 本研究中软件缺陷范围主要指软件代码中的缺陷, 如类代码上的缺陷. 此外, OO 度量阈值的确定不仅应用于软件代码的缺陷预测场景, 而且在软件设计质量的其他场景中也有重要价值. 这些其他应用场景包括代码异味 (code smell) 的检测、克隆代码的检测和设计质量的判定等. 在软件质量管理过程中, 软件实践者需要确定每一个度量指标的阈值或不同风险水平的门槛值以区分软件质量的好坏[4]. 因此, 在软件质量管理中, 确定 OO 度量的阈值成为理解和应用度量的关键.

在软件开发人员实践活动中, 一些集成开发环境 (如 JBuilder、Visual Studio 等) 除集成代码编写、编译和调试等常见功能外, 还提供了 OO 度量数据收集功能. 为了方便监控代码质量, 开发人员要花费工作量在开发集成环境中对于这部分 OO 度量设定阈值. 这样, 开发人员在编码时能随时接收 IDE 自动收集的度量信息, 并根据这些反馈信息仔细检查代码预防出现缺陷. 在此过程中, 仅有少量常见 OO 度量被设置了缺省阈值, 如微软设定继承深度阈值为 5 和复杂性阈值为 25 等 (https://docs.microsoft.com/zh-cn/previous-versions/visualstudio/visual-studio-2012/dd264982(v%3dvs.110)). 由于大多数 OO 度量没有设置阈值指南, 开发人员需要事先了解所用 IDE 的 OO 度量计算公式, 并根据以往开发实践经验来设定阈值检查代码.

为了帮助开发人员设定一些常见的 OO 度量 (如 CK 度量) 阈值参考值, 研究者们提出了多种 OO 度量阈值确定方法. 现有的 OO 度量阈值确定方法根据训练数据集中是否包含每个类或代码模块的缺陷信息划分有监督学习方法和无监督学习方法. 前者通过对有缺陷标签的度量值采用机器学习方法来设定阈值区分有缺陷和无缺陷类. 后者的训练集中只有 OO 度量值. 由于训练样本数据类别未知, 一般根据训练样本间相似性对数据集进行划分, 如根据不同百分位数区间的样本分布特征, 来划分不同类缺陷发生概率的风险等级. 由于训练集中缺少类的缺陷信息, 也使得应用无监督学习方法确定的 OO 度量阈值预测类是否包含缺陷的性能指标数值相对于有监督学习方法较低. 但是, 具有缺陷信息的样本数据收集工作也增加了研究者的工作量. 除了有监督和无监督学习方法分类, OO 度量阈值确定过程存在一些其他分类. 例如, Lanza 等人[4]提出基于统计的阈值和实际意义的阈值分类; Saraiva 等人[5]提出一种专家驱动的阈值确定和与之对应的度量数据驱动的阈值确定; 以及 Fernández 等人[6]提出敏感类内聚性度量 *SCOM* (sensitive class cohesion metric) 度量的分析性阈值和与之相反的 OO 度量的一般阈值 (非分析性阈值) 确定. 由于分析性阈值、专家驱动的阈值确定等这些阈值确定研究成果较少, 并未形成主流, 但他们的研究成果也丰富了 OO 度量阈值研究内容.

近年来, 关于 OO 度量阈值研究工作主要围绕有监督学习和无监督学习两类方法展开. 该领域目前的研究重点在于 OO 度量阈值标准的选择和阈值效应的检验. 截至 2020 年 12 月, 已有 20 多种 (仍在增长) 不同的阈值确定方法被提出. 这些方法在开发人员的实践活动中, 面临着一些困境.

(1) OO 度量阈值泛化性能差. 阈值确定方法依赖于训练集中代码的缺陷信息来选择有监督学习方法和无监督学习方法, 且生成的 OO 度量阈值在其他项目上的应用受限.

(2) OO 度量阈值应用场景不统一. 有监督学习和无监督学习方法在阈值数量上存在分歧: 前者确定的阈值只有一个, 超过 (或低于) 该阈值即被预测为有缺陷的代码; 后者生成若干分段阈值, OO 度量值超过不同的分段阈值所发生缺陷的风险水平不同. 不同数量的阈值造成应用场景不统一.

(3) OO 度量阈值理论基础薄弱. 不论是无监督学习还是有监督学习, 其面临的共同挑战主要是阈值效应的理论基础. 通过 OO 度量阈值假设检验, 可以判断阈值是否存在. 若 OO 度量阈值存在, 则其定义是什么? 其数量是一个还是多个? 是否存在理论基础?

值得注意的是, 目前学术界仅有对几种 OO 度量阈值确定方法比较得出最优的阈值确定方法[7−10]. 这些工作对多种特定的阈值确定方法进行了归纳和整理, 包括 ROC 曲线方法[7]、Bender 的基于逻辑回归的定量风险评估方法[11]等. 2016 年, Ronchieri 等人[12]使用 SCOPUS Web 搜索工具来检索 1970–2015 年发表的度量阈值论文, 但只是对这些软件度量阈值文献进行初步映射 (mapping) 研究, 并非深入阈值确定方法梳理、归纳和总结等研究工作. 据我们所知, OO 度量阈值确定方法研究领域的文献几乎全部发表于外文国际期刊或会议, 国内学术界对其介绍很少, 仅有 3 篇[2,13,14], 而且国内没有一项对该领域进行总结性的综述研究工作 (截至 2020 年 12 月). 为了完善该领域研究工作, 本文首次主要对近 10 年来 OO 度量阈值确定方法的研究历史进行回顾, 对现有成果进行分类介绍和总结, 并在此基础上指出该领域面临的挑战.

本文第 1 节概述主要研究问题. 第 2 节描述文献检索方式和现有文献概览信息. 第 3 节和第 4 节分别从有监督学习方法和无监督学习方法两个方面介绍近年的研究进展. 第 5 节介绍其他阈值确定方法. 第 6 节介绍 OO 度量阈值领域的相关研究工作. 第 7 节 分析 OO 度量阈值研究领域所面临的挑战及可行的研究方向. 第 8 节总结本文的内容.

1 研究问题

1.1 面向对象软件度量集

在面向对象程序设计出现初期, 软件度量主要集中在面向过程软件复杂性度量上, 包括程序结构复杂性度量、计算复杂性度量、逻辑复杂性度量、概念复杂性度量和文本复杂性度量[15]. 例如, 1976 年, McCabe[16]提出基于图论的复杂性度量, 并说明了如何使用它来管理和控制程序复杂性. Halstead[17]和 Curtis 等人[18]分别在 1977 年和 1979 年提出 Halstead 工作量 (复杂度) 度量. 1988 年, Basili 等人[19]基于目标、问题和度量范式概念, 提出创建新度量的一个基本框架模板. 该模板通过设定目标、生成可量化问题的大纲并最终生成可量化的度量.

随着面向对象程序设计方法发展, OO 度量逐步展示面向对象软件的内聚、耦合、继承、多态、信息隐藏和封装等特性[20,21]. 目前, 大多数 OO 度量是静态的源代码度量, 只能在编码阶段之后才能收集. 自从 Chidamber 和 Kemerer 在 1991 年提出著名 CK 度量集[22], 学术界在 30 年时间里一直持续不断地挖掘新的度量集.

图 1 汇总常见的 100 多个 OO 度量首次提出时间和作者名的简图[6,20,22−24,29−76]. 在这些研究中, 学术界和工业界的学者不断探索 OO 度量在面向对象软件的封装性、多态性、继承性、耦合性和内聚性等各类特征上量化研究. 甚至出现度量名重名现象, 如 Lorenz 等人[23]用 *NOP* 度量继承父类的数量, 而 Bansiya 等人[24]用 *NOP* 度量多态性方法数量. 附录中表 A1 (内聚性)[25]、表 A2 (耦合性)[26]、表 A3 (继承性)[27]和表 A4 (规模和复杂性)[28]列出当前常见的 100 多个 OO 度量, 分别从面向对象软件的内聚性、耦合性、继承性以及类的规模和复杂性 4 个方面来度量. 每张表中列出了各度量名简称、提出时间、定义内容和文献来源.

1.2 OO 度量数据收集工具和 OO 度量阈值模型

1.2.1 OO 度量数据收集工具

现阶段, 有许多 OO 度量数据收集工具供开发者在实践中自动化收集度量数据. 这也促进 OO 度量在实际软件工程活动中发展. 表 1 列出常见 OO 度量数据收集工具的类型、名称、适用的编程语言、度量集和下载网址.

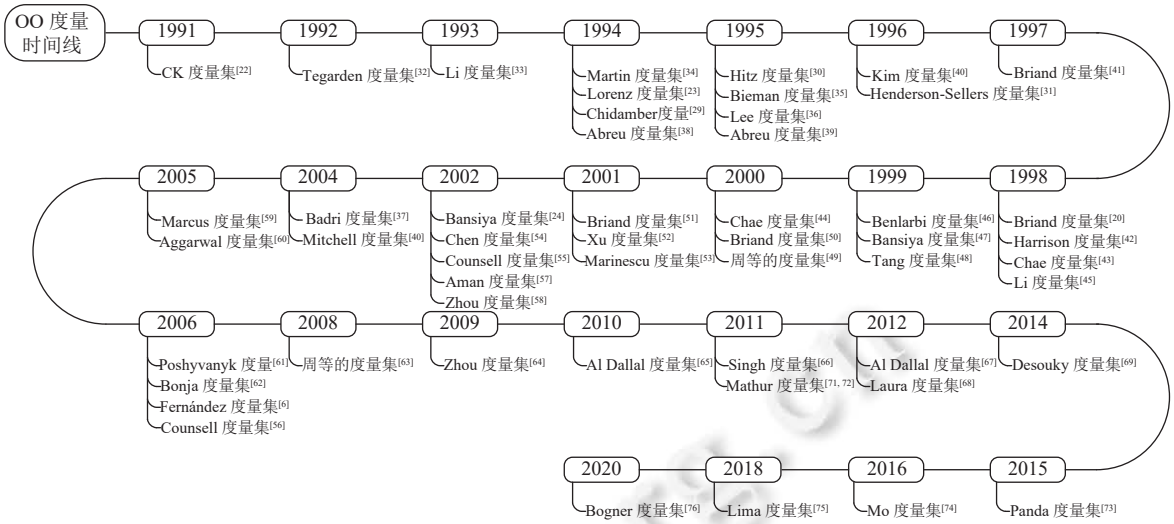


图 1　面向对象度量集提出时间表

表 1　面向对象软件度量收集工具汇总表

工具类型	工具名称	编程语言	度量集	网址
代码分析工具	Understand	C#/C++/C#/Java/Python等14种	包括圈复杂性、CK度量集在内的105种度量	www.scitools.com
	NDepend	C#/C++/VB/.NET平台语言	包括Application、Assemblies、Namespaces、Types、Methods和Fields在内的86个度量	www.ndepend.com
	JArchitect	Java	代码行、圈复杂度、耦合、嵌套深度等选项的度量	www.javadepend.com
	SonarQube	Java/C#/C/C++/Python等27种	复杂度、文件复制、可维护性、安全、规模和测试等选项的度量	www.sonarqube.org
	PMD	Java和Apex及其他6种语言	包括圈复杂度、部分CK度量等在内的10多种度量	pmd.github.io/latest/
	SIG	Java和.NET语言等350种程序语言	基于ISO25010标准质量的各类度量指标	www.softwareimprovementgroup.com
	JHawk	Java	Halstead度量、McCabe度量和可维护性指数(maintainability index)	www.virtualmachinery.com
	ckjm	Java	6个CK度量、Ca (afferent couplings)、NPM (number of public methods)度量	www.spinellis.gr/sw/ckjm/
	sourcemonitor	C++/C/C#/Java/VB/.NET /Delphi	圈复杂度等多种度量	www.campwoodsw.com/sourcemonitor.html
集成度量收集功能	JBuilder	Java/UML模型	包括CK度量集的OO度量	jbuilder.embarcadero.com
	Rational Rose	UML模型	常见CK度量集等	www.ibm.com/support/pages
	Project Analyzer	VB/VB.Net/VBA	包括CK、MOOD度量集的180多种OO度量	www.aivosto.com
	wakatime (IDE插件)	Python、C++、Java等	主要包括对代码工作量, 即开发人员编程时间度量	wakatime.com
专门度量收集工具	SDMetrics	UML模型	可自己定义的OO度量	www.sdmetrics.com
	Krakatau Essential	Java/C++	包括CK、MOOD度量集的80多种OO度量	www.powersoftware.com
	McCabe	Java/C++/Ada	包括圈复杂性、软件科学法、CK度量集在内的40多种OO度量	www.mccabe.com
	SourceMeter	C/C++/Java/C#/Python	60多个组件、文件、包、类和方法层次的代码度量	www.sourcemeter.com
	Designite	C#和Java	包括解决方案层次度量、项目层次度量、类层次度量和方法层次度量	www.designite-tools.com

这些工具可粗略分为如下 3 类.

(1) 代码分析工具. Understand 和 NDepend 是这类工具的代表. Understand 是一个源代码分析工具, 可以提供项目级、类级、程序单元级和文件级度量报告. 除单独用来分析代码外, 还提供了 API 接口供用户使用. 先用 Understand 扫描软件源代码生成扩展名为.udb 的数据库, 再调用 API 接口查询数据库中自定义的度量. 这些度量包括 29 个 OO 度量、27 个复杂性度量和 47 个计数度量. NDepend 是一个适用于微软的.NET 环境代码分析工具, 可与 Visual Studio 和 Azure DevOps/TFS 集成. 其收集的度量可以在 Visual Studio 扩展中可视化. 开发人员可以自己设定阈值, 在可视化界面中显示实际值与阈值的差异.

(2) 集成开发环境 (IDE) 中度量收集工具. JBuilder 和 Rational Rose 是这类的度量收集工具的代表. 由于集成了 Together 软件, JBuilder 能够从 UML 模型和 Java 源代码中收集百余种度量, 其中包括 CK 度量. Rational Rose 提供了与 JBuilder 相似的度量收集功能. Visual Studio 中也提供了圈复杂度、继承深度、类耦合度、代码行数和一个总的可维护指数等常见的度量. 此外, 一些 IDE 插件中也提供对源代码的度量工具. 如 WakaTime 可以统计常见的 IDE 中项目的编程活动来自动生成度量指标、统计及时间追踪.

(3) 专门度量收集工具. SDMetrics 和 Krakatau Essential 是这类工具的代表. 用户可以从 UML 建模工具或者逆向工程工具中导出 UML 模型到 XMI 文件中, SDMetrics 读取 XMI 文件来计算各种度量, 并检查与设计规则的符合性. Krakatau Essential 能从 Java/C++源代码中收集项目级、文件级、类级和方法级上的多种度量. McCabe IQ 是一款软件质量管理套件, 主要对软件进行质量分析和覆盖率测试, 它提供的软件度量主要包括代码行和注释行统计、嵌套深度和数据变量统计, 以及 Halstead 度量集和 McCabe 度量集上的 OO 度量.

1.2.2 面向对象软件质量预测模型

OO 度量可以作为解释变量用来预测软件代码发生缺陷倾向 (defect-proneness)、变化倾向 (change-proneness) 和软件测试难易 (testability) 等外部质量属性. 选择这些外部质量属性的代理变量作为被解释变量便可建立质量预测模型. 例如, 缺陷倾向的代理变量可以是被度量代码块中的缺陷数量; 变化倾向的代理变量可以是不同版本间每个模块修改次数或变更代码量; 软件测试难易的代理变量可以是测试该模块所需要的工作量. 在确定这些代理变量后, 可以在 OO 度量解释变量基础上增加代码规模度量[77]等作为控制变量建立软件质量预测模型. 在软件工程领域, OO 度量通过质量预测模型对类的缺陷信息预测通常有 3 类应用场景: 第 1 类应用场景是对每个类中的缺陷数量或代码修改行数的预测 (测“量”); 第 2 类应用场景为类是否包含缺陷或是否发生代码修改的预测 (测“类别”); 第 3 类应用场景是类发生缺陷概率从高到低的顺序预测 (测“序”). 其中, 应用较多的还是通过 OO 度量量化面向对象软件的各类特征信息来预测哪些类可能包含缺陷[3,78].

1.2.3 基于 OO 度量阈值的缺陷预测模型

在 OO 度量作为解释变量与“类别”作为被解释变量之间建立的质量预测模型中, 常见的有 logistic 模型. 当只选择一个 OO 度量为解释变量时, 通过对该度量代入模型后输出的预测值设定阈值来预测模块是否有问题. 若 OO 度量与类别呈正相关, 则将预测值大于或等于其设定阈值的模块预测为有问题模块, 低于阈值的模块预测为没有问题模块. 若 OO 度量与类别呈负相关, 则将预测值低于或等于其设定阈值的模块预测为有问题模块, 高于阈值的模块预测为没有问题模块. 此处比较的是模型中“类别”被解释变量的预测值与其设定的阈值. 由于这类模型的解释变量与被解释变量具有一一映射特征, 可将模型中“类别”预测值的阈值转换为 OO 度量的阈值. 这样, 在新系统中预测缺陷时, 通过 OO 度量与其阈值直接比较来预测模块是否有问题.

图 2 列示了基于 OO 度量阈值的缺陷预测模型, 包括模型建立、阈值确定与阈值应用. 其基本步骤如下.

(1) 提取 OO 度量数据. 在软件版本控制库 (如 Git) 中下载 OO 软件项目的某一版本. 利用 OO 度量数据收集工具 (如 Understand) 提取该版本源代码各种粒度下的实体及其引用信息, 并据此计算各种 OO 度量.

(2) 收集缺陷度量数据. 在同一软件项目的缺陷追踪系统上, 收集该版本上与度量相同粒度的各个代码块的缺陷数据, 如基于 Java 架构的缺陷追踪系统 Jira 中, 提供代码中 Bug 数量和各自优先级等信息. 这一步实现的关键是要把同一项目版本软件在版本控制系统和缺陷追踪系统中建立链接. 例如, 通过 Jira 中提供的关键字可以在 GitHub GraphQL API 中查询到该版本修改的文件等信息.

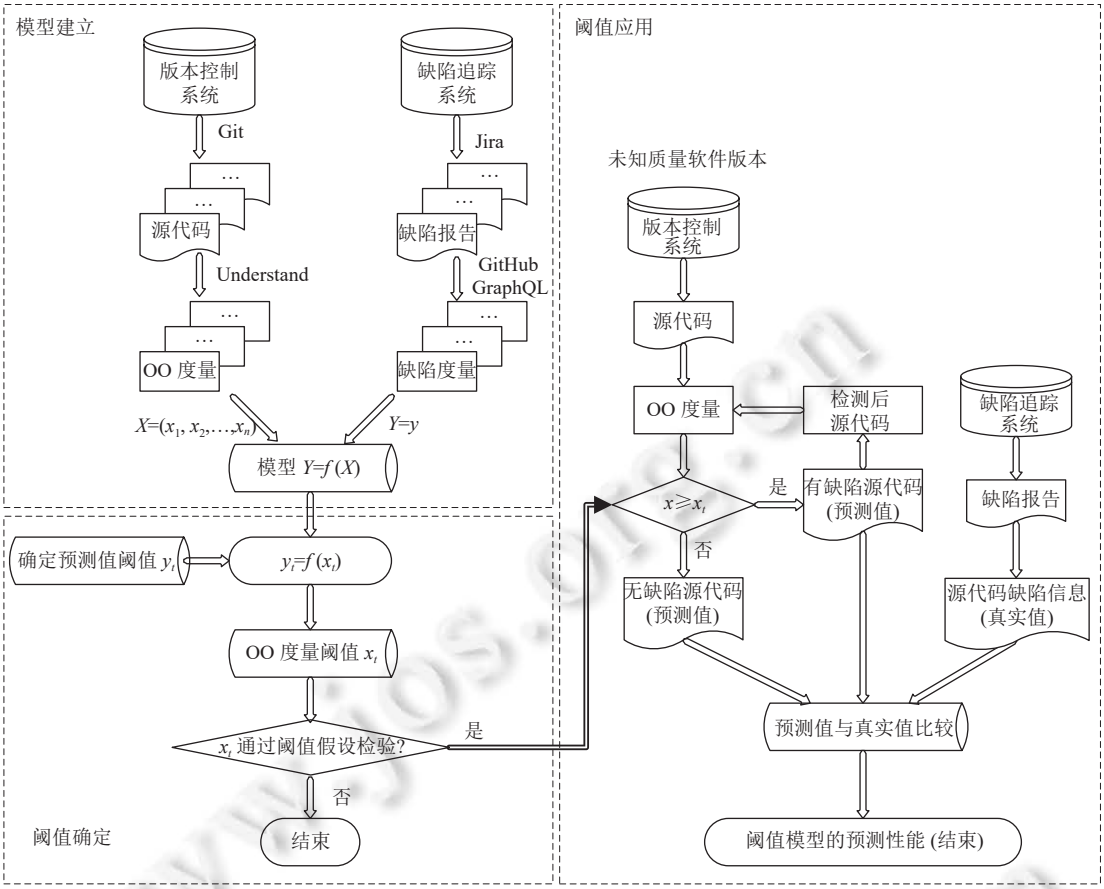


图 2　基于 OO 度量阈值的缺陷预测模型建立与应用过程

(3) 建立 OO 度量阈值模型. 重复前两个步骤以收集足够多的训练数据. 在训练集上, 以 OO 度量为解释变量与以缺陷度量为被解释变量 (如二分类变量) 建立预测模型. 在解决数据类别不平衡和异质性等会影响模型参数估计的问题后, 经参数检验与参数估计后得到模型各参数估计值, 从而完成模型建立.

(4) OO 度量阈值确定. 将收集的 OO 度量数据依次代入已建立的模型并输出模型的预测值. 在有监督学习方法中, 结合模型和阈值判断标准确定模型预测值的阈值 y_t. 通常会选择模型输出的某个预测值作为 y_t. 然后, 通过模型转换, 将预测值的阈值 y_t 转换成 OO 度量的阈值 x_t. 若转换后的阈值 x_t 通过阈值假设检验则可进入应用阶段; 若阈值检验不通过, 则结束当前度量阈值 x_t 的判断.

(5) OO 度量阈值应用. 对未知质量的软件项目源代码采用相同步骤收集 OO 度量和缺陷度量数据. 当某 OO 度量与缺陷度量正相关时, 未知质量代码的 OO 度量值大于等于阈值 x_t, 则预测为有缺陷的代码. 若经检测后真实存在缺陷, 则修复后重新度量再进行阈值判断. 若 OO 度量低于阈值 x_t, 则该未知质量的源代码则被预测为无缺陷的源代码. 当某 OO 度量与缺陷度量负相关时, 阈值判断与正相关情况相反.

(6) OO 度量阈值预测缺陷的性能评估. 在得到源代码缺陷信息的预测值后, 对预测结果与实际质量数据 (被预测代码的真实缺陷信息) 比较后得出阈值模型性能评估. 阈值应用的性能评估可为后续度量阈值研究提供参考资料.

基于 OO 度量阈值的缺陷预测模型是在有源代码缺陷信息的有监督学习方法上, 应用 OO 度量阈值进行缺陷预测. 当缺少源代码缺陷信息时, 应用无监督学习方法来生成 OO 度量阈值. 图 3 展示了 OO 度量阈值确定的无监督学习和有监督学习方法流程简图. 如图 3(a) 所示, 在无监督学习方法中, 仅根据 OO 度量数据自身分布特征, 如利用代码块行数[79]和实体数量[80]等作为权重等信息, 依据不同权重来设置不同风险水平 (等级) 上的分段阈值. $(n-1)$ 个分段阈值 T_i $(i=1, 2, \ldots, n-1)$ 将 OO 度量值域空间分为 n 个风险等级区间. 当 OO 度量值在某一风险等级

(水平) 空间上, 则将被度量的代码预测在该风险水平上存在缺陷. 各分段阈值还需要通过检验阈值左右两侧的度量数据是否存在显著差异. 放弃不存在显著差异的分段阈值. 最终, 左右两侧度量值显著不同的分段阈值被验证. 在图 3(b) 中, 有监督学习方法通过对缺陷预测模型中预测值选择符合某种阈值判断标准作为预测值阈值, 再通过模型转换为 OO 度量阈值. 然后进行阈值 (效应) 假设检验. 若通过检验, 则被验证为 OO 度量的阈值; 反之, 则放弃该 OO 度量阈值.

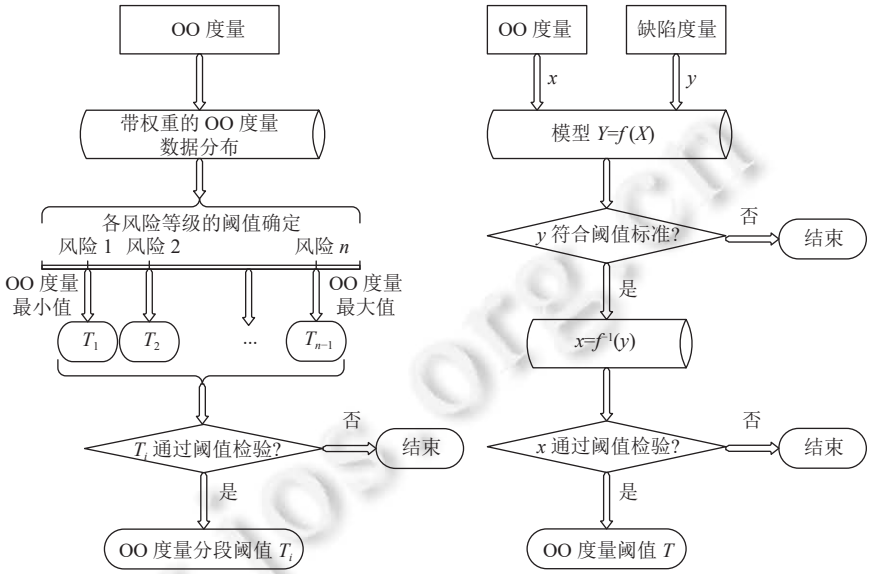


图 3 OO 度量阈值确定方法流程简图

由于无监督学习方法中没有缺陷标签信息, 使得该方法确定的阈值往往多于 1 个, 一般有 3 个分段阈值 (用于度量低风险、中风险和高风险) 或 5 个分段阈值 (用于度量极低风险、低风险、中风险、高风险和极高风险). 而有监督学习方法确定阈值只有一个. 无监督学习方法中 OO 度量权重数据分布和风险等级标准与有监督学习方法中模型预测值的阈值标准, 以及两类方法中阈值检验是确定 OO 度量阈值的关键步骤, 也是 OO 度量阈值确定方法的主要研究内容.

1.3 OO 度量阈值确定方法的研究内容

目前, OO 度量阈值确定方法的研究重点是阈值标准和阈值检验. 以下分别列出 OO 度量阈值研究面临的主要挑战和对应的研究内容. 具体内容将分有监督学习方法和无监督学习方法分别在第 3 节和第 4 节详细介绍.

1.3.1 OO 度量阈值标准

OO 度量阈值确定的有监督学习方法中最为关键的一步是如何选择模型的某一预测值作为阈值标准, 包括有监督学习方法中模型预测值阈值标准和无监督学习方法中 OO 度量分段阈值标准. 这一过程受主观因素影响较大. 在 OO 度量阈值模型的训练集中, 有缺陷代码的度量值域与没有缺陷代码的度量值域并没有清晰的界限. 这使得在模型预测值上选择某一预测值作为阈值存在挑战.

在无监督学习方法中, 阈值确定的关键在度量值域上设置分段阈值标准. 根据 OO 度量所在的值域, 可以对被度量的对象做出软件质量 (如是否为高风险代码) 的可靠评估[4]. 对无监督学习方法来说, 挑战主要在于 OO 度量阈值对应的被度量对象 (即模块) 的风险等级没有客观标准. 两类方法的主要挑战如下.

(1) OO 度量数据特征影响其阈值 (效应) 存在

在 100 多种 OO 度量中, 有些 OO 度量数据与其被度量的代码模块发生缺陷之间联系较弱. 这使得应用 OO 度量很难预测这些被度量的代码是否存在缺陷. 在有缺陷和没有缺陷的代码上, 存在某些 OO 度量数据特征无明显区分. 这些 OO 度量可以被认为不存在阈值 (效应). 这一挑战面临的直接问题是阈值 (效应) 定义.

现有文献中对阈值定义有如下 3 个阐述. Lorenz 等人[23]认为 OO 度量阈值是设置软件度量在可取区间和不可取区间的启发性数值. 这些阈值可以识别导致出现或不出现实际问题的异常值. Morasca[81]将度量阈值定义为一个函数 t: $TR\times TS\rightarrow[0, 1]$, 其中, TR 和 TS 分别是训练集和测试集. 阈值 $t(tr, ts)$ 不单是一个数字, 其值很可能会根据训练和测试集的使用而改变. Foucault 等人[82]认为: 某个给定度量 (μ_k) 的阈值 ($t_{\mu_k}\in\mathbb{R}$) 指将软件实体集合 U_k 分为 Low 和 $High$ 两组的值 ($Low\cup High=U_k$ 和 $Low\cap High=\varnothing$). Low 组中所有实体的度量值都小于或等于阈值; $High$ 组中所有实体的度量值都大于阈值 ($\forall e\in Low, \mu_k(e)\leqslant t_{\mu_k}$ 和 $\forall e\in High, \mu_k(e)>t_{\mu_k}$).

(2) 逆阈值效应的存在

对某些继承类度量可能存在逆阈值效应. 例如, Cartwright 等人[83]发现: 继承的类更容易出现错误倾向. 即在其他条件相同情况下, 没有继承的类可理解性稳定, 而继承的类在可理解性上降低了. 然而, Unger 等人[84]通过记录 3 个对等 Java 子程序发生修改的维护时间度量后发现: 继承深度为 3 的 Java 子程序的维护时间比继承深度为 0 的 Java 子程序长; 而继承深度为 5 的 Java 子程序与继承深度为 0 的 Java 子程序的维护时间几乎一样多. Unger 等人[84]把这一现象归因于这样一个事实: 完成深度继承子程序的维护任务所需要的更改量更小. Benlarbi 等人[78]把这一现象称之为继承深度 OO 度量的逆阈值效应.

(3) 对 OO 度量的阈值标准选择主观性较大

在有监督学习方法中, Shatnawi[85,86]在应用逻辑模型 (logistic) 确定 OO 度量阈值时, 对模型的输出值分别选择 0.06、0.065、0.075 和 0.1 概率作为阈值标准, 然后按性能评估指标进行筛选; Mauša 等人[87]用训练集中有缺陷的模块数与所有模块数之比作为阈值标准, 然后将确定的 OO 度量阈值在测试集上验证其 G-mean 值是否大于 0.6; 在无监督学习方法中, Alves 等人[79]选择 OO 度量各数值在基准数据中代码量权重上取 70%、80% 和 90% 作为分段阈值标准. 这些阈值标准选择都存在很大的主观因素.

为了解决上述挑战, OO 度量阈值研究的一个重点在于阈值标准选择上. 充分利用 OO 度量数据特征, 尤其是在有缺陷和无缺陷代码特征. 在挖掘出两类代码数据特征基础上, 还需要降低 OO 度量阈值标准中主观因素 (理想状态为不存在主观因素), 为后续的 OO 度量阈值假设检验提供可靠依据.

1.3.2 OO 度量阈值检验

由于无监督学习方法的多个分段阈值与不同风险等级相关, 所以需要详细阐述每个风险等级与对应阈值关系, 而不同风险等级的定义往往过于主观, 这增加了阈值假设检验难度. 因为有监督学习方法确定的阈值用于区分两种类型标签数据 (有缺陷和无缺陷模块), 所以需要检验两类模块的 OO 度量值是否有显著差异. 在 OO 度量阈值检验阶段, 主要有以下 3 方面挑战.

(1) 阈值检验不充分. 由于两类方法上的阈值标准判断过于主观, 导致阈值假设检验变得越发困难, 以至于大多数阈值研究中对 OO 度量阈值检验不充分, 甚至缺少阈值检验.

(2) 阈值形式不统一. 有监督学习方法只有一个分段阈值, 而无监督学习方法有若干分段阈值. 这给两类方法的比较带来困难. 在无监督学习方法的若干分段阈值中, 选择哪一个分段阈值与有监督学习方法所确定的阈值进行比较. 即当 OO 度量超过该分段阈值时, 被度量的类可以被预测为有缺陷的类.

(3) 泛化性能不确定. 两类方法确定阈值后, 需要在新项目上应用 OO 度量阈值来检测其预测性能. 在现实开发工作中, 仅在有限的项目上确定 OO 度量阈值, 需要在新项目上取得可靠的评估结果.

上述挑战使得 OO 度量阈值良好的缺陷预测表现大多停留在实验阶段或是在企业的项目内部应用而无法在现实项目中被广泛应用. 为了解决上述挑战, 该领域的研究重点还包括规范 OO 度量阈值确定过程, 尤其是阈值检验过程、收集更加广泛的数据集和拓展丰富的阈值应用 (包括阈值形式统一和实用工具推广阈值应用等).

