软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(2):582−624 [doi: 10.13328/j.cnki.jos.006714]

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

软件缺陷预测模型间的比较实验: 问题、进展与挑战*

刘旭同[1,2], 郭肇强[1,2], 刘释然[1,2], 张 鹏[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: 周毓明, E-mail: zhouyuming@nju.edu.cn

摘 要: 近年来, 研究者提出了大量的软件缺陷预测模型, 新模型往往通过与过往模型进行比较实验来表明其有效性. 然而, 研究者在设计新旧模型间的比较实验时并没有达成共识, 不同的工作往往采用不完全一致的比较实验设置, 这可能致使在对比模型时得到误导性结论, 最终错失提升缺陷预测能力的机会. 对近年来国内外学者所做的缺陷预测模型间的比较实验进行系统性的总结: 首先, 阐述缺陷预测模型间的比较实验的研究问题; 然后, 分别从缺陷数据集、数据集划分、基线模型、性能指标、分类阈值这 5 个方面对现有的比较实验进行总结; 最后, 指出目前在进行缺陷预测模型间比较实验时面临的挑战, 并给出建议的研究方向.

关键词: 缺陷预测; 比较实验; 软件维护; 质量保障

中图法分类号: TP311

中文引用格式: 刘旭同, 郭肇强, 刘释然, 张鹏, 卢红敏, 周毓明. 软件缺陷预测模型间的比较实验: 问题、进展与挑战. 软件学报, 2023, 34(2): 582–624. http://www.jos.org.cn/1000-9825/6714.htm

英文引用格式: Liu XT, Guo ZQ, Liu SR, Zhang P, Lu HM, Zhou YM. Comparing Software Defect Prediction Models: Research Problem, Progress, and Challenges. Ruan Jian Xue Bao/Journal of Software, 2023, 34(2): 582−624 (in Chinese). http://www.jos.org.cn/1000-9825/6714.htm

Comparing Software Defect Prediction Models: Research Problem, Progress, and Challenges

LIU Xu-Tong[1,2], GUO Zhao-Qiang[1,2], LIU Shi-Ran[1,2], ZHANG Peng[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: In recent years, a large number of software defect prediction models have been proposed. Once a new defect prediction model is proposed, it is often compared with previous defect prediction models to evaluate its effectiveness. However, there is no consensus on how to compare the newly proposed defect prediction model with previous defect prediction models. Different studies often adopt different settings for comparison, which may lead to misleading conclusions in the comparisons of prediction models, and consequently lead to missing the opportunity to improve the effectiveness of defect prediction. This study systematically reviews the comparative experiments of software defect prediction models conducted by worldwide scholars in recent years. First, the comparisons of defect prediction models are introduced. Then, the research progress is summarized from the perspectives of defect dataset, dataset split, baseline models, performance indicators, and classification thresholds, respectively, in the comparisons. Finally, the opportunities and challenges are summarized in comparative experiments of defect prediction models and the research directions in the future are outlined.

Key words: defect prediction; comparative experiment; software maintenance; quality assurance

软件缺陷预测(software defect prediction, SDP)模型的目的是预测软件项目中缺陷的潜在位置, 即预测哪个(些)模块(文件、类、函数, 等等)中可能包含缺陷[1]. 这样的预测信息在软件质量保障过程中非常重要: 一方面, 它赋予待检查或者待测试模块优先级, 从而尽早地在被测软件项目中找到有缺陷的模块; 另一方面,它指

* 基金项目: 国家自然科学基金(62172205); 江苏省研究生科研与实践创新计划(KYCX22_0153)
收稿时间: 2021-09-16; 修改时间: 2022-03-11; 采用时间: 2022-06-07; jos 在线出版时间: 2022-07-22

导测试人员为各个模块更加高效、合理地分配代码审查或测试资源. 为包含缺陷可能性高的模块分配更多的审查资源, 可以帮助质量保障人员在给定预算中发现到尽可能多的软件缺陷[1,2].

在过去的十几年中, 大量的缺陷预测模型被提出. 从建模技术来看, 现有的预测模型可以被大致分为两类: 有监督和无监督. 有监督模型使用带标签的数据进行训练获得预测函数[2], 而无监督模型从未标记的数据中推断出一个隐藏结构的函数[3]. 从应用场景来看, 现有的预测模型也可分为两类: 分类和排序. 分类模型将目标项目中的模块预测为有缺陷或无缺陷两个类别, 被预测为有缺陷的那些模块将被检查/测试[1]. 而排序模型将目标项目中的模块按照包含缺陷可能性从高到低进行排列[4]. 维护团队从排序中选择用于测试/检查的潜在缺陷模块, 直至可用资源被分配完毕. 从性能评估的角度看, 现有的预测模型可以通过两种方式进行评估: 非工作量感知和工作量感知. 在非工作量感知评估过程中, 所有模块所需要的审查/测试工作量被视作一样的, 并且审查/测试过程中, 在不同模块之间进行的上下文切换所需工作量不被考虑. 在工作量感知评估过程[4,5]中, 被审查/测试的模块所需要的工作量被视作各不相同, 并在不同模块之间进行的上下文切换所需工作量被考虑在内.

每当新的预测模型被提出, 一个重要的问题是如何评估其缺陷预测的有效性, 即它帮助开发人员在项目中找到缺陷的能力. 现有工作中, 对新提出的预测模型的评估遵循以下几个步骤: (1) 选择若干不久前新提出来的过往缺陷预测模型作为被比较的模型; (2) 为比较实验选择合适的评价指标、缺陷数据集、数据集划分方式; (3) 使用训练集建模并在测试集上得到预测结果, 并使用新模型和被比较的过往模型各自默认的分类阈值来计算依赖于阈值的评价指标; (4) 使用统计学方法来检测新模型是否显著优于过往模型, 且这样的优势有实践意义. 迄今为止, 大量被提出的缺陷预测模型采用了上述过程进行评估, 并且报告了相比于过往模型的显著优势. 考虑到这些论文的巨大数量以及几乎每个新提出的模型都宣称优于当时前沿的模型的结果, 学术界理应已极大提升了缺陷预测能力. 因此, 一个合理的预期是, 缺陷预测模型已经非常实用并且应当已被软件行业所接受. 然而, 现实并非如此. 由于学术界与实践质量保障开发人员的认知存在差距, 缺陷预测模型在工业界的应用效果并不如学术界实验所获得的同样高效[6−8]. 换言之, 缺陷预测模型的有效性极有可能被学术界高估了. 例如: 微软公司的 Zimmermann 和他的同事[6]表示, 按照他们的标准, 在工业级软件项目的 622 条缺陷预测中, 只有3.4%是实际有效的; 谷歌公司的Lewis等人[7]发现, 当尝试使用缺陷预测模型来支持代码审查时, 缺陷预测所提供的信息并没有导致谷歌开发者的行为可识别变化. 为何学术界倾向于高估缺陷预测模型的有效性? 根据我们的观察, 一个基本原因是缺乏评估新缺陷预测模型的有效性的一致框架. 具体来说, 研究者们对于如何进行模型间的比较实验缺乏一致的认识.

(1) 不同比较实验所选择的数据集质量不一. 当前被用于缺陷预测模型间的比较实验的缺陷数据集存在涵盖的编程语言单一[9]、数据集质量不齐[10]、数据集中所包含样本数量小[5,11]等问题, 可能威胁到比较实验得出的结论的有效性.

(2) 不同比较实验所采取的数据集划分方式不同. 某些比较实验仅使用交叉验证这种可能违反训练集-测试集的时序原则的数据集划分方式下进行验证[12]. 另外, 大量的比较实验倾向于仅在项目内缺陷预测[13]或跨项目缺陷预测[14]的单一场景下进行模型检验.

(3) 不同比较实验常常选择不同的过往模型与之比较, 缺乏一个被广为接受的基线模型. 结果是, 我们缺乏一个全局的参考点来确定缺陷预测领域进步的实际程度. 考虑到近年来许多提出来的模型并未开源且实现复杂(有多个参数需要被仔细地调节)[15−18], 新模型在与这类模型进行比较时需要重新实现这些复杂模型, 这可能导致被比较模型的性能退化, 从而在性能比较中无意间得到误导性的结论.

(4) 不同比较实验常常选择不同的评价指标来比较模型. 在哪些指标应当被用于评估缺陷预测的有效性上目前还并未达成共识. 在实际研究中, 不同的研究倾向于选择各自偏好的指标[4,9,14], 而如果不持续使用相同的性能指标, 则无法知道正在进行的预测性能中的真正进展.

(5) 参与比较的各个模型所花费的质量保障工作量往往不同(不同的代码审查工作量/上下文切换工作

量). 从实际应用的角度来看, 缺陷预测模型推荐项目中的部分被认为具有较高的缺陷倾向的模块用于审查或测试. 在文献中, 许多缺陷预测研究[9,18−20]是在不同的阈值也就是不同的给定工作量下比较这些模块的, 即它们的比较不是在一个公平的设置下进行的. 因此, 它可能导致对新提出的模型与过往模型的比较结果具有一定的偏差.

值得注意的是: 目前, 国内外学术界已有多份综述性的工作从多个角度对缺陷预测模型的技术研究进行了整理和归纳. 在 2008 年, Wang 等人[21]研究和讨论了软件缺陷预测技术的起源, 分析了 20 世纪 70 年代初到 2000 年初缺陷预测研究的发展和挑战, 并对这一时期的缺陷预测技术进行了分类讨论和比较. 在 2009 年, Catal 等人[22]从度量、预测方法和数据集这 3 个方面对过往的 74 项缺陷预测研究进行了系统回顾. 在 2011 年, Catal 等人[23]总结了从度量、预测方法、数据集、评价指标和实验结果这 5 个方面所进行的简单、有效的调研过往研究的流程, 系统回顾了 1990−2009 年间的 90 项研究, 并讨论了当时研究的发展趋势. 在 2012 年, Hall 等人[24]对 2000−2010 年间的 208 项研究进行了分析, 并综合了 36 项报告了充分背景和方法信息的研究的定量和定性结果. 他们发现: 大多数研究报告缺乏上下文和方法信息, 使潜在的模型用户很难选择一个匹配其环境的模型, 这也使得其他研究人员很难跨模型地进行元分析. 在 2013 年, Catal[25]表示, 半监督方法可以为标签数据有限的缺陷预测问题提供解决方案. 因而, 他在 NASA 数据集上评估了 4 种用于半监督缺陷预测的半监督分类方法. 在 2015 年, Malhotra 等人[26]对 1991 年 1 月−2013 年 10 月使用机器学习技术进行软件缺陷预测的文献进行了系统回顾, 并比较了机器学习技术与统计技术的性能, 总结了机器学习技术的优缺点. 在 2016 年, Chen 等人[27]认为缺陷预测的研究框架中的 3 个重要因素包括度量元的设定、缺陷预测模型的构建方法和缺陷预测数据集, 并从这 3 个因素归纳总结了截止到 2015 年 7 月的缺陷预测研究. 在 2018 年, Zhou 等人[28]以简单的无监督模型为基准, 对 CPDP 模型的性能进行了大规模评估. 在 2018 年, Herbold 等人[29]复现、开源并评估了从 2008−2015 年间的 24 个 CPDP 方法. 在 2019 年, Hosseini 等人[30]对 CPDP 模型从度量、模型、数据集、数据处理方法以及对应的预测性能的大规模文献加以回顾. 在 2019 年, Cai 等人[31]从数据标注、特征提取和模型评估等方面对近年来即时缺陷预测研究进展进行了梳理和总结. 在 2019 年, Gong 等人[32]汇总了 2010−2017 年国内外的缺陷预测研究进展情况, 从软件缺陷数据集、构建模型的方法及评价指标这 3 个方面进行了归纳分析. 在 2020 年, Tantithamthavorn 等人[15]总结了缺陷预测模型中的类不平衡技术对模型性能和解释性的影响, 并且对缺陷预测模型的模型验证技术进行了大规模对比[33]. 在 2020 年, Amasaki 等人[34]对跨项目缺陷预测技术(cross-project defect prediction, CPDP)进行了回顾, 并对 CPDP 技术迁移到项目内缺陷预测(within-project defect prediction, WPDP)场景的预测效果进行了大规模实验验证. 在 2020 年, Li 等人[35]通过大规模的文献回顾, 用荟萃分析等研究方法调查了无监督学习技术在软件缺陷预测中的应用和性能. 在 2021 年, Chen 等人[36]对 5 种异构缺陷预测方法和 4 种无监督缺陷预测方法从非工作量感知、工作量感知和识别缺陷模块的多样性这 3 个方面进行了大规模的实验比较, 实验结果表明, 异构缺陷预测方法的性能并不明显优于无监督缺陷预测方法. 在 2021 年, Xu 等人[37]对基于聚类的无监督缺陷预测模型进行了综合比较研究, 归纳整理并实验比较了 40 种聚类模型在 27 个项目版本的 3 类特征上的预测性能. 在 2022 年, Ni 等人[38]利用 Herbold 等人[29]提出的跨项目缺陷预测比较的基准工具包 Crosspare, 在相同的实验设置下, 比较了有监督跨项目缺陷预测方法和被 Zhou 等人[28]推荐为基线模型的无监督缺陷预测方法在工作量感知和非工作量感知情景下的预测性能.

但是据我们所知: 迄今为止, 还没有一项工作对缺陷预测模型间的比较实验进行总结(截至 2022 年 3 月). 新的缺陷预测模型要通过与过往模型的比较实验来评价其性能, 不同技术提升缺陷预测性能的效果同样需要通过模型间的比较实验来分析. 缺陷预测模型间的比较实验的实验结果将为缺陷预测的研究方向提供指导——在不断进行的比较实验中选出更优的模型, 进而不断提升缺陷预测的性能. 因此, 规范并公平地进行缺陷预测模型间的比较实验, 对于缺陷预测领域而言极为重要. 先前的进行模型间比较实验的论文体量已较为庞大, 然而对于如何规范并公平地进行缺陷预测模型间的比较实验却鲜有讨论. 为了弥补这一空白, 调研如何建立规范公平的比较实验框架, 本文大规模地评估了当前的缺陷预测模型间的比较实验. 具体来说, 本文

回顾了 2005–2021 年间的缺陷预测模型间的比较实验, 从缺陷数据集、数据集划分、基线模型、评价指标、阈值选择这 5 个方面对现有做法进行了分类介绍和总结, 分析了先前的比较实验所存在的挑战, 并在此基础上提出未来有关缺陷预测模型的比较实验的研究方向.