2 文献检索

2.1 文献检索与甄别

OO 度量阈值确定方法一直是软件度量领域的热点研究问题. 近年来, 出现大量的有监督学习方法和无监督

学习方法为主的阈值确定方法. 为了更好地深入分析、比较和总结该问题, 本文的参考文献力图覆盖近 10 年来与 OO 度量阈值确定方法相关的主要研究工作. 具体来说, 本文使用以下的步骤来进行相关文献检索和甄别.

(1) 检索文献来源. 谷歌学术搜索、IEEE Xplore Digital Library、Digital Library ACM、DBLP Computer Science Bibliography、Springer Link Online Library、Sicence Direct、ISI Web of Knowledge 平台和中国知网等计算机领域电子数据库. 检索内容主要包括但不限于《CCF 推荐国际学术会议和期刊目录》(2019 版) 中各类期刊和会议的电子格式文献.

(2) 检索关键词. “metric thresholds”“software metric thresholds”“object-oriented software metric thresholds” “software metric threshold effect”“thresholds of software metrics”和“thresholds of object-oriented software metrics”等 OO 度量阈值相关主题关键词.

(3) 检索起止时间. 2010–2019 年之间发表的文献. 由于 2010 年前关于 OO 度量阈值确定方法的相关文献并不多 (10 篇), 因此, 重点关注 2010 年及以后的文献.

(4) 检索方法. 除按上述关键词检索外, 还辅以滚雪球 (snowballing) 方法[88]. 首先, 通过检索选择时间最近的论文作为初步选定的研究文献; 其次, 检查选定研究文献的参考文献, 这些文献大多数与被选定的文献研究相关; 最后, 根据上一步中的参考文献, 继续检查其参考文献 (即滚雪球式的前进).

(5) 文献甄别. 文献需要围绕一个或多个 OO 度量阈值确定方法研究; 写作语言是英语和中文; 文献通过公开的电子数据库可以下载获得. 本文研究重点为 OO 度量阈值确定方法, 而部分文献采用已有 OO 度量阈值进行缺陷预测研究. 如 Alan 等人[89]使用 McCabe IQ 工具提供的度量阈值; Catal 等人[90]使用 ISM 公司提供的度量阈值等. 这类文献题目中也包括度量阈值检索关键词, 但原文中未阐明确定阈值方法, 而是引用现有度量阈值. 因此, 这类文献将被人工淘汰.

2.2 历史文献的汇总

经过文献的检索与甄别, 本文收集了 OO 度量阈值确定方法领域中近年来的学术论文共 80 篇. 这些研究论文将会按照有监督和无监督学习方法分别在第 3 节和第 4 节进行详细介绍. 对于 OO 度量阈值确定方法相关研究工作将在第 5 节和第 6 节中对其进行分类和介绍.

表 2 列出了近年来 OO 度量阈值确定方法相关文献列表[2,4–10,13,14,68,75–82,85,86,91–148]. 在这些文献中, 有监督学习方法与无监督学习方法文献数量相当: 前者有 37 篇, 后者有 39 篇, 其他方法的文献有 3 篇. 其中, 有监督学习方法中, 发表在会议上文献有 15 篇, 期刊有 23 篇; 无监督学习方法中, 发表在会议上的文献有 17 篇, 期刊有 15 篇, 硕博学位论文有 5 篇及书籍有 1 篇. 79 篇文献中, 中文文献有 3 篇[2,13,14]. 图 4 按年份展示 OO 度量阈值领域的研究论文发表的数量分布情况. 如图 4 所示, 该领域的逐年论文发表数量呈现一个波动上升趋势. 2009 年以前论文数量不多, 但有些文献[11,78]为后续研究提供探索方向. 表 2 和图 4 中展示内容说明 OO 度量阈值这个领域正越来越受研究者的重视.

3 有监督学习方法

表 3 列出确定 OO 度量阈值的各种有监督学习方法或模型的汇总信息. 这些汇总信息包括方法名称、年份、阈值确定标准、实验数据集、度量集、阈值检验和研究作者. 根据汇总信息, 发现存在如下 3 个典型的 OO 度量阈值确定方法, 每种典型方法会有若干学者进一步拓展研究并产生 OO 度量阈值的新方法.

(1) Benlarbi 等人[78]提出的单变量逻辑回归阈值模型 (BLRT).

(2) Bender[11]提出的基于逻辑回归的定量风险评估方法 (QRA), 也称 Bender 方法.

(3) Shatnawi 等人[118]提出的 ROC 方法.

除这 3 种典型 OO 度量阈值确定的有监督学习方法外, 还有一些其他的有监督学习方法. 例如, BPP、BCE、MFM、FPH-index、Lavazza 方法、矩形学习方法、基于遗传算法的调整方法和 C5.0 方法等. 在这些方法中, 研究者应用其他学科 (如文献计量学) 或机器学习中的方法来确定 OO 度量阈值. 这些方法都拓展了 OO 度量阈值研究工作.

表 2 面向对象软件度量阈值确定方法相关文献列表

类型	出版物缩写	文章数	有监督学习方法文献引用列表	无监督学习方法文献引用列表	其他方法引用列表
国际会议	ASE	1	—	[91]	—
	ISSRE	3	[78,81,85]	—	—
	ESEM	2	[92,93]	—	—
	ICSME	3	—	[79,94,95]	—
	EASE	4	[68,96,97]	[98]	—
	QRS	3	[9,99]	[100]	—
	SCAM	1	—	[101]	—
	SEKE	1	—	—	[5](专家驱动)
	其他	15	[121,123,124,127,130]	[80,82,132−135,137,138,142,144]	—
国际期刊	TOSEM	1	[77]	—	—
	TSE	4	[86,102,103]	[104]	—
	IS	1	[105]	—	—
	ESE	2	[106,107]	—	—
	JSS	3	[108]	[75,109]	—
	JSEP	1	—	[110]	—
	IST	2	[111]	[7]	—
	JCST	1	[112]	—	—
	IET	1	—	—	[113](综合方法)
	SQJ	1	—	[114]	—
	ES	1	[115]	—	—
	ESWA	1	[10]	—	—
	MSR	2	—	[116,117]	—
	其他	18	[8,118−120,122,125,126,128,129,131]	[76,136,139−141,143,145]	[6](分析阈值)
国内会议期刊	—	1	—	[2]	—
硕博学位论文	—	5	—	[13,14,146−148]	—
书籍	—	1	—	[4]	—
合计	—	79	37	39	3

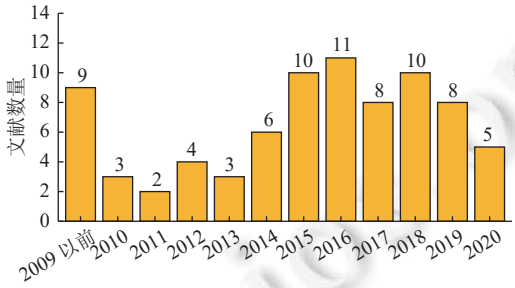


图 4 OO 度量阈值研究文献的年代分布图

3.1 单变量逻辑回归阈值模型 (BLRT)

3.1.1 BLRT 的基本原理

1991 年, Ulm[149]在流行病学研究领域提出基于单变量逻辑回归的阈值模型 (binary logistic regression model with a threshold, BLRT), 用于评估煤矿粉尘浓度的阈值. 若超过这个阈值, 矿工就会发生慢性支气管炎反应. Ulm 最初的模型如公式 (1) 所示:

$$\text{logit}P(x)=\beta_0+\beta_1(x-\tau)I_+(x-\tau) \tag{1}$$

表 3 OO 度量阈值有监督学习方法 (模型) 汇总

方法名称	年份	阈值确定标准	实验数据集	度量集	阈值检验	研究作者
BLRT*	2000	最大化似然函数法估计模型中阈值τ: $\pi=\frac{1}{1+e^{-(\beta_0+\beta_1 size+\beta_2(M-\tau)I_+(M-\tau))}}$	两个C++系统	WMC、DIT、CBO、RFC和NOC	有	Benlarbi等人[78]
	2002	最大化对数似然函数法估计模型中阈值τ: $\pi=\frac{1}{1+e^{-(\beta_0+\beta_1(size-\tau)I_+(size-\tau))}}$	两个C++系统和一个Java系统	Stmts、LOC、NM和NAI度量	有	Emam等人[102]
QRA* (Bender)	2006	$VARL=p^{-1}(p_0)=\frac{1}{\beta_1}\left(\log\left(\frac{p_0}{1-p_0}\right)-\beta_0\right)$ p_0取值为1%和5%	Eclipse 3.0	CBO、RFC和WMC	无	Shatnawi等人[85]
	2010	p_0取值为0.06、0.065、0.075和0.1. 在测试集上G-mean值最大的*VARL*值	Eclipse 2.0, 2.1和Rhino软件系统	CBO、RFC和WMC	无	Shatnawi等人[86]
	2013	p_0值为0.5、0.55、0.6、0.65和0.7时, *VARL*为阈值	jfreechart 1.0.0pre1和1.0.1	WMC、RFC、LCOM、CBO、DIT和LOC	无	Kaur等人[119]
	2014	p_0值分别取0.05、0.065、0.075、0.1和0.125. *AUC*(>0.7)最大时, *VARL*为阈值	Mozilla Firefox 1.5、2.0和3.0	PuF、EncF、WMC、DIT和NOC等12个	无	Singh等人[120]
	2015	p_0值分别为0.15、0.07和0.02时, *VARL*为阈值	Ivy 2.0、Ant 1.7、Tomcat 6.0等	6个CK度量和LOC度量	无	Malhotra等人[115]
	2015	选择0.05的p_0值确定度量阈值	Freemind和Frinika两个项目	CBO、NLM、NLDM和LCOM等8个	无	Malhotra等人[126]
	2016	BLR模型中最大凸性(convexity)处和最大斜率的一半处	PROMISE数据集	WMC、CBO、RFC等	有	Morasca等人[96]
	2016	*VARL*公式中p_0值的取值为0.1	jEdit 4.0和Ant 1.3	13个OO度量	无	Hussain等人[9]
	2016	p_0为训练集中有缺陷模块数与所有模块数之比, 且测试集上G-mean≥0.6验证	PROMISE数据集[150]中10个项目	WMC、DIT、NOC、CBO和RFC等20个	无	Arar等人[10]
	2016	选择0.05、0.065、0.08和0.095的p_0值确定度量阈值	NASA和PROMISE库	WMC、CBO、RFC、Ca和Ce等15个	无	Hussain等人[127]
	2017	测试集上验证其G-mean值是否大于0.6	Eclipse JDT和PDE	LOC、DIT、LCOM、CBO和RFC等16个	无	Mauša等人[87]
	2019	基于对错误倾向函数标准偏差来识别BLR模型中不确定区域范围	PROMISE数据集[150]中54个项目	WMC、CBO、RFC、CA和CE等9个	无	Lavazza等人[97]
	2019	测试集上*AUC*最大	Apache Click 2.0–2.1	LOC、DIT、CBO和RFC等14个	无	Malhotra等人[129]
	2019	测试集上G-mean≥6	Eclipse JDT	LOC、DIT、CBO和RFC等48个	无	Mauša等人[130]
	2019	测试集上G-mean≥6	SofaAudit工具收集的WDL数据	6个CK度量	无	Padhy等人[131]
ROC-Euclidean distance*	2009	$t_{opt}=\arg\min_t distance(pf,pd)$ ROC曲线上任意一点到(0, 1)欧氏距离最小的点(*pf*, *pd*)作为最佳决策阈值	PROMISE数据集中13个项目	未列出软件度量, 只列出单个项目上的阈值性能	无	Tosun等人[92]
ROC*	2010	遍历OO度量值, 当敏感性值和特异性值之和最大时为阈值, 通过*AUC*≥0.7验证	Eclipse 2.0、2.1和3.0	CBO、RFC、WMC、LCOM、DIT等12个	无	Shatnawi等人[118]
	2011	经过(0, 0), (1, 1)和(*pd*, *pf*)这3点的ROC曲线下面积*AUC*最大时, 其值为最佳阈值	NASA的5个公共数据集	LOC、v(g)和uniq_Op等13个	无	Catal等人[105]

表 3 OO 度量阈值有监督学习方法 (模型) 汇总 (续)

方法名称	年份	阈值确定标准	实验数据集	度量集	阈值检验	研究作者
ROC*	2012	选择ROC曲线上到对角线最大距离的点作为最佳阈值	两个家庭实验数据	NID、NCD、NPF等	无	Laura等人[68]
	2012	选择ROC曲线上到对角线最大距离的点作为最佳阈值	429个EPC流程模型的数据集	规模、连通性、模块性和复杂度等度量	有	Mendling等人[108]
	2012	ROC中*sensitivity*与(1−*specificity*)相等时的缺陷发生概率 p_0, 其*VARL*为阈值	PROMISE[150]中Apache Poi项目	WMC、DIT、NOC、CBO和RFC等20个	无	Malhotra等人[8]
	2013	$AUC \geqslant 0.7$	Eclipse (version 2.0, 2.1和3.0)	NOM、LCOM、NOS、AVCC和RFC等24个	无	Kumari等人[125]
	2016	假正例率、假反例率和错误率3个预测性能指标至少都在可接受水平(≤0.3)	Apache ANT和KC1等5个项目	SLOC、CBO、RFC和WMC这4个度量	无	Boucher等人[121]
	2017	遍历OO度量值, 当敏感性值等于特异性值时为最佳阈值. 通过$AUC \geqslant 0.55$验证	Android 2.3、4.0、4.1、4.2和4.3	LCOM、CBO、RFC、DIT和NOC等15个	无	Malhotra等人[122]
	2017	与ROC中(0, 1)点之间欧式距离最小; 似然比*LR*(*sensitivity*/(1−*specificity*))大于1	Ant、Camel和jEdit等5个系统	WMC、CBO、RFC和LCOM	有	Shatnawi[128]
	2018	增加检验*AUC*是否显著不等于采用随机方法预测变化倾向的数值(*AUC*=0.5)	D'Ambros等人的5个项目数据[151]	CBO、RFC、WMC、DIT、NOC和LCOM	有	Shatnawi等人[123]
ROC-ROI*	2020	RRA指标参考值(阈值)由实践者和研究者通过利益区域RoI的边界来确定	SEACRAFT[152]中67个数据集	WMC、DIT、NOC、CBO和RFC等19个	无	Morasca等人[106]
基于遗传算法的调整方法	2005	不断搜索每个潜在的解决方法(候选阈值)使得适应度函数(fitness function)最小化	7个Java和C++面向对象软件系统	ATFD、WMC、TCC、WOC和NOPA等6个	无	Mihancea等人[124]
BCE	2006	平衡误分类为有缺陷倾向和无缺陷倾向类别的数量来确定阈值	NASA度量数据集KC1	CK度量集和SLOC度量	有	Zhou等人[103]
矩形学习方法	2011	利用轴对齐的*d*维矩形的机器学习方法确定阈值	Eclipse platform等8个项目	WMC、CBO、RFC、NOM和VG等11个	无	Herbold等人[107]
BPP、BCE、MFM	2014	*balance*指标最大; 假正例率(FPR)和假反例率(FNR)两个分类错误比率大致相等; *F*-measure指标最大	NASA度量数据集KC1	CK度量集和SLOC度量	无	Zhou 等人[77]
FPH-index	2015	在*FP*的非单调递减顺序中, *fph*阈值是满足$FP(x_m) \geqslant z/n$的$FP(x_m)$最后一个值.	PROMISE中MC2和Tomcat数据集	BRANCH_COUNT度量和WMC度量	无	Morasca等人[81]
悲观和乐观阈值	2016	通过悲观和乐观模型(BLR_{pess}和BLR_{opt})分别设置悲观和乐观阈值t_{pess}和t_{opt}	PROMISE数据集中9个项目	WMC、CBO、RFC等	无	Lavazza等人[93]
基于风险规避的斜率阈值	2017	BLR和PBR模型中最大斜率值一半阈值、最大凸性阈值、斜率均值阈值和斜率中位数阈值	PROMISE数据集[150]中52个项目	WMC、CBO、RFC、CA和CE等9个	无	Morasca等人[111]
C5.0方法	2018	独立处理*N*个问题以构建*N*个分类树, 通过C5.0分类树学习器构建评估标准.	Komatsu公司C++嵌入式系统	FUN、ELOC、DN、CC和AveDN等	无	Tsuda等人[99]
聚类方法	2019	期望最大化聚类方法	16个开源的Java项目	LOC、LCOM和CBO度量等21个	有	Alqmase等人[112]

注: (1) *为3种典型的有监督学习方法及其发展而来的方法. (2) Alqmase等人[112]的期望最大化聚类方法虽然是无监督方法, 但是作者应用带标签的数据作为训练数据来得出度量阈值, 故归纳入有监督学习方法

公式 (1) 中, x 为解释变量, τ 为阈值, 且 β_0、β_1 和 τ ($\min x \leqslant \tau \leqslant \max x$) 是待估参数. 当 $x \leqslant \tau$ 时, $I_+(x-\tau)$ 等于 0; 当 $x>\tau$ 时, $I_+(x-\tau)$ 等于 1. $P(x)=P(Y=1|X=x)$ 为个体发生疾病的概率. 该模型的另一种表达形式如公式 (2) 所示:

$$\text{logit}P(x)=\begin{cases}\beta_0, & \text{当}x\leqslant\tau\text{时}\\ \beta_0+\beta_1(x-\tau), & \text{当}x>\tau\text{时}\end{cases} \tag{2}$$

Ulm[149]通过执行蒙特卡罗模拟方法来评估阈值 τ 的假设检验. 若 P 值小于给定的显著性水平 α, 则拒绝零假设; 反之, 则接受零假设.

3.1.2 面向对象度量阈值的确定

2000 年, Benlarbi 等人[78]和 El Emam 等人[102]认为 BLRT 模型可以应用于任何假定存在阈值的剂量-反应关系中来确定阈值, 并将该模型引入软件工程领域. 为了检测 OO 度量的阈值效应, Benlarbi 等人[78]在逻辑回归阈值模型中增加类规模 (SLOC) 潜在混合效应控制变量[153]. 在此基础上, Benlarbi 等人[78]提出软件工程领域中 OO 度量逻辑回归阈值模型, 如公式 (3) 所示:

$$\pi=\frac{1}{1+\mathrm{e}^{-(\beta_0+\beta_1 size+\beta_2(M-\tau)I_+(M-\tau))}} \tag{3}$$

其中, π 为面向对象程序中某个类有一个错误 (fault) 的概率, *size* 变量是该类的 SLOC 度量值, M 是一个具体的 OO 度量, τ 为 M 的阈值. 当 $M\leqslant\tau$ 时, $I_+(M-\tau)$ 等于 0; 当 $M>\tau$ 时, $I_+(M-\tau)$ 等于 1.

在公式 (3) 中, Benlarbi 等人[78]根据 El Emam 等人[154]关于类规模 (*size*) 没有阈值效应的结论, 把类规模设置为连续变量. 即 *size* 变量 SLOC 度量不存在阈值效应. 根据 Ulm[149]对逻辑回归阈值模型的二变量关系的要求, 在每个具体的 OO 度量阈值或非阈值模型中, *size* 变量被假定为常量. 这样, 可以使得逻辑回归模型为两变量关系模型. 如图 5 所示, 在 BLRT 模型简图中实线表示二元逻辑回归阈值 (BLRT) 模型, 而虚线表示逻辑回归非阈值模型. 当 OO 度量值超过阈值 时, 在阈值模型中的被度量类或模块发生缺陷的概率开始上升; 而在非阈值模型中的 OO 度量发生缺陷概率是从其最小值就开始逐步上升.

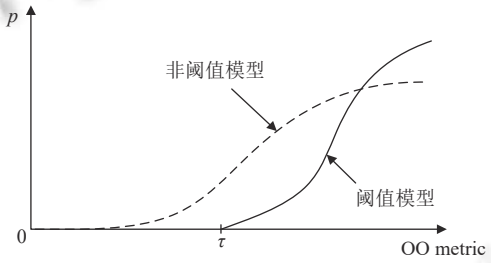


图 5 二元逻辑回归阈值模型和非阈值模型

对模型参数 β_0、β_1 和 τ 估计的常用方法是最大化似然函数法 $L(\beta_0, \beta_1, \tau)$ 或最大化对数似然函数法 $LL(\beta_0, \beta_1, \tau)$. 其对数似然函数如公式 (4) 所示:

$$LL(\beta_0,\beta_1,\tau)=\ln L(\beta_0,\beta_1,\tau)=\sum_{j=1}^{n}[\delta_j\ln P(x_j)+(1-\delta_j)(1-\ln P(x_j))] \tag{4}$$

其中, x_j 为解释变量 x 在第 j 个类或模块上 OO 度量值, n 为样本数; 当类或模块 j 没有缺陷时, δ_j=0; 当类或模块 j 有缺陷时, δ_j=1. 对该对数似然函数采用广义线性交互式建模 (GLIM) 方法[155]可以得出比较精确的阈值 τ 估计值. 当一个具体的 OO 度量阈值 τ, 以及模型的参数 β_0 和 β_1 算出估计值后, 需要通过假设检验对参数进行评估.

对参数 β_0 和 β_1 是否为 0 的假设检验中, Benlarbi 等人[78]的零假设为 H_0: $\beta_0=\beta_1=0$, 而 El Emam 等人[102]认为截距是否为零并不能提供实质性结论, 故其零假设中只有 β_1 参数, 即 H_0: $\beta_1=0$. 但是, 对 OO 度量阈值 τ 的假设检验与模型的参数 β_0 和 β_1 不同, 其零假设 H_0 和备择假设 H_1 如公式 (5) 所示:

$$H_0:\tau\leqslant M^{(1)}=\min M\text{ 和 }H_1:\tau>M^{(1)} \tag{5}$$

其中, $M^{(1)}$ 左上角括号里数字表示在 M 度量数值中从小到大的排序号. 当接受零假设时, 表明阈值 τ 等于或小于

OO 度量的最小值. 如图 5 所示, 对于阈值 τ 小于 OO 度量最小值的情况, 可以确定阈值模型和非阈值模型是相同的; 而对于阈值 τ 等于 OO 度量最小值的情况, 由于只有很小部分比例观察值是 OO 度量的最小值, 这使得阈值模型与非阈值模型非常相似. 因此, 在这两类情况下, 逻辑回归阈值模型和非阈值模型没有差别, 也即该 OO 度量没有阈值效应.

当拒绝零假设时, 表明阈值 τ 大于 OO 度量的最小值, 即该 OO 度量存在阈值效应. 如果被估计出的 OO 度量阈值 τ 靠近其在数据集上的最大值 $M^{(n)}$, 如图 5 所示, 会使得大多数类或模块发生缺陷的概率的观察值为 0, 从而会影响模型的参数估计值. Benlarbi 等人[78]认为这种 OO 度量在数据集上也不存在阈值效应. 在理想状态下, OO 度量阈值不应靠近数据集中 OO 度量值的最大值或最小值.

在 BLRT 模型中, 通过似然比 G 统计量检验 β_0、β_1 以及 τ 参数的显著性. 计算 G 统计量如公式 (6) 所示:

$$G = 2(\ln L(H_1) - \ln L(H_0)) \tag{6}$$

似然比 G 统计量在各自零假设条件下, 服从自由度为 1 的卡方分布. 对 OO 度量阈值 τ 参数的假设检验而言, 若 P 值小于给定的显著性水平 α, 则拒绝零假设; 反之, 则接受零假设.

3.1.3 阈值有效性的实证研究

Benlarbi 等人[78]分别在两个 C++语言系统数据集上, 应用 BLRT 模型进行实证分析后发现: WMC、DIT、CBO、RFC 和 NOC 这 5 个 OO 度量均不存在阈值效应. 从图 5 中可以看出, 如果这 5 个 OO 度量与错误倾向 (fault-proneness) 相关, 则属于非阈值模型的连续关系, 不存在阈值模型中缺陷发生概率的稳定区间 [min M, τ], 以及当 OO 度量超过阈值 τ 后, 缺陷发生概率出现急剧上升的现象.

1998 年, Hatton[156]基于人类记忆模型提出认知理论来解释复杂性度量的阈值效应及其与缺陷关系. 该理论同样也被推广至面向对象软件. Benlarbi 等人[78]利用上述实证研究验证该理论, 但未能得到有效验证. Benlarbi 等人[78]认为研究中的两个数据集来自不同国家的不同团队, 会使得验证削弱部分说服力. 但是, 他们声称首次使用具体的阈值模型来验证 OO 度量阈值效应理论. 这为后续研究者提供了 OO 度量阈值效应研究的新思路.

2000 年, El Emam 等人[102,154]也对类规模是否存在阈值效应进行实证分析. 其 BLRT 模型如公式 (7) 所示:

$$\pi = \frac{1}{1 + \mathrm{e}^{-(\beta_0 + \beta_1 (size - \tau) I_+ (size - \tau))}} \tag{7}$$

其中, τ 是类规模 ($size$) 度量的阈值. 当 $size \leqslant \tau$ 时, $I_+(size - \tau)$ 等于 0; 当 $size > \tau$ 时, $I_+(size - \tau)$ 等于 1. 类规模度量除了 SLOC 度量外, 还采用 Briand 等人[50]对类规模的 OO 度量: Stmts、NM 和 NAI, 其中 Stmts 是类中方法的声明和可执行语句数目; NM 是类中执行的方法数目; NAI 是类中不包括继承的属性数目, 如基础数据类型字符串和整型等. 4 个度量在数据集上的逻辑回归阈值模型均通过多重共线性检验, 解释变量前的系数均通过显著性检验且都为正数. 虽然通过对数似然最大化估计出各自阈值, 但都没有通过阈值模型与非阈值模型比较的显著性检验. 即类规模的 OO 度量不存在阈值.

综上, Ulm[149]在流行病学的提出的阈值模型在被 Benlarbi 等人[78]和 El Emam 等人[102,154]引入软件工程领域, 并试图求出部分 OO 度量的阈值. 虽然他们的实验结果都没有通过阈值的假设检验, 但该方法为后续 OO 度量研究提供重要借鉴意义.

3.2 定量风险评估方法 (quantitative risk assessment, QRA)

3.2.1 QRA 的基本原理

基于逻辑回归的定量风险评估方法 (QRA) 也称 Bender 方法. Ulm[149]将阈值定义为风险响应曲线的不可微断点, 在断点以下风险恒定, 在断点以上风险递增. Pastor 等人[157]通过双分段逻辑回归定义阈值为变化点的数值, 在变化点以下或以上均存在不同的风险增加. 然而, 当阈值定义为断点或变化点在无法识别或不可信的情况下, Bender[11]认为定量风险评估方法是一种有用的替代方法. 该方法假设预测模型中自变量低于其阈值时, 某一事件发生风险是一个常量且是可接受的, 则该阈值也可以被认为是有效阈值 (valid threshold). 该方法具体做法如下.

(1) 设 $p(x)=P(Y=1|Y=x)$ 为 $X=x$ 事件发生的概率, 则简单逻辑回归 (LR) 模型假设 p 的对数概率 (log odds) 与 X 线性相关, 其两种形式如公式 (8) 所示:

$$\log\left(\frac{p}{1-p}\right)=\beta_0+\beta_1 x \text{ 或 } p(x)=\frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \tag{8}$$

其中, β_0 和 β_1 是 LR 模型中的回归系数. 此处, 发生事件的概率 p 是 X 的连续函数且在每一点上都是可微的. 假设 p 与 X 之间的关系为正相关 ($\beta_1>0$), 其负相关的情况可依此类推.

(2) Bender[11]认为在流行病学研究领域中的剂量-反应曲线特征点上, 基准值 (benchmark value) 为某一事件发生风险急剧上升的特征点, 且具有实际意义上的相关性. 为此, 该基准值可定义为基于逻辑回归的可接受风险水平 (value of an acceptable risk level, *VARL*) 和可接受风险梯度 (value of an acceptable risk gradient, *VARG*) 两种数值. 当给定概率 p_0 时, *VARL* 的定义如公式 (9) 所示:

$$VARL=p^{-1}(p_0)=\frac{1}{\beta_1}\left(\log\left(\frac{p_0}{1-p_0}\right)-\beta_0\right) \tag{9}$$

当 X 取值小于 *VARL* 时, 则其对应的事件发生概率小于 p_0; 对于高于 *VARL* 的 X 值, 其风险发生概率大于 p_0. 然而, 公式 (9) 并没有考虑 p 与 X 之间关系的形状, 即当 X 在 *VARL* 的附近小部分区域没有考虑是否存在实际意义上的相关性 (X 在 *VARL* 点处是否可微). 为此, 将剂量-反应曲线的形状考虑在内的一个可行的方法是用 logistic 函数的导数来表示. 逻辑回归曲线的导数如公式 (10) 所示:

$$d(x)=p^{-1}(x)=\frac{\beta_1 e^{\beta_0+\beta_1 x}}{(1+e^{\beta_0+\beta_1 x})^2} \tag{10}$$

公式 (10) 的图形为具有最大值的单峰对称曲线. 当 $x_{max}=-\beta_0/\beta_1$ 时, 最大值为 $d_{max}=\beta_1/4$. 因此, 当给定 d_0 (d_{max} 除外) 时, 都存在两个对应的 x 值: 一个比 x_{max} 小, 另一个比 x_{max} 大. 图 6(a) 展示 *VARL* 值; 图 6(b) 展示 *VARG* 值.

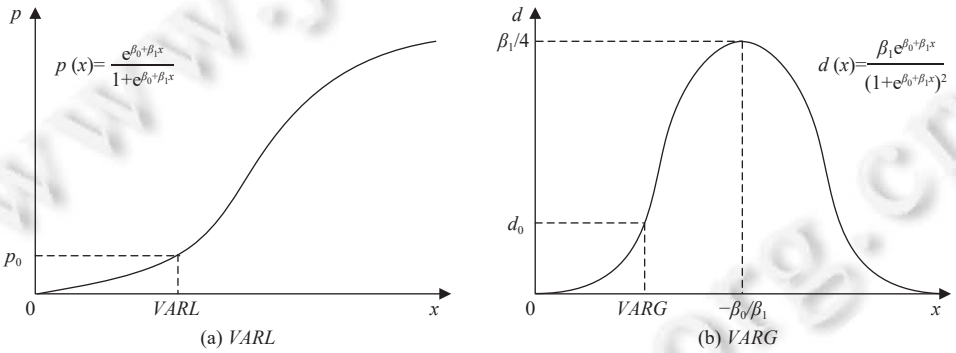


图 6 QRA 方法中的 *VARL* 值和 *VARG* 值示意图

在给定可接受风险梯度 $d_0\leqslant\beta_1/4$ 下, 以梯度类型定义的基准值 *VARG* 为比 x_{max} 小的数值, 如公式 (11) 所示:

$$VARG=d^{-1}(d_0)=\frac{1}{\beta_1}\left(\log\left(\frac{\beta_1-2d_0-\sqrt{\beta_1^2-4d_0\beta_1}}{2d_0}\right)-\beta_0\right) \tag{11}$$

其中, $0<d_0\leqslant\beta_1/4$. Bender[11]认为影响 *VARL* 和 *VARG* 值的 p_0 和 d_0 限制值受主观因素影响, 即很难给出 p_0 和 d_0 限制值的判定理由. 然而, 如果某个基准值应该具有临床意义而不是纯粹的数学意义, 则不可避免地需要选择 p_0 和 d_0 限制值, 或者是 p_0 和 d_0 限制值的组合.

3.2.2 面向对象度量阈值的确定

在软件工程领域, Shatnawi[85]于 2006 年将 QRA 方法引入面向对象度量阈值研究中. Shatnawi[85]首先用 0.05 的 P 值检验并筛选出 LR 模型中自变量 OO 度量前系数显著的度量; 然后, 分别用概率 p_0 值为 0.01 和 0.05 代入公式 (9) 中计算 *VARL* 值. 在 Eclipse 3.0 数据集上, 分别生成 CK 度量中 CBO、RFC 和 WMC 度量在类模块发生

缺陷概率为 1% 和 5% 的两个可接受风险水平下的阈值. Shatnawi[85]未给出选择 1% 和 5% 两个风险水平做相关阈值判断说明.

2010 年, Shatnawi[86]应用 QRA 方法先在 Eclipse 2.0 数据集上确定 3 个 CK 度量的阈值, 然后在 Eclipse 2.1 数据集上评估阈值效应. 其具体做法是对 VARL 公式中 p_0 值分别取 6%、6.5%、7.5% 和 10% 这 4 种风险水平构造决策树, 采用分类回归树方法找出能产生最佳分类精度的 p_0 值. 实验发现: 当 p_0 值为 0.065 时, CBO、RFC 和 WMC 度量阈值的预测缺陷性能指标 G-mean 值达到最大值. 然而, Shatnawi[86]发现 QRA 方法无法在 Mozilla 和 Rhino 软件系统上确定 CK 度量阈值. 当用概率 p_0 值分别为 0.01、0.05 和 0.1 代入 *VARL* 公式时, 只有 Rhino 软件系统在 1% 风险水平上可以确定 RFC 和 WMC 度量的阈值. 在 Rhino 软件系统其他风险水平上和 Mozilla 软件系统 3 种风险水平上均未能发现有效阈值. 有效阈值指阈值的取值范围在对应 OO 度量值域上.

3.2.3 阈值有效性的实证研究

在 Bender[11]提出 QRA 方法的基础上, 除了 Shatnawi[85,86]的阈值研究, Kaur 等人[119]、Singh 等人[120]、Malhotra 等人[115,126,129]、Hussain 等人[9]、Arar 等人[10]、Mauša 等人[87,130]、Morasca 等人[96,97,111]和 Padhy 等人[131]也通过选择符合判断标准的 *VARL* 或 *VARG* 值来确定 OO 度量阈值. 这类方法有一个共同特征: 把 OO 度量的阈值确定转化为 OO 度量值对应的模块发生缺陷概率值 (即 *VARL* 公式中的 p_0) 来确定. 随后, 选择判断标准确定 p_0, 从而确定 OO 度量阈值.

2013 年, Kaur 等人[119]采用 QRA 方法在 jfreechart 软件 1.0.0pre1 和 1.0.1 版本上分别确定 WMC、RFC、LCOM、CBO、DIT 度量以及 LOC 度量的阈值. 概率 p_0 值分别用 0.5、0.55、0.6、0.65 和 0.7 代入 *VARL* 公式, 得出 6 个 OO 度量在坏气味 (bad smell) 代码 5 种风险水平上的阈值. 除了 WMC 和 RFC 度量在 jfreechart 软件两个版本上的 50% 风险水平上以及 WMC 度量在 1.0.0pre1 版本的 55% 风险水平上未能求出有效阈值, 其他 OO 度量均在 jfreechart 软件系统的两个版本上求出有效阈值. 然而, Kaur 等人[119]未对上述 5 种风险水平上的阈值做进一步选择判断, 只是提出用不同的 OO 度量阈值识别不同风险水平上的类模块.

2014 年, Singh 等人[120]首先在 Mozilla Firefox 1.5、2.0 和 3.0 版本上收集了 12 个 OO 度量数据, 分别是 PuF[66]、EncF[66]、WMC、DIT、NOC、NOA、NOM[33]、NOAM、LCOM、Co、RFC 和 CBO 度量. 然后, 根据 QRA 方法计算 5%、6.5%、7.5%、10% 和 12.5% 这 5 种风险水平上的度量阈值. 接着, 分别使用每种风险水平上的阈值在 Mozilla Firefox 软件 3 个版本数据集上进行缺陷预测. 若 *AUC* 值大于 0.7, 则对应风险水平上的阈值被验证. 不仅如此, Singh 等人[120]认为 *AUC* 值越大, 其度量阈值越有效和实用. 因此, 在 Mozilla Firefox 软件 3 个版本上, 选择缺陷预测性能指标 *AUC* 值最大时, 对应风险水平上的度量阈值为最佳阈值.

2015 年, Malhotra 等人[115]在 Ivy 2.0、Ant 1.7、Tomcat 6.0、jEdit 4.3 和 Sakura 2.0.2.0 数据集和 KC 1 (NASA) 数据集上, 收集 6 个 CK 度量、LOC 度量和缺陷度量数据. 首先应用 QRA 方法在 KC1、Ivy 和 jEdit 数据集上对通过单变量逻辑回归系数检验的度量计算不同风险水平上的阈值, p_0 值分别取 0.01 至 0.15 之间的数值. 其阈值判断标准是使得各自软件系统上所有 OO 度量的 *VARL* 值都在其观察值范围内的最小 p_0 值. 其理由是随着风险等级值的增大, 类模块发生缺陷的风险也随之增大. 最后, 在 KC 1、Ivy 2.0 和 jEdit 4.3 系统的有效阈值基础上, 采用贝叶斯网络、朴素贝叶斯、随机森林、支持向量分类机和多层感知器 5 种机器学习方法在同一项目或相似项目上验证生成 OO 度量阈值.

2015 年, Malhotra 等人[126]直接选择 0.05 的 p_0 值确定度量阈值. 2016 年, Hussain[127]也直接选择 0.05、0.065、0.08 和 0.095 的 p_0 值确定度量阈值.

2016 年, Hussain 等人[9]用 PROMISE 数据集中 jEdit 4.0 和 Ant 1.3 项目分别作为训练集, 应用 QRA 方法在两个项目上分别生成 13 个 OO 度量阈值, 其中, *VARL* 公式中 p_0 值的取值为 0.1, 然后, 再分别在各自项目的后续版本上评估每个 OO 度量阈值在分类性能指标上的有效性 (accuracy). 与 Erni 等人[133]和 Ferreira 等人[109]的阈值确定方法 (同样实验流程) 进行比较, accuracy 指标值有显著提高.

2016 年, Arar 等人[10]将 PROMISE 数据集[150]中 10 个项目最早版本的系统合并作为训练集. 每个模块发生缺陷的风险水平分为 3 种: 没有缺陷、有 1 个缺陷和有 3 个缺陷. 与这 3 种风险水平对应, 在训练集上应用 QRA 方

法确定 OO 度量两种类型阈值, 即区分有 1 个缺陷和没有缺陷的阈值及区分有 3 个缺陷和没有缺陷的阈值. 两种类型阈值的 *VARL* 公式中 p_0 取值为各自训练集中标签为 1 的模块数与所有模块数之比. 在确定两种类型的阈值后, 分别利用训练集中各项目后续版本共 27 个系统作为测试集并计算 G-mean 值. 若阈值在测试集上 G-mean 平均值大于等于 0.6, 则该度量阈值被验证. 2019 年, Padhy 等人[131]也采用测试集上 G-mean 值大于等于 0.6 来验证阈值.

2017 年, Mauša 等人[87]在 Eclipse JDT 和 PDE 项目 14 个版本系统上, 采用分层 10 折交叉验证方法, 在每个项目上应用 QRA 方法计算 *VARL* 值为 OO 度量阈值. *VARL* 公式中 p_0 值的取值为分层 10 折交叉验证方法中 9 折训练集上有缺陷的模块数与所有模块数之比. 然后, 将生成的 OO 度量阈值在 1 折测试集上验证其 G-mean 值是否大于 0.6. 若 G-mean 值大于 0.6, 则对应的 OO 度量阈值被验证; 若 G-mean 值小于等于 0.6, 则阈值未能被有效验证. 2019 年, Mauša 等人[130]在 Eclipse JDT 项目 5 个版本数据集上采用上述相同方法确定 OO 度量阈值.

2017 年, Morasca 等人[111]提出风险规避 (risk-averse) 下基于单变量线性二元 Logistic 回归 (BLR) 和 Probit 回归 (PBR) 的缺陷倾向预测模型斜率的阈值确定方法, 包括 4 种阈值确定方法: (1) 两个模型斜率具有最大凸性 (maximum convexity) 处, 即模型的三阶导数为零处阈值 x_{MC}; (2) 两个模型最大斜率值一半 (maximum slope/2) 处的阈值 $x_{MS/2}$; (3) 两个模型斜率中位数的阈值 x_{med}; (4) 两个模型斜率均值的阈值 x_{avg}. 同时, 还提出 4 种缺陷倾向 $fp(x)$ 值 (*Fifty*, *Tr*, *Ts* 和 *All*) 作为阈值参考值. 他们在 PROMISE 数据集[150]中 52 个项目上分别建立 BLR 和 PBR 模型. 实验结果表明, 所有阈值的 *F* 度量值都呈现相似的分布特征. $x_{MS/2}$、x_{MC} 和 x_{avg}. 在召回率性能指标上比其他阈值的性能指标数值高, 且 x_{med} 的召回率比 4 种类型阈值 (*All*, *Fifty*, *Tr* 和 *Ts*) 参考值高.

2019 年, Lavazza 等人[97]提出基于对错误倾向函数标准偏差的计算来识别 BLR 模型中不确定区域范围上限和下限阈值的方法. BLR 模型中, 对所有 x 值的估计是一个概率值 $fp(x)$. 但 $fp(x)$ 不是与值 x 相关的模块为正例的概率 $p(x)$, 而是 $p(x)$ 的一个估计值. 通过方差的平方根 (即样本标准差 $sd(x)=\sqrt{fp(x)(1-p(x))}$) 来量化 $p(x)$ 值分布在期望值附近的程度. 即 $fp(x)$ 是 $p(x)$ 的估计有多可靠. 在 PROMISE 库的 54 个数据集上进行了测试后发现, 在大多数情况下, 从可靠范围内的变量获得的估计比从不确定范围内的变量获得的估计更加准确. 即不确定范围的上限和下限作为 OO 度量两个分类阈值, 能很好地区分正例 (大于上限阈值) 和负例 (小于下限阈值).

2019 年, Malhotra 等人[129]在 Apache Click 2.0–2.1 版本上收集 6 个 CK 度量, 以及 NPM、MOA、MFA[24]、IC[48]、LCOM3[31]、CBM、AMC[48]和 LOC 度量数据. 先用显著性水平 0.05 对 LR 模型回归系数筛选出符合条件 OO 度量, 再应用 QRA 方法计算 *VARL* 值. *VARL* 公式中 p_0 值的取值范围为 0.05 至 0.13. 阈值标准是使得所有 OO 度量在 Apache Click 2.0–2.1 版本上的 *VARL* 值都在其观察值范围内的最小 p_0 值. 然后, 将在 Apache Click 2.0–2.1 版本上得出的 OO 度量阈值在 2.1–2.2 版本上验证. 先将所有的原始度量数据转换成二分类数据, 应用随机森林和多层感知机两种机器学习方法在二分类数据集上建立各自模型并进行 10 折交叉验证. 实验结果显示, 在 Apache Click 的 2.1–2.2 版本上 *AUC* 值比在 2.0–2.1 版本上有所增加, 即 OO 度量阈值被验证.

综上, 这类方法将 OO 度量阈值确定问题转化为在模型输出值中寻求对应的阈值标准确定. 但缺少对阈值的假设检验, 且大多数阈值标准 (如 G-mean 值大于 0.6) 受主观因素影响较大.

3.3 ROC 曲线方法

3.3.1 ROC 的基本原理

ROC 曲线可以用来评估二值分类所提供信息的质量. Spackman 等人[158]将 ROC 曲线引入机器学习领域. 根据学习器的预测结果对样例进行排序, 按此顺序逐个把样本作为正例进行预测, 每次计算出假正例率和真正例率两个数值, 分别作为横、纵坐标作图, 就得到 ROC 曲线.

为了画出 ROC 曲线, 需要定义两个变量: 二分类变量 (0 或 1) 和连续变量. 如图 7(a) 所示, ROC 曲线纵轴是真正例率, $TPR=TP/(TP+FN)$, 也称敏感性 (*sensitivity*); 横轴是假正例率, $FPR=FP/(FP+TN)$, 也称 1−特异性 (1−*specificity*). 当候选阈值设为最大时, 即把所有的样例均预测为反例, 此时真正例率和假正例率均为 0, 如图 7(a) 中原点 (0, 0) 所示. 当候选阈值变小时, 以增加假正例率 (成本, false alarms) 为代价增加真正例率 (收益, hits); 当候选阈值变大时, 以减少真正例率为代价减少假正例率.

3.3.2 面向对象度量阈值的确定

2010 年, Shatnawi 等人[118]通过 ROC (receiver operating characteristic) 曲线方法确定 OO 度量阈值, 即 ROC 曲线可以用来评估使用 OO 度量进行二值分类所提供信息的质量. Shatnawi 等人[118]选择 OO 度量作为连续变量. 正例表示被度量的代码块有缺陷; 反例表示没有缺陷.

如表 4 所示, *TP*、*FP*、*TN*、*FN* 分别表示混淆矩阵中的真正例、假正例、真反例、假反例, 其含义分别为:

(1) 当样本真实值是正例, 而 OO 度量值大于等于阈值预测为正例时, 则为真正例 *TP*.

(2) 当样本真实值是正例, 而 OO 度量值小于阈值预测为负例时, 则为假负例 *FN*.

(3) 当样本真实值是负例, 而 OO 度量值小于阈值预测为负例时, 则为真负例 *TN*.

(4) 当样本真实值是负例, 而 OO 度量值大于等于阈值预测为正例时, 则为假正例 *FP*.

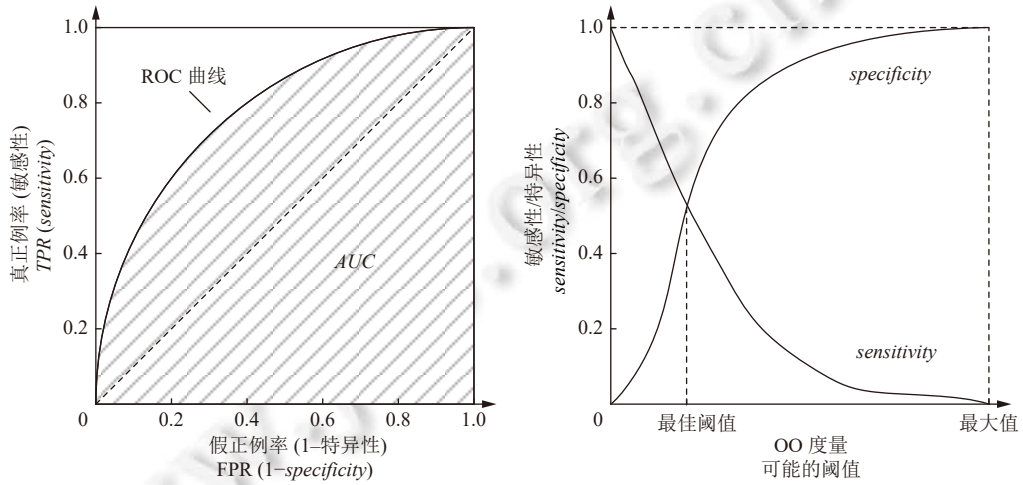


图 7 ROC 曲线与最佳 OO 度量阈值确定示意图

表 4 混淆矩阵表

分类估计值	真实值		
	反例	正例	
OO度量< 阈值(反例)	*TN* (true negatives)	*FN* (false negatives)	*EN=TN+FN* (estimated negatives)
OO度量≥阈值(正例)	*FP* (false positives)	*TP* (true positives)	*EP=FP+TP* (estimated positives)
	AN=TN+FP (actual negatives)	*AP=FN+TP* (actual positives)	*n=AN+AP=EN+EP*

所有正例估计值之和为 *EP*, 它等于真正例 *TP* 和假正例 *FP* 之和; 所有反例估计值之和为 *EN*, 它等于真反例 *TN* 和假反例 *FN* 之和; 所有正例真实值之和为 *AP*, 它等于真正例 *TP* 和假反例 *FN* 之和; 所有反例真实值之和为 *AN*, 它等于真反例 *TN* 和假正例 *FP* 之和. 数据集中样本总数为 *n*, 它等于 *AN* 与 *AP* 之和, 也等于 *EN* 和 *EP* 之和.

需要注意的是, Shatnawi 等人[118]并未计算每个类模块发生缺陷或错误的概率来排序, 而是直接用 OO 度量值排序 (一般由大到小的顺序), 按此顺序逐个把样本作为正例进行预测, 从而得出 ROC 曲线. 即在 OO 度量值域上, 按序依次选择某个度量值作为候选阈值, 排在候选阈值前面的度量值预测为正例, 排在后面的预测为负例. 从图 7(a) 原点开始, 将候选阈值依次设为从 OO 度量最大值至最小值 (一般为 0) 之间的度量值, 即依次将 OO 度量值大于候选阈值的模块划分为有缺陷的模块 (正例) 后, 可为每个候选阈值构建一个混淆矩阵用于评估分类性能. 每个混淆矩阵对应在 ROC 曲线上产生一个点, 而每个点是一对 (1–特异性) 和敏感性值 (*FPR* 和 *TPR*).

为了平衡候选阈值变动时成本与收益, Shatnawi 等人[118]选择敏感性值和特异性值之和最大的一对作为阈值判断标准, 并认为其对应的候选阈值是实际使用中的最佳阈值. 如图 7(b) 所示, 最佳阈值近似于在敏感性和特异

性曲线交点处的 OO 度量值[159]. 在最佳阈值处, 满足真正例率 (*TPR*) 和真反例率 (*TNR*) 同时最大 (即敏感性值和特异性值之和最大), 也满足假正例率 (*FPR*) 和假反例率 (*FNR*) 同时最小. 因此, Shatnawi 等人[118]确定 OO 度量阈值的判断标准还可以表述为敏感性值等于 (或近似等于) 特异性值时的 OO 度量值为其阈值.

在确定 OO 度量阈值后, Shatnawi 等人[118]通过 *AUC* (area under ROC curve, ROC 曲线下面积) 验证阈值的分类性能. 如图 7(a) 所示, ROC 曲线下的阴影面积为 *AUC*. 图中对角线表示随机猜测下的 ROC 曲线, 其 *AUC* 值为 0.5. Hosmer 等人[159]对 *AUC* 分类性能的规则为: *AUC*=0.5 表示无区分; 0.5<*AUC*<0.6 表示差; 0.6≤*AUC*<0.7 表示公平; 0.7≤*AUC*<0.8 表示可接受; 0.8≤*AUC*<0.9 表示优秀; *AUC*≥0.9 表示极好. Shatnawi 等人[118]认为实际应用中阈值的分类性能至少在可接受层次上. 即当 *AUC*≥0.7 时, OO 度量阈值被验证有效; 反之, 当 *AUC*<0.7 时, 对应 OO 度量并不能找到可实际应用的阈值. 根据上述 ROC 曲线方法, Shatnawi 等人[118]确定 OO 度量阈值的具体做法如下.

首先, 在 Eclipse 2.0、2.1 和 3.0 版本上收集 12 个 OO 度量数据: CBO、RFC、WMC、LCOM、DIT、NOC[22]、CTA、CTM[45]、NOAM、NOOM、NOA 和 NOO[23]度量. 为了绘制 ROC 曲线, Shatnawi 等人[118]提供两种场景下的分类变量: 二分类 (binary) 和顺序分类 (ordinal). 其中, 二分类是将类模块分为有错误和没有错误; 顺序分类是将类模块分为 4 种类别: 没有错误、低风险、中风险和高风险. 顺序分类中的类别需要逐个转为二分类, 即分别绘制低、中、高风险与无错误类别之间二分类的 ROC 曲线.

其次, 根据这两种场景下二分类变量, 分别提出两个零假设. 假设 1: 在 Eclipse 2.0、2.1 和 3.0 版本上, 没有实际可用的 OO 度量阈值区分有错误类模块和没有错误类模块. 假设 2: 没有实际可用的 OO 度量阈值区分任何一个错误类别 (低风险、中风险和高风险) 和无错误类别模块.

然后, 通过 ROC 曲线方法, 计算在二分类场景下各 OO 度量阈值、*p* 值和 *AUC* 值. 虽然绝大数 OO 度量通过 *p* 值检验, 即 *AUC* 显著不等于 0.5, 但所有 OO 度量阈值的分类性能 *AUC* 值均小于可接受范围 (*AUC*<0.7). 故未能拒绝假设 1. 在顺序分类场景下, 低风险类别的所有 OO 度量阈值的分类性能 *AUC* 值也都小于 0.7; 在中风险和高风险类别下, 只有 CBO、RFC、WMC、CTM 和 NOO 度量的阈值被验证 (*AUC*≥0.7).

最后, Shatnawi 等人[118]将 Eclipse 项目 3 个版本上的中风险和高风险类别的类模块合并, 应用 ROC 曲线方法, 确定 RFC、CTM、CBO、WMC 和 NOO 度量阈值, 并选择最大敏感性值为最终的阈值.

3.3.3 阈值有效性的实证研究

在 Shatnawi 等人[118]提出 ROC 曲线方法基础上, Tosun 等人[92]、Catal 等人[105]、Malhotra 等人[122]、Kumari 等人[125]、Boucher 等人[121] 、Shatnawi 等人[123,128]、Morasca 等人[106]、Sánchez-González 等人[68]和 Mendling 等人[108]通过选择不同阈值验证方法, 来确定度量阈值. 这类方法有一个优点: 在数据集不满足正态分布和存在类别不平衡情况下, 依然可以借助 ROC 曲线来确定阈值. 即遍历 OO 度量所有的候选阈值, 当其预测性能指标敏感性和特异性之和最大时即为最佳阈值, 再通过各自的方法 (常见的方法是 *AUC*≥0.7) 进行有效验证.

2009 年, Tosun 等人[92]选择 ROC 曲线上最接近 (0, 1) 点的阈值作为最佳决策阈值 t_{opt}, 其理由是 ROC 曲线图中左上点 (0, 1) 表示最好的分类性能. ROC 曲线上任意一点 (*pf*, *pd*) 到 (0, 1) 的欧氏距离 (Euclidean distance) 如公式 (12) 所示, 其中 *pf* (probability of false alarms) 即 FPR, *pd* (probability of detection rate) 即 *TPR*.

$$distance(pf,pd)=\sqrt{(0-pf)^2+(1-pd)^2} \tag{12}$$

若选择 t_{opt} 左边的点作为阈值, 则 *pf* 变小了, 但 *pd* 也同样减小; 若选择 t_{opt} 右边的点作为阈值, *pf* 和 *pd* 均增大. 因此, Tosun 等人[92]选择欧氏距离最小的点 (*pf*, *pd*) 作为最佳决策阈值 t_{opt}. 如公式 (13) 所示, 即当 $distance(pf,pd)$ 取最小值时, 点 (*pf*, *pd*) 作为阈值 t_{opt}.

$$t_{opt}=\arg\min_t distance(pf,pd) \tag{13}$$

2011 年, Catal 等人[105]在 ROC 曲线方法上提出噪音检测算法. 通过 5 个 NASA 公共数据集和基于朴素贝叶斯缺陷预测模型, 应用噪音检测算法比较建模前后度量数据集. 实验结果发现, 该噪音检测方法高效且提高缺陷预测性能. Catal 等人[105]认为选择敏感性和特异性之和最大的一对作为阈值判断标准并不总是存在, 如某个候选阈值会使得 *pd* (敏感性) 达到最大值, 而另一个候选阈值会使得 *pf* (1–特异性) 达最小值. 为了平衡 *pd* 与 *pf* 之间这种

矛盾, Catal 等人[105]提出使用经过 (0, 0), (1, 1) 和 (*pd*, *pf*) 这 3 点的 ROC 曲线下面积 *AUC* 的最大值对应的候选阈值作为最佳阈值.

2012 年, Malhotra 等人[8]把 Bender 等人[11]的 QRA 方法与 Shatnawi 等人[118]的 ROC 曲线方法两者结合起来确定 OO 度量阈值. 先为每个 OO 度量构建 ROC 曲线. 将 ROC 曲线上敏感性与 (1–特异性) 相等的点作为候选阈值标准, 通过 QRA 方法求出在该标准下候选阈值发生缺陷的基础概率 p_0, 再将 p_0 代入 *VARL* 公式求出 OO 度量阈值. 同年, Sánchez-González 等人[68]和 Mendling 等人[108]通过在 ROC 曲线上找到使敏感性和特异性最大化的点来选择最佳阈值, 其阈值标准是选择 ROC 曲线上到对角线 (如图 7(a) 中对角线) 最大距离的点作为最佳阈值. Sánchez-González 等人[68]是基于逻辑回归模型得出 ROC 曲线. 即 ROC 曲线上每一点 (*FPR*, *TPR*) 值是通过模型预测后得出的性能值. 通过比较 ROC 曲线方法和 Bender 方法确定 OO 度量阈值的召回率和精度性能指标后, Sánchez-González 等人[68]发现利用 ROC 曲线方法确定的阈值可以获得更准确的测量评估.

2013 年, Kumari 等人[125]应用 ROC 方法确定度量阈值时选择的标准是敏感性和特异性都具有最大值的一对, 即同时最小化假正例数和假负例数. 再通过 $AUC \geqslant 0.7$ 对确定的阈值进行有效验证.

2016 年, Boucher 等人[121]通过比较 ROC 曲线方法和 Alves 排序法两种阈值确定方法, 以便更好地进行软件错误预测. 在 Apache ANT、Ivy 和 KC1 等 5 个项目数据集上, 应用 ROC 曲线方法在每个项目上生成 SLOC、CBO、RFC 和 WMC 这 4 个度量阈值后, 通过假正例率、假反例率和错误率 ($(FN+FP)/(TP+FP+FN+TN)$) 这 3 个预测性能指标 (*PM*) 来验证阈值. 这 3 个指标值越低, 则阈值的分类性能则越好. 这 3 个性能指标规则为: $PM>0.5$ 表示无区分; $0.4<PM\leqslant 0.5$, 表示差; $0.3<PM\leqslant 0.4$, 表示公平; $0.2<PM\leqslant 0.3$, 表示可接受; $0.1<PM\leqslant 0.2$, 表示优秀; $PM\leqslant 0.1$, 表示极好. 依据 Shatnawi 等人[118]对 OO 度量阈值验证规则 ($AUC\geqslant 0.7$), Boucher 等人[121]认为在错误预测场景下, OO 度量阈值被验证的条件是 3 个预测性能指标至少都在可接受水平 (即都小于等于 0.3) 上.

2017 年, Shatnawi[128]在其 ROC 曲线方法[118]上, 增加了数据不平衡和特征选择对 OO 度量阈值确定方法的影响. 数据不平衡是指数据集中少数类有缺陷, 而绝大多类都没有缺陷; 特征选择是指并不是所有 OO 度量都与被度量模块存在缺陷有很强的相关性. Shatnawi[128]在 Ant、Camel 和 jEdit 等 5 个系统上的实验结果发现, WMC、CBO、RFC 和 LCOM 这 4 个度量与大多数系统版本的缺陷之间存在显著关系. 对于通过 *AUC* 显著不等于 0.5 检验的 OO 度量, 在 5 个系统各版本上, ROC 曲线方法确定的阈值并不一致. 对数据进行过采样和欠采样的结果与不进行采样情况下 ROC 分析没有显著差异. 通过 ROC 分析选择 OO 度量的分类性能与通过在所有 OO 度量上逐步进行特征选择构建的 4 种机器学习模型没有显著差异. 与 3 种特征选择技术相比, ROC 分析选择出相同 OO 度量时更可靠、更一致.

2017 年, Malhotra 等人[122]应用 ROC 方法, 在开源操作系统 Android 2.3、4.0、4.1、4.2 和 4.3 版本数据集上, 遍历每个 OO 度量所有度量值, 当其预测性能指标敏感性等于特异性时即为最佳阈值. 15 个 OO 度量阈值确定后用于预测代码变更倾向. 代码变更倾向被定义为在软件的后续版本中预测发生变化的概率. 阈值验证主要采用版本间验证的方法, 即用前一个版本上度量阈值预测后一个版本上的发生代码变更倾向. 如用 Android 2.3 版本上的 OO 度量阈值预测 Android 4.0 版本上的代码变更倾向. 当 *AUC* 值大于 0.55 时, 被验证为有效阈值.

2018 年, Shatnawi[123]在 D'Ambros 等人[151]提供的 5 个项目数据集上, 应用 ROC 曲线方法确定 CBO、RFC、WMC、DIT、NOC 和 LCOM 度量的阈值来对类模块发生变化倾向进行预测. 对于通过 *AUC* 显著不等于 0.5 检验的 OO 度量, 遍历每个数值作为候选阈值, 选取敏感性和特异性两个值接近相等且总和最大的点作为最佳阈值. 通过 Hosmer 规则验证 5 个项目上确定的度量阈值是否有效. 结果表明, DIT 和 NOC 两个继承类 OO 度量的 *AUC* 接近 0.5, 未能通过阈值有效验证. 另外 4 个度量基本上都得到有效验证, 即 *AUC* 值在可接受水平上.

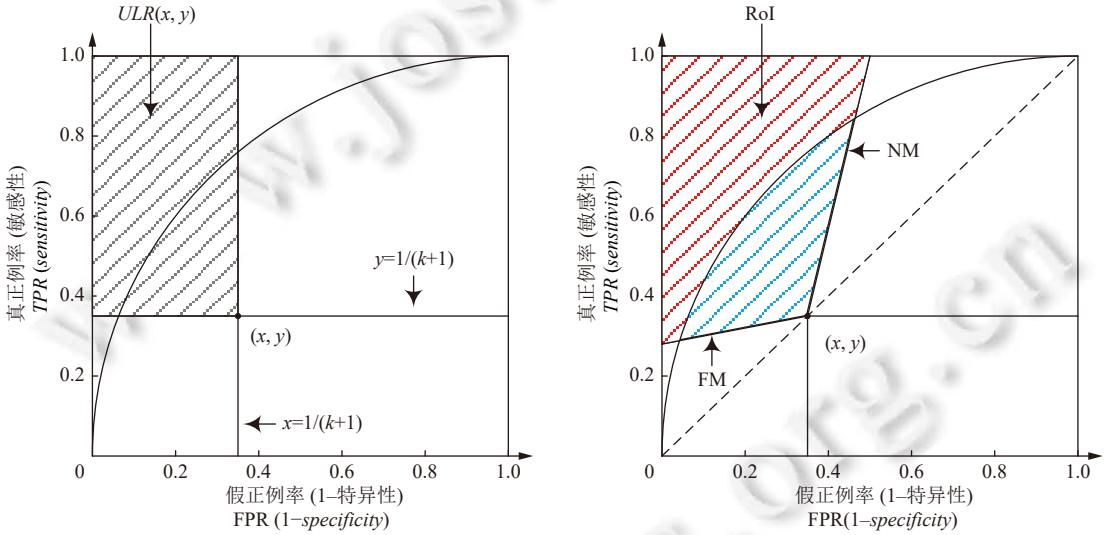
2020 年, Morasca 等人[106]认为, 当一个缺陷倾向预测模型的 *AUC* 大于另一个模型的 *AUC*, 并不能表示前者是可取的, 并提出通过基于随机策略和确定性策略方法在性能指标上设定最小值标准. 这些最小值标准有助于确定度量阈值. Morasca 等人[106]在 SEACRAFT 库[152]中 67 个数据集上构建缺陷预测模型, 生成 RRA (ratio of relevant areas)、*AUC* 和基尼系数 3 个性能指标值. 通过比较基于缺陷倾向模型和所有可能阈值的缺陷预测模型性能指标后, 发现 RRA 性能指标比传统的 *AUC* 和基尼系数 (Gini coefficient, *G*) 性能指标更加可靠且表现出更少的

错误分类. 基尼系数 (G=2AUC−1) 也是用于评价缺陷预测模型的分类性能. 两者的分类性能如表 5 所示. G 的取值范围在 [0, 1], 对应在 ROC 曲线图中对角线 y=x 的上方. 当 AUC 取值大于等于 0.7 或基尼系数值大于等于 0.4 时, 对应的预测指标为可接受水平及以上的分类性能.

表 5 AUC 与基尼系数的分类性能表

AUC取值范围	基尼系数取值范围	分类性能 (evaluation)
AUC=0.5	G=0	完全随机 (totally random)
0.5<AUC<0.7	0<G<0.4	差 (poor)
0.7≤AUC<0.8	0.4≤G<0.6	可接受 (acceptable)
0.8≤AUC<0.9	0.6≤G<0.8	优秀 (excellent)
0.9≤AUC<1	0.8≤G<1	极好 (outstanding)

Morasca 等人[106]提出新的 RRA 性能指标定义过程如下: 首先, 如图 8(a) 所示, 在 ROC 曲线图中阴影部分表示点 (x, y) 的左上角矩形 ULR(x, y). 对于任何合理的性能指标, ULR(x, y) 中所有点的性能指标值都不比点 (x, y) 本身差. 因此, 点 (x, y) 可以用来作为一个最低标准参照点. 其次, 如图 8(b) 所示, ROC 曲线以下浅蓝色阴影和 ROC 曲线以上红色阴影组合区域包含了 ULR(x, y) 中所有的点. 组合阴影区即是利益区域 RoI. 最后, ROC 曲线中 RoI 的相关面积之比 RRA 是低于 ROC 曲线的 RoI 面积与 RoI 的总面积之比. 如图 8(b) 所示, RRA 值等于浅蓝色阴影区域与所有阴影区域 (浅蓝色和红色) 面积总和之比.



(a) ULR 示意图 (b) RoI 示意图

图 8 左上角矩形 (ULR) 与利益区域 (RoI) 示意图

Morasca 等人[106]认为 AUC 与基尼系数 (G) 都是 RRA 的特例. 当利益区域 RoI 为整个 ROC 空间时, RRA 值等同 AUC 值; 当利益区域 RoI 为 ROC 空间中左上三角形时, RRA 值等同基尼系数值. 与 AUC 或 G 类似, 实践者和研究者可以通过设定缺陷倾向预测模型的 RRA 指标参考值来确定 OO 度量阈值. Morasca 等人认为 RRA 指标参考值可以由实践者和研究者通过利益区域 RoI 的边界来确定, 并提出两种可供选择的边界标准 FM 和 NM.