本文第 1 节首先概述 SDP 模型间比较实验的主要研究问题. 第 2 节说明本综述的文献检索方式和文献汇总信息. 第 3 节汇总 SDP 模型间比较实验中的缺陷数据集并分析在数据集使用上存在的问题. 第 4 节介绍 SDP 模型间比较实验所采取的数据集划分方法. 第 5 节介绍 SDP 模型间比较实验中的基线方法. 第 6 节介绍 SDP 模型间比较实验的评价指标. 第 7 节介绍 SDP 模型间比较实验中的阈值选择. 第 8 节分析 SDP 比较所面临的挑战以及未来的研究方向. 最后, 第 9 节总结本文的内容.

1 研究问题

1.1 SDP模型的构建流程

软件缺陷预测模型的基本假设是, 特定的软件度量(即属性)与软件模块(包、文件、类或函数)的缺陷之间存在相关性. 换句话说, 可以使用与模块缺陷有关联性的软件度量来构建缺陷预测模型. 其中, 有监督技术是缺陷预测领域的主流技术. 图 1 展示了应用有监督技术构建和评估缺陷预测模型的通用框架.

- 首先, 给定一组可用的软件项目, 每个项目都包含一个或多个版本, 这些版本中每个模块的缺陷情况是已知的(有缺陷或无缺陷). 提取这些版本的模块粒度的度量, 并将模块度量与模块的缺陷标签相关联, 以构建缺陷数据集;
- 然后, 从缺陷数据集中划分训练集和测试集;
- 接着, 在训练集上使用学习算法(例如随机森林)构建学习器, 并将该学习器用于预测测试集中模块的缺陷倾向性: 测试集中, 每个模块将被预测为有缺陷或无缺陷. 有些学习器不仅可以生成二分类结果, 还可以产生一个具体的风险数值, 降序排列后, 缺陷可能性大的模块排在缺陷可能性小的模块前面. 对无监督模型来说, 没有模型训练的过程, 这类模型通常直接被设计出来为测试集中的模块生成预测结果;
- 最后, 根据分类和排序结果计算分类和排序性能指标, 以评估预测模型在测试集上的性能. 性能评估过程通常涉及统计分析.

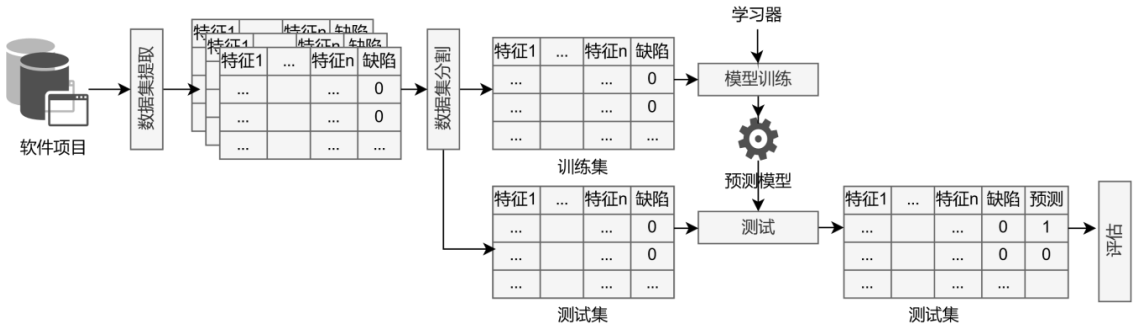


图 1 构建与评估有监督缺陷预测模型流程概览

1.2 SDP模型的比较实验

在第 1.1 节中, 我们介绍了单个 SDP 模型为测试数据集中的模块生成预测结果的过程. 在软件缺陷预测研究中, 常常进行多个 SDP 模型在同一测试数据集上的比较实验. 这样的比较实验通常在下面两种场景下进行: (1) 为了评估新提出的 SDP 模型的有效性, 因而选择一些当前已被评估过表现良好的 SDP 模型作为基线模型, 通过比较实验, 表明新模型的预测能力优于基线模型; (2) 同时比较评估多种 SDP 模型, 以得到某种规律或得到某个问题的答案, 如“跨项目缺陷预测模型是否可以预测项目内缺陷”和“哪个缺陷预测模型在实验

设置下表现最好”. 如图 2 所示, 本节介绍当前 SDP 模型间比较实验的一般流程.

(1) 选择或自行收集合适的数据集, 通常使用已有的公开数据集;

(2) 使用特定的数据集划分方式, 得到一系列训练集-测试集对;

(3) 对每个被比较的模型, 先使用训练集构建模型, 然后在测试集上进行预测, 得到模块的缺陷可能性的排序;

(4) 每个模型根据各自的阈值选择策略, 将模块分类为有缺陷或无缺陷, 比如图中红色模块意为被划分为有缺陷, 绿色模块意为被划分为无缺陷;

(5) 根据模块缺陷可能性的排序以及模块的实际缺陷标签来计算各个模型的一系列评价指标(排序指标和分类指标);

(6) 重复步骤(3)–步骤(5), 直到所有的训练集-测试集对都被用于模型训练和预测, 得到在每个测试集上的一系列评价指标值;

(7) 对上述过程得到的评价指标值进行统计分析, 得到比较结论, 例如“新模型是否能在大多数测试集上获得显著优于基线模型的召回率”.

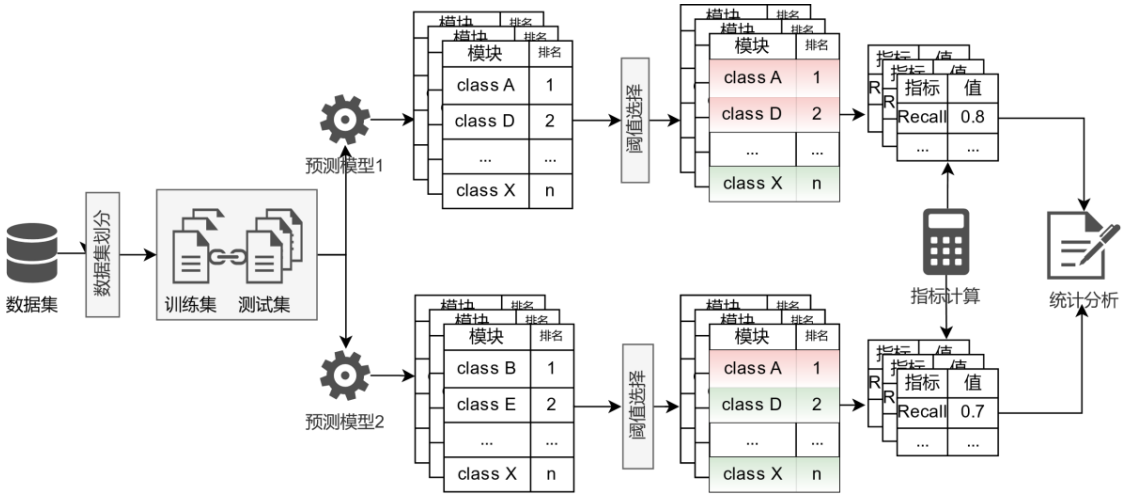


图 2 缺陷预测模型间比较实验的一般流程

从上述场景我们可以看出, 在 SDP 模型间的比较实验中, 除了预测模型本身的特性以外, 以下主要因素可能会影响模型间的比较结果: 模型比较中基线模型的选择、模型比较所选用的评价指标、模型比较的阈值设定、模型比较所选用的缺陷数据集和数据集的划分方法. 本文通过大规模的文献回顾发现: 在现有的 SDP 模型间的比较实验中, 以上这些因素的实验设置上存在影响比较实验公平性的问题. 因此, 本文分章节分析总结, 指出现有的影响 SDP 模型间的比较实验的公平性的问题(第 3 节–第 7 节), 并在第 8 节总结目前 SDP 模型间的比较实验有欠公平的设置对 SDP 研究带来的挑战以及未来可能的研究方向.

1.3 SDP模型的评估场景

本节进一步解释第 1.1 节和第 1.2 节中给出的 SDP 模型间的比较实验的流程中, SDP 模型在测试集上进行预测并最终得到评价指标值的过程, 并解释为什么 SDP 模型得到分类结果的一般方法是在排序结果上使用分类阈值. 这引出了为什么分类阈值的选择对于模型间的比较实验来说非常重要(将在第 7 节给出详细解释).

如前文所述, 被比较的 SDP 模型各自得到在测试集上的预测结果后, 通常计算两类指标: 分类指标和排序指标. 这两类指标也就对应了 SDP 模型的两种常见的评估场景: 分类场景和排序场景. 如第 1.1 节所述, 分类模型将目标数据集中的模块预测为有缺陷或无缺陷, 而排序模型将目标数据集中的模块从高到低按照包含缺陷的可能性排序. 一个 SDP 模型属于分类模型还是排序模型与其所使用的学习器的特性有关. 部分学习器

(例如决策树或规则集)仅产生类别决策, 本身就无法产生一个表示模块缺陷可能性大小的分值; 而另外一些学习器(例如朴素贝叶斯分类器或神经网络)则能自然地产生一个概率值或分数, 表示实例多大程度上属于某一类. 前者被称为硬分类器, 而后者被称为软分类器[39]. 显然, 使用硬分类器的 SDP 模型将只能产生二分类的预测结果(有缺陷或无缺陷), 而使用软分类器的 SDP 模型将得到按照缺陷可能性从大到小的模块排序. 在分类场景中, 预计开发人员将测试/审查所有被预测为有缺陷的模块; 而在排序场景中, 开发人员可以从排序中从前往后选择模块以进行测试/审查, 直到可用的资源耗尽.

然而, 分类和排序场景并不是完全分立的.

- 一方面, 排序模型可以简单地像分类模型一样对模块进行分类: 排序模型在目标数据集上得到每个模块的预测数值并据此产生模块缺陷可能性的排序后, 可以使用分类阈值来将模块分为有缺陷模块和无缺陷模块两部分. 例如, 基于随机森林分类算法的预测模型为测试项目中的模块给出了预测分数(范围为 0–1), 使用默认分类阈值 0.5, 可以将模块分为两类: 预测分数大于 0.5 的模块将被分类为有缺陷, 其余模块将被分类为无缺陷;
- 另一方面, 通过添加一些简单的规则, 使用硬分类器的分类模型可以像排序模型一样给出目标数据集中的模块的缺陷可能性排名. 这里举出一个可行的排序规则: 首先, 按分类标签对模块降序排序(预测为有缺陷的模块标签为 1, 预测为无缺陷的模块标签为 0); 然后, 对于具有相同标签的模块, 按照模块的源代码行数(SLOC)降序排列. 这个示例规则基于模块越大越可能有缺陷的假设.

由于从分类模型得到模块的缺陷可能性排序和从排序模型得到模块的分类结果都不难, 因此分类场景和排序场景的区分并不严格. 在 SDP 研究领域中, SDP 模型通常在分类和排序两种场景下被评估[15,18,19,40,41]. 缺陷预测领域常见的分类器大部分是软分类器, 因此通常在得到排序结果后, 使用分类阈值得到分类结果(如第 1.2 节所示).

2 文献检索

2.1 文献筛选与检索

本文通过文献检索来梳理过往的 SDP 相关工作中, SDP 模型间的比较实验. 本文主要关注的是先前相关工作中: 在比较实验中, 选择了哪些缺陷数据集以及比较实验采用的数据集划分方式? 使用了哪些基线方法与新提出的模型进行比较? 在比较中使用了哪些评价指标? 被比较的模型如何选择各自的分类阈值? 提出新的 SDP 模型以及评估 SDP 模型一直是 SDP 领域的热门方向. 本文的参考文献力图覆盖 2005–2021 年发表在国际重要期刊及会议的符合“进行了 SDP 模型间的比较实验”的研究工作. 具体来说, 本文使用以下步骤来进行相关文献的检索和筛选.

(1) 检索目标: 我们使用 DBLP Computer Science Bibliography 论文数据库提供的批量检索 API. 缺陷预测模型相关的研究工作众多, 由于要在每项研究上进行缺陷数据集、数据集划分方法、基线模型、评价指标和阈值设定等多方面的详细实验设置, 我们力求从众多缺陷预测工作中选择具有代表性的那部分进行调研. 因此, 我们将检索的来源限定在软件工程方向权威的国际会议 ESEC/FSE、ICSE、ASE 和国际期刊《IEEETSE》和《ACMTOSEM》上;

(2) 检索关键词: 为了找到软件缺陷预测相关的工作, 我们使用如下字符串进行搜索: (“defect” OR “defects” OR “defectiveness” OR “defective” OR “bug” OR “buggy” OR “fault” OR “faults”) AND (“predict” OR “predicting” OR “predicts” OR “predicted” OR “prediction”). 我们在目标数据库中对文献的题目使用上述关键词进行搜索;

(3) 检索起止时间: 检索 2005–2021 年共 17 年间的文献;

(4) 文献审查: 图 3 给出了文献检索与过滤方法以及经过每个步骤后得到的文献数量: 首先, 在 DBLP 中检索到了 132 篇文献; 接着, 为保证文献质量对文献进行初步过滤, 考虑到会议中的 workshop 文章通常篇幅短小、描述简介, 不便于我们从中提取归纳实验设置的细节, 因此进一步去掉会议中的

workshop 文章，得到 89 篇文献，然后通过阅读文献去除与研究主题无关的文献，得到最终的 66 篇文献. 在最后一步中，我们采取了预设的选入与排除标准来决定文献是否与研究主题相关. 表 1 详细说明了文献选入与排除的标准.

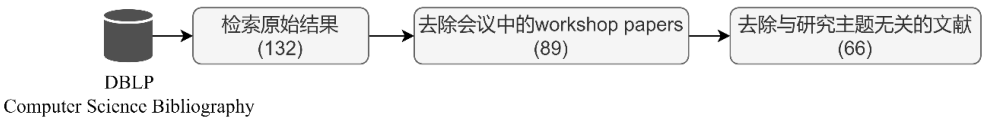


图 3 文献检索与过滤流程

表 1 系统性的文献回顾的选入/排除标准

选入标准	
(1)	论文必须是关于缺陷预测的实证研究：它必须包括在一组数据上训练模型，并对模型的有效性进行分析，目的是预测软件模块中的缺陷，而不是缺陷预测相关的其他种类的研究，如评估开发人员对缺陷预测技术的信念、根据逻辑和经验评价过往预测模型的优缺点等
(2)	论文必须对涉及到的缺陷数据集、数据集划分方法、评价指标、基线模型和阈值设定进行清晰的说明
(3)	论文必须是同行评议的完整研究论文，发表在会议论文集或期刊上
排除标准	
(1)	进行缺陷预测技术与其他技术的比较，而不是对不同的缺陷预测技术进行比较. 如比较缺陷预测技术和静态程序分析技术
(2)	预测的目标不是模块有缺陷与否，而是缺陷数量、缺陷密度、缺陷定位(代码行)、缺陷修复时间、已关闭缺陷的重开与否、哪些缺陷在下一版本中会被修复等
(3)	论文对过往的缺陷预测研究进行纯粹的元分析，没有模型训练与测试的过程

2.2 历史文献的汇总

经过文献的检索与筛选，本文收集了 SDP 模型相关的学术论文共 66 篇. 这些研究论文将会按照其在缺陷数据集选择、数据集划分、基线模型、评价指标以及阈值设定上的实验设置，分别在第 3 节–第 7 节进行重点介绍. 图 4 展示了 2005 年以来，SDP 模型比较相关的研究发表的论文数量、论文涉及到的作者数量、论文发表时作者所属机构数量的分布情况.