综上, ROC 曲线方法及其衍生方法均以候选阈值的缺陷预测性能最优为导向, 大多数缺少对阈值的假设检验过程或存在阈值检验不通过情况[92]. 虽然在数据集不满足正态分布和存在类别不平衡情况下, ROC 曲线方法依然可以确定阈值, 但是 OO 度量的阈值效应检验对其应用来说仍是不可或缺的一步.

3.4 其他有监督学习方法

本节介绍基于遗传算法的方法、矩形学习方法、BPP 方法、BCE 方法、MFM 方法、FPH 指数法和基于分

类树学习器算法 C5.0 方法等等其他有监督学习方法.

3.4.1 基于遗传算法的阈值确定方法

2005 年, Mihancea 等人[124]在研究面向对象软件系统设计缺陷自动检测优化的过程中, 提出基于遗传算法的调整方法 (tuning machine method based on a genetic algorithm) 来确定 OO 度量阈值, 使得阈值能够最大限度地正确分类的实体数量. 遗传算法不断搜索每个潜在的解决方法 (候选阈值) 使得适应度函数 (fitness function) 最小化. 如公式 (14) 所示:

$$f(x)=A*b*Fn\_No(X)+A*(1-b)*Fp\_No(X)+1 \tag{14}$$

其中, X 是潜在的解决方法 (候选阈值); $Fn\_No(X)$ 和 $Fp\_No(X)$ 分别是当使用编码为 X 的阈值对调优集应用调优策略时获得的假负例数和假正例数; A 是由用户设定的处罚幅度 (penalty amplitude) 且为正整数, 它帮助用户调整由 X 产生不一致的处罚值; b 是 0 至 1 之间的系统平衡参数, 它实现了调整组件的平衡调整点, 当 b 为 0.5 时, 假负例数 $Fn\_No(X)$ 和假正例数 $Fp\_No(X)$ 将以相同的分数惩罚. 在公式 (16) 中, $f(X)$ 最小时的候选阈值 X 值为阈值.

3.4.2 矩形学习方法

2011 年, Herbold 等人[107]利用轴对齐 d 维矩形 (axis-aligned d-dimensional rectangles) 的机器学习方法确定阈值. 应用矩形学习算法[160]时, 一般分为两个阶段: 第 1 阶段根据训练数据分布进行分区; 第 2 阶段基于此分区计算矩形. 第 1 阶段目的是找到 d 维实空间的一个分区; 而第 2 阶段是计算目标矩形的边界. 其阈值确定过程如下.

已知给定的度量集 $M=\{m_1, \ldots, m_d\}$和软件代码块 $\mathbf{X}$ 与其对应的已知分类 $\mathbf{Y}$ 集合. 在度量集 M 上确定阈值 $T=\{t_1,\ldots, t_d\}$用来区分软件代码块 $\mathbf{X}$ 达到与 $(\mathbf{X}, \mathbf{Y})$ 相同的分类. 在软件代码块 $\mathbf{X}$ 上度量 M, 可将 $\mathbf{X}$ 映射到 d 维实空间中得到 $M(\mathbf{X}):=\{(m_1(x),\ldots, m_d(x)): x\in\mathbf{X}\}\subset\mathbf{R}^d$. 轴对齐矩形学习算法的输入是一对 $(M(\mathbf{X}), \mathbf{Y})$, 输出是一组 d 维的下限值和上限值 (l_i, u_i), 其中维度 i=1, …, d. 假设第 i 维软件度量 m_i 的值越大越不好, 则第 i 维中矩形上限 u_i 解释为度量 m_i 的阈值. 因此, 在得出所有的 $t_i=u_i$ 后, 度量集 M 的阈值集为 $T=\{t_1, \ldots, t_d\}$.

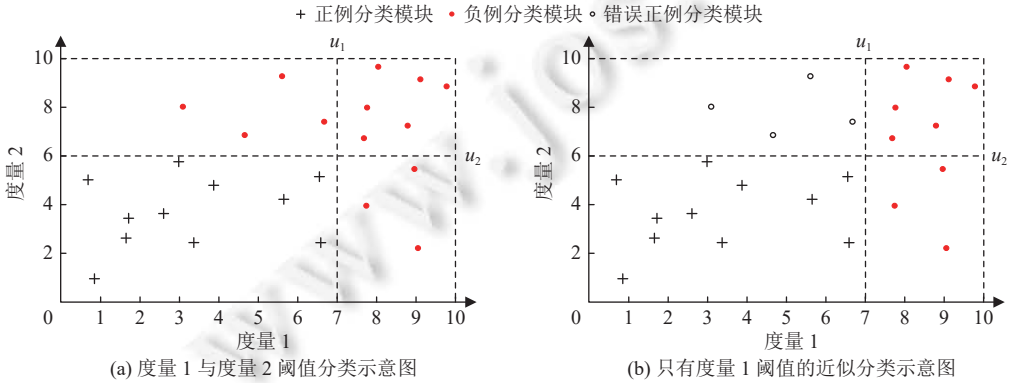
对于某个实体 x, 度量集 M 和阈值 T 的分类结果定义如公式 (15) 所示:

$$f_0(x,M,T)=\begin{cases}1, & \text{if } |\{i\in\{1,\ldots,d\}: m_i(x)>t_i\}|=0\\ 0, & \text{if } |\{i\in\{1,\ldots,d\}: m_i(x)>t_i\}|>0\end{cases} \tag{15}$$

当至少有一个度量 m_i 的值超过其阈值 t_i 时, $f_0(x, M, T)$ 的值为 0 (表示分类错误); 当没有度量超过其阈值时, $f_0(x, M, T)$ 的值为 1 (表示分类正确). 图 9 展示矩形学习在二维状态中用于软件模块分类的度量阈值. u_1 和 u_2 虚线展示在训练集上分类的两个度量阈值. 图 9(a) 展示两个度量均用于分类; 图 9(b) 展示只有度量 1 用分类, 即仅通过两个度量之一来近似通过两个度量获得的分类. 分类误差定义为随机抽取样本 (X, Y) 被错误分类的概率, 即 $\varepsilon=\mathbb{P}(f_0(X, M, T)\neq Y)$. 由此, 在给定训练样本 $(\mathbf{X}, \mathbf{Y})$ 上实证分析的分类误差定义如公式 (16) 所示:

$$\varepsilon_{\mathbf{X},\mathbf{Y}}(M,T)=\frac{1}{|\mathbf{X}|}\sum_{(x,y)\in(\mathbf{X},\mathbf{Y})}\mathbf{1}_{f(x,M,T)\neq Y} \tag{16}$$

其中, $\mathbf{1}$ 函数的定义为当 $f(x, M, T)\neq Y$ 为正确时取值 1; 当 $f(x, M, T)\neq Y$ 错误时取值 0.



(a) 度量 1 与度量 2 阈值分类示意图 (b) 只有度量 1 阈值的近似分类示意图

图 9 应用矩形学习方法确定阈值示意图

3.4.3 BPP、BCE 和 MFM 方法

2014 年, Zhou 等人[77]在研究类规模混合效应对软件缺陷预测影响时, 为了与随机模型达到相同召回率条件下检测缺陷预测模型在减少检查或测试工作量方面的效果, 提出 *ER* (effort reduction) 性能指标, 如公式 (17) 所示:

$$ER(m)=\frac{Effort(random)-Effort(m)}{Effort(random)} \tag{17}$$

其中, $Effort(m)=\left(\sum_{i=1}^{n}s_i\times p_i\right)/\sum_{i=1}^{n}s_i$ 为预测模型 m 中需要被检查或测试代码量与总代码量的比率, n 为某个软件系统中类模块数量, s_i 为第 i 个类模块的代码量 (SLOC), 当第 i 个类模块超过设定阈值时被预测为有缺陷, p_i 的值为 1, 反之则为 0; $Effort(random)=\left(\sum_{i=1}^{n}f_i\times p_i\right)/\sum_{i=1}^{n}f_i$ 为与预测模型 (m) 达到相同召回率, 随机模型中需要被检查或测试代码量与总代码量的比率, f_i 为第 i 个类模块中包含的缺陷数. 在计算 ER 指标中 p_i 值时, Zhou 等人[77]提出如下 3 种 OO 度量阈值确定方法.

(1) BPP 方法 (balanced-pf-pd method)——平衡 *pf* 和 *pd* 值的方法.

BPP 方法是基于缺陷预测模型应用 ROC 曲线方法来确定阈值. 与 Tosun 等人[92]选择最小欧式距离相似, BPP 方法是通过计算平衡 ROC 曲线中 *pf* 和 *pd* 值的 *balance* 指标实现的, 其公式如公式 (18) 所示. 其 OO 度量阈值判断标准为: 在给定的训练集上, 使得 *balance* 指标最大的候选阈值为最佳阈值.

$$balance=1-\frac{\sqrt{(0-pf)^2+(1-pd)^2}}{\sqrt{2}} \tag{18}$$

(2) BCE 方法 (balanced-classification-error method)——平衡分类错误的方法.

BCE 方法是在给定的训练集上, 选择假正例率 (FPR) 和假反例率 (FNR) 两个分类错误比率大致相等时的候选阈值为最佳阈值. 由于这种方法给了更多的权重由假正例带来错误分类, 所以 BCE 方法充分考虑了训练集中类不平衡 (大多数类模块都没有缺陷) 问题, 从而确定有效阈值.

(3) MFM 方法 (maximum-*F*-measure method)——最大化 *F* 度量值方法.

MFM 方法是在给定的训练集上, 选择使 *F* 度量最大时的候选阈值为最佳阈值[161]. *F* 度量是召回率和精度的调和平均数. 当候选阈值变小时, 其预测性能值召回率可适当拉升, 但由于增加了假正例而降低了精度; 当候选阈值变大时, 召回率相应会降低, 但也会使假正例的减少而提升精度. 事实上, *F* 度量也是平衡候选阈值变小或变大时出现召回率与精度两方面矛盾的一个性能指标.

通常, 较低的 OO 度量阈值往往会产生较高的召回率 (completeness) 和较低的精度 (correctness); 反之, 较高的 OO 度量阈值会产生较低的召回率和较高的精度. Zhou 等人[103]说通过在给定的应用项目权衡召回率和精度来寻找最佳的 OO 度量阈值. 具体做法是大致平衡误分类为有缺陷倾向和无缺陷倾向类别的数量来确定阈值, 即在混淆矩阵中 *FP* 值大致等于 *FN* 值. 该方法与 BCE 方法思路一致, 不同之处在于前者选择 *FP* 和 *FN* 的绝对数比较; 而后者选择 *FPR* 和 *FNR* 的相对数比较.

3.4.4 *FPH* 指数法

2015 年, Morasca 等人[81]扩展文献计量学 (bibliometrics) 中 H 指数[162]概念, 引入 *FPH* 指数 (fault-proneness H-index) 方法来设置 OO 度量阈值. 其基本思想是: 类似于用 H 指数法标识研究人员顶级出版物的方法设置阈值识别出最容易发生错误的模块.

Hirsch[162]定义一个科学家的 H 指数 (Hirsch 指数) 为: 如果他 (她) 的 N_p 篇论文中的 h 篇至少被引用 h 次, 而其他 (N_p–h) 篇论文每篇被引用不超过 h, 则该科学家的 H 指数为 h. H 指数是文献计量指数, 可以量化一组科学文章的影响, 但 H 指数重点关注研究人员引用最多的文献. 如果研究人员的 H 指数为 30, 则被引用次数少于 30 的论文引用次数实际上并不重要.

Morasca 等人[81]先提出分类模型, 它包括一个错误倾向模型和一个错误倾向阈值. 错误倾向模型是估计模块发生错误可能性的函数. 分类模型将错误倾向性超过阈值的模块估计为有错误模块, 否则估计为无错误模块. 错误倾向阈值为一个函数 t:$TR\times TS\rightarrow[0, 1]$, 其中 *TR* 和 *TS* 分别是所有可能的训练集和测试集. 阈值 $t(tr, ts)$ 不只是一个

数字, 而是一个正式 (full-fledged) 的函数, 它的值很可能会根据训练和测试集的使用而改变.

在 H 指数基础上, Morasca 等人[81]提出 *FPH* 指数. 设 $PF(X_M)$ 为取值区间 [0, 1] 上错误倾向函数, 给定一个 *MODULES* 模块集中的模块 m, 令 $PF(x_M)$ 为 m 的错误倾向值和 $ge(m)$ 为模块集中错误倾向值大于或等于 $PF(x_M)$ 模块所占的比例, 则 *MODULES* 模块集的 *FPH* 指数为 $PF(x_M)$ 的最小值 *fph*, 并使得错误倾向大于或等于 *fph* 的模块的比例 $ge(m)$ 满足 $fph \geqslant ge(m)$.

在 *FPH* 指数定义上可以得出 *FPH* 阈值为 $t(tr, ts)=fph$, *fph* 为在测试集模块上 *FPH* 指数. Morasca 等人[81]在 PROMISE 库中 MC2 数据集的 *BRANCH_COUNT* 度量和 Tomcat 数据集的 WMC 度量上应用 *FPH* 指数法确定阈值, 并提出用于比较的 4 个不同阈值: $t(tr, ts)=0.5$、$t(tr, ts)=AP_{tr}/n_{tr}$、$t(tr, ts)=AP_{ts}/n_{ts}$ 和 $t(tr, ts)=AP_{all}/n_{all}$, 后 3 个阈值分别是有缺陷的模块在训练集、测试集中以及两者的并集 ($all=tr \cup ts$) 中所占的比例. 实验结果表明, $t(tr, ts)=fph$ 阈值比另外 4 个阈值 $t(tr, ts)=0.5$、$t(tr, ts)=AP_{tr}/n_{tr}$、$t(tr, ts)=AP_{ts}/n_{ts}$ 和 $t(tr, ts)=AP_{all}/n_{all}$ 有更好的召回率, 但是精度比它们差. $t(tr, ts)=AP_{tr}/n_{tr}$ 阈值在所有阈值中 *F* 度量指标最好.

3.4.5 Lavazza 方法

2016 年, Lavazza 等人[93]提出悲观阈值 x_p 和乐观阈值 x_o. 在包含已知和未知错误模块的数据集上构建两个错误倾向模型 (BLR): 悲观模型假设数据集所有模块中未知错误信息模块为真实的错误模块, 而乐观模型假设所有未知错误信息模块为真实无错误模块. 在两个模型中, 分别使用两个缺陷倾向阈值: 已知真实有错误模块的比例和未知错误的模块比例. 模型和阈值的定义取决于所选的训练集, 并使用以下 3 个不同的训练集来构建模型和设置阈值, 而测试集 *ts* 将使用错误未知的模块数据. 即 $ts_{maxNeg}=ts_{maxPos}=ts_{neut}=UnK$.

(1) 最大正例训练集 tr_{maxPos} 将所有未知错误模块 (*UK*) 视为正例 (*AP*). 即 $AP_{maxPos}=AP+UK$ 和 $AN_{maxPos}=AN$. 在训练集 tr_{maxPos} 上, 构建悲观模型 BLR_{pess}, 并设置乐观阈值 $t_{opt}=AP_{maxPos}/n=(AP+UK)/n$.

(2) 最大负例训练集 tr_{maxNeg} 将所有未知错误模块 (*UK*) 视为负例 (*AN*). 即 $AP_{maxNeg}=AP$ 和 $AN_{maxNeg}=AN+UK$. 在训练集 tr_{maxNeg} 上, 构建乐观模型 BLR_{opt}, 并设置悲观阈值 $t_{pess}=AP_{maxNeg}/n=AP/n$.

(3) 中立的训练集 tr_{neut} 不使用未知错误模块. 即 $AP_{neut}=AP$ 和 $AN_{neut}=AN$. 在训练集 tr_{neut} 上, 构建中立模型 BLR_{neut}, 并设置中立阈值 $t_{neut}=AP_{neut}/(AN_{neut}+AP_{neut})=AP/(AP+AN)$.

通常, 悲观阈值小于乐观阈值 ($t_{pess} \leqslant t_{opt}$). 图 10 展示 RFC 度量应用 Lavazza 方法确定乐观和悲观阈值示意图. 下边虚线为悲观阈值 t_{pess}; 上边虚线为乐观阈值 t_{opt}; 左边实线为悲观模型 BLR_{pess}; 右边虚线为乐观模型 BLR_{opt}. 两类模型和两类阈值交点的 x 值, 组合出 4 种类型阈值: (1) BLR_{pess} 和 t_{pess} 的交点 x_{pp}; (2) BLR_{pess} 和 t_{opt} 的交点 x_{po}; (3) BLR_{opt} 和 t_{pess} 的交点 x_{op}; (4) BLR_{opt} 和 t_{opt} 的交点 x_{oo}.

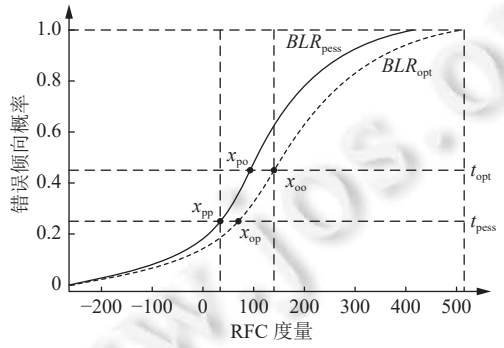


图 10 *RFC* 度量应用 Lavazza 方法确定阈值示意图

Lavazza 等人[93]从 BLR_{pess} 取一个交点 (用 x_p 表示), 再从 BLR_{opt} 取一个交点 (用 x_o 表示), 得出以下 x_p 和 x_o 的 4 种可能的组合策略: (I) $x_p=x_{pp}$ 和 $x_o=x_{op}$; (II) $x_p=x_{pp}$ 和 $x_o=x_{oo}$; (III) $x_p=x_{po}$ 和 $x_o=x_{op}$; (IV) $x_p=x_{po}$ 和 $x_o=x_{oo}$. 在给定 $X=x$ 的模块, 提供 4 种错误估计规则. (a) $x<x_p \wedge x<x_o$. 模块被估计为负例 (无错误). (b) $x<x_p \wedge x_o<x$. 第 1 个阈值预测模块为负例, 而第 2 个阈值预测模块为正例. 两者矛盾, 故模块被估计为错误性未知. (c) $x_p<x \wedge x<x_o$. 与 (b) 类似.

模块被估计为错误性未知. (d) $x_p<x \wedge x_o<x$. 模块被估计为正例 (有错误).

Lavazza 等人[93]将两个阈值 x_p 和 x_o 中的最小值称之为“软”阈值, 最大值称之为“硬”阈值. “软”阈值与“硬”阈值之间称为“灰色地带”. 通过规避风险的方法, 将没有充分证据的“灰色地带”模块估计为轻度易错的模块. 例如, 在软件开发期间, 通过使用常规方法开发人员会收到一条指南, 必须使用 RFC 小于等于 41 的度量值编写代码模块. 相反, 应用 Lavazza 方法的指南规定, 开发人员应该使用 RFC 小于等于 22 度量值 (“软”阈值) 开发代码模块, 必须使用 RFC 小于等于 48 度量值 (“硬”阈值) 开发代码模块.

3.4.6 基于分类树学习器算法 C5.0——处理度量之间的非正交关系

2018 年, Tsuda 等人[99]提出基于分类树学习器 C5.0 算法确定度量阈值的方法, 并认为该方法比 4 种传统方法 (分位数方法、Alves 方法、Bender 方法和 ROC 曲线方法) 有更准确的评估. 由于传统方法通常为每个度量正交地确定阈值, 这使得这些方法不能完全解释在给定背景 (开发语言和项目领域) 下各度量之间关系.

Tsuda 等人[99]认为考虑度量之间的非正交关系有助于对演化性缺陷倾向性在给定背景下的精确评估. Mäntylä 等人[163]提出缺陷的两种分类: 演化性缺陷和功能性缺陷, 并定义演化性缺陷为使代码不符合标准、更易出错或更难修改、扩展或理解的代码缺陷. 演化性缺陷在运行时是不可见的. 为此, 度量的标准和目标并不是很清晰. 而功能性缺陷会导致代码运行时出现可见故障. 修复功能性缺陷也相对明显 (如变量错误、内存管理、函数调用等). 通过上下文阈值 (contextual threshold) 的确定和应用, Tsuda 等人[99]构建精确的评估标准来检测易于演化的缺陷倾向代码文件. 其方法包括以下 6 步.

(1) 为给定项目背景下构建基准数据. 根据 Zhang 等人[164]研究, 一些常用度量的分布受背景因素影响, 影响最大的因素分别是应用程序领域、编程语言和变更数量.

(2) 定义一个如何在这些上下文因素中解释演化性的缺陷倾向基本模型. 目标问题度量 (goal-question-metric, GQM) 是定义解释模型的典型方法.

(3) 通过专家的手动检查来收集实际可演化性缺陷倾向性的数据.

(4) 通过静态分析衡量度量指标.

(5) 通过 C5.0 分类树学习器构建评估标准. 此步骤独立处理 N 个问题以构建 N 个分类树.

(6) 讨论标准的有效性. 如果不可接受, 则返回至步骤 (5).

Tsuda 等人[99]在 Komatsu 公司 C++语言嵌入式系统的数据集上, 构建了文件级别的评估标准. 目标问题度量 (GQM) 方法中的目标是评估源代码演变性缺陷倾向的严重程度. 问题有两个: 第 1 个问题为文件大小是否正确? 该问题的度量有 *FUN* (文件中函数定义数量) 和 *ELOC* (文件中的可执行代码行). 第 2 个问题是函数不是太复杂吗? 该问题关注控制流程的复杂程度. 其度量有嵌套深度度量 *DN* 和图复杂度量 *CC*. 由于 *DN* 和 *CC* 都是函数级别度量, Tsuda 等人分别用它们的均值和最大值来替代. 即 *AveDN*、*MaxDN*、*AveCC* 和 *MaxCC*.

实验结果表明: 问题 1 的分类树了解释 *FUN* 和 *ELOC* 之间的非正交关系. 即 *ELOC* 有两个阈值, 分别以 *FUN* 小于或等于 6 来区分, 而问题 2 的分类树仅使用 *MaxCC*. 即 C5.0 算法忽略了与问题 2 相关的另外 3 个度量.

3.5 小 结

本节介绍 OO 度量阈值确定的有监督学习方法. 介绍了 BLRT 模型、QRA 方法、ROC 方法和其他方法, 各类方法小结如下.

(1) 单变量逻辑回归阈值模型 (BLRT) 是首次将流行病学的阈值模型引入软件工程研究领域, 为后续 OO 度量阈值研究奠定了基础: 度量阈值确定过程包括阈值确定标准和阈值假设检验.

(2) 基于逻辑回归的定量风险评估方法 (QRA) 把阈值确定转化为 OO 度量值对应的模块发生缺陷概率值来确定, 这拓宽了 OO 度量阈值研究.

(3) ROC 方法可以在数据集不满足正态分布和存在类别不平衡情况下, 依然可以确定阈值.

除了这 3 种主要方法, 还存在其他方法确定阈值, 但只有 BLRT 模型中对阈值执行假设检验及 Shatnawi[123]等少数学者补充了假设检验, 其他方法都缺少必要的阈值检验过程.

4 无监督学习方法

表 6 列出各类无监督学习方法确定 OO 度量阈值的汇总信息, 包括方法名称、提出年份、模型主要公式和阈值确定标准、实验数据集、度量集、阈值检验和研究作者.

无监督学习方法确定 OO 度量阈值是在被度量代码模块缺陷信息未知的情况下, 根据 OO 度量数据特征, 如规模、分布和异常数据等, 确定 OO 度量在不同风险等级上的分段阈值. 如表 6 所示, Alves 排序法、Ferreira 方法、Vale 方法、中位数阈值方法和 Oliveira 方法均以度量值上设置百分位数方法来确定阈值. 这类方法具有一类共同特征: 将 OO 度量处理后, 在 0 至 100% 上设置若干分段阈值, 如常见的 70%、80% 和 90% 百分位数上对应的 OO 度量值作为分段阈值等. 此外, 还有对数转换法、幂律分布特征法、聚类方法、*SATT* 和贝叶斯层次模型等其他无监督学习确定阈值方法.

1996 年, Erni 等人[133]在服从正态分布的软件度量值域上设定阈值 T: $M1=\{Z|Z\leqslant T\}$和 $M2=\{Z|Z>T\}$. 把度量值分为两个子集, 分别称之为正常值和异常值. 通过度量均值 μ 和标准差 σ 计算出 $T_{min}=\mu-\sigma$ 和 $T_{max}=\mu+\sigma$, 并设定其为下限阈值和上限阈值. 当软件度量值越大越易出现问题时, 采用 T_{max} 与度量值进行比较; 反之, 采用 T_{min} 来比较.

2006 年, Shatnawi[147]在其博士学位论文中提出在有错误与无错误类的度量均值基础上确定度量阈值. 由于第 2 类错误的成本 (将有错误类别误分为无错误类别) 比第 1 类错误的成本 (将无错误类别误分为有错误类别) 的成本高, 通过降低第 2 类错误来确定阈值 $T=\mu_{no\text{-}error}+\theta\times(\mu_{error}-\mu_{no\text{-}error})$. 其中, $\mu_{no\text{-}error}$ 是没有错误的度量均值和 μ_{error} 是有错误的度量均值. θ 是用来降低第 2 类错误的变量, 其最小值为 0, 此时 $T=\mu_{no\text{-}error}$; 最大值为 1, 此时 $T=\mu_{error}$. 较小的 θ 值, 会降低第 2 类错误. 因此, θ 值 ($0<\theta<0.5$) 越小越好. 然而, Shatnawi[147]建议 θ 值取 0.5, 并建议在后续研究中寻找一个最优 θ 值.

2006 年, Lanza 等人[4]提出基于统计的阈值和实际意义的阈值. 基于统计的 4 种阈值分别是低值阈值 (AVG-STDEV)、均值阈值 (AVG)、高值阈值 (AVG+STDEV) 和极高阈值 [(AVG+STDEV)×1.5], 其中, AVG 表示均值, STDEV 表示标准差. 基于实际意义的阈值是被普遍接受和容易理解的, 包括两种情况: 第 1 种是常用的分数阈值, 如 1/4、1/3 和 1/2 等. 第 2 种是具有普遍接受意义的阈值, 如与公认语义相关联的阈值, 用 0 表示没有 (none)、用 2–5 表示少数或数个 (few/several) 及 7–8 表示短期记忆容量 (short memory capacity).

除了 Erni 等人[133]、Lanza 等人[4]和 Shatnawi[147]早期关于无监督学习方法确定阈值, 自 2010 年以来, 这类方法由于缺少类模块的缺陷标签信息而使得 OO 度量阈值确定变得相对简便, 不需要通过复杂建模、设置参数、模型检验等步骤. 这在一定程度上也促进此类方法的进一步发展.

4.1 Alves 排序法

2010 年, Alves 等人[79]认为阈值确定方法需要满足数据驱动、稳健性和实用性 3 个需求: (1) 不应由专家意见驱动, 而应由一组代表性系统的度量数据驱动; (2) 应尊重度量的统计属性, 如度量规模和分布, 并且应能够抵御 (识别) 度量值和系统规模的异常值; (3) 应可重复、透明且易于执行. Alves 等人依据这 3 个需求将 100 个私有和开源的软件系统中度量数据汇总和整合, 在各系统之间差异性和源代码量的合理百分比基础上确定各度量阈值. 其做法如下.

设 X 是某一具体 OO 度量, 并用“模块”表示软件系统的任何片段, 如子系统、组件、类、方法等. Lavazza 等人[138]通过数学表达式阐述 Alves 排序法的 X 阈值确定方法, 包括以下 6 个步骤.

(1) 度量提取. 从基准数据的软件系统中提取 OO 度量 X, 包括 OO 度量在每个系统中每个模块上的数值和每个模块代码数 LOC.

(2) 权重比率计算. 在给定软件系统 j 中每个模块 i 上设定权重 $w_{i,j}$, 其值等于模块 i 的 LOC 值 $loc_{i,j}$ 与项目 j 的 LOC 值总和$\left(LOC_j=\sum_{i\in j}loc_{i,j}\right)$之比: $w_{i,j}=loc_{i,j}/LOC_j$. 各系统内所有模块权重总和为 100%.

(3) 模块整合. 计算系统 j 中 X 每个数值的权重 $w_j(x)$. $w_j(x)=\sum_{\forall i \text{ such that } X_i=x}w_{i,j}=\sum_{\forall i \text{ such that } X_i=x}loc_{i,j}/LOC_j$. $w_j(x)$ 值等于系统 j 中 $X=x$ 所有模块的 $w_{i,j}$ 之和. 这等价于计算一个 X 各个数值权重总和为 100% 的加权直方图.

表 6 OO 度量阈值无监督学习方法 (模型) 汇总

方法名称	年份	阈值确定标准	实验数据集	度量集	阈值检验	研究作者
Erni方法	1996	在服从正态分布的假设下, 度量X值的中心区域$[\mu-\sigma, \mu+\sigma]$为“正常”取值	Semlib的3个版本系统	复杂度、耦合和内聚度量	无	Erni等人[133]
帕累托启发式方法	1998	选择度量值分布中从低值开始第80分位数作为度量阈值标准	TPM、FIS和SLB信息系统	WMC、DIT、NOC、CBO、RFC、LCOM	无	Chidamber等人[104]
均值系数方法	2006	$T=\mu_{\text{no-error}}+\theta*(\mu_{\text{error}}-\mu_{\text{no-error}})$, $\mu_{\text{no-error}}$和μ_{error}分别是没有和有错误的度量均值, $0<\theta<0.5$	Eclipse 2.0、2.1和3.0	LOC、DIT、CBO和RFC等12个	有	Shatnawi [147]
Lanza方法	2006	正态分布数据: 低值阈值$\mu-\sigma$、高值阈值$\mu+\sigma$和极高阈值$1.5(\mu+\sigma)$, μ为均值和σ为标准差	45个Java和37个C++项目	AMW、LOC和CYCLO等度量	无	Lanza等人[4]
Alves排序法	2010	在所有系统的基准数据集上, 根据70%、80%和90%的百分比来确定阈值	100个Java和C#项目	McCabe和LOC度量	无	Alves等人[79]
	2016	将确定的阈值分别与单个代码文件中缺陷和高风险文件组中缺陷密度相联系.	4 575个开源项目数据集	代码规模和复杂度的度量	无	Yamashita等人[100]
	2018	重点关注两个分段阈值(即90%和95%)	会计和通信等15类软件系统	规模、复杂度和两个继承性度量	无	Mori 等人[142]
	2018	同上	同上	LOC、NOA等8个	无	Mori[146]
	2020	在代码量权重上取70%、80%和90%作为分段阈值标准	1 489个项目数据集	11种Test Smell度量	无	Spadini等人[116]
Ferreira方法	2012	泊松分布取均值为度量的典型值; 威布尔(Weibull)分布则设置好、常规和差3个范围	SourceForge上40个Java项目	LCOM、Ca、NPM、DIT、COF和NPF	无	Ferreira等人[109]
	2015	设置70%和90%两个百分位数为阈值标准	Eclipse Java的111个系统	DIT、LCOM和MLOC等23个	无	Filó等人[132]
	2017	EasyFit工具对OO度量数据进行拟合. 若为重尾分布, 则得出好、常规和差3类分段阈值; 反之, 则将该度量的均值确定为阈值	SourceForge和F-Droid上的Android项目	NOM、RFC、DIT、NOC和CBO	无	Stojkovski[148]
	2018	若为幂律分布, 则在第90分位数上确定阈值; 反之, 算术平均值加标准差的方法确定阈值	SourceForge上Java、C++和C#各100个系统	AMS、ACC、NS、CBO和LCOM等8个度量	无	Beranič等人[143]
	2019	设置70%、80%和90%两个百分位数为阈值标准	Java、C++、C＃和Python各100个系统	CountLineCode和AvgLineCode等9个	无	Tina等人[145]
幂律分布方法	2013	最小化$D=\max\|S(x)-P(x)\|$得出$x_{\min}$最佳估计值. $S(x)$和$P(x)$分别是包括$x_{\min}$观察值数据和在$x\geqslant x_{\min}$区域最佳拟合模型的累积分布函数	5个Java开放源代码系统	CBO、NOC、DIT、RFC、WMC等8个	有	Shatnawi等人[136]
Oliveira方法	2014	RTTool工具: 至少p%的系统实体(class)不应超过阈值k	Tempero库中106个系统	FAN-OUT、LOC、NOA和NOM	无	Oliveira等人[94]
	2014	在给定的语料库上, 至少p%的系统实体(class)不应超过阈值k, 即$M\leqslant k$	Qualitas Corpus中106个系统	NOM、RFC和WMC等8个度量	无	Oliveira等人[134]
	2015	至少p%的系统实体(class)不应超过阈值k	Pharo/Smalltalk上79个语料库	NOM、LCOM和LOC等7个度量	无	Oliveira等人[95]
Vale方法	2014	选择度量值分布中从低值开始第80分位数作为度量阈值标准	Jetty和Weka等5个项目	NOA+NOM 和 CBO	无	Foucault等人[82]
	2015	通过与整个基准模块相关的X值计算出分位数来确定X的阈值	33个软件产品线的数据集	LOC、CBO、WMC和NCR	无	Vale等人[80]
	2015	比较Alves排序法、Ferreira方法和Oliveira方法	33个软件产品线的数据集	LOC、CBO、WMC和NCR	无	Vale等人[135]

表 6 OO 度量阈值无监督学习方法 (模型) 汇总 (续)

方法名称	年份	阈值确定标准	实验数据集	度量集	阈值检验	研究作者
	2019	将总体度量值90%和95%设置为上限阈值标准; 3%和15%设置为下限阈值标准	33个软件产品线的数据集	LOC、CBO、WMC和NCR	无	Vale等人[114]
基于P-B-d定理的模型	2014	Hill图法和平均超出量函数法	Camel和Xalan等9个系统	SLOC、CBO和RFC等5个度量	有	王志坚[13]
中位数阈值方法	2015	选择每个度量值域上的中位数作为该度量阈值	NetGene/ReLin上7个项目	Safe数据集中26个度量	无	Nam等人[91]
对数转换法	2015	通过自然对数变换后, 应用Erni方法确定阈值, 再使用指数函数将T'的值取反得出阈值	PROMISE中11个系统	6个CK度量	有	Shatnawi等人[110]
	2017	同上	PROMISE数据集	6个CK度量	无	Gupta等人[140]
	2018	同上	PROMISE库中10个系统	WMC、CBO和RFC	无	Shatnawi [144]
	2018	转换前检查度量数据是否呈正态分布	JHotdraw-7.5.1	CC、LSCC、SCOM和SCC	无	Zhou等人[2]
Lanza方法	2015	用25%、50%、75%分位数确定度量的低、中、高阈值	*QualitasCorpus* 中74个系统	WMC、NOAM和LOC等11个	无	Fontana等人[137]
TDTool工具法	2016	Alves排序法、Ferreira方法、Oliveira方法和Vale方法	33个软件产品线的数据集	LOC、CBO、WMC和NCR	无	Veado等人[98]
*SATT*方法	2016	在Alves方法中增加提取架构角色信息确定阈值	120个SpringMVC和301个Android系统	McCabe度量	有	Aniche等人[101]
3种方法的比较	2016	比较ROC曲线方法、VARL方法和Alves排序法	PROMISE中11个项目和一个Eclipse项目	SLOC、CBO、RFC、WMC和LCOM等7个	无	Boucher等人[7]
3种方法的比较	2016	比较Erni、Alves和Vale等人的方法	PROMISE中berek和zuzel等16个项目	WMC、DIT、CBO、RFC和CA等11个	无	Lavazza等人[138]
聚类方法	2016	基于灰色关联等级的聚类算法	PROMISE数据集	6个CK度量	无	Kaur等人[139]
	2017	K-means聚类算法	ArgoUML0.24、0.28、0.30和0.34系统	LCOM5、CBO、CBOI、NII、NOI和RFC	无	Meena等人[141]
3种阈值策略方法	2018	用Pysmell工具实现3种阈值策略: 基于经验的阈值、基于统计的阈值和基于训练机的阈值	ansible、boto和django等9个Python项目	LPL、LM和LSC等10个代码异味度量	无	陈芝菲[14]
Lima方法	2018	度量3个频率阈值: 高频值 (0至90分位数)、中频值 (90至95分位数)和低频值 (95至99分位数)	25个GitHub项目中24 947个Java类的数据	AA、LOCAD、ANL、AED、AC、UAC和ASC度量	无	Lima等人[75]
贝叶斯层次建模方法	2018	OO度量平均log值得分的70%、80%和90%设置为阈值	120个Spring MVC数据	CBO度量	无	Ernst等人[117]
分位数方法	2020	[最小值, 第一分位数]、[第一分位数, 中位数]、[中位数, 第三分位数]和[第三分位数, 最大值]这4个范围	1 737个公共APIs数据	APL、APO、BRC和WSIC等10个度量	无	Bogner等人[76]

注: 度量集一列, 尽可能将每篇文献中涉及到的度量(在括号中显示)列出, 至少列出3个或5个度量的情况

(4) 系统整合. 先对 $w_j(x)$ 通过基准数据中系统数量 n 进行归一化处理: $w_j(x)/n$. 然后, 对所有系统上归一化的 $w_j(x)$ 求和后, 得出基准数据中 $X=x$ 的权重 $w(x)=\sum_{\forall j\in benchmark} w_j(x)/n$. 归一化可确保直方图所有数值权重总和保持 100%, 而整合是 OO 度量各个数值的权重之和.

(5) 权重比率整合. 如图 11 所示, 对基准数据中所有系统的 OO 度量 X 各个数值按递增顺序进行排序 (y 轴), 并按权重为 1%, 2%, …,100% 顺序 (x 轴) 排列 OO 度量 X 值. 这等价于计算密度函数 (density function).

(6) 阈值确定. 通过在权重 $w(x)$ 的数据分布上设定分数 F, 并定义最小值 $\bar{x}$ 满足 $\sum_{X\in\min(X)...\bar{x}} w(x)\geqslant F$, 得出 X 的一个分段阈值 $\bar{x}$. 当模块的度量值 $x\leqslant\bar{x}$ 时被认为是“好”模块, 反之被认为是“坏”模块. 当 F 值分别设定为 70%、80% 和 90% 作为分段阈值标准时, 可计算出对应的 $\bar{x}$ 作为 X 的分段阈值.

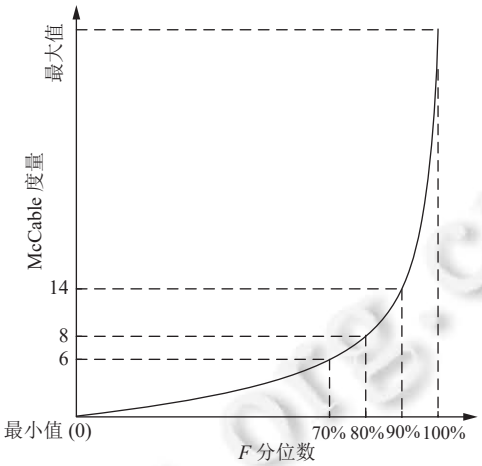


图 11 McCabe 度量应用 Alves 排序法生成分段阈值示意图

Alves 等人[79]采用荷兰的 SIG 公司的阈值判断标准: OO 度量各数值在基准数据中代码量权重上取 70%、80% 和 90% 作为分段阈值标准. 在这 3 个分段阈值上的代码 (模块) 可以分别在长期、中期和短期内修复. 此外, 这 3 个百分比还可以将软件质量分为 4 个等级: 低风险 (0–70%)、中风险 (70%–80%)、高风险 (80%–90%) 和极高风险 (>90%). 如图 11 所示, 在 100 个 Java 和 C#系统上应用该方法确定 McCabe 度量的 3 个分段阈值为 6、8 和 14.

在 Alves 排序法的基础上, Lavazza 等人[138]、Boucher 等人[7,121]、Yamashita 等人[100]、Mori 等人[142,146]和 Spadini 等人[116]通过选择在基准数据集中 OO 度量各个数值不同的百分比权重, 来确定 OO 度量不同分段阈值标准.

2016 年, Lavazza 等人[138]认为 Alves 排序法对较小系统中模块在最终的 (如上述步骤 (5) 中) 权重分布中存在不合理的比例影响. 即放大了较小系统中较小模块的比例在所有系统 (基准数据) 中的影响. 例如, 假定基准数据中只包括软件系统 A 和 B. A 的 LOC 值比 B 大, 并且 B 中最大的模块都比 A 中最小模块的 LOC 值要小. 若 B 中最大模块在 B 中占 60%, 则通过上述步骤 5 的归一化整合后, 该最大模块中 X 度量值所占的比例为 30%. 该比例有可能比 A 中所有的模块在最终所占的权重比例都要大 (A 中取 X 的某个度量值的模块数较少). 因此, 用分数 F 代表 LOC 的相同比例并不一定是正确的. 即在 F=90% 的情况下, Lavazza[138]认为通常无法将包含 90%的 LOC 的一组模块评估为“好”.

2016 年, Boucher 等人[7,121]在 12 个公开数据集 (11 个 PROMISE 项目和一个 Eclipse 项目) 上比较 ROC 曲线方法、VARL 方法和 Alves 排序法. 通过比较缺陷倾向模型的预测性能指标后发现, ROC 曲线方法最好, Alves 排序法也是一个较好的选择. 事实上, Alves 排序法比 ROC 曲线方法优越之处在于, Alves 排序法是一个无监督方法并且是在无缺陷标签信息下通过度量数据直接确定阈值.

2016 年, Yamashita 等人[100]在 4575 个开源项目数据集上基于代码规模 (代码和方法数量) 和复杂度 (圈复杂度和耦合接口模块) 应用 Alves 排序法确定圈复杂度 CC 度量的阈值. 将确定的阈值分别与单个代码文件中缺陷和高风险文件组中缺陷密度相联系. 研究发现: 除少数例外, 在极高风险文件组中的单个代码文件存在缺陷的可能性更高. 然而, 在极高风险文件组中, 其缺陷密度始终较低. 与规模较小和较不复杂的文件相比, 同样数量的代码在规模较大或复杂文件中存在较少的缺陷. 一般来说, 较小规模的代码会产生更少的缺陷. 因此, Yamashita 等人[100]得出结论, 基于规模和复杂度的风险阈值须谨慎使用.

2018 年, Mori 等人[142,146]应用 Alves 排序法确定各领域内的度量阈值. 在方法最后一步中, 重点关注两个分段阈值(即 90% 和 95%). 这些领域软件包括会计软件、游戏软件和通信软件等 15 类软件系统.

2020 年, Spadini 等人[116]在 1 489 个项目数据集上应用 Alves 排序法确定 9 种测试代码异味 (test smell) 的严重性阈值 (severity threshold). 同样, 在基准数据中代码量权重上取 70%、80% 和 90% 作为分段阈值标准, 并将代码库中的测试代码异味分类为 4 个不同的类别: 当权重比率在 [0, 70) 区间上时为未发现或低于阈值的测试代码异味; 在 [70, 80) 区间上时为用于长期重构的中等严重性测试代码异味; 在 [80, 90) 区间上时为用于短期重构的高严重性测试代码异味; 在 [95, 104] 区间上时为用于即时重构的极高严重性测试代码异味.

综上, Alves 排序法虽然考虑代码块行数权重对度量值分布的影响, 但也存在放大较小系统中较小模块的权重影响. 而且, 对 70%、80% 和 90% 的 F 值作为分段阈值标准, 也存在主观因素影响和缺少必要的阈值检验.

4.2 Ferreira 方法

2012 年, Ferreira 等人[109]从 SourceForge 网站上搜集 40 个开源 Java 项目作为数据集来源项目. 从内聚、耦合和信息隐藏等面向对象程序特征提取 6 个 OO 度量组成的度量集: LCOM、DIT、COF[39]、Ca、NPM 和 NPF.

实验发现, 6 个度量均服从幂律分布: 分布函数中随机变量 X 取值 x 的概率与 x 的负幂成比例: $P(X=x)\propto cx^{-k}$.

幂律分布是一种重尾分布, 其特征是随机变量取大值频率非常低, 而取小值的频率很高; 均值不具有代表性, 且没有一个数值可以作为随机变量的典型值. Ferreira 等人[109]确定阈值方法主要包括数据拟合和数据分析. 具体做如下.

首先, 应用 EasyFit 工具将 OO 度量数据拟合为各种可能的概率分布, 如伯努利分布、二项式分布、均匀分布、几何分布、超几何分布、对数分布、泊松分布、正态分布、学生 t 分布、卡方分布、指数分布、对数正态分布、帕累托分布和威布尔分布.

每一种概率分布有两个主要函数: 概率密度函数 $f(x)$ 表示随机变量取值 x 的概率; 累积分布函数 $F(x)$ 表示随机变量取小于或等于 x 值的概率. 实验结果发现, OO 度量数据对泊松分布和威布尔分布拟合较好. 泊松分布的概率密度函数 $f_p(x)$ 和累积分布函数 $F_p(x)$ 如公式 (19) 所示, λ 参数为随机变量 X 的均值.

$$f_p(x)=P(X=x)=\frac{\mathrm{e}^{-\lambda}\cdot\lambda^x}{x!}\ 和\ F_p(x)=P(X\leqslant x_0)=\sum_{x=0}^{x=x_0}\frac{\mathrm{e}^{-\lambda}\cdot\lambda^x}{x!} \tag{19}$$

威布尔分布的概率密度函数 $f_w(x)$ 和累积分布函数 $F_w(x)$ 如公式 (20) 所示. 其中, α 参数为形状参数, 当 α 参数小于 1 时, 威布尔分布是重尾分布; β 参数为比例参数. 当 β 参数增加时, 曲线高度减小且有拉伸曲线作用.

$$f_w(x)=P(X=x)=\frac{\alpha}{\beta}\left(\frac{x}{\beta}\right)^{\alpha-1}\mathrm{e}^{-(x/\beta)^\alpha}\ 和\ F_w(x)=P(X\leqslant x)=1-\mathrm{e}^{-(x/\beta)^\alpha}(\alpha>0,\beta>0) \tag{20}$$

然后, 对每个 OO 度量生成两张图: 散点图和双对数刻度图 (log-log scale). 散点图用于显示 OO 度量每个取值的频率; 双对数刻度图用于观察其分布是否为幂律分布. 若度量数据的概率分布有一个代表性均值, 如泊松分布, 则该均值为度量的典型值; 若度量数据不存在代表性均值, 如威布尔分布, 则在度量值域上设置 3 个范围: 好 (good) 指度量标准中最常见的 (高频) 值; 常规 (regular) 指度量值不太频繁但也不会有很低的频率; 差 (bad) 指度量值出现频率很低. 通过可视化分析后, 得出每个 OO 度量的阈值参考值. Ferreira 等人认为其实验结果得到的 6 个 OO 度量阈值可以区分出违反设计原则的类与设计良好的类.

2015 年, Filó等人[132]对 Ferreira 方法提出了两方面改进: 修改分段阈值名称为普遍 (common/good)、偶然 (casual/regular)、不普遍 (uncommon/bad); 基于图形可视化分析和度量值频率概念, 设置 70% 和 90% 两个百分位数为阈值标准进而确定其阈值, 而不是直接设置度量值为阈值. 这样, 可以通过 R 语言代码复现阈值.

2017 年, Stojkovski[148]在其硕士学位毕业论文中应用 Ferreira 方法在安卓应用程序项目上确定度量阈值. 应用 EasyFit 工具对 OO 度量数据进行拟合. 若为重尾分布, 则应用频率图、频率表和可视化分析的方法得出好、常规和差 3 类分段阈值; 若不服从重尾分布时, 则将该度量的均值确定为阈值.

2018 年, Beranič等人[143]使用 Ferreira 方法在 SourceForge 上搜集 Java、C++和 C#这 3 种语言的开源软件各

100 个软件产品数据, 确定 SLOC、AMS、NOM、WMC、ACC、NS、CBO 和 LCOM 度量阈值. 实验结果发现: 前面 7 个度量服从幂律分布, 可以在第 90 分位数上确定阈值. 然而, LCOM 度量并不服从幂律分布而是正态分布. Beranič 等人定义 LCOM 度量值在 0 到 100 之间来表示该类内缺乏内聚性的百分比, 并用算术平均值加标准差的方法确定 LCOM 度量的阈值.

2019 年, Tina 等人[145]应用 Ferreira 方法确定 4 种语言 (Java, C++, C＃和 Python) 项目上各度量阈值. 通过比较后发现, 阈值在不同的编程语言之间是不同的.

综上, 由于 Ferreira 方法对 OO 度量数据拟合出各类分布后, 采用可视化分析方法来确定阈值, 使得该方法存在人为影响因素, 且不易推广.

4.3 Vale 方法

2015 年, Vale 等人[80]在 33 个软件产品线组成的基准数据集上确定度量阈值, 通过召回率和精度指标评估上帝类 (God class) 和惰性类 (lazy class) 两种代码异味检测策略. 软件产品线 (SPL) 是共享通用且可变组件的一组软件系统. 软件度量阈值可以直接反映 SPL 组件测量过程的有效性. 上帝类是指软件系统中“知道”或“做”得太多的类. 它通过 NCR 来度量, 表明软件组件正在积聚许多其他组件功能的实现. 惰性类是指软件系统中“知道”或“做”得太少的类.

Vale 等人[80]在确定阈值前提出 7 点需求: (1) 一组有代表性的基准数据集; (2) 与基准数据中实体数量有很强的依赖性; (3) 与基准数据中软件系统数量有弱依赖性; (4) 应计算上限阈值和下限阈值; (5) 以分步形式推导阈值; (6) 重视度量数据的统计特性; (7) 应是系统的、可重复、透明和直接执行的方法. 在确定度量阈值之前, Vale 等人[80]先调查 OO 度量与被度量模块的 LOC 度量之间相关性, 确定该相关性是否影响阈值计算. 然后, 检测各 OO 度量是否服从威布尔分布 (重尾分布). 最后, 再确定度量阈值. 该方法包括以下 5 个步骤.

(1) 度量提取. 从基准数据每个软件系统中的实体上提取度量数据.

(2) 权重比率计算. 计算每个实体在基准数据所有实体上的权重. 每个实体有相同的权重且所有实体权重总和为 100%. 例如, 基准数据中有 10 000 个实体, 则每个实体的权重为 0.01%.

(3) 按升序排序. 将度量每个值按升序排序, 并按 1%, 2%,…,100% 权重比例叠加至最大值. 这与计算密度函数相似. 在密度函数图中, x 轴表示权重比例, 其值域为 0 至 100%; y 轴表示度量值, 其值域为度量值域.

(4) 实体整合. 整合度量每个值上的权重. 例如, 若基准数据中 WMC 度量在 4 个实体上取值为 4, 且每个实体占 0.01%, 则基准数据中 WMC 度量值为 4 的权重为 0.04%. WMC 度量所有值的权重总和为 100%.

(5) 确定阈值. 在基准数据中度量值整体权重上设置 3%、15%、90% 和 95% 分别作为分段阈值标准, 得出 5 种风险水平: 极低值 (0–3%); 低值 (3%–15%); 中等值 (15%–90%); 高值 (90%–95%); 极高值 (95%–100%).

Vale 等人[80]应用 VSD[165]工具搜集基于软件产品线的基准数据, 通过该方法确定 LOC、CBO、WMC 和 NCR 度量的 4 个分段阈值. 将 33 个软件产品线分为 3 个基准数据分别确定度量阈值. Vale 等人[80]研究发现度量分段阈值与基准数据中实体 (如 class) 数量相关. 2019 年, Vale 等人[114]对自己的方法略做调整. 在第 (5) 步的确定阈值过程中, 将总体度量值的 90% 和 95% 设置为上限阈值标准、3％和 15％设置为下限阈值标准.

与 Vale 等人[80,114]方法类似, Foucault 等人[82]、Fontana 等人[137]、Lima 等人[75]和 Bogner 等人[76]直接在度量值的不同分位数上确定阈值.

2014 年, Foucault 等人[82]参考 Chidamber 等人[104]的方法, 选择度量值分布中从低值开始第 80 分位数作为度量阈值标准. 1998 年, Chidamber 等人[104]依据常见的帕累托 (80/20) 启发式方法来定义度量的高值临界值不低于第 80 分位数. Foucault 等人确定度量分位数阈值方法包括 3 步: (1) 采用双重抽样方法, 从软件项目中随机提取各类背景的实体; (2) 计算抽样的实体软件度量值; (3) 通过 bootstrap 统计方法计算度量阈值的置信区间. 双重抽样指: 先从一组给定的已知项目中随机选择具有相同背景 (如同为 Java 语言) 的一组软件项目, 再创建一个包含选定项目中所有可用实体的有限数量总体. 最后随机选择此有限总体中的实体来构建样本.

2015 年, Fontana 等人[137]根据 Lanza 等人[4]的阈值确定方法分别用 25%、50%、75% 分位数确定度量的低、

中、高阈值. 该方法排除用比率表示的度量, 尤其是取值范围在区间 [0, 1] 内 (如 TCC) 的度量. 该方法主要包括 3 个步骤: (1) 度量计算. 即计算选定系统上所有类和方法的度量值; (2) 汇总度量的分布分析. 计算每个度量分布的分位数函数; (3) 阈值确定. 在分位数函数的 25%、50%、75% 各点上设置低、中、高阈值.

2018 年, Lima 等人[75]收集了 25 个项目中 24 947 个 Java 类模块的度量数据. 由于这些度量数据服从指数分布, 所以各个度量的均值和中位数没有实际意义, 不能作为阈值参考值, 并根据度量频率分布, 将软件度量值区间分为: 高频值 (very frequent)、中频值 (frequent) 和低频值 (less frequent). Lima 等人[75]使用第 90 分位数作为参考点: 0 至 90 分位数为高频值、90 至 95 分位数为中频值、95 至 99 分位数为低频值.

2020 年, Bogner 等人[76]提出模块化且可扩展的 RAMA (RESTful API metric analyzer) 方法计算可维护软件度量集. 在每个度量分位数分布上定义分段阈值, 使得度量值落入各自取值范围. 这 4 个范围依次是 [最小值, 第一分位数]、[第一分位数, 中位数]、[中位数, 第三分位数] 和 [第三分位数, 最大值]. 每个范围分别标记为绿色、黄色、橙色和红色 (从好到差的排序). 如果软件系统度量结果在红色或橙色区间内, 则建议改善相关的设计属性.

综上, 由于 Vale 方法确定度量阈值过程中未考虑被度量模块的代码量权重, 使得阈值受被度量模块代码量影响较小, 但也带来未充分考虑代码量对度量阈值影响的弊端.

4.4 中位数阈值方法 (median threshold)

2015 年, Nam 等人[91]提出在无标签数据集上缺陷预测的方法 CLA 和 CLAMI. 这两个方法的核心在于通过度量值的大小判断来对被度量的实例标记标签 (buggy 或 clean). Nam 等人认为典型缺陷预测数据集中缺陷倾向趋势是较高的复杂度会导致更多的缺陷倾向. 由于 OO 度量一般是测量源代码和开发过程的复杂度, 所以有缺陷的实例比没有缺陷的实例会有更高 OO 度量值的趋势.

Nam 等人[91]比较 11 个不同分位数候选阈值, 即 P10 (第 10 百分位数)、P20、P25 (第 1 个四分位数)、P30、P40、P50 (中位数)、P60、P70、P75 (第 3 个四分位数)、P80 和 P90 后发现, P50 的 F 值和 AUC 值排名最好. 因此, Nam 等人[91]选择每个度量值域上的中位数作为该度量阈值, 并将中位数阈值应用于缺陷预测.

CLA 和 CLAMI 方法的前两步分别为 (1) 将实例进行聚类 (clustering instances) 和 (2) 在聚类的实例上注标签 (labeling instances). CLAMI 还包括两个步骤 (3) 度量选择 (metric selection) 和 (4) 实例选择 (instance selection). 如图 12 所示, 每一步骤详细阐述如下.

图 12 CLA 和 CLAMI 方法应用中位数阈值的缺陷预测示意图

(1) 将实例进行聚类. X_1– X_7 为无标签数据集上 7 个度量; 实例 A–G 为数据集中 7 个实例. M_1 为度量 X_1 在 {$\mathbf{a}_1$, b_1, $\mathbf{c}_1$, d_1, e_1, f_1, g_1}上的中位数 (阈值), 其中 $\mathbf{a}_1$ 和 $\mathbf{c}_1$ 用粗体表示其值大于 M_1. 其他 6 个度量也类似用粗体表示超过其中位数阈值. K 为某一实例中超过中位数阈值的度量数量, 如实例 A 中, 有 X_1, X_3, X_7 这 3 个度量其对应度量值 $\mathbf{a}_1$,$\mathbf{a}_3$, $\mathbf{a}_7$ 大于中位数阈值. 根据 K=4, 3, 2 和 0, 可以将 7 个实例分为 4 类.

(2) 在聚类的实例上注标签. K=4 的类有实例 C; K=3 的类有实例 A 和 E; K=2 的类有实例 B、D 和 F; K=0 的类有实例 G. 将这 4 类实例分为两组, 并将上半组的实例标为有缺陷 (buggy) 的实例, 其他的为无缺陷 (clean) 实

例. 即将 K=4 的实例 C 和 K=3 的实例 A、E 标注为 buggy (粗体), 其他的实例标注为 clean.

(3) 度量选择. 在两类标签基础上, 计算每个度量的扰动分数 (metric violation scores, MVS). 由于缺陷预测模型质量高度依赖于度量质量, 提供有用信息的度量已被广泛应用于各类缺陷预测. CLAMI 方法通过去除与缺陷预测趋势不一致的度量以达到缺陷数据的缺陷倾向趋势受到扰动最小. 每个度量的扰动分数为: MVS_i=C_i/F_i, 其中, C_i 为第 i 个度量发生扰动的数量, F_i 为第 i 个度量所有度量值的数量. 图 12 中度量扰动分数行显示了 7 个度量的扰动分数, 且 X_1 和 X_4 度量具有最小的 MVS 值 (1/7). Nam 等人[91]选择 X_1 和 X_4 这两个具有最小的 MVS 值的度量作为训练集数据.

(4) 实例选择. 在第 (3) 步中得出的 X_1 和 X_4 度量中, 实例 A 的 X_4 值和实例 E 的 X_1 值还存在扰动项, 故将实例 A 和 E 去除后, 形成最后的训练集数据. 经过实例选择后, 若训练集中未见有 buggy 标签的数据, Nam 等人[91]在第 (3) 步中最小的 MVS 值上再增加一个仅次于该值的度量, 直至在产生训练集中 buggy 标签和 clean 标签数据都出现为止.

Nam 等人[91]从 NetGene[166]和 ReLink[167]两组数据集上搜集 7 个项目的度量数据集. 随机将该数据集分成两等份, 先将前 50% 和后 50% 分别作为训练集和测试集数据. 应用有监督学习方法在训练集上生成有监督模型, 再在测试集上检测有监督学习方法的预测性能, 然后将前 50% 和后 50% 分别作为测试集与训练集数据, 重复相同工作. CLA 和 CLAMI 方法作为无监督学习方法只在上述两次实验中的测试集上进行训练并缺陷预测. Nam 等人[91]重复这种随机将数据集分成两等份 500 次, 执行 1 000 次缺陷预测并报告其平均值. 实验结果发现, CLA 和 CLAMI 方法不需要最初标记的实例, 但在召回率、F 值和 AUC 性能指标上可达到与大多数有监督学习方法相当的预测性能.

4.5 Oliveira 方法

2014 年, Oliveira 等人[134]提出相对阈值 (relative threshold) 概念, 并提出确定相对阈值时需要满足的两个条件: 在类层次上 OO 度量服从重尾分布; 度量值较低的类比度量值较高的类更可取. 事实上, 大多数代码度量值服从重尾分布[168]且与被度量的代码块发生缺陷倾向正相关. 由于复杂的需求、性能的优化和机器自动生成代码等原因, 使得源代码实体在大多数软件系统中不遵循度量阈值. 例如, 在耦合性度量中, Taube 等人[169]认为高耦合永远不会从软件设计中完全消除, 某种程度的高耦合可能是相当合理的. 受这些发现的启发, Oliveira 等人[95,134]提出一种基于语料库相对阈值 [p, k], 其概念是至少 p% 的系统实体不应超过阈值 k, 即 M≤k. 其中, M 为给定的代码度量.k 是上限, p 是应遵循该上限的最小实体百分比. 该相对阈值可容忍 (100−p)% 的类 M>k. 但这 (100−p)% 的类并不代表“理想”的类. 只是由于度量服从重尾分布, 允许这部分类的存在.

事实上, 相对阈值表示在给定的语料库中, 某一百分比 p% 的实体不应超过某个阈值 k. 每个阈值有一个限定的百分比, 而通常意义上所说的阈值是指整个实体上的绝对阈值. Oliveira 等人认为绝对阈值是相对阈值的一种补充, 即 p% 取其上限值 100%. 例如, “85% 的方法应具有 McCabe 度量值小于等于 14”, 该相对阈值 [85%, 14] 表示当高风险方法占系统方法总数的 15% 以上时, 可能会影响系统的质量. 公式 (21)–公式 (24) 展示了计算某个 OO 度量 M 相对阈值中的 p 和 k 两个参数的过程.

$$ComplianceRate[p,k]=\frac{|\{S\in Corpus|p\%\text{ of the class }in\ S\text{ have }M\leqslant k\}|}{|Corpus|} \tag{21}$$

$$penalty_1[p,k]=\begin{cases}\dfrac{Min-ComplianceRate[p,k]}{Min}, & \text{if } ComplianceRate[p,k]<Min\\ 0, & \text{otherwise}\end{cases} \tag{22}$$

$$penalty_2[k]=\begin{cases}\dfrac{k-MedianTail}{MedianTail}, & \text{if } k>MedianTail\\ 0, & \text{otherwise}\end{cases} \tag{23}$$

$$ComplianceRatePenalty[p,k]=penalty_1[p,k]+penalty_2[k] \tag{24}$$

公式 (21) 中的函数 $ComplianceRate[p, k]$ 返回语料库中满足由 [p, k] 定义相对阈值的系统百分比数. 公式 (24) 中的 $ComplianceRatePenalty$ 是 $Penalty_1$ 和 $Penalty_2$ 之和. 若 $ComplianceRate[p, k]$ 小于 Min, 则 $Penalty_1$ 表示 $ComplianceRate[p, k]$ 受到其与 Min(%) 距离成正比的惩罚. 同样, 若阈值 k 大于语料库中每个系统中 OO 度量值

M 的第 *MedianTail* 分位数 (每个系统 S 中 *M* 值第 *Tail* 位数数组的中位数), 则 $Penalty_2$ 表示 *ComplianceRate*[*p*, *k*] 受到 *k* 与 *MedianTail* 距离成正比的惩罚. 通过最小化 *ComplianceRatePenalty* 获得相对阈值 [*p*, *k*]. 当出现 *ComplianceRatePenalty* 相等 (ties) 时, 先选择 *p* 最高的相对阈值, 然后选择 *k* 最低的相对阈值.

2015 年, Oliveira 等人[95]在 79 个 Pharo/Smalltalk 应用程序的语料库上确定 NOM、LOC、FAN-OUT、RFC、WMC、PUBA/NOA 和 LCOM 度量的相对阈值, 并由 5 位专家和 25 位开发人员进行验证. 实验结果表明, 专家认为高质量的应用程序重视度量相对阈值; 开发人员通常很难指出设计不良的应用程序. 同时, 通过相对阈值发现的不合规应用程序在很大程度上没有被视为比其他应用程序需要更多的维护工作.

4.6 其他无监督学习方法

本小节介绍幂律分布特征法、基于 Pickands-Balkema-de Haan (P-B-d) 定理的阈值模型、对数转换法、聚类方法、SATT 方法和贝叶斯层次模型等其他无监督学习方法.

4.6.1 基于幂律分布特征的阈值确定方法

2013 年, Shatnawi 等人[136]在 5 个 Java 开放源代码系统上使用最大似然估计将 OO 度量拟合到幂律分布上来确定阈值, 进而可以提供有关度量解释的更多信息. 幂律分布具有许多特征, 如宽尾分布 (fat-tail)、右偏分布 (right-skewed) 和无标度网络 (scale-free networks). 无标度网络指存在一些数据中心使得某些类遵守最复杂的规则. 一个无标度行为意味着无论软件系统如何发展, 总是有小部分比例的实体与许多度量值相对应.

幂律分布公式为 $P(x)=Cx^{-\alpha}$, 其中参数 α 为幂律分布的指数, 通常介于 2 和 3 之间. 常量 *C* 是调整系数, 使得 $P(x)$ 等于 1. 参数 α 可以通过对数变换后应用最小二乘法及最大似然估计法来估计. Clauset 等人[170]发现最大似然估计法比最小二乘法获得更为准确的估计值. 因此, Shatnawi 等人[136]选择极大似然法估计幂律分布的指数 α.