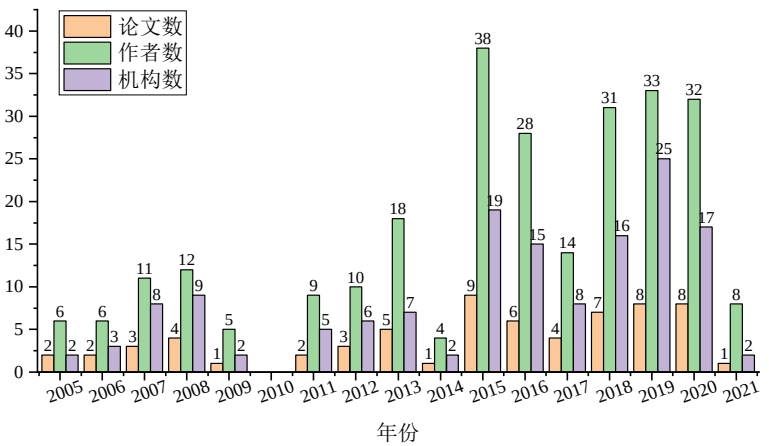


图 4 SDP 模型研究文献的年代分布

从图 4 中可以看出：该领域每年发表的论文数量、作者数量与机构数量都整体呈现逐年上升的趋势，尤其是 2011–2021 年的这 10 年间，几乎每年都有多篇相关论文发表在国际权威期刊和会议上.

图 5 列出了 SDP 模型的研究论文在出版物上的发表分布情况，其中，期刊类论文有 34 篇，包括 TOSEM 论文 2 篇[28,42]和 TSE 论文 32 篇[1,4,5,12,15,17,18,20,24,30,33,40,43–62]；会议类论文有 32 篇，包括 ASE 论文 5 篇[2,11,19,63,64]、

FSE 论文 6 篇[3,6,16,65–67]、ICSE 论文 21 篇[9,13,14,29,41,68–83]，可见软件工程领域对 SDP 模型的重视.

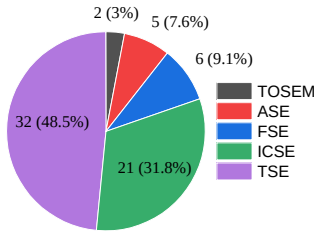


图 5　SDP 模型研究文献的出版物分布

我们进一步分析了近年来逐渐多样化的 SDP 模型的研究方向及进展. 图 6 列出了 SDP 模型多样化的研究进展及其相关的重要文献, 相同颜色代表同一类研究课题.

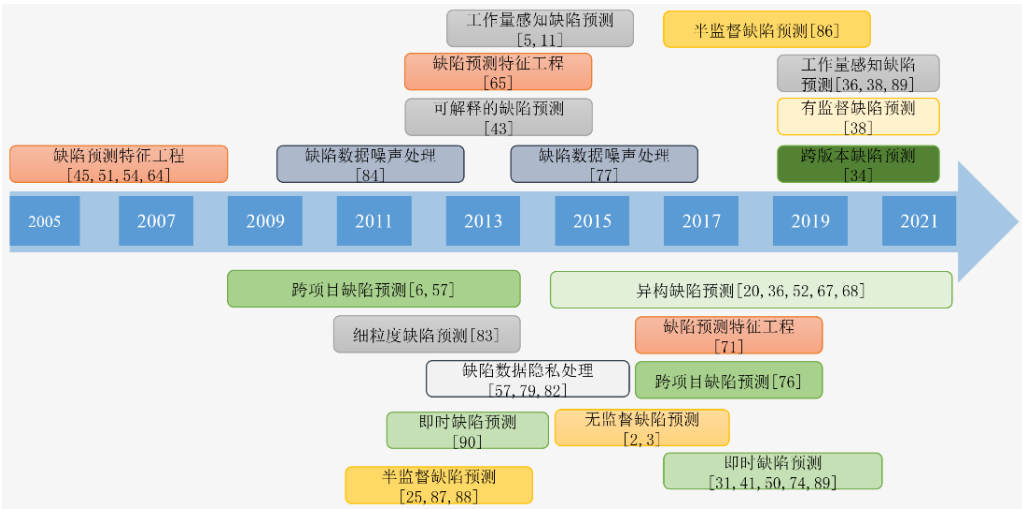


图 6　SDP 模型多样化的研究进展

可以看出: 逐年来, 多样化的分类算法[2,3,25,38,84–86]、缺陷预测场景[6,20,34,36,51,55,65,66,74]、工作量感知[5,11,36,38,87]、缺陷预测特征工程[45,50,52,62,63,69]、缺陷数据集噪音处理方法[75,82]、缺陷数据隐私处理方法[55,77,80]被陆续提出. 从分类算法的种类来看, 主要有无监督[2,3]、有监督[38]及半监督[25,84–86]缺陷预测方法被提了出来; 从缺陷预测场景来看, 有跨项目缺陷预测[6,55,74]、跨版本缺陷预测[34]、异构缺陷预测[20,36,51,65,66]、即时缺陷预测[31,41,49,72,87,88]等被提出; 从工作量感知与否, 又可分为工作量感知的缺陷预测[5,11,36,38,87]和非工作量感知的缺陷预测. 从整体上逐年增多的相关论文发表数量、在国际权威期刊和会议发表的论文数量以及 SDP 模型的研究方向的多样化趋势来看, SDP 模型的研究越来越受到研究者的重视.

为了更进一步地了解现有的大量缺陷预测模型在实践者中的实际应用情况以及工业界对于软件缺陷预测模型的看法, 我们收集汇总了 2009 年以来的部分调查软件从业人员对缺陷预测模型技术评价的研究[6–8,87,89]. 表 2 中的“组织”一列指的是作者在文中明确提到的被调研对象所属的组织, 可以看到, 该项汇总囊括了谷歌、微软、阿里巴巴这 3 家大规模企业的从业人员对于缺陷预测模型的看法[6,7,87]以及 GitHub 开发者和招募于 Amazon Mechanical Turk 的开发者的意见[8,89]. 通过简要介绍他们的实验方法以及相应的结论我们发现: 整体上, 工业界对缺陷预测模型的评价趋于消极, 缺陷预测模型与投入实际应用之间还有一定距离. 例如: 在 2020 年, Yan 等人[87]的调研发现, 仅在 33%的情况下, 从业人员会按照部署在阿里巴巴内部的缺陷预测工具提供的缺陷预测来进行代码修改. 这意味着在绝大多数情况下, 从业人员认为缺陷预测工具提供的预测是不可修改或者无需修改的. 而在学术界, 逐年提出的新的缺陷预测模型却在不断被证明提升了过往模型的预测性

能，在这样逐年新旧模型的比较实验所证实的性能提升情况下，令人惊讶的是，这些模型并未被大规模地投入到工业界使用，因此，我们考虑缺陷预测模型的有效性极可能被学术界高估了．这促使本文从缺陷数据集、数据集划分、基线模型、评价指标、阈值选择这 5 个方面对现有的缺陷预测模型间比较实验的实验设置进行分类介绍和总结，分析先前的比较实验存在的影响模型评估有效性的挑战．

表 2　调查软件从业人员对缺陷预测技术评价的研究汇总

年份	研究	作者	组织	简介	主要结论
2009	Ref.[6]	Zimmermann 等人	微软	作者认为在实际应用中，跨项目缺陷预测模型需要在精度、召回率和准确率这 3 个指标上都达到 0.75 以上的性能，才能被认为是有实用价值的	在 622 个跨项目的训练集-测试集对上，仅有 3.4%达到了他们的标准．说明跨项目缺陷预测模型与实际应用之间还存在很大距离
2013	Ref.[7]	Lewis 等人	谷歌	为研究缺陷预测模型是否对指导人类开发人员有价值，作者在谷歌的某项目上部署了一个缺陷预测模型	实验发现，该缺陷预测模型对于开发人员的行为没有可见的影响．开发者对缺陷预测模型的反馈褒贬不一但总体倾向于消极
2020	Ref.[8]	Wan 等人	–	开放式访谈结合调查问卷通过定性和定量的混合研究调查 Github 开发者对于缺陷预测的看法、行动和预期．该研究收集了五大洲超过 33 个国家的从业人员的 395 份答复	超过 90%的受访者有使用缺陷预测技术的意愿；在缺陷密度分布和文件大小与缺陷性之间的关系方面，从业人员的看法与得到充分支持的研究证据之间存在脱节；在缺陷修复过程中，超过 40%的受访者表示他们会做一个“变通”的修复，而不是修正实际的致错代码
2020	Ref.[87]	Yan 等人	阿里巴巴	在阿里巴巴部署了一个基于 CBS+方法的工作量感知即时缺陷预测工具，进行用户研究以调查该工具在真实的工业环境中的有效性	用户调研发现：仅在 33%的情况下，开发者会去修复缺陷预测工具提供的被预测为有缺陷的模块
2021	Ref.[89]	Jiarpakdee 等人	–	定性调查从业人员(招募于 Amazon Mechanical Turk)对缺陷预测模型的目标和用于生成缺陷预测模型的可视化解释的模型诊断技术的看法	82%–84%的受访者认为，缺陷预测模型能够为更有缺陷风险的文件赋予更高的优先级、能够理解与缺陷相关的软件特性、能够理解哪些软件特性对软件缺陷贡献最大；该研究强调了提高模型可理解性对于缺陷预测走向实际应用的意义

3　缺陷数据集

本节首先汇总过往文献中使用的缺陷数据集，并简要介绍它们的收集情况；接着，分析这些缺陷数据集在被现有的 SDP 模型间的比较实验使用时存在的问题；最后，分析总结现有的 SDP 模型间的比较实验中缺陷数据集的使用情况．

3.1　缺陷数据集汇总

在 SDP 研究领域，存在一些研究者收集并公开的缺陷数据集，这些数据集提供了项目中模块的某些特征值和模块的缺陷情况．SDP 模型的比较实验往往在公开的缺陷数据集上进行实验，以供其他研究人员复现、对比以及继续进一步的研究工作．由于不同的数据集是在不同软件开发团体、不同语言和不同项目上收集的，且采用的缺陷识别方法以及收集的特征各有不同，因而在不同数据集上进行的模型比较实验的结论可能存在出入．表 3 汇总了 SDP 领域的公开可用数据集，包括提出年份、项目的编程语言、数据集来源、缺陷收集的粒度，最后两列分别是在本文的文献回顾范围内使用缺陷数据集的次数及使用文献列表．另外，为了厘清汇总到的缺陷数据集的版本迭代情况，我们通过检查数据集原始文献的后续引用文献的方法，检查数据集的初始版本后是否有迭代版本．具体而言，现有的数据集版本迭代可分为如下两种情况：(1) 原作者对数据集进行扩充(如 Jureczko 数据集)；(2) 其他研究人员对数据集进行噪声清洗(如 NASA 数据集)或特征扩充(如 Eclipse 数据集)、实例粒度增加(如 GithubBugDataset)等．

表 3 缺陷预测研究使用的公开数据集汇总

缺陷数据集	首次发表年份	语言	来源	粒度	使用数	使用列表
NASAMDP (PROMISE)	2005	C/C++/JAVA/Perl	Ref.[90]	函数	10	Ref.[1,19,29,42, 43,50,57,62,66,78]
Eclipse	2007	JAVA	Ref.[94]	文件/包	7	Ref.[13,15,17, 33,42,43,70]
Kim	2008	C/C++/JAVA/Perl/ Python/Java Script/ PHP/XML	Ref.[96]	代码提交	2	Ref.[33,82]
SOFTLAB	2009	C	Ref.[97]	函数	9	Ref.[12,19,20, 42,46,51,65,66]
AEEEM	2010	JAVA	Ref.[98]	类	16	Ref.[4,12,14,15,17,19,20, 29,33,46,51,64–66,70,74]
JURECZKO	2010	JAVA	Ref. [99,100]	类	27	Ref.[4,9,12,14,15,17–20, 29,33,40,51,54,55,59,64, 66,68–70,73,74,77,78,80]
ReLink	2011	JAVA	Ref.[101]	文件	14	Ref.[2,12,14,15,17,19, 20,29,46,51,64–66,70]
JIRA	2013	JAVA	Ref.[102]	文件	–	–
Clean NASA	2013	C/C++/JAVA/Perl	Ref.[10]	函数	12	Ref.[12,15,17,20,33, 46,51,65,70,74,78]
COMMIT	2013	C/C++/JAVA	Ref.[88]	代码提交	2	Ref.[3,12]
NetGene	2013	JAVA	Ref.[103]	文件	3	Ref.[2,12,29]
Bugcatcher	2014	JAVA	Ref.[104]	文件	–	–
GithubBugDataset	2016	JAVA	Ref.[105]	类和文件	–	–
ELFF	2016	JAVA	Ref.[106]	方法和类	1	Ref.[4]
UnifiedJava	2018	JAVA	Ref.[107]	类和文件	–	–
Cabral	2019	Java/JavaScript/Python	Ref.[41]	代码提交	2	Ref.[41,72]

数据集相关介绍如下.

(1) NASAMDP 数据集来自于 NASAMetricsData Programs[90]. 2005 年, Tim 和 Jelber 主导了 Int'l Workshop on Predictor Models in Software Engineering (PROMISE)以提升研究界对软件工程模型的信心[91]. 一系列软件工程相关的数据集被公布在 PROMISE 中, 而在缺陷预测领域, PROMISE 数据集[92]主要指的是 Tim 和 Koru 等人贡献的来源于 NASAMDP 的缺陷数据集. NASA MDP 中, 不同项目所包含的度量数目各不相同, 从 21 到 39 不等. 在 2013 年, Shepperd 等人发现 NASAMDP 数据集存在大量噪音, 并且对 NASAMDP 数据集进行了清洗, 得到 CleanNASA 数据集[10]. 尽管 CleanNASA 数据集可以视作 NASAMDP 数据集的一个后续版本, 但是考虑到提出 CleanNASA 的这篇论文对于缺陷预测领域的影响力较大, 后续大量文献特别指出使用的是 CleanNASA 数据集而不是原始 NASAMDP 数据集, 因为为了区分使用这两个数据集的文献, 我们在表 2 中分别列出了原始 NASAMDP 数据集及 CleanNASA 数据集. 在 2016 年, Petrić 等人[93]又在 Shepperd 等人[10]的基础上额外增加了规则, 以进一步移除 CleanNASA 数据集中仍然存在的错误数据, 但我们并未在该文献[93]中找到进一步清洗后的数据集链接;

(2) Eclipse 数据集包含了 EclipsePlatform 系统(来自开源 Eclipse 基金会)在 2002 年、2003 年、2004 年的 3 个版本(release 2.0、2.1 和 3.0)的数据, 由 Zimmermann 等人[94]在 2007 年收集. 在 2008 年, Moser 等人[95]为 Eclipse 数据集中的实例收集了一组与软件开发过程相关的过程度量, 并实验对比了过程度量与 Eclipse 原本的静态代码度量构建的缺陷预测模型的性能. 但我们在 Moser 等人[95]的论文中并未找到过程度量版本的 Eclipse 数据集的开源链接;

(3) Kim 数据集是指 Kim 等人[96]在 2008 年收集的代码提交粒度上的缺陷数据集, 该数据集包含 12 个开源项目(包括 Eclipse、Bugzilla 和 Mozilla 等知名开源项目), 这些项目所使用的编程语言涵盖 C/C++、Java、Perl、Python、Java Script、PHP 和 XML. 随后, 在 2013 年, Kim 等人提出了一种名为 CLNI 的缺陷数据集中的噪音实例识别方法, 并在论文中使用 CLNI 对 Kim 数据集中的部分项目进行了清洗;

(4) Turhan 等人[97]从一家土耳其软件公司收集了包含 6 个项目的 SOFTLAB 数据集, 它们是 C 语言编写的家用电器嵌入式控制器. SOFTLAB 项目包括 29 个静态代码特征, 其中 17 个与 NASA MDP 数据集一样;

(5) AEEEM 是由 D'Ambros 等人[98]收集的包含 5 个 Java 项目的缺陷数据集, 最初用于比较不同度量集合的表现. 因此, AEEEM 中含有多达 61 个特征[74];

(6) JURECZKO 数据集是由 Jureczko 等人[99,100]在开源 Java 项目上收集的, 包括 McCabe 圈复杂度、CK 特征和其他面向对象的特征. 在 2010 年, Jureczko 和 Spinellis[100]收集并发布了一个缺陷数据集, 其中包含了 11 个开源项目和 5 个商业软件项目. 随后在同一年, Jureczko 和 Madeyski[99]收集并发布了一个扩充版本以支撑进一步的实验, 该扩充版本包含了 15 个开源项目的 48 个版本、6 个商业软件项目的 27 个版本以及 17 个由学生所开发的学术软件项目. 通常来说, JURECZKO 数据集指的是在这两篇工作中[99,100]由 Jureczko、Spinellis 和 Madeyski 这 3 人所收集到的全部数据集, 而后续的使用者们绝大多数使用的是 JURECZKO 数据集中收集自开源项目的那一部分[4,9,12,14];

(7) ReLink 数据集由 Wu 等人[101]收集, 数据集包含 2 个由 GoogleCode 托管的活跃的安卓项目以及 3 个 Apache 项目. 每个模块包含 60 个静态产品度量和 3 个不同的缺陷标签, golden 表示人工验证和修正过的缺陷标签, relink 表示用他们提出的方法自动生成的缺陷标签, traditionalheuristic 表示基于传统方法自动生成的缺陷标签. 通常来说, ReLink 数据集指代的是使用 golden 标签的 ReLink 数据集, 即缺陷信息是经过人工验证和修正的数据集版本;

(8) JIRA 数据集是 Bissyandé 等人[102]利用问题跟踪系统 JIRA 中的软件项目缺陷信息在 10 个 Apache 托管的开源项目上收集的. Bissyandé 等人[102]表示: ReLink 数据集的缺陷标签收集过程中使用的算法和过滤方法存在一些漏洞, 而 JIRA 数据集则是用更严谨的策略收集到的干净数据集, 但是作者并未用自己提出的新策略对 ReLink 数据集进行清洗来给出一个迭代版本;

(9) Kamei 等人[88]收集了 6 个知名的开源项目(BugZilla、Columba、Eclipse JDT、Eclipse Platform、Mozilla 和 PostgreSQL)的代码提交粒度的缺陷数据集, 因此该数据集常被称作 COMMIT. COMMIT 数据集经常出现在即时(just-in-time, 即在代码提交粒度进行缺陷预测)的 SDP 研究中[74];

(10) NETGENE 是 Herzig 等人[103]收集的包含了 4 个开源项目的缺陷数据集, 包含 465 个度量, 包括静态产品度量、网络度量、与模块的历史相关的度量, 如每次代码提交时的平均作者数量;

(11) Bugcatcher 指的是 Hall 等人[104]收集的 Eclipse、ApacheCommons 和 ArgoUML 项目上文件的 5 种代码异味以及对应的缺陷情况的缺陷数据集, 该数据集被 Hall 等人[104]用于研究代码异味对软件缺陷的影响. 本文的文献回顾范围内并未见到对该数据集的使用;

(12) GithubBugDataset 指的是 Tóth 等人[105]收集并生成的 15 个 Github 开源的 Java 项目上的 105 个版本缺陷数据集, 该数据包含类粒度和文件粒度的数据. 本文的文献回顾范围内并未见到对此数据集的使用;

(13) Shippey 等人[106]使用改进的 SZZ 算法提取缺陷信息并使用 JHawk 计算度量, 生成了包含 23 个系统共 138 个版本的方法和类粒度的缺陷数据集 ELFF;

(14) UnifiedJava 数据集由 Ferenc 等人[107]收集, 他们汇总了前人提出的 5 个缺陷数据集 AEEEM[98]、Eclipse[94]、PROMISE[92]、Bugcatcher[104]和 GithubBugDataset[105], 通过分析处理这些数据集获得了这些数据集共同的度量集合, 并用共同的度量集合建立并公开了类粒度和文件粒度的统一数据集. 本文的文献回顾范围内并未见到对该数据集的使用;

(15) Cabral 数据集是指 Cabral 等人[41]从 10 个 Github 开源项目收集了代码提交粒度的缺陷数据集, 这 10 个项目是在持续时间超过 5 年、历史丰富(提交数大于 1 万)和引发缺陷的变更比例恰当(总体在 20%左右)的项目中随机选择的. 这个数据集提供的 14 个度量涵盖了 5 个维度的代码提交粒度指标: 更改的大小、更改文件的历史、更改的扩散、开发人员的经验和变更的目的.

3.2 缺陷数据集存在的问题

第 3.1 节表明: 近几年, 仍不断有新的缺陷数据集被收集, 反映了 SDP 领域对有标签数据的需求以及 SDP 领域仍然是软件工程的活跃领域. 尽管该领域已有一定数量的数据集被收集和使用, 但目前我们发现, 在 SDP 模型的比较实验中用到的缺陷数据集的收集和使用过程中仍存在一些问题需要得到重视, 这些问题可能会影响缺陷预测模型的比较实验的公平性.

(1) 涵盖的编程语言不全面. 从表 2 可以发现: 在缺陷预测模型比较实验中, 主要是在 Java 项目上进行比较实验, 其次是 C/C++, 并没有太多的比较实验是通过其他近年来流行并被大量使用的编程语言来实现, 例如在 Python、Javascript、Perl 等语言构成的数据集上进行的. 从我们文献综述的结果上来看: 很多近年来流行的编程语言在现有的缺陷数据集中甚至是缺失的, 例如 Go、Swift、Kotlin、Ruby 等. 这会导致比较实验的结论是否可以推广到其他编程语言上仍是未知的. 更多其他语言的度量抽取和缺陷识别方法应该被给予讨论, 以得到编程语言更丰富的缺陷数据;

(2) 数据集的质量参差不齐. 以上数据集的收集过程各有差异, 这将会导致数据集的质量各不相同. 例如: Gray 等人[108]指出, NASAMDP 数据集存在一些质量问题; 随后, Shepperd 等人[10]进一步分析了 NASA MDP 数据集[90]中存在的质量问题, 提出了一个预处理算法, 发布了一个被认为是干净的数据集 CleanNASA; 在 2013 年, Kim 等人[109]发现缺陷数据标签收集中存在广泛的误分类问题, 这会直接影响那些使用了误分类的数据集的工具和研究的准确性; 另外, 在 2022 年, Liu 等人[110]对多版本项目缺陷数据集中不一致的缺陷标签(即在一个软件项目的多个版本中许多实例具有相同的源代码但标签不同)进行了系统的研究, 并建议研究人员对现有的缺陷标签收集方法进行改进, 以减少不一致标签的产生, 并在使用缺陷数据集时, 检测并排除其中的不一致标签, 以避免它们对缺陷预测的潜在的负面影响. 为了确保 SDP 模型评估的结果可信, 模型间的比较实验必须在质量可信的数据集上进行. 因此, 当前数据集质量参差不齐的问题可能会对模型间比较的有效性产生潜在威胁;

(3) 可获取的数据集版本迭代不足、更新缓慢. 在文本汇总到的缺陷数据集范围内, 从噪音清洗的角度看, 仅有 NASAMDP 数据集[90](2005 年)先后经历了 CleanNASA[10](2013 年)和 Petrić 等人[93](2016 年)对此数据集的噪音清洗, 以及 Kim 数据集[96](2008 年)有一个清洗了部分项目上的噪音的迭代版本[82](2013 年); 从数据集规模扩充的角度看, 仅有 Jureczko 数据集[99,100]在 2010 年同一年, 先后发布了一个少量项目版本[100]和一个扩充版本[99]; 从特征增加或融合的角度看, 仅有 Eclipse 数据集(2007 年)有一个增加了过程度量的扩充版本(2008 年), 以及 UnifiedJava 数据集(2018 年)是对前人提出的 5 个缺陷数据集 AEEEM[98](2010 年)、Eclipse[94](2007 年)、PROMISE[92](2005 年)、Bugcatcher[104](2014 年)和 GithubBugDataset[105](2016 年)从度量上进行整合得到的一个统一版本. 除此之外, 其他数据集尚未有版本迭代和更新. 可见, 目前研究领域内可获取的数据集版本迭代不足. 另外, 在已有的数据集版本迭代与更新中, 可以发现总体上数据集更新较慢. 例如: 从 NASAMDP 数据集[90]到 CleanNASA[10]数据集之间 8 年的空白期间, 研究者们已经在不可靠的 NASAMDP 数据集上做了大量研究, 而这些研究的有效性都有可能由于 NASAMDP 数据集的噪音问题受到潜在威胁;

(4) 比较实验常在不充分的数据集上进行. SDP 模型间的比较实验倾向于选择一个或少量几个数据集进行实验, 而不是在多个缺陷数据集上进行大规模实验. 首先, 由于不同缺陷预测研究选用的缺陷数据集通常不同, 这不利于模型之间直接通过文献报告的实验结果进行性能比较; 其次, 由于各个缺陷数据集中的样本数量普遍不多, 因此仅在少量数据集上得到的模型间的比较结果的泛化性缺乏保障.

3.3 小 结

本节从缺陷数据集汇总和缺陷数据集上存在的问题这两个方面介绍了 SDP 模型间的比较实验中的缺陷数据集的使用情况. 下面简述主要发现.

(1) 通过汇总文献回顾中使用到的 12 个缺陷数据集，我们发现，其中使用最频繁的数据集包括 JURECZKO、AEEEM、ReLink、CleanNASA 和 NASAMDP;

(2) SDP 模型间的比较实验中用到的缺陷数据集在收集和使用过程中存在包括涵盖编程语言不全面、数据集质量参差不齐、数据集维护不活跃和比较实验常在不充分的数据集上进行这几个问题.

4 数据集划分方法

本节首先汇总不同的数据集划分方法，并介绍它们所对应的使用场景；接着，描述不同使用场景中的缺陷预测研究的进展；随后，分析现有的 SDP 模型间的比较实验中在选择数据集划分方法上存在的问题；最后，对数据集划分方法进行总结.

4.1 数据集划分方法汇总

表 4 汇总了在 SDP 研究领域中现有的主要数据集划分方法. 考虑一个缺陷数据集中有多个项目，每个项目包含一个或多个版本. 对于一个待预测的目标版本，所有可能用来对其进行预测的数据包括它的历史版本，以及目标版本所在项目以外的外部项目. 项目内缺陷预测(within-project defect prediction, WPDP)指的是仅使用目标版本的历史版本来进行模型训练；跨项目缺陷预测(cross-project defect prediction, CPDP)指的是使用目标版本所在项目以外的外部项目来进行模型训练；而交叉验证(cross-validation, CV)则指的是把项目或版本视作整体，在其中(通常以随机划分的方式)划分训练集和测试集.

表 4 缺陷预测评估中常见数据集划分方法汇总

类别	划分方法	描述
WPDP	SFV (single farthest version)	用项目内最旧版本(如果可用)训练模型，可能受到数据漂移的影响，但节省了反复用更新的版本训练模型的开销
	SPV (single prior version)	用项目内距离目标版本最近的先前版本(如果可用)来训练模型
	APV (all prior version)	用项目内目标版本的所有先前版本(如果可用)来训练模型
CPDP	CSFV (CPDP data with SFV)	CPDP 数据加上 SFV
	CSPV (CPDP data with SPV)	CPDP 数据加上 SPV
	CAPV (CPDP data with APV)	CPDP 数据加上 AFV
	C-one	仅用外部项目训练模型，且仅使用一个外部项目或一个外部版本来训练
	C-all	只用其他项目训练模型，使用全部外部项目来训练
Cross-validation	10×10 cross-validation	10×10 折交叉验证
Random splitand repeat	90/10 training/test set split	将数据集按 90%/10%的比例分割成训练集和测试集，重复实验 N 次得到结果

表 4 中, WPDP 和 CPDP 类别下的划分方法的命名部分沿用了 Amasaki[34]的命名.