在软件工程领域, OO 度量都是离散变量. 然而, 极大似然法可应用于离散变量和连续变量两种类型的数据. Clauset 等人[170]提供幂律分布指数 α 的估计程序. 如公式 (25) 所示, 左右两式分别展示连续变量和离散变量情况下, 幂律分布指数 α 的极大似然估计值.

$$\alpha=1+n\left[\sum_{i=1}^{n}\ln\frac{x_i}{x_{\min}}\right]^{-1} \text{ 和 } \alpha=1+n\left[\sum_{i=1}^{n}\ln\frac{x_i}{x_{\min}-1/2}\right]^{-1} \tag{25}$$

其中, x_i (i=1,…,n) 为 x 的观察值, 且满足 $x_i \geqslant x_{\min}$, 表示 x 值从尾部分布开始.

从幂律分布中提取 $x_i \geqslant x_{\min}$ 并做假设检验非常重要. $x_{\min}$ 的估计应使得幂定律分布有较好的拟合优度. 如果选择的 $x_{\min}$ 值太低, 则由于拟合了非幂律分布数据而使得模型获得有偏参数; 另一方面, 如果选择的 $x_{\min}$ 值过高, 则将排除低于 $x_{\min}$ 的有效数据点. 为了平衡这两方面的限制, 通过最小化观察数据和最佳拟合模型的概率分布之间的差异来获得 $x_{\min}$ 的有效估计值. 为此, Shatnawi 等人[136]使用 Kolmogorov-Smirnov (KS) 拟合优度检验[171]来实现. KS 测量两个概率分布之间的距离. 当数据右偏时, KS 会产生合理的结果. KS 统计量[170]为 $D=\max|S(x)-P(x)|$. 通过最小化 *D* 可得出 $x_{\min}$ 的最佳估计值. $S(x)$ 是包括 $x_{\min}$ 观察值数据的累积分布函数. $P(x)$ 是在 $x \geqslant x_{\min}$ 区域上最佳拟合模型的累积分布函数.

估计这些参数还不足以得出给定的数据在 $x \geqslant x_{\min}$ 区域上服从幂律分布. 因此, Shatnawi 等人[136]选择拟合优度检验以验证数据是否遵循幂律分布的假设, 并选择小于 0.1 的 *p* 值用于排除幂律分布, 而大于 0.1 的 *p* 值并不表示幂律最合适, 即不能排除幂律分布对数据合理的拟合.

一个服从幂律分布的 OO 度量意味着其数据可以分为两组: 小于 $x_{\min}$ 和大于等于 $x_{\min}$. 每一组都有不同的幂律分布行为 (如复杂度). 大于等于 $x_{\min}$ 的这一组数据具有无标度网络的特征, 即少量类导致软件系统的最大复杂性. 对于不同的幂律分布的行为可以通过设置阈值来区分. 因此, Shatnawi 等人[136]选择 $x_{\min}$ 作为 OO 度量的一个合理的阈值.

4.6.2 基于 Pickands-Balkema-de Haan 定理的阈值模型

2014 年, 王志坚[13]在硕士毕业论文中提出基于 Pickands-Balkema-de Haan 定理的阈值模型, 并介绍两种阈值的选择方法: Hill 图法和平均超出量函数法. 该模型通过预先设定一个阈值, 收集所有超过这个阈值的数据构成新

的极值数据序列, 再对这些超过阈值的极值数据使用广义帕累托分布进行建模. 具体做法如下.

假设总体 X 的分布函数为 $F(x)$, $X_1, X_2, \ldots, X_n$ 为取自总体的一个随机样本序列. 将样本按取值大小排序, 则称为 $X_{(1)} \leqslant X_{(2)} \leqslant \ldots \leqslant X_{(n)}$ 次序统计量. 阈值模型仅关注超过某个预设好的阈值 μ 的样本点, 即尾部的次序统计量. 根据 Pickands-Balkema-de Haan 定理, 超过阈值 μ 的样本在总体样本中的分布情况, 即尾部估计如公式 (26):

$$F(x)=\begin{cases}1-\dfrac{N}{n}\left(1+\xi\dfrac{x-\mu}{\beta}\right)^{-\frac{1}{\xi}}, & \xi\neq 0\\ 1-\dfrac{N}{n}\exp\left(-\dfrac{x-\mu}{\beta}\right), & \xi=0\end{cases} \tag{26}$$

从公式 (26) 可以看出, 只有在阈值 μ 的取值足够大的情况下, 才满足超过阈值的分布函数近似等于帕累托分布. 阈值 μ 的选择决定了用来建模的极值数据, 进而决定广义帕累托分布的参数估计情况. 王志坚[13]给出两种阈值的选择方法: Hill 图法和平均超出量函数法.

(1) Hill 图法

设 $X_1>X_2>\ldots>X_n$ 为独立同分布的次序统计量. Hill 统计量如公式 (27) 所示. Hill 图为点集$\{(k, H^{-1}_{k,n}), 1\leqslant k\leqslant n-1\}$构成的曲线[172]. Hill 统计量在 k 大于一定值后, $H_{k,n}$ 趋近于一个常数. 通过观察 Hill 图, 可以选择位于图形中部和尾部, 纵坐标趋近于稳定的起始点的横坐标 K 所对应的数据点 X_k 作为阈值 μ.

$$H_{k,n}=\frac{1}{k}\sum_{i=1}^{k}\ln\left(\frac{X_i}{X_k}\right) \tag{27}$$

(2) 平均超出量函数

对随机变量 X, $e(\mu)=E(X-\mu|X>\mu)$ 为平均超出量函数. 对给定样本 $X_1, X_2, \ldots, X_n$, 样本的平均超出量函数如公式 (28) 所示:

$$e_n(\mu)=\frac{1}{N_\mu}\sum_{i\in\Delta_n(\mu)}(X_i-\mu),\ \mu>0 \tag{28}$$

其中, N_μ 表示超出量的数目, $\Delta_n(\mu)=\{i,\ i=1,2,\ldots,n,\ X_i>\mu\}$. 设 $X_{1,n}\geqslant X_{2,n}\geqslant\ldots\geqslant X_{n,n}$ 样本的次序统计量. 则定义点集$\{(X_{k,n}, e(X_{k,n})), k=1, 2, \ldots, n\}$称为平均超出量图. 如果对于某个阈值 μ_0 的值 μ, 样本的平均超出量函数图应该近似形成一条直线. 故选择使得 $e(X_{k,n})$ 关于横坐标近似为直线的那个起始点作为阈值.

为了检验模型的准确性, 即拟合优度 (goodness of fit), 王志坚[13]在实验中先用 Kolmogorov-Smirnov (KS) 检验给出数值化的 p 值, 然后通过拟合的分布与经验分布的概率图 (probability plot)、分位数图 (quantile plot)、重现水平图 (return level plot) 和频率分布图 (frequency plot) 来检验模型.

4.6.3 对数转换方法

2015 年, Shatnawi[110]提出通过对数转换方法确定度量阈值. 通过对数转换, 提高数据质量并减少数据偏斜, 且确定的阈值比使用未经对数转换的数据确定的阈值更好地识别错误类别. Shatnawi[110]在 PROMISE 数据集的 11 个系统上发现 6 个 CK 度量均向右偏, 进行转换后, 产生较少偏斜且更接近正态分布的数据.

常用的转换有对数变换、平方根变换和逆变换[173]. 对数转换不能产生零值的转换. 为此, 需要添加一个常量 (即增加数值 1). 平方根不适用于负值; 逆变换使软件模块的顺序相反, 在缺陷预测中未能起实质作用. Shatnawi[110]建议使用自然对数变换, 因为它可以减少数据点之间的相对距离, 即减少度量数据的偏度. 例如, Ant 1.7 系统中, 未转换之前 WMC、DIT、NOC、CBO、RFC 和 LCOM 的偏度分别为 2.992、0.437、15.787、12.819、2.459、11.961; 转换后的偏度分别为 0.125、–0.129、3.61、0.749、–0.529、0.523.

Shatnawi[110]提出使用偏度指数 (skewness index) 显示转换前后的差异. 偏度指数是数据分布中不对称性的度量. 正态分布的偏度为零, 任何对称数据的偏度都应接近零[173]. 经过转换后, 通过度量均值 μ 和标准差 σ 计算出 $T'=\mu+\sigma$ 和 $T''=\mu-\sigma$, 并设定其为上限阈值和下限阈值. 但是, Shatnawi[110]建议仅使用 $\mu+\sigma$ 得出一个阈值, 该阈值标识所研究系统的最复杂部分 (大于阈值部分). 由于 T'的结果代表了转换后的数据, 需要将其转换逆转以对原始数据产生阈值. 使用指数函数将 T'的值取反, 其计算公式为 $T=Exp(T')$.

2017 年, Gupta 等人[140]应用 Shatnawi[110]提出通过对数转换方法在 PROMISE 数据集上确定 6 个 CK 度量阈值. 与 Shatnawi 方法[110]不同在于进行对数转换前, Gupta 等人[140]从其中删除了各度量的重复记录值, 只有唯一的一组记录值保留在数据集中.

2018 年, Shatnawi[144]应用对数转换方法确定度量阈值后, 将错误预测模型与阈值结合以提高预测性能. 首先, 在原先二分类数据集上应用对数转换方法确定阈值, 应用该阈值将依赖变量二分类数据转换为三分类变量: 当度量值大于阈值且无错误的模块归类为中等分类, 而有错误的模块归类为高等分类, 剩下的都归为无错误分类. 将新增中等分类的模块也标记为有错误倾向的模块. 然后, 在新数据上使用 5 个机器学习模型 (朴素贝叶斯、逻辑回归、K 近邻、C4.5 和 JRip 决策树) 来构建预测模型. 在 PROMISE 数据集中 10 个系统构建这 5 个分类器. 实验结果发现, 与传统分类相比, 该方法改进了所有分类器的分类性能.

4.6.4 基于灰色关联等级的聚类算法

2016 年, Kaur 等人[139]提出基于灰色关联等级的聚类算法确定度量阈值. 其思路是在现有的错误预测框架一下, 将异构数据通过归一化分类为同质形式的数据. 然后, 在同质的数据集上执行聚类算法后, 找到每一个 (聚) 类的代表阈值. 最后, 在其他数据集上分析各类阈值的准确性.

Kaur 方法的主要步骤是: 先将数据集中各度量通过灰色关联 (等级) 法转换为灰色关联值; 再通过 K-means 聚类技术归一化数据集中所有度量的数据; 最后, 在聚类的数据集上, 计算各度量的灰色关联值的均值和方差, 以均值和方差之和作为度量阈值.

4.6.5 K-means 聚类方法

2017 年, Meena 等人[141]提出 K-means 聚类算法确定内聚和耦合类度量阈值. 通过 SourceMeter 工具生成的一个内聚性度量 (LCOM5) 和 5 个耦合性度量 (CBO、CBOI、NII、NOI 和 RFC). 数据来源为 ArgoUML 0.24, 0.28, 0.30 and 0.34 系统. 该方法包括 3 步.

(1) 统计分析. 在训练集上, 每个度量上应用 K-means 聚类算法生成中心值 (centroid values). 计算在 K 簇 (cluster) 对应的最终中心值和每个簇的描述性统计值.

(2) 确定阈值. 选择聚类的中心值作为阈值, 计算每个度量在各版本 ArgoUML 项目上阈值 (k=2, 3, 4) 的准确性性能指标. 即选择阈值预测性能高的那一簇的中心值作为各度量的最终阈值.

(3) 阈值有效性. 将这些阈值应用于更高的版本的 ArgoUML 项目测试阈值有效性.

4.6.6 期望最大化聚类方法

2019 年, Alqmase 等人[112]在 LOC、LCOM 和 CBO 度量上提出一种基于期望最大化算法的自动聚类框架确定软件度量阈值. 在不同的分类集群上, 通过确定不同分类阈值反映不同级别的软件质量. 该框架与语言无关, 并且基于面向对象的特性和属性. 聚类是根据特定的相似性/距离度量将数据对象分组为可变数量的簇 (cluster) 或类 (class). 期望最大化 (expectation maximization, EM) 算法将每个数据对象分配给具有不同成员资格概率的簇 (类). 通过迭代其期望值 (E) 和最大化 (M) 的两个主要步骤来最大化总体似然分配. 在计算期望值步骤中, 将数据对象与模型参数的初始估计值相关联, 并在其中创建概率分布. 然后, 通过似然最大化过程来完善初始模型. 当模型分布变得稳定并且在 E 步和 M 步之间不再变化时, 将获得 EM 算法的最终聚类决策. Alqmase 等人[112]的具体做法如下.

(1) 采用 He 等人[174]方法, 在 Weka 中应用 CfsSubsetEval 评估分类器 (evaluator) 和贪婪逐步搜索算法 (greedy stepwise search algorithm) 从原始数据中自动进行特征选择.

(2) 使用 EM 聚类算法, 将概率分布分配给每个度量观察值. EM 算法的输入是数据集中每个实体 (类、组件或代码块) 度量数据; 输出是不同的簇 (类). 每个簇 (类) 反应一个具体的软件质量级别.

(3) 通过上一步聚类后, 可以使用任何更好的方法从每个聚类中提取阈值. Alqmase 等人选择上一步骤中每个簇 (类) 中度量均值作为该簇 (类) 中度量的阈值. EM 算法可以根据用户自己的经验来指定分类数量, 从而可以产生指定数量的阈值.

4.6.7 为软件架构定制的阈值确定方法 (software architecture tailored thresholds, *SATT*)

2016 年, Aniche 等人[101]提出为软件架构定制的度量阈值确定方法 (*SATT* 方法). 该方法检测不同的体系结构

在代码度量方面是否很大的不同, 并提供一个基于软件体系结构度量的特定阈值. 在两个不同体系结构 (MVC 和 Android) 的 400 多个项目应用 *SATT* 方法后发现, *SATT* 可以克服传统方法所存在的问题, 尤其是在某些架构下的度量值与其他方法完全不同的情况下. Aniche 等人[101]在 GitHub 上收集了 120 个 Spring MVC 和 301 个 Android 系统数据. 它们都要求软件工程师在其应用程序中使用具有特定架构信息的类. Spring MVC 是一个 Java 框架, 支持开发人员构建 Web 应用程序; Android 是一个丰富的应用程序框架, 允许开发人员在 Java 语言环境中为移动设备构建应用程序. 以 MVC 系统中 Controller 类定义 McCabe 阈值为例, *SATT* 方法包括以下 8 步.

(1) 创建数据集. 选择遵循某一架构的系统, 如 Spring MVC 应用程序.

(2) 提取架构角色信息. 对于 Spring MVC, Controller 类始终使用@Controller 进行注释.

(3) 计算度量值. 计算基准数据中所有类的代码度量值, 如 McCabe, 无论它们在体系结构中的作用如何.

(4) 统计度量. 通过执行统计检验, 以测量该架构的类 (组 1) 和其他类 (组 2) 之间的代码度量值之间的差异. Aniche 等人[101]建议在两组之间使用非配对的 Wilcoxon 检验和 Cliff 的 Delta 值比较.

(5) 分析统计检验结果. 根据 Romano 等人[175]的分类来阐述效应值. 设 D 为效应值规模, 其取值范围为 [−1, 1]. 当$|D|<0.147$ 时, 表示很小效应 (negligible effect); 当 $0.147\leqslant|D|<0.33$ 时, 表示小效应 (small effect); 当 $0.33\leqslant|D|<0.474$ 时, 表示中等效应 (medium effect); 当$|D|\geqslant0.474$ 时, 表示大效应 (large effect).

(6) 计算权重比率. 从此步骤开始, 仅关注已分析架构的类. 计算所有类的 LOC 作为其权重并将其在基准数据中属于该架构所有的类归一化处理. 归一化保证所有权重总和为 100%. 假设在基准数据中 Controller 类的代码行总和为 10 万行, 若某个类 A 有 100 行代码, 则类 A 的权重为 0.001.

(7) 汇合权重比率. 与 Alves 排序法类似, 先将每个度量值的权重汇合, 再按每个度量值升序排序.

(8) 确定阈值. 从权重汇合后最接近 70% (中等)、80% (高) 和 90% (非常高) 的类中提取代码度量值分别作为分类阈值.

Aniche 等人[101]比较 *SATT* 方法与 Alves 排序法后, 发现在某一架构下的类与其他类有明显不同的度量值分布. 通过 Alves 排序法会返回一些可疑的分类阈值, 而 *SATT* 方法通过使用具有架构信息的度量值分布来定义分类阈值, 并对其进行改进. 在 MVC 和 Android 架构中的应用, 表明 *SATT* 方法可以克服传统方法中存在的问题, 尤其是当某些架构下的度量值与其他架构完全不同时. 因此, Aniche 等人[101]要求在应用代码质量评估工具, 如 SonarQube (www.sonarqube.org) 和 PMD (pmd.github.io), 对类级别度量进行特定于软件架构的处理.

4.6.8 贝叶斯层次模型 (Bayesian hierarchical modelling for tailoring metric thresholds)

2018 年, Ernst[117]在 Aniche 等人[101]的 120 个 Spring MVC 数据上应用贝叶斯层次建模方法确定度量阈值. 先使用 Aniche 等人的数据拟合全局汇合 (global pooling)、非汇合 (unpooled) 和部分汇合 (partial pooling) 这 3 个不同的回归模型, 然后使用均方根误差 (RMSE) 验证这些模型的准确性.

全局汇合指汇合所有单个文件, 并拟合单个线性回归模型 (固定效应模型); 非汇合指每个项目都有一个单独的回归模型, 可以比较每个项目的系数. 部分汇合指分层的部分汇合方法, 它根据全局估计来调整局部估计. 层次模型中的层次是项目中的文件, 部分汇合带来了细微差别. 这是由于软件架构[101]的局部影响而存在的, 但是通过全局参数对其进行了规范化 (又称为收缩, shrinkage). 实验结果表明, 与全局汇合方法相比, (部分汇合的) 分层模型可将模型预测误差减少多达 50%.

Aniche 等人[101]的 SATT 方法采用了 Alves 排序方法确定阈值. Alves 方法通过按代码对文件在所有项目中进行排名来设置, 然后找到最接近 70%/80%/90%(中等/高/非常高的风险) 的代码文件度量值. 同样, Ernst[117]使用部分汇合回归模型将阈值设置为 LCBO (CBO 度量的平均 log 值得分) 正态分布的 70%/80%/90%. 该方法确定的阈值因项目特征和正则化全局数据集而异.

4.7 小 结

本节介绍确定 OO 度量阈值的无监督学习方法. 在这些方法中, 以带权重的度量值百分位数来确定 OO 度量各分段阈值占多数, 如 Alves 排序、Ferreira 方法、Vale 方法、中位数阈值方法和 Oliveira 方法. 但也有幂律分布

特征法、基于 Pickands-Balkema-de Haan 定理的阈值模型、对数转换法、聚类方法、SATT 和贝叶斯层次模型等其他无监督学习方法.

这类方法的共性问题在于缺少各分段阈值左右两端不同风险水平上度量值是否显著不同的假设检验. 只有幂律分布特征的阈值确定方法、3 种聚类方法确定阈值方法和基于 Pickands-Balkema-de Haan 定理的阈值模型等少数方法通过检验后得出的各分段阈值.

5 其他方法

除了有监督和无监督学习方法确定 OO 度量阈值, 还可以通过其他其他方法确定 OO 度量阈值. 例如, 专家驱动、分析性阈值 (analytical threshold) 和多个系统上阈值中位数方法等.

5.1 专家驱动的阈值确定方法

2012 年, Sánchez-González 等人[68]认为阈值的定义需要理论和实践基础, 同时也要满足特定需求. 它应满足以下条件[79]: 阈值不应该基于专家的意见, 而是基于度量数据, 即度量数据的统计特征, 如度量规模、分布和对异常数据的弹性 (resilient); 它应该是可重复的, 透明的并且易于执行.

然而, Saraiva 等人[5]于 2019 年提出一种在缺少数据时, 基于专家的知识来定义 (确定) 度量阈值的系统方法. 其目的是通过开发模型来模仿阈值决策时人类思考过程, 为管理人员在决策有关度量的解释 (即语义) 时提供支持. 该方法基于影响度量阈值确定的背景因素 (context factor), 并且得到模糊逻辑概念 (fuzzy logic concept) 的支持, 将收集的明确数据 (crisp value) 建模转换为解释的信息 (linguistic variable).

Marinescu[176]提出了一个语义过滤准则, 用于支持在基于阈值检测设计缺陷的情况下对源代码度量进行分析. 该准则定义了两种与阈值相关的过滤准则: 边界 (marginal) 和区间 (interval). 边界过滤准则定义阈值和方向, 即指定阈值是上限还是下限. 区间过滤准则定义为诸如“在 20 和 30 之间”的区间内. Saraiva 等人[5]使用 Marinescu[176]中提出的阈值分类, 并通过系统的方法来定义阈值.

Saraiva 等人[5]介绍了 3 位具有 5 年以上开发经验的项目经理 (巴西专家). 他们从度量代码覆盖率、静态代码分析警告数和缺陷数中得出的背景因素, 用于决定软件产品是否满足质量要求来发布. 使用度量代码覆盖率来确定是否已经执行了足够的测试, 及缺陷数是否足够低到可以将产品交付给客户. 代码覆盖率 (crisp value) 的取值范围在 [0, 100%]. 可以设定 85% 阈值发布产品. 相反, 当代码覆盖率为 50% 时, 则不应该发布该产品. 专家驱动阈值定义的策略是由 3 个主要步骤循环组成: 阈值表征、阈值建模和阈值评估.

(1) 阈值表征 (threshold characterization). 通过识别相关背景因素和阈值类型来表征阈值. 首先, 定义度量的语义范围 (semantics scale). 通常有布尔和序数两种类型. 如用好与不好标记代码覆盖率的布尔值. 序数类型如差、中等和好. 其次, 选择阈值类型是边界还区间. 边界阈值有两个类型: $HigherThan(\Theta)$ 和 $LowerThan(\Theta)$, 其中, Θ 表示参考值. 区间阈值定义为 $Between(\alpha, \beta)$, 其等价于 $HigherThan(\alpha) \wedge LowerThan(\beta)$, α 和 β 分别为下限阈值和上限阈值. 然后, 识别背景因子 (context factor). 背景因子有两种类型: 递减因子和增强因子. 例如, 对于代码覆盖率度量, 专家认为产品复杂性和项目关键性是增强因素, 而团队经验和对演化成本的影响则是减少因素. 最后, 这些因素按其对阈值影响的相对大小排序.

(2) 阈值建模 (threshold modeling). 由于该方法基于模糊逻辑的概念, 需要将阈值建模为语义变量 (linguistic variable), 即“自然或人工语言中的单词或句子的变量”. 将步骤 (1) 中度量的语义范围映射到语义变量的术语集. 例如, 代码覆盖率的布尔类型语义变量术语集由{好, 不好}组成. 此步骤的主要目标是将度量的明确数值 (crisp value) 转换为模糊语言术语. 为了达到该目的, 需要定义一个成员函数. 有 6 种常见的函数. 若是边界阈值, 专家可以选择 Z-型、Sigmoid 型或 S 型作为参考函数. 若是区间阈值, 专家可以选择三角、梯型和高斯类型. 之后, 专家利用过去项目中的经验来定义“假设”(what-if) 场景, 以指导他们配置一组函数值, 即将可能的数值映射到每个可能术语. 这些数值并不是直接定义概率, 而是用口头量度 (verbal scale). 随后, 再将这些口头量度根据 Renooij 等人[177]的规则转换为数值. 可以使用一种算法 (如 Akima Cubic Spline) 将专家从每个术语中得出的数据拟合到适当

的分布中. 最后, 专家可以直观地分析结果分布, 并判断其是否反映了他的直觉, 否则, 它们必须反映出不一致之处, 并且应该重新启动该步骤.

(3) 阈值评估 (threshold evaluation). 与其他专家驱动的过程中一样, 必须有一个经验循环. 在该循环中, 将基于阈值的决策进行分析以评估模型. 为此, 阈值设计者应根据项目的背景安排会议来评估阈值. 如果模型的结果与实际情况不一致, 则可能会出现 3 种结果: 例外情况、范围限制和模型需要改进. 例如, 根据公司的现状, 所有项目的下限阈值分别为: 代码覆盖率为 80%; 静态代码分析警告数的阈值为 10; 缺陷数的阈值为 5.

5.2 分析性阈值确定方法

2006 年, Fernández 等人[6]通过度量推导过程分析得出敏感类内聚性度量 *SCOM* (sensitive class cohesion metric) 度量的阈值. 敏感的类内聚性度量 *SCOM* 是所有方法对之间的相似性总和与方法对总数的比值. 其计算过程如下.

设某个类有属性$\{A_1, A_2,\ldots, A_n\}$和方法$\{M_1, M_2,\ldots, M_n\}$. 设 I_k 表示在给定方法 M_k 下属性的子集, 设 a 是类中实例属性的总数. 因为两个集合交集最大可能的基数 (cardinality, 集合元素个数) 是它们基数的最小值, 则一对方法的连通强度 $C_{i,j}$ 如公式 (29) 所示. 当某个类的一对方法中仅有一个属性时, 则连通强度 $C_{i,j}$ 公式中分母值为 1.

$$C_{i,j}=\begin{cases}0, & \text{if } I_i\cap I_j=\emptyset\\ \dfrac{card(I_i\cap I_j)}{\min(card(I_i),\ card(I_j))}, & \text{otherwise}\end{cases} \tag{29}$$

当一对方法涉及更多属性时, 它们的连接强度必须给予更大权重. 因此, 在考虑权重 $\alpha_{i,j}=(card(I_i\cup I_j))/a$ 的情况下, $SCOM=[2/m(m-1)]\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}C_{i,j}\times\alpha_{i,j}$, 其中, 求和前的系数是方法对总数的倒数. 当某个类仅有一个方法和一个属性时, 其对内聚的贡献很小. *SCOM* 度量的取值范围为 [0, 1]. 当每个方法都有独立的属性集, *SCOM* 值为 0, 表示没有内聚; 当每个方法使用类中所有属性时, *SCOM* 值为 1, 表示全内聚 (full cohesion).

Fernández 等人[6]认为奇异点 (singular points) 的理论研究给出了如何解释 *SCOM* 度量值的信息. 将使用不相交属性子集的方法的子集数称为某个类的簇数 (clusters). Fernández 等人给出具有一个簇的类 *SCOM* 度量最小值和具有至少两个簇的类 *SCOM* 度量最大值的解析表达式.

(1) 具有一个簇的类 *SCOM* 度量最小值. 设某类有 m 个方法和 a 个属性. 当 a=3 和 a=4 时, Fernández 等人[6]从 m=2 开始向类中逐一增加方法数后归纳发现, 对内聚起重要作用的项数 (称为 S) 具有线性特征为 $S=a\cdot t+r$. 即 $S(m,a)=(1/2)[1+\text{int}[(m-1)/a]]\cdot[Mod[(m-1)/a]+m-1]$. 当两种方法都有相同属性为 $1/a$ 时, 会出现最小权重因子 $\alpha_{i,j}$. 因此, $SCOM_{\min}>[2/m(m-1)a]S(m,a)=SCOM_{\text{minK}}$. Fernández 等人[6]将 $SCOM_{\text{minK}}$ 称为 *SCOM* 度量已知的最小值. 当某个类的 $SCOM<SCOM_{\text{minK}}$ 时, 则其不满足具有一个簇的前提条件. Fernández 等人认为它至少有两个簇, 并且必须细分为更小、更加内聚的类.

(2) 具有两个簇的类 *SCOM* 度量最大值. 当某个类具有两个不相交属性子集方法的簇时, 可以评估出 *SCOM* 度量的最大值. 在此情况下, 该类 *SCOM* 度量公式中至少有 $m-1$ 个空项. 换句话, 该 *SCOM* 度量公式中至多有 $(m-1)(m-2)/2$ 个非空项. 连通强度 $C_{i,j}$ 的最大值为 1. 当一个簇有 $a-1$ 个属性而其他簇 (另一个簇) 仅有一个属性, 且这个簇中所有方法包含该簇内所有的属性时, 会出现最大的权重因子 $\alpha_{i,j}=(a-1)/a$. 最后, 具有两个簇的类 (包含 m 个方法和 a 个属性) *SCOM* 度量最大值为 $SCOM_{2\max}=(a-1)(m-2)/ma$. $SCOM_{2\max}$ 提供的分析值可确保类中一个簇的存在.

Fernández 等人[6]把一个项目中每个类的方法数平均值设为 12 和一个类中属性数阈值设为 3, 得出在 a=3 和 m=12 时, $SCOM_{\text{minK}}$ 值为 0.13, $SCOM_{2\max}$ 值为 0.56. 图 13 展示 *SCOM* 度量上各重要值. 若某类中存在使用不相交属性子集方法的两个簇, 则表明该类可以被拆分. 然而, 并非所有具有多个簇的类都可以被 $SCOM_{\text{minK}}$ 检测到. Fernández 等人[6]认为 *SCOM* 度量阈值可考虑的范围在 $SCOM_{\text{minK}}$ 与 1 之间. 如果可以保证类中只有一个簇, 则 *SCOM* 度量值将在 $SCOM_{2\max}$ 与 1 之间. 这些边界的具体数值依赖具体项目中类的方法数和属性数.

5.3 多个系统上的阈值中位数方法

2020 年, Shatnawi[113]在 Liu 等人[178]的文献中选择 8 种阈值确定方法进行比较, 发现不同的阈值确定技术应

用于不同领域多个系统时, 可以得出每个度量的一致阈值. 这 8 种方法主要思路是通过最大化或最小化敏感性、特异性、精度和召回率这些评估标准来优化阈值.

图 13 *SCOM* 度量值范围中各重要值

对于不平衡数据使用单一评估标准 (如最大敏感性或特异性) 可能也会导致阈值不平衡. 为此, Shatnawi 选择一对评估标准来确定阈值, 如 (敏感性, 特异性) 或 (召回率, 精度). 它们的公式分别为: 敏感性 (或召回率)=$a/(a+c)$; 特异性=$d/(d+b)$; 精度=$a/(a+b)$, 其中, a (*TP*)、b (*FP*)、c (*FN*)、d (*TN*) 分别表示混淆矩阵中的真正例、假正例、假反例、真反例的数据. 每个度量的所有值均被看作候选阈值 (t), 然后测试以下 8 种方法以找到最佳阈值.

(1) Kappa 值最大化方法[178]. Kappa 广泛应用于医疗领域评价者之间的一致性 (interrater agreement). 对于每个候选阈值 (t), 可根据公式 (30)–公式 (32) 计算出 Kappa 指数, 其取值范围为 [−1, 1]. 但在软件工程领域, Shatnawi[113]认为其应至少大于 0. Kappa 指数值为 0 表示偶然达成一致, 而 Kappa 值为 1 表示对有缺陷和无缺陷类别的完美评价. 因此, 最佳阈值应该选择观察一致性 (observed agreement) 与机会一致性 (chance agreement) 差异最大的候选阈值, 即候选阈值=max{*Kappa*(t)}.

$$Kappa(t)=\frac{observed\ agreement-chance\ agreement}{1-chance\ agreement} \tag{30}$$

$$observed\ agreement=(a+d)/n \tag{31}$$

$$chance\ agreement=\left[\frac{(a+c)(a+b)}{n}+\frac{(b+d)(c+d)}{n}\right]/n,\ (n=a+b+c+d) \tag{32}$$

(2) MaxSpSe. 同时最大化敏感性和特异性方法[179]. 在 ROC 曲线图中, 为了使敏感性和特异性同时最大, 而不是敏感性最大或特异性最大, 可以通过两类错误分类 (*FP* 和 *FN*) 概率最小化的交点处获得. 为此, Shatnawi[113]搜索每个候选阈值 (t) 的 (特异性 (t), 敏感性 (t)) 最小值. 在这些测试阈值中, 选择最大值, 从而保证敏感性和特异性同时最大, 即候选阈值=max{min(*specificity*(t), *sensitivity*(t))}.

(3) Youden 指数[180]. 敏感性和特异性之和最大化方法. Shatnawi[113]计算每个候选阈值的 Youden 指数: *YI*(t)=*sensitivity*(t)+*specificity*(t)−1. 与 MaxSpSe 不同, Youden 指数是搜索候选阈值中 (敏感性, 特异性) 最大的一对. 即候选阈值=max{*YI*(t)}.

(4) SpEqualSe (敏感性-特异性相等的方法)[159]. 敏感性和特异性差异的绝对值最小化. 它使得预测的真正例和真负例尽可能的相等. 敏感性衡量的是对错误类别的预测, 而特异性衡量的是对非错误类别的预测. 两者很难同时达到最大. 通常需要在两者之间进行权衡. 最小差异意味着应尽可能减少两种误分类率. 因此, 候选阈值=min{|*sensitivity*(t)−*specificity*(t)|}.

(5) 基于 ROC01 图的方法[181]. 在 ROC 曲线图中, 候选阈值对应的点到左上角 (0, 1) 点的距离最短. 其距离计算公式为 *ROC*01(t)=(1–*specificity*(t)–0)^2+(*sensitivity*(t)–1)^2. 左上角的点 (0, 1) 是分类完美点, 接近该点的候选阈值, 可以最大程度上减少错误分类. 据此, 其候选阈值=min{*ROC*01(t)}.

(6) *PrecisionEqRecall*. 精度和召回率差异的绝对值最小化[178]. 该方法试图使得精度和召回率都达到最大值, 它通过精度和召回率差异的绝对值最小化来度量. 精度和召回率差异的绝对值计算公式为 *PrecisionEqRecall*(t)=|*Precision*(t)–*Recall*(t)|. 据此, 候选阈值=min{*PrecisionEqRecall*(t)}.

(7) 基于 *Precision-Recall* 图的方法[178]. 在 Precision-Recall 图 (*PR*11) 中, 候选阈值对应的点到右上角 (1, 1) 点的距离最短. 其距离计算公式为 *PR*11(t)=(*Precision*(t)–1)^2+(*Recall*(t)–1)^2. *Precision-Recall* 图中的 (1, 1) 点是所有分类问题中最好的分类点. 因此, 搜索候选阈值到该点的距离最小的点同样也使得候选阈值分类效果最好. 即候选阈值=min{*PR*11(t)}.

(8) F 度量值最大化方法. F 度量值是精度和召回率的调和平均数[178], 其取值在 [0, 1]. F 度量值越大, 精度和召回率也最大, 其对应的分类效果最好. F 度量的公式为 $F(t)=[\alpha/Precision(t)+(1-\alpha)/Recall(t)]^{-1}$. 当 α=0.5 时, 只有在精度和查全率都很高的情况下 F 才能很高. 为此, Shatnawi[122]在公式中 α 取 0.5. 候选阈值=max{$PR11(t)$}.

在这 8 种阈值确定方法上, Shatnawi[113]用 PROMISE 数据集[150]中 11 个系统上 6 个 CK 度量分别确定其阈值. DIT 和 NOC 度量未能得出可实用的阈值. 在 11 个项目上应用 8 种方法可得出每个度量的 88 个阈值. 由于有些方法产生阈值的标准差偏大以及发生偏斜, 他们计算每一种方法的变异系数 (其值等于标准差除以均值) 后, 发现有 5 种方法的变异系数在剩下 4 种 CK 度量上变异系数较大, 只有 MaxSpSe、SpEqualSe 和 ROC01 图 3 种方法的变异系数较小. 即通过这 3 种方法确定的阈值并未在很大程度上发生分散. 因此, 他们建议应用这 3 种方法确定 OO 度量阈值, 并选择 11 个系统上各度量阈值中位数作为该度量所有阈值的代表.

5.4 小 结

本节从专家驱动、分析性阈值和多系统阈值中位数方法这 3 个角度介绍 OO 度量领域的其他相关研究. 下面简述主要发现.

(1) 专家驱动方法在缺少数据, 尤其缺少项目早期版本数据时, 可作为 OO 度量阈值确定方法的重要补充.

(2) 分析性阈值是一种从 OO 度量自身定义的基础上确定阈值方法.

(3) 多系统阈值中位数方法本质上是一个综合各类方法来确定阈值. 近年来, 也逐渐受到研究者的重视.

6 阈值工具和阈值评估

本节汇总在 OO 度量阈值确定过程中, 研究者提供的阈值确定工具、评估分类性能指标和实验数据集.

6.1 阈值确定工具

本节展示现有研究文献中 OO 度量阈值确定的实用工具. 表 7 列示这些工具名称、开发者、阈值确定方法和网址.

表 7 OO 度量阈值确定的实用工具

时间	工具名称	开发者	阈值确定方法	网址
2014年	RTTool	Oliveira等人[94]	Oliveira方法 (相对阈值)	aserg.labsoft.dcc.ufmg.br/rttool/
2014年	CTTool	Foucault等人[82]	基于分位数方法 (第80分位数)	se.labri.fr/data/articles/thresholds/
2016年	TDTool	Veado等人[98]	Alves排序法、Ferreira方法、Oliveira方法、Vale方法	labsoft.dcc.ufmg.br/doku.php?id=about:tdtool

2014 年, Oliveira 等人[94]提供一种基于语料库自动确定相对阈值的工具 RTTool. 在公式 (24) 中, 设置 $Penalty_1$ 中 Min 距离为 90%, 即在给定的语料库中, 至少应有 90% 的系统遵循真实的设计规则; 在公式 (25) 中, 设置 $Penalty_2$ 中 M 的第 $Tail$ 分位数为 90. 即假设重尾分布的尾部始于语料库中每个系统 S 中度量 M 值的第 90 个百分位数. Oliveira 等人[94]在 Tempero[182]的语料库中 106 个系统上, 应用 RTTool 工具计算出 FAN-OUT、LOC、NOA 和 NOM 这 4 个 OO 度量的相对阈值分别为: [80%, 15]、[75%, 222]、[80%, 8] 和 [80%, 17].

2014 年, Foucault 等人[82]提出计算基于上下文度量阈值的工具 CTTool (contextual metric thresholds tool). 该工具建立在 Harmony (一组 Java 插件) 的基础上, Harmony 是一个开放源代码研究平台, 旨在简化挖掘软件存储库工具的开发. 首先从 GitHub 上选择超过 200 万个随机项目; 然后通过 Harmony 分析得出 Java 类层级上度量值并存储在 MariaDB/MySQL 数据库中; 最后使用 R 语言计算阈值, 由 Harmony 分析计算出目标分位数和指标值并返回度量阈值.

2016 年, Veado 等人[98]在 Vale 等人[135]比较 Alves 排序法、Ferreira 方法和 Oliveira 方法基础上, 应用 VSD 工具[165]搜集基于软件产品线的基准数据. 数据以 CSV 文件格式存储. 在此基础上, 应用 TDTool 开源工具, 实现文献中 Alves 排序法、Ferreira 方法、Oliveira 方法和 Vale 方法来确定度量阈值. 这些方法都考虑 OO 度量的偏斜分布的数据特征. Veado 等人[98]指出 TDTool 已应用于面向功能的软件产品线及一些开源 Java 系统.

6.2 评估指标

评价指标是度量方法有效性的重要依据之一. 对于不同阈值模型或方法, 研究者使用多种类型指标对阈值应用输出结果进行评价. 本节从分类性能角度介绍阈值模型领域的评价指标.

表 8 列出 16 种 OO 度量阈值的分类性能指标, 每种指标根据表 4 中混淆矩阵 *TP*、*FP*、*TN* 和 *FN* 这 4 种情况二分类情况列出其计算公式. 这些指标的具体含义如下.

表 8 评价 OO 度量阈值的分类性能指标汇总

评估指标	计算公式	使用列表
Precision (精度)	$TP/(TP+FP)$	[9,68,80,81,91,99,103,107,108,110−112,114,131,135,136,138−143,146]
Recall (召回率)	$TP/(TP+FN)$	[8,9,68,80,81,91,92,96,99,103,107,108,110−115,131,135,136,138−143,146]
FPR (假正例率)	$FP/(FP+TN)$	[7,92,105,121]
FNR (假反例率)	$FN/(FN+TP)$	[7,105,121]
TNR (真反例率)	$TN/(FP+TN)$	[8,113,115]
Accuracy (准确率)	$(TP+TN)/(TP+FP+FN+TN)$	[8,9,103,106,131,140,141,143]
Error rate (错误率)	$(FN+FP)/(TP+FP+FN+TN)$	[105,121]
F-measure (*F*度量)	$\frac{2\times Precision\times Recall}{Precision+Recall}$	[9,68,81,91,93,96,97,99,107,110−112,131,136,138,140,141,143]
MCC (Matthews相关系数)	$\frac{TP\cdot TN-FP\cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	[97,107]
Balance (平衡值)	$1-\sqrt{(0-pf)^2+(1-pd)^2}$	[92]
G-mean (G均值)	$\sqrt{(TP/(TP+FN))\times(TN/(TN+FP))}$	[7,11,86,87,115,126,127,130,131]
AUC	Area under the ROC curve	[8,77,91,105,115,118,122,123,125−128,144]
RRA (RoI相关面积比例)	The ratio of relevant areas	[106]

(1) *Precision* (精度). 在预测为有缺陷的代码块 (如类模块等) 中, 有多少比例的代码块真正与该缺陷相关.

(2) *Recall* (召回率). 在所有与缺陷相关的代码块中, 有多少比例的代码块被正确预测为与缺陷相关.

(3) *Accuracy* (准确率). 在所有的分类结果中, 正确分类的代码块 (包括正确分为与缺陷相关和正确分为与缺陷无关的代码块) 占所有代码块的比例.

(4) Error rate (错误率). 在所有的分类结果中, 错误分类的代码块 (包括错误分为与缺陷相关和错误分为与缺陷无关的代码块) 占所有代码块的比例, 它等于 1–*Accuracy*.

(5) *FPR* (假正例率). 在所有与缺陷无关的代码块中, 有多少比例的代码块被错误预测为与缺陷相关.

(6) *FNR* (假反例率). 在所有与缺陷相关的代码块中, 有多少比例的代码块被错误预测为与缺陷无关.

(7) *TNR* (真反例率). 在所有与缺陷无关的代码块中, 有多少比例的代码块被正确预测为与缺陷无关.

(8) *F*-measure (*F* 度量). 精度与召回率的调和平均值, 用于客观地综合评价分类结果.

(9) MCC (Matthews 相关系数). 真实类别与预测类别二元分类之间的相关系数.

(10) *AUC*. ROC 曲线下与坐标轴围成的面积.

(11) Balance (平衡值). 1 减去 ROC 曲线上一点 (*pf*, *pd*) 到点 (0, 1) 的欧式距离.

(12) G-mean (G 均值). 召回率与真反例率的几何平均值.

(13) RRA (RoI 相关面积比例). 低于 ROC 曲线的 RoI 面积与 RoI 的总面积之比.

上述指标中, Error rate (错误率)、*FPR* (假正例率) 和 *FNR* (假反例率) 这 3 个指标与分类性能成反比; 其他指标均与分类性能成正比.

在机器学习领域, 缺陷预测模型中数据存在类别不平衡时, 通常不能得出准确的性能评价. 单个性能评价指标

对数据中某一较多的类别赋于更多的权重, 而且未考虑错误分类成本. 因而, 为了综合考虑两类数据 (如有缺陷和无缺陷代码块两种类别), 在存在数据类别不平衡时, 通常会选择综合性指标[144], 如 ROC 曲线. 表 8 中前 7 种指标 (1–7), 只从某一方面评价分类性能; 后面 6 种指标 (8–13), 则属于综合性分类性能指标. 虽然综合性指标考虑了类别不平衡等数据特征对评价的影响, 但文献中用得最多的还是召回率, 有 28 篇文献使用过, 然后是精度和 *F* 度量, 分别有 23 篇和 18 篇文献使用过. 此外, 还存在少量工作量感知相关的评价指标, 如 ER[77]. 不同文献对同一指标有多种表述, 如 *Recall* (召回率) 又称为 *TPR* (真正例率)、*Sensitivity* (敏感性)、*pd* (probability of detection rate, 检测率概率); *FPR* (假正例率) 又称 *pf* (probability of false alarms, 误警率); *TNR* (真反例率) 又称 *specificity* (特异性), 它等于 (1–*FPR*); *F*-measure (*F* 度量) 又称 *F*1-score 或调和平均值.

6.3 实验数据集

目前, OO 度量阈值研究领域已有较多研究者收集并且公开了它们的实验数据集. 这对于继续在 OO 度量阈值领域进行研究提供便利. 事实上, 应用 OO 度量阈值预测代码缺陷是 OO 度量预测代码缺陷内容之一. 为此, OO 度量预测代码缺陷研究领域的数据集都可以应用到 OO 度量阈值研究领域.

表 9 列出了 OO 度量阈值领域的公开可用数据集的汇总信息, 包括数据集提供者、所使用的项目语言、代码粒度、项目数和网址等. 这些数据集的收集和使用过程中仍存在一些值得重视的问题.

表 9 OO 度量阈值模型或方法的数据集汇总

提供者	项目语言	粒度	项目个数	缺陷标签?	使用列表	网址
Menzies等人[183]	Java/C/C++	Class	20	有	[7,8,92,96,121,127]	promise.site.uottawa.ca/SERepository/
Menzies等人[152]	Java/C/C++	Class	215	有	[7,97,106]	zenodo.org/communities/seacraft
Jureckzo等人[150]	Java/C/C++	Class	33	有	[9,10,93,110−113,121, 128,138−140]	openscience.us/repo (unavailable)
Tempero等人[182]	Java	Class	112	无	[94,132,134,137]	qualitascorpus.com
Oliveira等人[95]	Java	Class	79	无	[95]	aserg.labsoft.dcc.ufmg.br/software/
Vale等人[114]	Java/C/C#	Class/Method/ Project	64	无	[114]	labsoft.dcc.ufmg.br/doku.php?id=%20about:spl_list
D'Ambros等人[151]	Java	Class	5	有	[7,77,121,123]	bug.inf.usi.ch/download.php
Mausa等人[184]	Java	Class	2	有	[87,130]	www.seiplab.riteh.uniri.hr/?pageid=834
Mo等人[74]	Java/C/C++	File	38	无	[74]	www.cs.drexel.edu/~rm859/DL.php

(1) 数据集质量差异. 数据集差异主要有两方面. 一方面为是否包含缺陷标签信息. 由于缺少代码块上的缺陷信息, 不包含缺陷标签信息数据集只能通过无监督方法确定度量阈值, 如 Tempero 等人[182]的 Qualitas 语料库、Oliveira 等人[95]和 Vale 等人[114]的数据集. 另一方面, 不同研究者的数据集收集过程存在一定的偏差, 尤其是将同一代码块的度量信息与缺陷信息链接的过程.

(2) 数据集数量偏少. 虽然已有较多的数据集, 大多数研究者还是集中于使用个别数据集来确定 OO 度量阈值, 如 Jureckzo 等人[150]和 Menzies 等人[152]的数据集. 这方便使用同一数据集的阈值确定方法比较, 但也存在其他数据集未充分利用而影响数据集的可靠性, 而且在个别数据集上 OO 度量阈值获得好的缺陷预测性能, 其泛化性能需要更多实验数据的证实.

(3) 数据集语言结构偏向于 Java. 研究者提供的绝大多数数据集中实验项目是由 Java 开发, 其他语言开发的数据集相对较少且没有提供下载网址, 故而很少被大量引用. 不同语言在项目开发过程中有各自的特点, 其 OO 度量阈值存在差异, 仅仅在一种语言上确定的 OO 度量阈值不能直接应用到其他语言开的项目上. 如 Beranič等人[145]收集 Java、C++、C＃和 Python 项目上各度量值, 应用 Ferreira 等人[109]方法确定 4 种语言项目上度量阈值, 研究发现 CountLineCode 度量阈值从大到小的 4 种语言分别是 C＃、C++、Java 和 Python.

6.4 小 结

本节从阈值确定工具、评估分类性能指标和实验数据 3 个方面简要介绍了 OO 度量阈值确定方法的相关研究. 下面简述主要的发现.

(1) 现有的阈值确定工具 RTTool、CTTool 和 TDTool 均是应用无监督学习方法确定 OO 度量阈值. 缺乏有监督学习方法确定阈值的工具, 这也增加阈值确定方法应用推广难度.

(2) 现有 OO 度量阈值研究主要评价其分类性能, 绝大多数研究缺少工作量感知性能评价, 这导致现有的阈值研究方法难以满足实际应用的需求.

(3) 现有阈值研究中数据集偏向于 Java 语言, 且大多数研究集中于少数测试数据集上, 无法将各自研究方法推广至其他项目数据集上.

7 挑战与方向

从现有文献来看, OO 度量阈值确定方法的相关研究已经取得很好的成效. 然而, 现有的研究技术在工业界中并没有被广泛应用, 因为它还难以满足实际软件开发和维护过程的需求. 本节从规范性、泛化性、准确性和实用性四个角度来介绍 OO 度量阈值领域的研究面临着以下挑战.

(1) 规范性. 从 OO 度量阈值确定方法流程图 (图 2 和图 3) 中可以看出, OO 度量阈值确定过程主要包括两步: 阈值标准和阈值检验. 就阈值标准而言, 存在有监督学习方法和无监督学习方法各自阈值标准不统一、阈值标准判断过于主观和标准判断对度量训练数据集分布要求等局限性. 现有研究中仅有少数研究者执行了阈值假设检验[13,78,96,101–103,108,110,112,123,128,136,147], 大多数研究者在用各自方法确定阈值后, 直接在测试集评价其缺陷预测性能而缺少必要的阈值假设检验[8,79,82,86,105,109,118,134]. 若 OO 度量阈值未能进行统计学上假设检验, 则无法判断阈值在训练集上各模块度量值左右两边发生缺陷的概率是否发生显著变化. 此外, 若缺少阈值假设检验, 也无法解释继承性 OO 度量的逆阈值效应[78]挑战, 如 DIT 存在多个矛盾的阈值.

(2) 泛化性. 影响泛化性因素: (1) 度量概念不同. 由于存在众多收集度量数据的工具 (如表 1 所示), 这使得一些度量存在多种定义, 如 LCOM 存在至少 5 种[20,22–31]定义. 而由于同一度量定义不一致[110], 使得该度量阈值确定之后的应用带来概念上的误差. (2) 测试集数据偏少. 虽然现有多个数据集可供下载 (如表 9 所示), 但大多数阈值确定方法仍在集中在很少的测试项目上进行实验验证. 这些项目数据也大多数是由 Java 语言开发的. 目前的阈值确定方法很难获得其他语言开发的人员信任. 同时, 由于缺少在不同语言上确定阈值的工具, 增加了各自方法实现泛化的难易程度.

(3) 准确性. 准确性分为两方面: 1) 阈值确定的准确性. 现有的两类监督学习方法已有了 20 多种, 而且还会有新的阈值确定方法不断地被提出. 在度量阈值存在的假设检验基础上, 需要通过数学和统计学上的方法来综合各类方法的研究成果, 使得度量阈值研究结果更加接近于的度量阈值真实值. 2) 准确的实验数据集. 它是确定 OO 度量阈值预测性能的重要依据. 对于有监督学习方法而言, 对缺陷信息数据收集准确性会影响阈值预测模型和阈值假设检验[185]. 缺陷数据准确性需要注意两方面, 一是在缺陷信息系统收集代码块缺陷数据准确性; 另一方面是同一代码块的度量信息与缺陷信息的链接准确性. 为了保证 OO 度量阈值客观公正, 研究者需要确保数据集的准确性.

(4) 实用性. 目前, OO 度量阈值确定方法种类有 20 多种, 但活跃在实验研究阶段居多, 缺乏在工程实践中广泛应用. 除了度量阈值确定缺少客观阈值标准和阈值假设检验、准确性和泛化性低的因素外, 另一个重要原因是这些方法只能够提供给开发者建议性确定阈值方法和参考性 OO 度量阈值. 在生产实践中, 开发人员需要根据这些方法来确定各自开发环境下的 OO 度量阈值, 缺少功能强大 (如自动收集 OO 度量数据和项目历史版本上的缺陷数据) 的阈值确定的实用工具.

针对 OO 度量阈值确定方法研究领域面临的问题挑战, 我们从过程规范、方法创新、实验验证和阈值应用总结了以下几方面可能将会成为进一步深入研究的重要方向.

(1) 过程规范. 包括 OO 度量数据收集规范和 OO 度量阈值确定过程规范两方面内容. 一方面, IEEE (电气与电子工程师协会) 制定的 IEEE Std 1061-1998《IEEE 软件质量度量方法学标准》[186]定义了一个软件质量度量的框架, 并对度量描述和度量各数据项描述进行规范. 研究者们需要在统一的软件质量度量框架下完成各类度量数据收集工作. 另一方面, 完整的 OO 度量阈值确定过程, 主要包括阈值标准和阈值检验两部分. 在阈值确定过程中, 应该选择客观的判断标准, 尽量减少主观因素的影响. 同时, 阈值检验用于发现 OO 度量值在阈值左右两边是否存在显著不同. 若缺少必要的假设检验, 则该阈值失去统计学上的判断意义.

(2) 方法创新. OO 度量阈值确定方法的创新主要包括两方面. 一方面, OO 度量阈值的具体确定方法可以引入机器学习中先进分类技术方法, 提升 OO 度量阈值在测试集上的分类性能; 另一方面, 在解决泛化性问题上, 可以应用统计学上元分析方法综合各类度量阈值研究, 得出的综合阈值可以应用于训练数据以外的软件系统. 元分析 (meta-analysis)[187]方法合成来自于一系列单个研究的数据, 包括固定效应和随机效应元分析. 当前, 元分析已经成为医学、药理学、教育学、经济学、生态学和软件工程等多个学科和领域重要的研究工具.

(3) 实验验证. 在解决准确性差问题上, 一个研究方向是对目前方法进行科学准确对比实验研究. 这主要包括两方面. 一方面是全面精准的收集数据集. 重点在缺陷信息数据和度量缺陷链接技术上准确性. 为 OO 度量阈值确定方法研究收集更多准确数据集, 需要涵盖各种语言开发的项目以及各个领域的项目, 用来验证其有效性和泛化性. 另一方面是设置科学的实验流程和设置. 缺少关键的阈值假设检验的实验设置, 会影响实验结果的准确性. 为此, 需要科学的设计实验, 且符合现实开发场景, 以确保其合理性.

(4) 阈值应用. 为了提高 OO 度量阈值应用的实用性, 一个重要研究方向是提供包括实用工具在内的全面指导过程. 本质上, OO 度量阈值预测代码缺陷是 OO 度量缺陷预测重要内容之一. 同样, OO 度量阈值可以在跨项目、跨公司、跨语言等背景下进一步深入应用研究. 如果 OO 度量阈值确定方法为这些跨项目的泛化研究提供功能强大的实用工具, 可以在很大程度上帮助开发人员在现实开发环境部署, 并顺利投入生产实践工作.

8 结束语

近年来, 基于 OO 度量阈值的软件缺陷预测方法由于具有计算成本低、部署简便等优势, 成为 OO 度量缺陷预测领域的研究热点. 本文主要围绕 OO 度量阈值确定的有监督和无监督学习方法两大方面, 对当前研究工作进行了梳理和总结. 基于当前的研究进展, 本文总结了该领域面临的主要挑战并指出了未来可能的研究方向. 主要工作总结如下:

首先, 本文总结了 OO 度量阈值确定过程包括阈值标准和阈值检验. 其次, 重点介绍了单变量逻辑回归阈值模型、定量风险评估方法和 ROC 曲线方法为主的有监督学习方法和基于带权重度量值的百分位数 (在 0 至 100% 选择阈值标准) 阈值确定方法为主的无监督学习方法, 以及专家驱动方法、分析性阈值方法和多个系统上阈值中位数方法. 然后, 展示了阈值确定工具、分类性能指标和实验数据集等阈值确定的相关研究工作. 最后, 本文从规范性、泛化性、准确性和实用性 4 个角度总结了该领域面临的主要问题挑战并针对性指出了未来可能的研究方向.

References:

[1] Conte SD, Dunsmore HE, Shen VY. Software Engineering Metrics and Models. Redwood City: Benjamin-cummings Publishing Co. Inc., 1986.

[2] Zhou XC, Lai W, Wen JF. Empirical study on the distribution of object-oriented software cohesion metrics. Ruan Jian Xue Bao/Journal of Software, 2018, 29(10): 3051–3067 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5290.htm [doi: 10.13328/j.cnki.jos.005290]

[3] Fenton NE, Pfleeger SL. Software Metrics: A Rigorous and Practical Approach. 2nd ed., Boston: PWS Publishing Press, 1997.

[4] Lanza M, Marinescu R. Object-oriented Metrics in Practice—Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-oriented Systems. Berlin: Springer, 2006. [doi: 10.1007/3-540-39538-5]

[5] Saraiva R, Perkusich M, Almeida H, Perkusich A. A systematic process to define expert-driven software metrics thresholds. In: Proc. of the 31st Int'l Conf. on Software Engineering and Knowledge Engineering. Lisbon: SEKE, 2019. 171–226. [doi: 10.18293/SEKE2019-217]

[6] Fernández L, Peña R. A sensitive metric of class cohesion. Information Theories & Applications, 2006, 13: 82–91.

[7] Boucher A, Badri M. Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. Information and Software Technology, 2018, 96: 38–67. [doi: 10.1016/j.infsof.2017.11.005]

[8] Malhotra R, Bansal AJ. Quantitative assessment of risks considering threshold effects of object-oriented metrics using open source software. Software Quality Professional, 2012, 14(4): 33–46.

[9] Hussain S, Keung J, Khan AA, Bennin KE. Detection of fault-prone classes using logistic regression based object-oriented metrics thresholds. In: Proc. of the 2016 IEEE Int'l Conf. on Software Quality, Reliability and Security Companion (QRS-C). Vienna: IEEE,

2016. 93–100. [doi: 10.1109/QRS-C.2016.16]
[10] Arar OF, Ayan K. Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies. Expert Systems with Applications, 2016, 61: 106–121. [doi: 10.1016/j.eswa.2016.05.018]
[11] Bender R. Quantitative risk assessment in epidemiological studies investigating threshold effects. Biometrical Journal, 1999, 41(3): 305–319. [doi: 10.1002/(SICI)1521-4036(199906)41:3<305::AID-BIMJ305>3.0.CO;2-Y]
[12] Ronchieri E, Canaparo M. A preliminary mapping study of software metrics thresholds. In: Proc. of the 11th Int'l Joint Conf. on Software Technologies (ICSOFT-EA). Lisbon: ICSOFT, 2016. 232–240. [doi: 10.5220/0005988402320240]
[13] Wang ZJ. An empirical study for predicting the extreme values of faults in software projects [MS. Thesis]. Nanjing: Nanjing University, 2014 (in Chinese with English abstract).
[14] Chen ZF. Code smell evaluation and detection based on mining software maintenance history [Ph.D. Thesis]. Nanjing: Nanjing University, 2018 (in Chinese with English abstract).
[15] Zuse H. Software Complexity: Measures and Methods. Berlin: Walter de Gruyter GmbH & Co KG, 1990.
[16] McCabe TJ. A complexity measure. IEEE Trans. on Software Engineering, 1976, SE-2(4): 308–320. [doi: 10.1109/TSE.1976.233837]
[17] Halstead MH. Elements of Software Science. New York: Elsevier, 1977.
[18] Curtis B, Sheppard SB, Milliman P, Borst MA, Love T. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. IEEE Trans. on Software Engineering, 1979, SE-5(2): 96–104. [doi: 10.1109/TSE.1979.234165]
[19] Basili VR, Rombach HD. The TAME project: Towards improvement-oriented software environments. IEEE Trans. on Software Engineering, 1988, 14(6): 758–773. [doi: 10.1109/32.6156]
[20] Briand LC, Daly JW, Wüst J. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering, 1998, 3(1): 65–117. [doi: 10.1023/A:1009783721306]
[21] Briand LC, Daly JW, Wust JK. A unified framework for coupling measurement in object-oriented systems. IEEE Trans. on Software Engineering, 1999, 25(1): 91–121. [doi: 10.1109/32.748920]
[22] Chidamber SR, Kemerer CF. Towards a metrics suite for object oriented design. In: Proc. of the Conf. on Object-oriented Programming Systems, Languages, and Applications. Phoenix: ACM, 1991. 197–211. [doi: 10.1145/117954.117970]
[23] Lorenz M, Kidd J. Object-oriented Software Metrics: A Practical Guide. Englewood Cliffs: PTR Prentice Hall, 1994.
[24] 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]
[25] https://github.com/meiyuanqing/MetaThreshold/blob/main/附录 (表 A1_内聚性).docx
[26] https://github.com/meiyuanqing/MetaThreshold/blob/main/附录 (表 A2_耦合性).docx
[27] https://github.com/meiyuanqing/MetaThreshold/blob/main/附录 (表 A3_继承性).docx
[28] https://github.com/meiyuanqing/MetaThreshold/blob/main/附录 (表 A4_规模与复杂性).docx
[29] 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]
[30] Hitz M, Montazeri B. Measuring coupling and cohesion in object-oriented systems. In: Proc. of the Int'l Symp. on Applied Corporate Computing. Monterrey: ISACC, 1995. 1–10.
[31] Henderson-Sellers B. Object-oriented Metrics. Upper Saddle River: Prentice Hall, 1996.
[32] Tegarden DP, Sheetz SD, Monarchi DE. A software complexity model of object-oriented systems. Decision Support Systems, 1995, 13(3–4): 241–262. [doi: 10.1016/0167-9236(93)E0045-F]
[33] Li W, Henry S. Object-oriented metrics that predict maintainability. Journal of Systems and Software, 1993, 23(2): 111–122. [doi: 10.1016/0164-1212(93)90077-B]
[34] Martin R. OO design quality metrics: An analysis of dependencies. Quality Engineering, 1994, 12(1): 151–170.
[35] Bieman JM, Kang B-K. Cohesion and reuse in an object-oriented system. ACM SIGSOFT Software Engineering Notes, 1995, 20(SI): 259–262. [doi: 10.1145/223427.211856]
[36] Lee YS, Liang BS, Wu FJ, Wang SF. Measuring the coupling and cohesion of an object-oriented program based on information flow. In: Proc. of the 1995 Int'l Conf. on Software Quality. Maribor: ICSQ, 1995. 81–90.
[37] Badri L, Badri M. A proposal of a new class cohesion criterion: An empirical study. Journal of Object Technology, 2004, 3(4): 145–159. [doi: 10.5381/jot.2004.3.4.a8]
[38] Abreu FB, Carapuça R. Object-oriented software engineering: Measuring and controlling the development process. In: Proc. of the 4th Int'l Conf. of Software Quality. McLean: ICSQ, 1994. 1–8.
[39] Abreu FB. The MOOD metrics set. In: Proc. of the 1995 ECOOP Workshop on Metrics. Aarhus: ECOOP, 1995. 150–152.

[40] Kim EM, Kusumoto S, Kikuno T, Chang OB. Heuristics for computing attribute values of C++ program complexity metrics. In: Proc. of the 20th Int'l Computer Software and Applications Conf. Seoul: IEEE, 1996. 104–109. [doi: 10.1109/CMPSAC.1996.542433]

[41] Briand L, Devanbu P, Melo W. An investigation into coupling measures for C++. In: Proc. of the 19th Int'l Conf. on Software Engineering. Boston: ACM, 1997. 412–421. [doi: 10.1145/253228.253367]

[42] Harrison R, Counsell S, Nithi R. Coupling metrics for object-oriented design. In: Proc. of the 5th Int'l Software Metrics Symp. Metrics (Cat. No. 98TB100262). Bethesda: IEEE, 1998. 150–157. [doi: 10.1109/METRIC.1998.731240]

[43] Chae HS, Kwon YR. A cohesion measure for classes in object-oriented systems. In: Proc. of the 5th Int'l Software Metrics Symp. Metrics (Cat. No. 98TB100262). Bethesda: IEEE, 1998. 158–166. [doi: 10.1109/METRIC.1998.731241]

[44] Chae HS, Kwon YR, Bae DH. A cohesion measure for object-oriented classes. Software: Practice and Experience, 2000, 30(12): 1405–1431. [doi: 10.1002/1097-024X(200010)30:12<1405::AID-SPE330>3.0.CO;2-3]

[45] Li W. Another metric suite for object-oriented programming. Journal of Systems and Software, 1998, 44(2): 155–162. [doi: 10.1016/S0164-1212(98)10052-3]

[46] Benlarbi S, Melo WL. Polymorphism measures for early risk prediction. In: Proc. of the 1999 Int'l Conf. on Software Engineering (IEEE Cat. No. 99CB37002). Los Angeles: IEEE, 1999. 334–344. [doi: 10.1145/302405.302652]

[47] Bansiya J, Etzkorn L, Davis C, Li W. A class cohesion metric for object-oriented designs. Journal of Object-oriented Programming, 1999, 11(8): 47–52.

[48] Tang MH, Kao MH, Chen MH. An empirical study on object-oriented metrics. In: Proc. of the 6th Int'l Software Metrics Symp. (Cat. No. PR00403). Boca Raton: IEEE, 1999. 242–249. [doi: 10.1109/METRIC.1999.809745]

[49] Zhou YM, Xu BW. An object-extracting approach using module cohesion. Ruan Jian Xue Bao/Journal of Software, 2000, 11(4): 557–562 (in Chinese with English abstract). http://www.jos.org.cn/jos/article/abstract/20000421

[50] Briand LC, Wüst J, Daly JW, Porter DV. Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems and Software, 2000, 51(3): 245–273. [doi: 10.1016/S0164-1212(99)00102-8]

[51] Briand LC, Wust J. Modeling development effort in object-oriented systems using design properties. IEEE Trans. on Software Engineering, 2001, 27(11): 963–986. [doi: 10.1109/32.965338]

[52] Xu BW, Zhou YM. Comments on "a cohesion measure for object-oriented classes" by Heung Seok Chae, Yong Rae Kwon and Doo Hwan Bae (Softw. Pract. Exper. 2000; 30: 1405–1431). Software: Practice and Experience, 2001, 31(14): 1381–1388. [doi: 10.1002/spe.413]

[53] Marinescu R. Detecting design flaws via metrics in object-oriented systems. In: Proc. of the 39th Int'l Conf. and Exhibition on Technology of Object-oriented Languages and Systems. TOOLS 39. Santa Barbara: IEEE, 2001. 173–182. [doi: 10.1109/TOOLS.2001.941671]

[54] Chen ZQ, Zhou YM, Xu BW, Zhao JJ, Yang HJ. A novel approach to measuring class cohesion based on dependence analysis. In: Proc. of the 2002 Int'l Conf. on Software Maintenance. Montreal: IEEE, 2002. 377–384. [doi: 10.1109/ICSM.2002.1167794]

[55] Counsell S, Mendes E, Swift S, Tucker A. Evaluation of an object-oriented cohesion metric through Hamming distances. Technical Report, London: University of London, 2002.