(1) WPDP. 适用于目标版本有足够的历史数据用于模型训练的情况. 相比于外部项目，目标版本的历史版本通常被认为与目标版本拥有更相近的特征空间，而更少受到分布迁移对预测性能的影响. 因此, WPDP 通常被认为性能普遍优于 CPDP. 从理论上讲，对于数据集中某项目的多个可获取的版本 $(1,...,n)$，对于第 i+1 个版本，在 WPDP 场景下，它有 2^i–1 个可能的数据集划分方法，然而，这些划分方法中仅有少数常见于现有的 SDP 研究. WPDP 中常见的数据集划分方法有 SFV、SPV、APV，具体描述见表 3.

(2) CPDP. 适用于目标版本没有历史版本或历史版本中的数据不足以用来训练预测模型的情况. 由于可以使用外部项目进行训练，CPDP 可以利用外部组织发布的数据集来弥补训练样本的不足. 与 WPDP 的优点相对，CPDP 通常被认为预测性能不及 WPDP. CPDP 可大致分为严格 CPDP 与混合 CPDP: 严格 CPDP 指的是训练集只能使用目标项目之外的项目中的数据，而混合 CPDP 指的是训练集中的数据来自目标项目之外的其他项目中的数据以及部分本项目内的数据. CPDP 中常见的数据集划分方法有 CSFV、CSPV、CAPV、C-one 和 C-all，具体见表 3. 其中，CSFV、CSPV、CAPV 属于常见的混合 CPDP 方法: CSFV 指的是对于一个目标版本，使用它所在的项目中的最旧的版本以及

外部项目的数据作为训练集; CSPV 指的是对于一个目标版本, 使用它所在的项目中的前一版本以及外部项目的数据作为训练集; CAPV 指的是对于一个目标版本, 使用它所在的项目中的所有先前版本以及外部项目的数据作为训练集. C-one 和 C-all 则是严格的 CPDP 方法.

(3) *M*×*N* 折交叉验证. 该划分方法可以使所有的样本都能作为训练数据和测试数据, 并且该划分方法在缺陷预测中被广泛使用[54]. *M*×*N* 折交叉验证指的是将数据集分成 *N* 份, 在由 *N*–1 份数据合在一起训练得到预测模型后, 在剩下那一份数据上进行测试. 这个过程会重复 *N* 次, 直到每一份都既做过训练集也做过测试集. 最后, 上述过程会重复 *M* 次, 每次都把数据随机地分出 *N* 折. 总的来说, 预测并测试的过程会有 *M*×*N* 次. 那么, 10×10 折交叉验证重复实验 100 次. 尽管交叉验证方便、快速, 无需目标数据集的历史版本数据, 只需在目标数据集内部随机划分训练集和测试集, 但交叉验证方法由于可能违反时序原则(即训练集中的模块开发时间必须早于测试集中模块的开发时间, 必须用先开发的模块来预测后开发的模块), 因而并未有现实的应用场景与之对应.

(4) 随机划分并重复实验. 除了 *M*×*N* 折交叉验证之外, 随机划分并重复实验是另外一种把数据集的全部实例作为整体随机划分训练集和测试集的方法, 这种方法随机抽取 *x*%数据作为训练集, 其余 1–*x*%作为测试集, 并且重复 *N* 次, 得到模型性能. 该方法可以简单、快速地验证模型效果, 但与交叉验证方法一样, 可能会违反时序原则, 因而并未有现实的应用场景与之对应.

图 7 统计了本文的文献回顾范围内的数据集划分方式的分布情况. 横轴表示 3 种不同的数据集划分方式, 纵轴表示每种划分方式出现在本文所调查的模型间比较实验中的次数. 例如: 在本文的文献回顾范围内, 某篇文献既在 CV 划分方式下进行了模型间比较实验, 也在 CPDP 划分方式下进行了模型间比较实验, 那么图 7 所统计的相关文献中, 数据集划分方式的 CV 与 CPDP 的次数就各自加 1. 可以看出: CV(28 次)是最常用的数据集划分方式, 其次是 CPDP(22 次), 最后是 WPDP(9 次). 从使用场景上来看, 2005 年以来发表在国际权威期刊及会议上的 CPDP 场景下的缺陷预测研究工作远远多于 WPDP 场景下的缺陷预测研究工作.

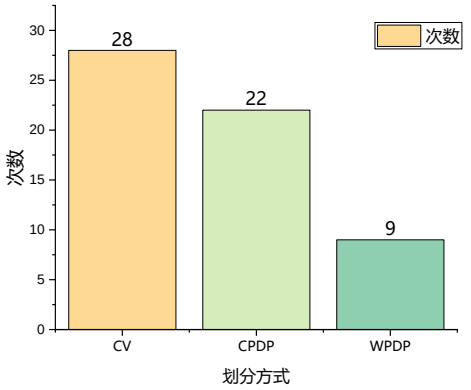


图 7 SDP 文献中数据集划分方式的分布情况

4.2 数据集划分方法对应的场景

不同的数据集划分方法被认为对应不同的使用场景. 因而在文献中, 数据集划分方法的选择通常是为了验证缺陷预测模型在某个特定的使用场景下的预测性能. 经常可以看到: 研究者为特定的使用场景提出缺陷预测模型, 并且只在这种使用场景下对该模型进行评估, 即仅在这种使用场景下进行模型间的比较实验.

4.2.1 WPDP 场景

在 2008 年, Moser 等人[13]以 SPV 划分方法在 WPDP 场景下, 在 Eclipse 数据集上比较了过程度量和代码度量进行缺陷预测的有效性, 结论是对 Eclipse 数据集来说, 过程度量比代码度量在缺陷预测上更有效. 值得注意的是, Eclipse 数据集只包含 3 个版本, 也就是说, Moser 等人的结论是仅在两个测试集上得到的. 在 2018 年, Wen 等人[18]在 JURECZKO 的部分项目上, 以 SPV 划分方法在 WPDP 场景下进行了缺陷预测模型间的比

较实验, 以验证他们提出的新度量是否优于传统度量. 除此之外, 在 2016 年, Wang 等人[69]在 WPDP (SPV)和 CPDP (C-one)两种场景下进行模型间的比较实验. 在 2018 年, Wang 等人[40]在 WPDP(文件粒度的 SPV 和文件修改粒度的 SPV)以及 CPDP 场景下进行了模型间的比较实验. 在 2019 年, Qu 等人[4]选择在交叉验证(10×3 折)和 WPDP (SPV)下评估他们所提出的缺陷预测模型 Top-core 的有效性. 由于这 3 篇研究使用了不止一种数据集划分方式来进行实验, 因此在第 4.2.4 节中我们将更为详细地进行介绍.

WPDP 场景的数据集划分方法要求缺陷数据集中的项目分版本进行特征抽取和缺陷标注, 也就是说, WPDP 场景下的缺陷预测实验仅能在单个项目中的包含多个版本的缺陷数据集上进行. 从第 3.1 节对目前常用的缺陷数据集的汇总结果来看, 多版本的缺陷数据集数量有限(仅有 Eclispe、JURECZKO 和 Shippey 等), 客观上限制了在更大规模数据集上进行 WPDP 场景下的 SDP 实验.

4.2.2 CPDP 场景

CPDP 模型的评估和验证是近年来 SDP 研究领域的热门方向. 为了在 CPDP 场景下找到更好的模型, 研究者们展开了一系列模型间的比较实验, 包括: 对比不同的 CPDP 技术(迁移学习方法与分类器的组合), 以研究自动参数优化对 CPDP 模型性能的提升[14]; 对 2015–2019 年间的 CPDP 文献的系统性回顾[30]; 使用简单的基于模块规模的无监督基线模型对 CPDP 模型性能进行大规模评估[28]; 对 2008–2015 年间的 24 个 CPDP 模型的统一实现并开源成为框架工具 Crosspare[29], 并在 Crosspare 上对这 24 个模型进行了性能比较[29], 等等.

SDP 模型是独立于数据集划分方法的, 因此, 专门为某种特定场景(即某个特定的数据集划分方法)提出的缺陷预测模型也可以被应用于其他场景. Amasaki[34]研究了专为 CPDP 场景提出并在 CPDP 场景下得到验证的 CPDP 模型用在 WPDP 场景下的性能, 得出的结论是, 一些 CPDP 方法可以提升 WPDP 的预测性能[34]. 这启发了我们来思考, 模型的有效性仅在单一数据集划分方法下进行验证是不全面的. 研究者应当在多种划分方法下对模型进行评估, 以最大程度地探索所提出的 SDP 模型在各种场景下对预测性能的提升情况.

4.2.3 交叉验证场景

在缺陷预测研究中, 常用的交叉验证的实验设置有 10×10 折交叉验证[42,78]和 10 折交叉验证(M=1)[77]. 例如: 在 2018 年, Song 等人[12]使用 10×10 折交叉验证来比较不同的类不平衡学习方法在缺陷预测问题上的性能; 在 2017 年, Palomba 等人[54]使用 10×10 折交叉验证来验证所提出的基于代码异味的缺陷预测模型.

不同于交叉验证中常用的 10×10 折交叉验证, Yang 等人[5,11]选择在缺陷预测模型的评估实验中选择 3×3 折交叉验证, 也就是将数据集随机分成 3 份, 每次取 2 份作训练集, 余下一份作测试集, 直到每一份都作过一次测试集. 重复这个过程 30 次, 也就是总共进行 90 次模型的训练和测试. 这样做的出发点是, 他们使用的缺陷数据集中样本数量不大且有缺陷模块占比很低. 考虑到样本数量小和有缺陷模块占比低是缺陷数据集的普遍共性, 因此 30×3 折交叉验证不失为一个有实践价值的缺陷预测研究的实验设置. 在 2019 年, Qu 等人[4]使用了交叉验证(10×3 折)来评估他们所提出的 SDP 模型 Top-core 的有效性. 另外, 在 2018 年, Agrawal 等人[73]使用 5×5 折交叉验证来进行比较实验, 验证他们所提出的缺陷数据集预处理方法 SMOTUNED. 可见: 研究者们在缺陷数据集上进行 M×N 折交叉验证时, 有选择较小 N 值的倾向. Bennin[68]等人和 Dejaeger 等人[43]则采用了 3 折交叉验证, 并表示, 应当在后续工作中进一步在 WPDP 场景下验证模型的性能.

4.2.4 随机划分场景

常用的随机划分的实验设置是按照 90%/10%、80%/20%、50%/50%或其他比例划分训练集和测试集, 得到在测试集上的预测结果. 重复该过程 N 次, 取所有迭代下的性能平均值作为指标结果. 由于在本文的文献综述范围内采用随机划分方式验证模型性能的文献不多, 本文扩大了文献检索范围进行检索, 并列出了部分采用随机划分实验设置的文献.

Turhan 等人[97]按照 90%/10%划分训练集和测试集、重复 10 次的数据集划分设置, 比较了跨公司缺陷预测和公司内部缺陷预测的预测效果.

Mende[111]按照 90%/10%划分训练集和测试集, 得到在测试集上的预测结果. 重复该过程 50 次取平均值作为模型的性能结果, 在该数据集划分设置下比较了不同的学习算法.

Yang 等人[112]使用 50%/50%的训练集和测试集划分比例, 重复实验 500 次来验证他们所提出的无监督缺陷预测模型 CEL 的有效性.

4.2.5 多种场景下的全面验证

近年来, 更多的 SDP 研究选择了在多种场景下全面验证 SDP 模型的效果. 在 2012 年, Peters 等人[80]在 10×10 折交叉验证和 CPDP (C-all)上验证了他们所提出的数据隐私算法 MORPH 的有效性. 在 2016 年, Wang 等人[69]在 WPDP (SPV)和 CPDP (C-one)两种场景下进行了模型间的比较实验, 以验证他们所提出的 DBN-CP 模型的有效性. 在 2017 年, Jing 等人[46]在交叉验证和 CPDP 场景下进行 SDP 模型间的比较实验, 以来验证他们的框架对于解决项目这两种情况下的类不平衡问题都有效果. 在 2018 年, Wang 等人[40]在 WPDP(文件粒度的 WPDP 和文件修改粒度的 WPDP)以及 CPDP(文件粒度的 CPDP 和文件修改粒度的 CPDP)多种场景下进行了模型间的比较实验, 以验证他们所提出的语义度量对于构建 SDP 模型的有效性. 在 2019 年, Qu 等人[4]采用选择在两种预测场景(10×3 折交叉验证和 SPV)下进行评估的方式来验证他们所提出的 SDP 模型 Top-core 的有效性, 鉴于交叉验证是最广泛用于评估缺陷预测模型的方法之一, 而 SPV 则被认为更合适和实用[4]. 在 2019 年, Gong 等人[19]在交叉验证(随机选 85%的样本作为训练集, 其余为测试集)和 CPDP (C-all)这两种划分方式下研究移除类交叉(classoverlap)与否对 SDP 性能的影响.

总体上, 部分研究者意识到: 应当在多种场景下进行模型间的比较实验, 以求实现全面的 SDP 模型评估. 我们已经讨论过, 为什么交叉验证方法被认为实用价值不高. 在涵盖了其他两类更有实用意义的划分(WPDP 和/或 CPDP)的研究中, 具体的数据集划分方法各有不同. 如 CPDP 场景下进行评估实验, 某些研究选用 C-one 而其他研究倾向于 C-all. 相同场景而具体划分不同, 未必得到一致的实验结论, 而选用不同划分设定的研究之间也难以对预测性能进行比较. 我们相信: 在多种数据集划分方式下进行全面的模型间比较实验, 并且更有依据地选择数据集划分方式, 有助于缺陷预测模型间比较实验得到更合理的结论.

4.3 数据集划分存在的问题

本节汇总了过往文献在进行 SDP 模型间的比较实验时选用的数据集划分方法, 并汇总不同数据集划分方法下的研究, 通过分析我们发现, 当前的比较实验中数据集划分的选择存在以下问题.

1) 交叉验证和随机划分并不符合现实场景. 大量实证研究使用交叉验证在预测数据上建模, 已经使其成为一种公认方法[68]. 然而, 公认的交叉验证方法并不符合缺陷预测模型的实际应用场合. 考虑到时序原则, 即只能使用已开发并标记的模块来预测未开发的模块, 交叉验证用某个版本中的一部分模块来预测另一部分模块尽管简单, 但可能会导致模型使用未来的知识导致模型性能被高估[16], 并不符合现实场景. 同理, 随机划分方法同样会违反时序原则, 不符合现实场景, 因而在缺陷预测问题上, 不是一种能够准确衡量模型实际应用价值的数据集划分方法.
2) 划分方式缺乏一致认知. 我们发现: 即便相同种类的划分场景, 在具体划分做法上, 不同的模型采取不同的做法, 导致各自产生的实验结果无法比较. 如: CPDP 场景下, 有些实验选择 C-one[69]而另外一些实验选择 C-all[19]; WPDP 场景下, 有些实验选择 SPV 而有些实验选择 APV[34]. 据我们所知: 并未有研究对数据集划分的选择依据进行深入探讨, 选择依据基本上是取决于研究者的偏好. 为了能够一致比较不同 SDP 研究中的结果, 我们建议: (1) 进一步探讨不同划分方式将如何影响实验; (2) 研究者们实现多种划分方式, 并公开自己的实验源代码.
3) 单一场景下的验证不够全面. SDP 模型与数据集划分是独立的, 同一个模型可以在不同的训练集-测试集上运行. 因而, SDP 模型独立于使用场景. 当选择的数据集划分方法不同时, 缺陷预测模型间的比较结果很可能不同. 应当在多种场景下进行 SDP 模型间的比较实验(目前来说, 至少应当涵盖 WPDP 和 CPDP 这两种场景), 这样才能得到对缺陷预测模型的更全面的评估.