[56] Counsell S, Swift S, Crampton J. The interpretation and utility of three cohesion metrics for object-oriented design. ACM Trans. on Software Engineering and Methodology, 2006, 15(2): 123–149. [doi: 10.1145/1131421.1131422]

[57] Aman H, Yamasaki K, Yamada H, Noda MT. A proposal of class cohesion metrics using sizes of cohesive parts. In: Proc. of the 5th Joint Conf. on Knowledge-based Software Engineering. Amsterdam: IOS Press/Ohmsha, 2002. 102–107.

[58] Zhou YM, Xu BW, Zhao JJ, Yang HJ. ICBMC: An improved cohesion measure for classes. In: Proc. of the 2002 Int'l Conf. on Software Maintenance. Montreal: IEEE, 2002. 44–53. [doi: 10.1109/ICSM.2002.1167746]

[59] Marcus A, Poshyvanyk D. The conceptual cohesion of classes. In: Proc. of the 21st IEEE Int'l Conf. on Software Maintenance (ICSM2005). Budapest: IEEE, 2005. 133–142. [doi: 10.1109/ICSM.2005.89]

[60] Aggarwal KK, Singh Y, Kaur A, Malhotra R. Software reuse metrics for object-oriented systems. In: Proc. of the 3rd ACIS Int'l Conf. on Software Engineering Research, Management and Applications (SERA2005). Mount Pleasant: IEEE, 2005. 48–54. [doi: 10.1109/SERA.2005.60]

[61] Poshyvanyk D, Marcus A. The conceptual coupling metrics for object-oriented systems. In: Proc. of the 22nd IEEE Int'l Conf. on Software Maintenance. Philadelphia: IEEE, 2006. 469–478. [doi: 10.1109/ICSM.2006.67]

[62] Bonja C, Kidanmariam E. Metrics for class cohesion and similarity between methods. In: Proc. of the 44th annual Southeast Regional Conf. Melbourne: ACM, 2006. 91–95. [doi: 10.1145/1185448.1185469]

[63] Zhou YM, Xu BW. Dependence structure analysis-based approach for measuring importance of classes. Journal of Southeast University

(Natural Science Edition), 2008, 38(3): 380–384 (in Chinese with English abstract). [doi: 10.3969/j.issn.1001-0505.2008.03.004]

[64] Zhou TL, Xu BW, Shi L, Zhou YM. Measuring package cohesion based on client usages. Ruan Jian Xue Bao/Journal of Software, 2009, 20(2): 256–270 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/561.htm

[65] Al Dallal J. Mathematical validation of object-oriented class cohesion metrics. Int'l Journal of Computers, 2010, 4(2): 45–52.

[66] Singh S, Kahlon KS. Effectiveness of encapsulation and object-oriented metrics to refactor code and identify error prone classes using bad smells. ACM SIGSOFT Software Engineering Notes, 2011, 36(5): 1–10. [doi: 10.1145/2020976.2020994]

[67] Al Dallal J, Briand LC. A precise method-method interaction-based cohesion metric for object-oriented classes. ACM Trans. on Software Engineering and Methodology, 2012, 21(2): 1–34. [doi: 10.1145/2089116.2089118]

[68] Sánchez-González L, García F, Ruiz F, Mendling J. A study of the effectiveness of two threshold definition techniques. In: Proc. of the 16th Int'l Conf. on Evaluation & Assessment in Software Engineering (EASE 2012). Ciudad Real: IEEE, 2012. 197–205. [doi: 10.1049/ic.2012.0026]

[69] Desouky AF, Etzkorn LH. Object oriented cohesion metrics: A qualitative empirical analysis of runtime behavior. In: Proc. of the 2014 ACM Southeast Regional Conf. Kennesaw: ACM, 2014. 1–6. [doi: 10.1145/2638404.2638464]

[70] Mitchell A, Power JF. Run-time cohesion metrics: An empirical investigation, In: Proc. of the Int'l Conf. on Software Engineering Research and Practice. Las Vegas: CSREA Press, 2004. 532–537.

[71] Mathur R, Keen KJ, Etzkorn LH. Towards a measure of object oriented runtime cohesion based on number of instance variable accesses. In: Proc. of the 49th Annual Southeast Regional Conf. Kennesaw: ACM, 2011. 255–257. [doi: 10.1145/2016039.2016105]

[72] Mathur R. Object-oriented runtime software quality analysis [Ph.D. Thesis]. Huntsville: The University of Alabama in Huntsville, 2011.

[73] Panda S, Mohapatra DP. ACCo: A novel approach to measure cohesion using hierarchical slicing of Java programs. Innovations in Systems and Software Engineering, 2015, 11(4): 243–260. [doi: 10.1007/s11334-015-0252-8]

[74] Mo R, Cai YF, Kazman R, Xiao L, Feng Q. Decoupling level: A new metric for architectural maintenance complexity. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Austin: IEEE, 2016. 499–510. [doi: 10.1145/2884781.2884825]

[75] Lima P, Guerra E, Meirelles P, Kanashiro L, Silva H, Silveira FF. A metrics suite for code annotation assessment. Journal of Systems and Software, 2018, 137: 163–183. [doi: 10.1016/j.jss.2017.11.024]

[76] Bogner J, Wagner S, Zimmermann A. Collecting service-based maintainability metrics from RESTful API descriptions: Static analysis and threshold derivation. In: Proc. of the 14th European Conf. on Software Architecture. L'Aquila: Springer, 2020. 215–227. [doi: 10.1007/978-3-030-59155-7_16]

[77] Zhou YM, Xu BW, Leung H, Chen L. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Trans. on Software Engineering and Methodology, 2014, 23(1): 10. [doi: 10.1145/2556777]

[78] Benlarbi S, El Emam K, Goel N, Rai S. Thresholds for object-oriented measures. In: Proc. of the 11th Int'l Symp. on Software Reliability Engineering. San Jose: IEEE, 2000. 24–38. [doi: 10.1109/ISSRE.2000.885858]

[79] Alves TL, Ypma C, Visser J. Deriving metric thresholds from benchmark data. In: Proc. of the 2010 IEEE Int'l Conf. on Software Maintenance. Timisoara: IEEE, 2010. 1–10. [doi: 10.1109/ICSM.2010.5609747]

[80] Vale GAD, Figueiredo EML. A method to derive metric thresholds for software product lines. In: Proc. of the 29th Brazilian Symp. on Software Engineering. Belo Horizonte: IEEE, 2015. 110–119. [doi: 10.1109/SBES.2015.9]

[81] Morasca S. Classifying faulty modules with an extension of the H-index. In: Proc. of the 26th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Gaithersbury: IEEE, 2015. 416–426. [doi: 10.1109/ISSRE.2015.7381835]

[82] Foucault M, Palyart M, Falleri JR, Blanc X. Computing contextual metric thresholds. In: Proc. of the 29th Annual ACM Symp. on Applied Computing. Gyeongju: ACM, 2014. 1120–1125. [doi: 10.1145/2554850.2554997]

[83] Cartwright M, Shepperd M. An empirical investigation of an object-oriented software system. IEEE Trans. on Software Engineering, 2000, 26(8): 786–796. [doi: 10.1109/32.879814]

[84] Unger B, Prechelt L, Philippsen M. The impact of inheritance depth on maintenance tasks: Detailed description and evaluation of two experiment replications. Technical Report, Karlsruhe: University of Karlsruhe, 1998.

[85] Shatnawi R. An investigation of CK metrics thresholds. In: Proc. of the 17th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE 2006). Raleigh: IEEE, 2006. 12–13.

[86] Shatnawi R. A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. IEEE Trans. on Software Engineering, 2010, 36(2): 216–225. [doi: 10.1109/TSE.2010.9]

[87] Mauša G, Grbac TG. The stability of threshold values for software metrics in software defect prediction. In: Proc. of the 7th Int'l Conf. on Model and Data Engineering. Barcelona: Springer, 2017. 81–95. [doi: 10.1007/978-3-319-66854-3_7]

[88] Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M. Lessons from applying the systematic literature review process within the

software engineering domain. Journal of Systems and Software, 2007, 80(4): 571–583. [doi: 10.1016/j.jss.2006.07.009]

[89] Alan O, Catal DC. An outlier detection algorithm based on object-oriented metrics thresholds. In: Proc. of the 24th Int'l Symp. on Computer and Information Sciences. Guzelyurt: IEEE, 2009. 567–570. [doi: 10.1109/ISCIS.2009.5291882]

[90] Catal C, Sevim U, Diri B. Clustering and metrics thresholds based software fault prediction of unlabeled program modules. In: Proc. of the 6th Int'l Conf. on Information Technology: New Generations. Las Vegas: IEEE, 2009. 199–204. [doi: 10.1109/ITNG.2009.12]

[91] Nam J, Kim S. Clami: Defect prediction on unlabeled datasets. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lincoln: IEEE, 2015. 452–463. [doi: 10.1109/ASE.2015.56]

[92] Tosun A, Bener A. Reducing false alarms in software defect prediction by decision threshold optimization. In: Proc. of the 3rd Int'l Symp. on Empirical Software Engineering and Measurement. Lake Buena Vista: IEEE, 2009. 477–480. [doi: 10.1109/ESEM.2009.5316006]

[93] Lavazza L, Morasca S. Identifying thresholds for software faultiness via optimistic and pessimistic estimations. In: Proc. of the 10th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Ciudad Real: ACM, 2016. 1–10. [doi: 10.1145/2961111.2962595]

[94] Oliveira P, Lima FP, Valente MT, Serebrenik A. RTTool: A tool for extracting relative thresholds for source code metrics. In: Proc. of the 2014 IEEE Int'l Conf. on Software Maintenance and Evolution. Victoria: IEEE, 2014. 629–632. [doi: 10.1109/ICSME.2014.112]

[95] Oliveira P, Valente MT, Bergel A, Serebrenik A. Validating metric thresholds with developers: An early result. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Bremen: IEEE, 2015. 546–550. [doi: 10.1109/ICSM.2015.7332511]

[96] Morasca S, Lavazza L. Slope-based fault-proneness thresholds for software engineering measures. In: Proc. of the 20th Int'l Conf. on Evaluation and Assessment in Software Engineering. Limerick: ACM, 2016. 1–10. [doi: 10.1145/2915970.2915997]

[97] Lavazza L, Morasca S. Dealing with uncertainty in binary logistic regression fault-proneness models. In: Proc. of the Evaluation and Assessment on Software Engineering. Copenhagen: ACM, 2019. 46–55. [doi: 10.1145/3319008.3319012]

[98] Veado L, Vale G, Fernandes E, Figueiredo E. TDTool: Threshold derivation tool. In: Proc. of the 20th Int'l Conf. on Evaluation and Assessment in Software Engineering. Limerick: ACM, 2016. 24. [doi: 10.1145/2915970.2916014]

[99] Tsuda N, Washizaki H, Fukazawa Y, Yasuda Y, Sugimura S. Machine learning to evaluate evolvability defects: Code metrics thresholds for a given context. In: Proc. of the 2018 IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). Lisbon: IEEE, 2018. 83–94. [doi: 10.1109/QRS.2018.00022]

[100] Yamashita K, Huang CY, Nagappan M, Kamei Y, Mockus A, Hassan AE, Ubayashi N. Thresholds for size and complexity metrics: A case study from the perspective of defect density. In: Proc. of the 2016 IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). Vienna: IEEE, 2016. 191–201. [doi: 10.1109/QRS.2016.31]

[101] Aniche M, Treude C, Zaidman A, Van Deursen A, Gerosa MA. SATT: Tailoring code metric thresholds for different software architectures. In: Proc. of the 16th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM). Raleigh: IEEE, 2016. 41–50. [doi: 10.1109/SCAM.2016.19]

[102] El Emam K, Benlarbi S, Goel N, Melo W, Lounis H, Rai SN. The optimal class size for object-oriented software. IEEE Trans. on Software Engineering, 2002, 28(5): 494–509. [doi: 10.1109/TSE.2002.1000452]

[103] Zhou YM, Leung H. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Trans. on Software Engineering, 2006, 32(10): 771–789. [doi: 10.1109/TSE.2006.102]

[104] Chidamber SR, Darcy DP, Kemerer CF. Managerial use of metrics for object-oriented software: An exploratory analysis. IEEE Trans. on Software Engineering, 1998, 24(8): 629–639. [doi: 10.1109/32.707698]

[105] Catal C, Alan O, Balkan K. Class noise detection based on software metrics and ROC curves. Information Sciences, 2011, 181(21): 4867–4877. [doi: 10.1016/j.ins.2011.06.017]

[106] Morasca S, Lavazza L. On the assessment of software defect prediction models via ROC curves. Empirical Software Engineering, 2020, 25(5): 3977–4019. [doi: 10.1007/s10664-020-09861-4]

[107] Herbold S, Grabowski J, Waack S. Calculation and optimization of thresholds for sets of software metrics. Empirical Software Engineering, 2011, 16(6): 812–841. [doi: 10.1007/s10664-011-9162-z]

[108] Mendling J, Sánchez-González L, García F, La Rosa M. Thresholds for error probability measures of business process models. Journal of Systems and Software, 2012, 85(5): 1188–1197. [doi: 10.1016/j.jss.2012.01.017]

[109] Ferreira KAM, Bigonha MAS, Bigonha RS, Mendes LFO, Almeida HC. Identifying thresholds for object-oriented software metrics. Journal of Systems and Software, 2012, 85(2): 244–257. [doi: 10.1016/j.jss.2011.05.044]

[110] Shatnawi R. Deriving metrics thresholds using log transformation. Journal of Software: Evolution and Process, 2015, 27(2): 95–113. [doi: 10.1002/smr.1702]

[111] Morasca S, Lavazza L. Risk-averse slope-based thresholds: Definition and empirical evaluation. Information and Software Technology, 2017, 89: 37–63. [doi: 10.1016/j.infsof.2017.03.005]

[112] Alqmase M, Alshayeb M, Ghouti L. Threshold extraction framework for software metrics. Journal of Computer Science and Technology, 2019, 34(5): 1063–1078. [doi: 10.1007/s11390-019-1960-6]

[113] Shatnawi R. A comparison of threshold identification techniques for object-oriented software metrics. IET Software, 2020, 14(6): 727–738. [doi: 10.1049/iet-sen.2020.0025]

[114] Vale G, Fernandes E, Figueiredo E. On the proposal and evaluation of a benchmark-based threshold derivation method. Software Quality Journal, 2019, 27(1): 275–306. [doi: 10.1007/s11219-018-9405-y]

[115] Malhotra R, Bansal AJ. Fault prediction considering threshold effects of object-oriented metrics. Expert System, 2015, 32(2): 203–219. [doi: 10.1111/exsy.12078]

[116] Spadini D, Schvarcbacher M, Oprescu AM, Bruntink M, Bacchelli A. Investigating severity thresholds for test smells. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: ACM, 2020. 311–321. [doi: 10.1145/3379597.3387453]

[117] Ernst NA. Bayesian hierarchical modelling for tailoring metric thresholds. In: Proc. of the 15th Int'l Conf. on Mining Software Repositories. Gothenburg: ACM, 2018. 587–591. [doi: 10.1145/3196398.3196443]

[118] Shatnawi R, Li W, Swain J, Newman T. Finding software metrics threshold values using ROC curves. Journal of Software Maintenance and evolution: Research and Practice, 2010, 22(1): 1–16. [doi: 10.1002/smr.404]

[119] Kaur S, Singh S, Kaur H. A quantitative investigation of software metrics threshold values at acceptable risk level. Int'l Journal of Engineering Research & Technology (IJERT), 2013, 2(3): 1–7.

[120] Singh S, Kahlon KS. Object oriented software metrics threshold values at quantitative acceptable risk level. CSI Trans. on ICT, 2014, 2(3): 191–205. [doi: 10.1007/s40012-014-0057-1]

[121] Boucher A, Badri M. Using software metrics thresholds to predict fault-prone classes in object-oriented software. In: Proc. of the 4th Int'l Conf. on Applied Computing and Information Technology/3rd Int'l Conf. on Computational Science/Intelligence and Applied Informatics/1st Int'l Conf. on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD). Las Vegas: IEEE, 2016. 169–176. [doi: 10.1109/ACIT-CSII-BCD.2016.042]

[122] Malhotra R, Bansal A. Identifying threshold values of an open source software using receiver operating characteristics curve (ROC). Journal of Information and Optimization Sciences, 2017, 38(1): 39–69. [doi: 10.1080/02522667.2015.1135592]

[123] Shatnawi R. Identifying threshold values of change-prone modules. In: Proc. of the 2018 Int'l Conf. on E-business and Mobile Commerce. Chengdu: ACM, 2018. 39–43. [doi: 10.1145/3230467.3230477]

[124] Mihancea PF, Marinescu R. Towards the optimization of automatic detection of design flaws in object-oriented software systems. In: Proc. of the 9th European Conf. on Software Maintenance and Reengineering. Manchester: IEEE, 2005. 92–101. [doi: 10.1109/CSMR.2005.53]

[125] Kumari D, Rajnish K. Finding error-prone classes at design time using class based object-oriented metrics threshold through statistical method. Infocomp Journal of Computer Science, 2013, 12(1): 49–63.

[126] Malhotra R, Bansal A. Prediction of change prone classes using threshold methodology. Advances in Computer Science and Information Technology (ACSIT), 2015, 2(11): 30–35.

[127] Hussain S. Threshold analysis of design metrics to detect design flaws: Student research abstract. In: Proc. of the 31st Annual ACM Symp. on Applied Computing. Pisa: ACM, 2016. 1584–1585. [doi: 10.1145/2851613.2852013]

[128] Shatnawi R. The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction. Innovations in Systems and Software Engineering, 2017, 13(2–3): 201–217. [doi: 10.1007/s11334-017-0295-0]

[129] Malhotra R, Sharma A. Estimating the threshold of software metrics for Web applications. Int'l Journal of System Assurance Engineering and Management, 2019, 10(1): 110–125. [doi: 10.1007/s13198-019-00773-1]

[130] Mauša G, Galinac Grbac T, Brezočnik L, Vili P, Heričko M. Software metrics as identifiers of defect occurrence severity. In: Proc. of the 8th Workshop on Software Quality Analysis. Ohrid: SQAMIA, 2019. 9.

[131] Padhy N, Panigrahi R, Neeraja K. Threshold estimation from software metrics by using evolutionary techniques and its proposed algorithms, models. Evolutionary Intelligence, 2019, 14(2): 315–329. [doi: 10.1007/s12065-019-00201-0]

[132] Filó TGS, Bigonha MAS, Ferreira KAM. A catalogue of thresholds for object-oriented software metrics. In: Proc. of the 1st Int'l Conf. on Advances and Trends in Software Engineering. Barcelona: IARIA, 2015. 48–55.

[133] Erni K, Lewerentz C. Applying design-metrics to object-oriented frameworks. In: Proc. of the 3rd Int'l Software Metrics Symp. Berlin: IEEE, 1996. 64–74. [doi: 10.1109/METRIC.1996.492444]

[134] Oliveira P, Valente MT, Lima FP. Extracting relative thresholds for source code metrics. In: Proc. of the 2014 Software Evolution

Week/IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). Antwerp: IEEE, 2014. 254–263. [doi: 10.1109/CSMR-WCRE.2014.6747177]

[135] Vale G, Albuquerque D, Figueiredo E, Garcia A. Defining metric thresholds for software product lines: A comparative study. In: Proc. of the 19th Int'l Conf. on Software Product Line. Nashville: ACM, 2015. 176–185. [doi: 10.1145/2791060.2791078]

[136] Shatnawi R, Qutaibah A. An empirical study of the effect of power law distribution on the interpretation of OO metrics. ISRN Software Engineering, 2013, 2013: 198937. [doi: 10.1155/2013/198937]

[137] Fontana FA, Ferme V, Zanoni M, Yamashita A. Automatic metric thresholds derivation for code smell detection. In: Proc. of the 6th IEEE/ACM Int'l Workshop on Emerging Trends in Software Metrics. Florence: IEEE, 2015. 44–53. [doi: 10.1109/WETSoM.2015.14]

[138] Lavazza L, Morasca S. An empirical evaluation of distribution-based thresholds for internal software measures. In: Proc. of the the 12th Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. Ciudad Real: ACM, 2016. 1–10. [doi: 10.1145/2972958.2972965]

[139] Kaur H, Singh J, Singh B. Metrics threshold analysis on the basis of clustering technique for fault prediction. Int'l Journal of Science and Research, 2016, 5(6): 158–162.

[140] Gupta S, Gupta D. Fault Prediction using metric threshold value of object-oriented systems. Int'l Journal of Engineering Science and Computing, 2017, 7(6): 13629–13643.

[141] Meena SS, Singh S. Threshold design for cohesion and coupling metrics using K-means clustering algorithm. Int'l Journal of Advanced Research in Computer Science, 2017, 8(5): 2297–2307.

[142] Mori A, Vale G, Viggiato M, Oliveira J, Figueiredo E, Cirilo E, Jamshidi P, Kastner C. Evaluating domain-specific metric thresholds: An empirical study. In: Proc. of the 2018 Int'l Conf. on Technical Debt. Gothenburg: ACM, 2018. 41–50. [doi: 10.1145/3194164.3194173]

[143] Beranič T, Podgorelec V, Heričko M. Towards a reliable identification of deficient code with a combination of software metrics. Applied Sciences, 2018, 8(10): 1902. [doi: 10.3390/app8101902]

[144] Shatnawi R. Improving software fault prediction with threshold values. In: Proc. of the 26th Int 'l Conf. on Software, Telecommunications and Computer Networks (SoftCOM). Split: IEEE, 2018. 1–6. [doi: 10.23919/SOFTCOM.2018.8555818]

[145] Beranič T, Heričko M. Comparison of systematically derived software metrics thresholds for object-oriented programming languages. Computer Science and Information Systems, 2020, 17(1): 181–203. [doi: 10.2298/CSIS181012035B]

[146] Mori AV. Design and evaluation of a method to derive domain metric thresholds. [MS. Thesis]. Minas Gerais: Universidade Federal de Minas Gerais, 2018.

[147] Shatnawi R. The validation and threshold values of object-oriented metrics [Ph.D. Thesis]. Huntsville: University of Alabama in Huntsville, 2006.

[148] Stojkovski M. Thresholds for software quality metrics in open source android projects [MS. Thesis]. Trondheim: Norwegian University of Science and Technology, 2017.

[149] Ulm K. A statistical method for assessing a threshold in epidemiological studies. Statistics in Medicine, 1991, 10(3): 341–349. [doi: 10.1002/sim.4780100306]

[150] 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. 1–10. [doi: 10.1145/1868328.1868342]

[151] D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches. In: Proc. of the 7th IEEE Working Conf. on Mining Software Repositories (MSR 2010). Cape Town: IEEE, 2010. 31–41. [doi: 10.1109/MSR.2010.5463279]

[152] Menzies T, Krishna R, Pryor D. The SEACRAFT repository of empirical software engineering data. 2017. https://zenodo.org/communities/seacraft.

[153] El Emam K, Benlarbi S, Goel N, Rai SN. The confounding effect of class size on the validity of object-oriented metrics. IEEE Trans. on Software Engineering, 2001, 27(7): 630–650. [doi: 10.1109/32.935855]

[154] El Emam K, Benlarbi S, Goel N, Melo W, Lounis H, Rai SN. The optimal class size for object-oriented software: A replicated study. Fredericton: National Research Council Canada, 2000.

[155] Baker RJ, Nelder JA. The GLIM System Release 3: Generalised Linear Interactive Modelling Manual. Oxford: Oxford Numerical Algorithms Group United Kingdom GB, 1978.

[156] Hatton L. Does OO sync with how we think? IEEE Software, 1998, 15(3): 46–54. [doi: 10.1109/52.676735]

[157] Pastor R, Guallar E. Use of two-segmented logistic regression to estimate change-points in epidemiologic studies. American Journal of Epidemiology, 1998, 148(7): 631–642. [doi: 10.1093/aje/148.7.631]

[158] Spackman KA. Signal detection theory: Valuable tools for evaluating inductive learning. In: Segre AM, ed. Proc. of the 6th Int 'l

Workshop on Machine Learning. San Mateo: Morgan Kaufmann, 1989. 160–163. [doi: 10.1016/B978-1-55860-036-2.50047-3]

[159] Hosmer DW, Lemeshow S. Stepwise Logistic Regression. Applied Logistic Regression. New York: Wilsy-Intsrscisncs, 2000. 116–121.

[160] Kearns M. Efficient noise-tolerant learning from statistical queries. Journal of the ACM, 1998, 45(6): 983–1006. [doi: 10.1145/293347.293351]

[161] Rahman F, Posnett D, Devanbu P. Recalling the "imprecision" of cross-project defect prediction. In: Proc. of the ACM SIGSOFT 20th Int'l Symp. on the Foundations of Software Engineering. Cary: ACM, 2012. 1–11. [doi: 10.1145/2393596.2393669]

[162] Hirsch JE. An index to quantify an individual's scientific research output. Proc. of the National Academy of Sciences of the United States of America, 2005, 102(46): 16569–16572. [doi: 10.1073/pnas.0507655102]

[163] Mäntylä MV, Lassenius C. What types of defects are really discovered in code reviews? IEEE Trans. on Software Engineering, 2009, 35(3): 430–448. [doi: 10.1109/TSE.2008.71]

[164] Zhang F, Mockus A, Zou Y, Khomh F, Hassan AE. How does context affect the distribution of software maintainability metrics? In: Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance. Eindhoven: IEEE, 2013. 350–359. [doi: 10.1109/ICSM.2013.46]

[165] Abilio R, Vale G, Oliveira J, Figueiredo E, Costa H. Code smell detection tool for compositional-based software product lines. BCSTS, 2014. 109–116.

[166] Herzig K, Just S, Rau A, Zeller A. Predicting defects using change genealogies. In: Proc. of the 24th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Pasadena: IEEE, 2013. 118–127. [doi: 10.1109/ISSRE.2013.6698911]

[167] Wu RX, Zhang HY, Kim S, Cheung SC. Relink: Recovering links between bugs and changes. In: Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering. Szeged: ACM, 2011. 15–25. [doi: 10.1145/2025113.2025120]

[168] Louridas P, Spinellis D, Vlachos V. Power laws in software. ACM Trans. on Software Engineering and Methodology, 2008, 18(1): 1–26. [doi: 10.1145/1391984.1391986]

[169] Taube-Schock C, Walker RJ, Witten IH. Can we avoid high coupling? In: Proc. of the 25th European Conf. on ECOOP 2011-Object-Oriented Programming. Lancaster: Springer, 204–228. [doi: 10.1007/978-3-642-22655-7_10]

[170] Clauset A, Shalizi CR, Newman MEJ. Power-law distributions in empirical data. SIAM Review, 2009, 51(4): 661–703. [doi: 10.1137/070710111]

[171] Flannery BP, Press WH, Teukolsky SA, Vetterling WT. Numerical Recipes in C: The Art of Scientific Computing. Cambridge: Cambridge University Press, 1993: 1002.

[172] Drees H, De Haan L, Resnick S. How to make a hill plot. Annals of Statistics, 2000, 28(1): 254–274. [doi: 10.1214/aos/1016120372]

[173] Osborne J. Notes on the use of data transformations. Practical Assessment, Research, and Evaluation, 2002, 8: 6. [doi: 10.7275/4vng-5608]

[174] He P, Li B, Liu X, Chen J, Ma YT. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 2015, 59: 170–190. [doi: 10.1016/j.infsof.2014.11.006]

[175] Romano J, Kromrey JD, Coraggio J, Skowronek J. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys? Annual Meeting of the Florida Association of Institutional Research, 2006, 177: 1–33.

[176] Marinescu R. Detection strategies: Metrics-based rules for detecting design flaws. In: Proc. of the 20th IEEE Int'l Conf. on Software Maintenance. Chicago: IEEE, 2004. 350–359. [doi: 10.1109/ICSM.2004.1357820]

[177] Renooij S, Witteman C. Talking probabilities: Communicating probabilistic information with words and numbers. Int'l Journal of Approximate Reasoning, 1999, 22(3): 169–194. [doi: 10.1016/S0888-613X(99)00027-4]

[178] Liu CR, Berry PM, Dawson TP, Pearson RG. Selecting thresholds of occurrence in the prediction of species distributions. Ecography, 2005, 28(3): 385–393. [doi: 10.1111/j.0906-7590.2005.03957.x]

[179] Gallop RJ, Crits-Christoph P, Muenz LR, Tu XM. Determination and interpretation of the optimal operating point for ROC curves derived through generalized linear models. Understanding Statistics, 2003, 2(4): 219–242. [doi: 10.1207/S15328031US0204_01]

[180] Youden WJ. Index for rating diagnostic tests. Cancer, 1950, 3(1): 32–35. [doi: 10.1002/1097-0142(1950)3:1%3C32::AID-CNCR2820030106%3E3.0.CO;2-3]

[181] Vermont J, Bosson JL, Francois P, Robert C, Rueff A, Demongeot J. Strategies for graphical threshold determination. Computer Methods and Programs in Biomedicine, 1991, 35(2): 141–150. [doi: 10.1016/0169-2607(91)90072-2]

[182] Tempero E, Anslow C, Dietrich J, Han T, Li J, Lumpe M, Melton H, Noble J. The qualitas corpus: A curated collection of Java code for empirical studies. In: Proc. of the 2010 Asia Pacific Software Engineering Conf. Sydney: IEEE, 2010. 336–345. [doi: 10.1109/APSEC.2010.46]

[183] Menzies T, Caglayan B, He Z, Kocaguneli E, Krall J, Peters F, Turhan B. The promise repository of empirical software engineering data. West Virginia University, Department of Computer Science, 2012.

[184] Mauša G, Galinac-Grbac T, Dalbelo-Bašić B. A systematic data collection procedure for software defect prediction. Computer Science and Information Systems, 2016, 13(1): 173–197. [doi: 10.2298/CSIS141228061M]

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

[186] IEEE. IEEE Std 1061-1998 IEEE standard for a software quality metrics methodology. New York: IEEE, 1998. [doi: 10.1109/IEEESTD.1994.9055496]

[187] Borenstein M, Hedges LV, Higgins JPT, Rothstein HR. Introduction to Meta-Analysis. Hoboken: John Wiley & Sons Ltd., 2009.

附中文参考文献:

[2] 周晓聪, 赖蔚, 温剑丰. 面向对象软件内聚度度量数据分布的实证研究. 软件学报, 2018, 29(10): 3051–3067. http://www.jos.org.cn/1000-9825/5290.htm [doi: 10.13328/j.cnki.jos.005290]

[13] 王志坚. 软件项目缺陷极值预测 [硕士学位论文]. 南京: 南京大学, 2014.

[14] 陈芝菲. 基于软件维护历史的代码异味评估与检测 [博士学位论文]. 南京: 南京大学, 2018.

[49] 周毓明, 徐宝文. 一种利用模块内聚性的对象抽取方法. 软件学报, 2000, 11(4): 557−562. http://www.jos.org.cn/jos/article/abstract/20000421

[63] 周毓明, 徐宝文. 基于依赖结构分析的类重要性度量方法. 东南大学学报(自然科学版), 2008, 38(3): 380–384. [doi: 10.3969/j.issn.1001-0505.2008.03.004]

[64] 周天琳, 徐宝文, 史亮, 周毓明. 基于客户程序度量包内聚性. 软件学报, 2009, 20(2): 256−270. http://www.jos.org.cn/1000-9825/561.htm

梅元清(1980－), 男, 博士生, 副教授, 主要研究领域为软件度量, 软件缺陷预测.

陈林(1979－), 男, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为软件分析测试.

郭肇强(1994－), 男, 博士, CCF 学生会员, 主要研究领域为软件度量, 缺陷定位.

卢红敏(1975－), 女, 博士, CCF 专业会员, 主要研究领域为软件分析与测试.

周慧聪(1996－), 女, 硕士, CCF 学生会员, 主要研究领域为缺陷定位.

周毓明(1974－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件维护, 测试与分析.

李言辉(1981－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为软件演化与维护, 软件测试.