4.4 小 结

本节从数据集划分方法汇总、不同数据集划分下的 SDP 研究、数据集划分中存在的问题这 3 个方面详细

介绍了 SDP 模型间的比较实验中的数据集划分方法. 下面简述主要发现.

(1) 当前, 主要的数据集划分方法可分为 WPDP、CPDP 和交叉验证. WPDP 中最常用的方法是 SPV(用项目内距离目标版本最近的可用先前版本来训练模型), CPDP 中常用的方法有 C-one 和 C-all, $M\times N$ 折交叉验证中比较常见的是 10×10 折交叉验证和 10 折交叉验证;

(2) 不同的数据集划分方法对应着不同的应用场景: WPDP 对应目标版本有足够多的历史版本数据用来训练模型的场景, 而 CPDP 对应目标版本的历史版本数据缺失或不足的场景. 尽管近年来有研究选择在多种应用场景下评估 SDP 模型, 但 SDP 研究主要还是倾向于在单一场景下进行模型评估;

(3) 在当前的 SDP 模型比较实验中, 数据集划分的选择存在大量使用不符合现实场景的交叉验证方法、在划分方式上缺乏一直认知和单一场景下的验证不够全面等问题.

5 SDP 模型比较实验中的基线模型

迄今为止, SDP 领域已经提出了大量的预测模型. 每当有新模型被提出, 往往要通过与基线模型比较来验证其缺陷预测效果. 因此, 在模型比较中, 基线模型的选取对于模型效果的验证至关重要. 通过文献回顾我们发现: 过往在 SDP 模型比较中被用作基线模型的 SDP 模型种类众多, 这些模型由不同的分类依据可以有不同的划分. 为了评估基线模型的使用现状, 本文依据“预测模型是否需要用到历史的标签信息”这一点, 主要从有监督缺陷预测模型的基线方法、无监督缺陷预测模型的基线方法、半监督缺陷预测模型的基线方法这 3 个方面对 SDP 模型间的比较实验中使用的基线模型进行分类总结.

5.1 有监督缺陷预测的基线模型

有监督技术是 SDP 领域的主流技术. 本节分基于经典的机器学习算法的基线模型和以领域内前沿模型作为基线模型两部分, 介绍文献回顾中有监督缺陷预测模型的常见基线模型. 领域内前沿模型作为基线模型指的是, 把当时被认为是最先进的模型当作基线模型. 通常认为: 如果通过大规模比较发现新模型优于经典的机器学习算法, 则新模型具有研究价值, 至少在缺陷预测上比起久经验证的经典算法性能上有所提升; 如果通过大规模比较发现新模型的预测性能优于当时最先进的模型, 则可以说明新模型提升了缺陷预测的性能.

5.1.1 基于经典的机器学习算法的基线模型

表 5 中汇总了常作为缺陷预测基线模型的经典的机器学习算法, 表中列出了技术名称、该技术对应的参考文献以及在本文的文献回顾范围内使用了相应的基线模型的文献.

表 5 经典机器学习算法作为有监督缺陷预测模型基线模型汇总

缩写	技术名称	来源	使用文献
NB	朴素贝叶斯(naive Bayes)	Ref.[113]	Ref.[2,19,74]
RF	随机森林(random forest)	Ref.[114]	Ref.[2,19,68,74]
DT	决策树(decision tree)	Ref.[115]	Ref.[2,68,74]
SVM	支持向量机(support vector machine)	Ref.[116]	Ref.[2,19,68]
LR	逻辑回归(logistic regression)	Ref.[117]	Ref.[2,19,74]
KNN	K 近邻(k-nearest neighbor)	Ref.[118]	Ref.[19,68]
NN	神经网络(nerual network)	Ref.[119]	Ref.[68]
ADTree	可变决策树(ADTree)	Ref.[120]	Ref.[40]

经典的机器学习算法经常被用作有监督缺陷预测模型的基线模型, 其主要原因有:

(1) 这些模型的有效性已经在各种应用领域得到验证, 为研究人员所熟知;

(2) 这些模型已经内置在各种语言的算法库中, 如 Python 的 scikit-learn 和 Java 的 Weka, 无需研究人员重复实现;

(3) 这些模型在早期的缺陷预测研究中就被使用, 并逐渐成为缺陷预测的常用和传统模型, 后续的使用或许是基于惯例.

有监督缺陷预测模型常用的经典机器学习算法有朴素贝叶斯(naive Bayes)[113]、随机森林(random

forest)[114]、决策树(decision tree)[115]、支持向量机(support vector machine)[116]、逻辑回归(logistic regression)[117]、*K* 近邻(*k*-nearest neighbor)[118]、可变决策树(ADTree)[120], 等等. 大量文献同时使用这些算法中的多个作为基线模型与新提出的模型进行比较[2,19,68,74]. 下面介绍部分使用传统有监督机器学习模型作为基线模型的研究.

在 2015 年, Nam 等人[2]提出了两个无监督缺陷预测方法 CLA 和 CLAMI. 为了评估这两个新模型的预测性能, 作者选择了一系列传统的有监督机器学习方法作为基线模型, 包括逻辑回归[117]、朴素贝叶斯[113]、贝叶斯网络[121]、J48 决策树[122]、随机森林[114]和支持向量机[116]. 另外, 他们还使用了基于阈值的方法[123]和基于专家意见的方法[124].

在 2016 年, Zhang 等人[74]提出了一个基于频谱聚类(spectral clustering)的无监督缺陷预测方法 SC. 为评估 SC 的预测性能, Zhang 等人使用了一系列有监督和无监督方法作为基线模型, 其中有监督方法有随机森林[114]、朴素贝叶斯[113]、逻辑回归[117]、J48 决策树[122]和逻辑模型树(LogicModelTree)[125]. 他们所使用的无监督基线模型将会在第 5.2 节中给出详细介绍.

在 2018 年, Bennin 等人[68]提出了 MAHAKIL, 一个基于多样性的过采样方法, 用于解决缺陷预测中的类不平衡问题. 为了验证 MAHAKIL 对预测性能的提升, 文献[68]比较了各种常见的有监督机器学习算法不结合 MAHAKIL 进行训练集预处理与结合 MAHAKIL 进行训练集预处理的预测性能的差距, 其中选取的有监督机器学习算法是神经网络[119]、C4.5 决策树[126]、支持向量机[116]、*K* 近邻[118]和随机森林[114].

在 2019 年, Gong 等人[19]提出: 缺陷数据集中有缺陷类和无缺陷类在特征空间中的交叠(class overlap)会对在数据集上构建出的缺陷预测模型的效果产生负面影响[19], 因此, 应当在数据的预处理环节就消除训练集的 Class overlap. 为此, 他们研究了 4 种方法: IKMCCA、NCL、KMCCA 和 Without removing 对 SDP 性能的提升. 他们使用不消除 Class Overlap 的过往的 SDP 方法作为基线模型, 包括传统的有监督机器学习算法朴素贝叶斯[113]、随机森林[114]、支持向量机[116]、逻辑回归[117]和 *K* 近邻[118].

5.1.2 以领域内前沿模型作为基线模型

表 6 中汇总了近年来常作为有监督缺陷预测模型的基线模型的领域内前沿模型, 表中列出了技术名称、该技术对应的参考文献以及在本文的文献回顾范围内使用了相应的基线模型的文献. 通常认为: 通过与缺陷预测领域内的前沿模型进行对比, 如果能够证明自身提出的模型比前沿模型的预测性能要好, 则可以证明模型的有效性.

表 6 前沿模型作为有监督缺陷预测模型基线模型汇总

缩写	模型简介	来源	使用文献
CamargoCruz09-DT	对目标数据和训练数据进行对数标准化	Ref.[127]	Ref.[38]
Menzies11-RF	通过用 WHERE 算法对训练数据进行聚类, 然后用 WHICH 规则学习算法对结果进行分类	Ref.[128]	Ref.[38]
Turhan09-DT	对特征值进行对数变换然后基于 *k*NN 算法 筛选出与目标实例最接近的训练实例进行训练	Ref.[97]	Ref.[38]
Watanabe08-DT	提出了一个新颖的数据标准化公式	Ref.[129]	Ref.[38]
TunedmanualUp	由一个缺陷预测器进行调参改良的 ManualUp 方法	Ref.[130]	Ref.[38]
TCA+	转移成分分析和归一化(transfer component analysis plus normalization)	Ref.[131]	Ref.[20,64,65,132]
NN-filter	最邻近邻居过滤器(nearest neighbor filter)	Ref.[97]	Ref.[20,64,65,132]
HYDRA	混合模型重建法(hybrid model reconstruction approach)	Ref.[59]	Ref.[20,132]

近年来, 有监督缺陷预测模型常用的缺陷预测前沿模型有 CamargoCruz09-DT[127]、Menzies11-RF[128]、Turhan09-DT[97]、Watanabe08-DT[129]、TunuedmanualUp[130]、TCA+[131]、NN-filter[97]、HYDRA[59], 等等. 在 Herbold 等人[29]的大规模比较实验报告中, CamargoCruz09-DT[127]、Menzies11-RF[128]和 Turhan09-DT[97]是 3 个表现最好的缺陷预测模型, 因此, Ni 等人[38]使用了这 3 个模型作为基线模型来验证他们所提出的缺陷预测方法 EASC 的有效性. 然而, 在 Herbold 等人的原始论文[29]发表后不久, Herbold 等人[133]就修正了原始论文中得出的结论, 并说明 CamargoCruz09-NB[127]、Amasaki15-NB[134]和 Peters15-NB[77]才是这个大规模对比实验中表现最好的 3 个缺陷预测模型. TunedmanualUp 由 Tim 等人[130]提出, 对简单的按照模块规模从小到大来预测模

块缺陷可能性从小到大的 ManualUp 方法[28]进行改良, 先用一个有监督的学习器进行预测, 为 ManualUp 确定分类阈值. NN-filter、TCA+和 HYDRA 是现阶段 CPDP 的代表性的基线模型[20], 因此常在提出 CPDP 场景下被用作基线模型, 并与新提出的有监督缺陷预测模型进行比较. TCA+在 2013 年由 Nam 等人[131]提出, TCA+结合了数据归一化技术和基于特征的迁移学习方法(transfer component analysis, TCA). NN-filter 在 2009 年由 Turhan 等人[97]提出, 是从外部项目中选择接近项目内数据的合适数据(nearest neighbor)作为训练集的迁移学习方法. Turhan 等人[97]的实验结果表明, NN-filter 通过消除外部项目的不相关性来降低 CPDP 的高误报率. 另外, HYDRA (hybrid model reconstruction approach)在 2016 年由 Xia 等人[59]提出, 是一种基于任务的增强技术, 用于实例迁移, 包括遗传算法和集成学习两个阶段[20]. 近年来, 也有越来越多的 CPDP 研究使用 HYDRA 作为基线模型. 下面介绍部分使用近年来缺陷预测领域前沿模型作为基线模型的研究.

在 2015 年, Jing 等人[65]提出了一个结合统一度量与迁移学习的异构跨项目缺陷预测模型 CCA+. 为了验证 CCA+的预测性能, Jing 等人选取了性能领先的 CPDP 模型 TCA+和 NN-filter 作为基线模型与之对比. 在一个外部项目作为训练集和多个外部项目作为训练集两种异构跨项目实验配置下, 表明在 MDP 数据集下, CCA+的召回率和假警报率优于 TCA+和 NN-filter.

在 2019 年, Li 等人[20]提出的 MSMDA 模型中同时使用了 TCA+、NN-filter 和 HYDRAY 作为基线模型. 2020 年, Li 等人[64]为 CPDP 场景提出了一个模型发现技术 Bi-Level Optimal technich for CPDP (BiLO-CPDP). 顾名思义, BiLO-CPDP 把分类器和迁移学习算法的选择以及它们的超参数优化放到双层规划的数学框架中, 上层优化问题的解依赖于下层优化问题的最优解. 在 Li 等人[64]的研究中, 上层的优化问题是从给定的组合中确定迁移学习者和分类器的最佳组合, 下层的优化问题搜索与迁移学习器和分类器的最佳参数设置. 为了验证 BiLO-CPDP 的性能, Li 等人选择了多种迁移学习算法与分类器进行组合, 与 BiLO-CPDP 进行比对, 其中就包括 TCA+和 NN-filter 这两个 CPDP 中常见的迁移学习算法. 结果表明, BiLO-CPDP 在 AEEEM、Relink 和 JURECZKO 这 3 个数据集上的 AUC 表现优于 TCA+和 NN-filter.

在 2019 年, Gong 等人[19]提出: 类重叠问题(class overlap)与类不平衡问题一样, 都可能影响缺陷预测模型的性能. 然而, 类重叠问题却不如类不平衡问题那样, 能够在缺陷预测领域得到重视. 类重叠指的是, 训练数据中往往包含一些特征值相似但属于不同类的实例. 为了解决类重叠对缺陷预测的影响, Gong 等人提出一种改进的 *K*-means 聚类清理方法(improved *K*-means clustering cleaning approach, IKMCCA)来解决类重叠和类不平衡问题. 为了衡量 IKMCCA 的缺陷预测能力, Gong 等人[36]选择了多个基线模型, 在 WPDP 和 CPDP 两种场景下进行了大规模的比较实验. WPDP 的基线模型是传统的有监督机器学习算法(朴素贝叶斯、随机森林、支持向量机、逻辑回归和 *K* 近邻), 而 CPDP 的基线模型则包括 NN-filter、TCA+和 HYRRA. 实验结果表明: IKMCCA 在 CPDP 场景下, 在 Recall、Balance 和 AUC 上获得了更好的表现, 意味着在缺陷预测中应当考虑到类重叠和类不平衡问题.

在 2020 年, Ni 等人[38]提出了 EASC (effort-aware supervised cross-project defect prediction), 该缺陷预测模型通过分段排序的方法来平衡工作量感知和非工作量感知的性能. 为了验证 EASC 在工作量感知和非工作量感知指标下的性能, Ni 等人以 4 个 Herbold 等人报告的缺陷预测领域的前沿方法 CamargoCruz09-DT[127]、Menzies11-RF[128]、Turhan09-DT、Watanabe08-DT 以及 2 个无监督的基于模块规模的简单方法 ManualDown[28]、ManualUp[28]还有一个 Tim 提出的结合了有监督方法去改良 ManualUp 的新方法 TunedManualUp 作为基线模型, 进行了大规模的实验比较.

5.2 无监督缺陷预测的基线模型

大部分的 SDP 模型通过利用历史缺陷数据来构建有监督预测模型, 不过, 历史缺陷数据并不总是可以获取到的. 无监督技术由于其不需要过往缺陷数据, 也就不需要花费成本对模块的缺陷进行标注. 同时, 无监督技术的预测效果不受训练集和目标集的分布差异的影响. 另外, 通常来说, 无监督算法比有监督算法实现起来更简单, 运行更快. 尽管普遍认为有监督技术的效果比无监督技术要好, 但基于无监督技术的上述优点, SDP 领域提出了一系列基于无监督技术的 SDP 模型, 也有后续的 SDP 研究使用无监督模型作为基线方法. 表

7 汇总了 SDP 领域常见的无监督基线模型.

表 7 无监督缺陷预测的基线模型汇总

类别	缩写	技术名称	来源	使用文献
随机方法	RANDOM	随机方法(random model or dummy predictor)	–	Ref.[29,47,82]
	FIX	FIX	–	Ref.[29]
基于模块规模	ManualDown	ManualDown	Ref.[28]	Ref.[28,42]
	ManualUp	ManualUp	Ref.[28]	Ref.[28,42]
聚类技术	CLAMI	Clustering, labeling, metric selection and instance selection	Ref.[2]	Ref.[51,66]
	SC	Spectral clustering	Ref.[74]	Ref.[20]
	K-means	*K*-means	Ref.[123]	Ref.[74]

本节将常用作基线方法的无监督缺陷模型分 3 类进行介绍, 分别是随机模型、基于模块规模的模型以及基于聚类技术的模型.

5.2.1 随机模型

当一个新的 SDP 模型被提出, 研究者们通常会将其与现有的最好的一些 SDP 模型进行对比, 希望借此了解新方法相比于过往方法是否提升了预测性能. 与此同时, 一个更基本的问题是: 新的 SDP 模型是否优于那些“最简单、最基本、性能最差”的 SDP 模型? 这是个非常重要的问题, 因为如果一个 SDP 模型甚至不能优于那些最简单的模型, 那么这样的方法是没有应用价值的. 基于这样的思想, 部分 SDP 研究[29,47,82]选择随机模型作为基线模型. 随机模型即以一定的概率(通常是 50%)来判定模块是否有缺陷, 以 50%概率判定模块有缺陷的模型在文献中也被称作虚拟预测器(dummy predictor). 如果新提出的 SDP 模型的表现不如随机模型, 也就意味着模型预测效果还不如乱猜. 因此, 新提出的模型的评价指标至少要优于随机模型. 下面介绍部分使用随机模型作为基线模型的缺陷预测研究.

在 2011 年, Kim 等人[82]在移除训练集的噪声和不移除噪声两种情况下, 对比了贝叶斯网络和 dummy predictor 这两种 SDP 方法. 在 2016 年, Lee 等人[47]为了验证他们所提出的开发人员微交互度量构建的 SDP 模型的性能, 选择了 dummy predictor 作为了基线模型之一. 在 2018 年, Herbold 等人[29]评估了多种 SDP 方法在 CPDP 上的性能, 他们采用了 4 种方法作为基线方法: 跨项目缺陷预测方法(CV)、all internal project 方法(ALL)、以 50%概率判定模块有缺陷的方法(也就是 dummy predictor, RANDOM)以及把所有模块都判定为有缺陷的方法(FIX). 可以将 FIX 方法理解成以 100%的概率判定模块为有缺陷.

除了随机模型之外, 常见于 SDP 性能评估的用于衡量 SDP 模型的一些无监督的“基本模型”还有最优模型(optimalmodel)和最差模型(worst model), 等等. 最优模型代表理想中表现最好的缺陷模型, 具体来说, 最优模型把模块按照缺陷密度降序排列, 用来衡量被评估模型和最好的缺陷预测结果之间的差距; 最差模型代表理想中表现最差的缺陷模型, 具体来说, 最差模型把模块按照缺陷密度升序排列, 用来衡量被评估模型和最差的缺陷预测结果之间的差距. 新提出的模型应当在性能上尽可能地接近最优模型, 尽可能地远离最差模型. 不过, 最优模型和最差模型极少被直接用作基线模型, 它们通常直接被包含在评估指标的设计中, 如Δ_{opt}就是衡量被评估模型和最优模型的 ROC 曲线之间的面积的(详见第 6.1.2 节).

5.2.2 基于模块规模的基线方法

在过去的几十年中, 大量研究报告了项目中的模块大小(例如 SLOC、源代码行)与模块的缺陷是否有相关性[28,42,130,135−140]. ManualDown 和 ManualUp 是两个常用作基线模型的、基于模块规模的简单 SDP 模型[28,130,136,138,139]. 前者认为较大的模块更容易出现缺陷, 而后者认为较小的模块更容易出现缺陷. 具体来说, 这两个模型利用目标项目/版本中的模块规模度量 SLOC(源代码行数)将模块分类为有缺陷或无缺陷. SLOC 表示收集到的非空白、非注释代码行的数量. 设 m 为当前版本中的一个模块, $R(m)$为模 m 的缺陷可能性的预测值. 对于 ManualDown 来说, $R(m)=SLOC(m)$; 而对于 ManualUp 来说, $R(m)=1/SLOC(m)$. 可见: ManualDown 和 ManualDown 需要较低的度量收集成本、较低的模型构建成本, 具有良好的可扩展性并且易于实施. 下面介绍部分使用了 ManualDown 和/或 ManualUp 作为基线模型的 SDP 研究.

在 2014 年, Zhou 等人[42]对模块规模在 SDP 中潜在的混杂效应进行了深入研究, 他们通过大规模的比较

实验来验证消除模块规模对其他特征的协同影响后, 训练出的 SDP 模型的性能. 在性能评估中, 他们使用了 ManualDown 和 ManualUp 作为基线模型. 使用 ManualDown 是因为许多研究报告说: 更大的模块往往有更多的错误, 因此建议大模块应首先被检查/测试[28,42,130,135−140]. 使用 ManualUp 是因为 Koru 等人发现, 小模块的缺陷密度更大. 因此, 他们建议先检查小模块[136,138,139].

在 2018 年, Zhou 等人对 CPDP 模型进行了大规模的回顾[28], 使用 ManualDown 作为基线模型, 与过往的缺陷预测模型在分类指标上进行比较; 使用 ManualUp 作为基线模型, 与过往的 SDP 模型在排序指标上进行比较. 结果发现: 相比于简单的基于模块大小的无监督方法 ManualDown 和 ManualUp, 过往的 CPDP 模型几乎对预测性能没有太大的提升. 基于这一令人惊讶的实验结果, Zhou 等人建议后续的 CPDP 模型评估中使用的基线模型都应包括 ManualDown 和 ManualUp.

5.2.3 基于聚类技术的基线方法

基于聚类技术进行缺陷预测的基本假设是: 有缺陷模型拥有相似的软件度量, 因此容易被分到同一个聚类中[124]. 新提出的基于聚类的 SDP 方法常把过往的基于聚类的缺陷预测方法作为基线模型并与之比较.

在 2015 年, Nam 等人[2]提出了在目标数据集上自动进行缺陷预测的方法. 换句话说, 他们提出的方法在无标签的数据集上训练预测模型, 得到的预测模型再用于预测该数据集中的模块. 他们的模型构建分为 5 个步骤: (1) 对模块进行聚类(clustering instances); (2) 标注聚类后的模块(labeling instances); (3) 特征选择(metric selection); (4) 模块选择(instance selection); (5) 使用机器学习分类器来构建预测模型. 根据以上 5 个步骤的缩写, 这个无监督缺陷预测模型称作 CLAMI. Nam 等人的实验结果表明: CLAMI 在大部分的项目上取得了不输于典型的 SDP 方法的召回率、F_1 和 AUC. 在之后的异构缺陷预测(heterogeneous defect prediction, HDP)研究中, Nam 等人[66]研究了基于 KSAnalyzer、PAnalyzer 和 ScoAnalyzer 的 HDP 模型, 并使用 WPDP 方法、训练集和目标集的特征相同的 CPDP 方法、基于不平衡的特征集合的 CPDP 方法以及无监督缺陷预测方法这 4 种缺陷预测方法作为基线模型与 HDP 方法进行比较. 其中, 无监督缺陷预测方法使用的就是 CLAMI. 在 2018 年, Nam 等人[51]将他们在 2015 年所做的工作进行了扩展, 他们在该研究中讨论了 HDP 所需的数据集规模的问题, 并结合近期的研究讨论了更多相关工作. 在该扩展工作中, 他们研究的 HDP 模型基于 KSAnalyzer, 选择的基线方法与 2015 年的工作一样.

在 2016 年, Zhang 等人提出了一种基于连通性的无监督分类模型, 称为频谱聚类(spectral clustering, SC)模型[74]. 对于给定的目标数据集, SC 模型以如下步骤判定数据集中的模块是否有缺陷.

(1) 以模块为节点、模块间的连接为边、模块间的相似度为权重, 构建一个有权图. 模块间的相似性通过两个模块采用 *z*-score 进行了归一化的特征向量的点积来表示;
(2) 使用频谱聚类算法把模块分成两个不重叠的聚类;
(3) 把平均行总和度量值更大的那个聚类中的模块判定为有缺陷, 另一个聚类中的模块判定为无缺陷. 原因是“对于大多数度量, 有缺陷模块通常比无缺陷模块具有更大的值”.

为了评估 SC 模型的预测性能, Zhang 等人[74]选择了 5 种有监督分类算法和 4 种基于聚类的无监督分类算法作为基线模型与 SC 进行比较. 结果表明: 在总共 10 个 SDP 模型中, SC 的表现属于最好之一. 5 种有监督分类算法已在第 5.1 节中给出介绍. 4 种基于聚类的无监督分类算法分别是 *K* 均值(*K*-means)聚类算法、Neural-gas 聚类算法、模糊 *c*-均值聚类算法(fuzzy *C*-means)、Partition Around Medoids (PAM)算法. 其中, *K*-means 聚类算法和 Neural-gas 聚类算法曾在 2004 年被 Zhong 等人[124]用来构造他们的聚类-专家判定无监督 SDP 模型. 选择这两种聚类算法的原因是: “选择 *K*-means 的原因是它的普及和效率, 而使用 Neural-gas 是由于它在产生相干簇上的良好性能”. 简单来说, 对于目标数据集, 该模型先使用聚类算法得到多个聚类, 接着再请专家判定每个聚类的标签(有缺陷或无缺陷). 但 Zhong 等人提出的无监督模型不是完全自动化的, 它依赖于专家的判断.

为了从缺陷预测模型中移除主观的专家意见, 在 2009 年, Catal 等人[123]提出了一个基于 *K*-means 聚类算法的无监督缺陷预测模型, 在得到聚类结果之后, 根据软件度量的阈值来判定每个聚类的标签(有缺陷或无缺陷). 具体来说, 如果一个聚类的任意一个度量的平均值高过这个度量的阈值, 那么这个聚类将被判定为有缺

陷. 这里使用的软件度量阈值是由集成软件度量公司(Integrated Software Metrics, Inc.)提出的. 但是这个模型仍然需要人为选择 *K*-means 算法中的聚类个数, 而聚类个数的选择可能会给缺陷预测的性能造成影响. 为了避免人为选择聚类个数, 得到完全自动的无监督缺陷预测模型, 在2010年, Catal等人[141]又在他们原有模型的基础上, 使用 *X*-means 取代了 *K*-means, *X*-means 可以自动选择最佳的聚类个数并进行聚类.

另外, 由于无监督缺陷预测方法和 HDP 方法都被用于解决缺乏历史缺陷数据的项目的缺陷预测问题, 因此, 研究者们在为解决这一问题而提出 HDP 方法时, 很自然地会想到这个新方法与无监督 SDP 方法相比性能如何. 因此, 也有 HDP 模型以无监督的聚类方法作为基线模型并与之比较.

在 2019 年, Li 等人[20]提出了一个多源 HDP 方法, 称作 MSMDA (multisource selection based manifold discriminant alignment). 对于目标项目, MSMDA 可以从许多可用的源中增量地选择类似分布的源项目. MSMDA 通过充分利用多个源的标签信息和有限数量的训练目标数据, 将目标项目和精心选择的多个源转化为具有相似数据分布和良好分类能力的判别公共子空间[20]. 在验证 MSMDA 的缺陷预测性能的比较实验中, 他们同时选择了 WPDP 方法、经典的 CPDP 方法(NN-filter 和 TCA+)、HDP 方法(CPDP-IFS、CCA+和 HDP-KS)、实例转换方法(MultiSourceTrAdaBoost 和 HYDRA)和无监督缺陷预测方法(SC)作为基线模型.

5.3 半监督缺陷预测的基线模型

如第 5.2 节所述, 用于构建有监督缺陷预测模型的历史缺陷数据并不总是容易获得的. 半监督缺陷技术可以利用少部分带标签的训练数据以及其余的无标签训练数据, 例如采用联合训练或者标签传播等方式来为无标签的训练数据生成伪标签, 再用标注的数据结合伪标签数据进行模型训练. 考虑到半监督方法仅需要少量有标签数据、能够降低数据收集成本的优点, SDP 领域提出了一系列半监督缺陷预测模型, 表 8 汇总了近年来提出的半监督缺陷预测模型在验证自身性能时常用的基线模型. 从表中可以看出, 半监督缺陷预测模型在选择基线模型时有两大类思路: 其一是选择其他半监督方法作为基线模型与之比较, 包括过往已被用于缺陷预测的半监督方法以及经典的半监督方法; 其二是选择经典的有监督模型与之比较, 通过“作为半监督模型达到了与有监督模型相当甚至更优的预测性能”来说明自身所提出的新的半监督模型的有效性.

表 8 半监督缺陷预测的基线模型汇总

类别	缩写	技术名称	来源	使用文献
半监督方法	Roca	RandOm committee	Ref.[142]	Ref.[142]
	FTF	Fitting-the-fits	Ref.[143,149]	Ref.[85,149]
	FTcF	Fitting-the-confident-Fits	Ref.[143]	Ref.[84]
	CoForest	Extending the co-training paradigm by using random forest	Ref.[144]	Ref.[86]
	LDS	Low-density separation	Ref.[145]	Ref.[25,85]
	SVMlight	An implementation of semi-supervised SVM	Ref.[146]	Ref.[25]
	EM-SEMI	Expectation-maximization SEMI-supervised approach	Ref.[147]	Ref.[25]
	CMN	Classmass normalization	Ref.[148]	Ref.[25,85]
	GKSLP	Gaussian kernel similarity based label propagation	Ref.[148]	Ref.[85]
	ROCUS	Random committee with under-sampling	Ref.[142]	Ref.[85]
有监督方法	ML	经典的有监督模型, 朴素贝叶斯、逻辑回归、随机森林等	Ref.[113,114,117]	Ref.[84,149,150]

半监督缺陷预测中, 使用的半监督基线模型有 Roca[142]、FTF[143]、FTcF[143]、CoForest[144]、LDS[145]、SVMlight[146]、EM-SEMI[147]、CMN[148]、GKSLP[148]、ROCUS[142]等. FTF[143]是传统的自训练算法的一个变体, 一个基础的监督学习器从当前标记的数据中被反复训练, 在每次迭代中, 未标记的模块的状态被反复更新, 直到满足某种停止条件[149]. 在 FTcF[143]中, 一个基础的监督学习器从当前标记的数据中被反复训练. 在 FTcF 的每次迭代中, 都会对未标记数据的标签进行预测. 对预测分数具有高信心的实例将逐渐从未标记的数据池迁移到标记的数据池, 当未标记的数据池中没有任何模块时, 该算法就完成了预测. CoForest[144]是一种基于分歧的半监督学习算法, 利用了半监督学习和集成学习的优势. LDS[145]在图形距离衍生内核(graph-distance-derived kernel)上训练 SVM, 并应用梯度 TSVM 算法进行半监督分类[85]. CMN[148]采用高斯随机场模型, 根据实例间的相似函数给出图的权值[85]. GKSLP[148]是一种基于高斯内核相似度的标签传播算法. ROCUS[142]是半监督学习结合欠采样数据平衡策略的缺陷预测模型. 半监督缺陷预测中的有监督基线模型主要是经典的有监

督模型, 包括朴素贝叶斯、逻辑回归、随机森林等. 下面介绍部分使用半监督缺陷预测模型研究以及他们使用的基线方法.

在 2011 年, Jiang 等人[142]提出了一种名为 ROCUS 的半监督学习结合欠采样数据平衡策略的方法用于缺陷预测. 研究表明: ROCUS 的预测性能优于忽略缺陷数据集的类不平衡性质的半监督学习方法, 且优于未能有效利用未标记数据的单纯的类不平衡学习方法. 在比较实验中, 除了使用有监督模型、未进行类平衡的半监督模型、进行了类平衡的有监督模型之外, 他们还使用了一种半监督缺陷预测方法 Roca (random committee)作为基线模型[142]. 相比于 ROCUS, Roca 忽略了不平衡类的分布.

在 2011 年, Lu 等人[149]利用半监督算法 FTF 进行缺陷预测, 通过与随机森林模型进行比较发现: 当半监督的 FTF 方法和有监督方法都使用随机森林作为基础的分类算法时, 基于 FTF 的半监督模型表现更优. 2012 年, Lu 等人[84]又提出了一种结合特征降维策略的基于 FTcF 半监督算法的缺陷预测模型, 他们的对比实验所采用的基线模型包括没有进行特征降维的原始的半监督方法 FTcF 以及有监督的随机森林模型. 结果发现: 结合了多维缩放(multidimensional scaling, MDS)特征降维的半监督方法的预测性能优于随机森林及 FTcF 方法.

在 2012 年, Li 等人[86]提出了通过主动选择和标记一些以前未标记的数据训练分类器的 ACoForest 方法, 他们期望该方法能够在采样中抽取出对训练出好模型最有帮助的那部分模块, 该方法是半监督方法 CoForest 的一个扩展. 在验证 ACoForest 性能的比较实验中, 作者将基于半监督算法 CoForest[144]的缺陷预测方法作为基线模型.

在 2013 年, Catal[25]探索了 4 种半监督分类方法在缺陷预测问题上的性能, 它们分别是低密度分隔(low-density separation, LDS)[145]、半监督支持向量机的 SVMlight 实现[146]、最大期望演算法(expectation-maximization)[147]和类别批量归一化(class mass normalization)[148]. 通过比较, 他们推荐在有标签训练数据不足的情况下, 使用基于 LDS 的缺陷预测方法.

在 2016 年, He 等人[150]提出了一种半监督缺陷预测方法 extRF, 它采用变化突发信息来提高软件缺陷预测的准确性. 他们在比较实验中选择了 3 个经典的有监督学习方法: 朴素贝叶斯[113]、逻辑回归[117]、随机森林[114]作为与 extRF 进行对比的基线模型.

在 2017 年, Zhang 等人[85]使用基于非负稀疏图的标签传播方法 NSGLP (nonnegative sparse graph based label propagation)来迭代预测未标记的实例, 以用于软件缺陷预测. 在验证 NSGLP 的比较实验中, 作者使用过往被证明在软件缺陷预测问题上有效的半监督模型: FTF[143]、ROCUS[142]、LDS[145]、CMN[148]、GKSLP[148]作为基线模型. 实验结果表明: NSGLP 优于几种有代表性的最先进的半监督软件缺陷预测方法, 它可以充分地利用静态代码度量的特点, 提高软件缺陷预测模型的泛化能力.

5.4 基线模型的使用现状

我们从被调研的 66 篇文献中选出了 2019–2021 年间发表的、使用了至少一种基线方法与他们新提出的缺陷预测模型进行比较的文献, 统计了他们所使用的基线方法. 如表 9 所示, 这 9 篇文献几乎使用了完全不相同的基线模型来与自己提出的新模型进行比较. 缺点如下.

(1) 从表 8 可以看出, 9 个缺陷预测研究中涉及 30 多个基线模型. 然而在这 9 项研究中, 没有一个共同的基线模型. 事实上, 在每项研究中, 作者都用他们喜欢的先验模型作为基线模型来研究新提出模型的有效性. 其中, 大多数缺陷研究所使用的基线模型与其他研究中使用的模型没有任何重叠. 在这 10 项研究的每一项中, 作者都表示他们提出的新模型的性能超过了基线模型, 因此推动了最先进的技术. 但是由于缺乏一个共同的基线模型, 我们无法确定这 9 个新模型的相对有效性. 更重要的是, 由于缺乏一个全局性的参考点, 我们无法知道在缺陷预测的性能提升上, 我们到底走了多远;

(2) 考虑到许多先前提出的监督模型是复杂的(例如许多参数需要仔细调整), 并且不是开源的, 因此后续的研究必须重新实现它们, 以便把它们作为基线模型. 然而, 一个微小的实施偏差就可能导致预测性能的“退化”. 因此, 当一个新提出的模型被证明比这些基线模型具有更高的有效性时, 就有可能报告一个虚假的进步.

表 9 2019–2021 年发表的部分缺陷预测文献以及它们所使用的基线模型

研究	年份	模型	基线模型描述	基线模型
Ref.[4]	2021	Top-core and top-degree	Other techiques belonging to the same theory	PageRank, Betweenness
Ref.[64]	2020	BiLO-CPDP (automated model discovery techique)	A state of art automated machine leanrning tool, a simplified version of BiLO-CPDP, and 21 prior CPDP techniques	Auto-Sklearn, SLO-CPDP, and 21 combinations of prior classifier and transfer learners
Ref.[9]	2020	DTL-DP	Existing deep learning-based defect prediction approaches for both WPDP and CPDP scenario	Semantic, PROMISE-DP, DP-LSTM, DP-CNN, DBN-CP, and TCA+
Ref.[40]	2020	Semantic features that are automatically learned by DBN	Two types of file-level traditional features and three types of change-level features which have been used in previous studies	PROMISE features, AST features, bag-of-words features, characteristic features, and meta features
Ref.[18]	2020	FENCES	Three baselines which leverage traditional process and static code metrics and state-of-the-art work that leverages deep learning techiques to learn static code metrics automatically	RH[151], Mo[152], SCC, and ASF (automatically learn sematic features)
Ref.[19]	2019	IKMCCA	State-of-the-Art learning models without removing the overlapping instances	NB, RF, SVM, LR, KNN, NN-filter, TCA+, VCB-SVM, DTB, MNB, and HYDRA
Ref.[41]	2019	ORB (oversampling rate boosting)	State-of-the-art class imbalance evolution learning approaches	OOB, UOB, OOB (FixedIR), OOB (FixedIR)*, and OOB-SW
Ref.[20]	2019	MSMDA (multi-source heterogeneous defect prediction approach)	Prior CPDP methods	NN-Filter, TCA+, CPDP-IFS, CCA+, HDP by KSAnalyzer, SC, MultiSource-TrAdaBoost, and HYDRA
Ref.[54]	2019	Code smell intensity-including model	Prior methods	BCCM (the basic code change model), DM (based on developers that worked on a code component), and DCBM (the developer changes based model)

5.5 小 结

本节从有监督基线方法、无监督基线方法、CPDP 基线方法这 3 个方面详细介绍了新缺陷预测模型在模型评估时常用的基线方法. 下面简述主要发现.

(1) 当前常见的被用作有监督缺陷预测的基线模型包括经典的有监督机器学习算法(朴素贝叶斯、随机森林、支持向量机、逻辑回归和 K 近邻等)和当时前沿的缺陷预测模型(CamargoCruz09-DT[125]、Menzies11-RF[126]、Turhan09-DT[96]、Watanabe08-DT[127]、TunuedmanualUp[128]、TCA+[129]、NN-filter[96]、HYDRA[61]等). 当前常见的被用作无监督缺陷预测的基线模型包括随机模型(主要是以 50%的概率判定模型有缺陷的 dummy predictor)、基于模块规模的 ManualDown[28]和 ManualUp[28]以及基于聚类技术的 CLAMI[2]和 SC[20]. 当前常见的被用作半监督缺陷预测的基线模型主要分为当时前沿的半监督缺陷预测模型(Roca[142]、FTF[143]、FTcF[143]、CoForest[144]、LDS[145]、SVMlight[146]、EM-SEMI[147]、CMN[148]、GKSLP[148]、ROCUS[142]等)和经典的有监督机器学习算法(朴素贝叶斯、逻辑回归、随机森林等);

(2) 无监督基线模型的优点是实现简单、运行快, 通常被认为是新提出的有监督模型性能的底线. 即: 如果一个新提出的复杂的有监督模型在利用了过往的缺陷信息之后, 预测能力达不到简单的无监督模型, 则新模型会被认为不具有实用价值;

(3) 对于哪个缺陷预测模型在何种情况下最适合被用作比较实验的基线模型, 研究领域内没有统一做法. 因而可以看到: 文献回顾中, 大量的缺陷预测研究各自选择多个不同的基线模型进行比较. 由于缺乏一个共同的基线模型, 无法确定这些模型的相对有效性. 也就是说, 缺陷预测领域缺乏一个统一的被持续使用的基线模型来作为一个全局参考点.

6 SDP 模型比较的评价指标

本节介绍 SDP 模型比较中采用的评价指标. 主要从评价指标的概览、评价指标使用偏好的变迁和目前缺陷预测领域对这些评价指标的优缺点的认知这 3 个部分展开介绍.

6.1 评价指标的计算

缺陷预测模型间的性能比较需要通过比较模型在相同目标数据集上的评价指标来实现. 过往的缺陷模型评估中使用了不同类型的各种评价指标, 本节从分类指标和排序指标两类来介绍文献综述的66篇相关工作中使用到的 SDP 模型评价指标.

6.1.1 评估分类性能的指标

表 10 对分类性能评估指标的计算方法和使用情况进行了汇总. 从表 10 的倒数第 2 列可以看出, 有些指标在不同的文献中可能有多个别名. 例如, FPR、PF 和 FAR 具有相同的含义: 预测为有缺陷的干净模块的比例. 在 SDP 研究的早期文献中, “被预测为有缺陷的缺陷模块的比例”通常被称为“正确性(correctness)”, 但随后, “召回(recall)”一词变得更加常见; *PD* 和 1–*PF* 的调和平均值在一些文献中称为 G_1, 在另一些文献中称为 *G*-measure; 召回率和准确率的几何平均值称为 G_2 或 *G*-mean. 为了保持对指标的清晰引用, 对于含义相同的指标, 我们在名称栏中放置了一个更常用的指标, 而在解释栏中给出了其别名(如果有的话). 在本文的其余部分中, 我们仅使用第 1 列中给出的指标名称来保持行文一致.

表 10 评估 SDP 模型的分类指标汇总

评估指标	计算公式	说明
Recall(召回率)	$TP/(TP+FN)$	被预测为有缺陷的模块占全部有缺陷模块的比例, 也称作 PD (probability of detection)
Precision(精度)	$TP/(TP+FP)$	被预测为有缺陷的模块中实际有缺陷模块占的比例
Accuracy(准确率)	$(TP+FP)/(TP+FP+TN+FN)$	被正确预测的模块占全部模块的比例
FPR(假警报率)	$FP/(FP+TN)$	假警报率, 也称作 FAR (false alarm rate) 或者 PF (probability of false alarm). 无缺陷模块中被预测为有缺陷模块的比例
Correctness(正确性)	$TP/(TP+FP)$	与精度相同
Completeness (完整性)	$Defects(TP)/Defects(TP+FN)$	召回率的一种变体, 在被判定为有缺陷模块中找到的缺陷占所有缺陷的比例
TNR(真阴性率)	$TN/(TN+FP)$	真阴性率, 也称作 clean recall
FNR(假阴性率)	$FN/(FN+TP)$	假阴性率, 有缺陷模块中被预测为无缺陷的比例
F_β	$\frac{(1+\beta^2)\times Recall\times Precision}{Recall+\beta^2\times Precision}$	精度和召回率的调和平均值(β=1 或 2)
G_1	$\frac{2\times PD\times(1-PF)}{PD+(1-PF)}$	*PD* 和 1–*PF* 的调和平均值, 也称作 *G*-measure
G_2	$\sqrt{Recall\times Precision}$	精度和召回率的几何平均值, 也称作 *G*-mean
G_3	$Recall\times(1-PF)$	召回率和 1–*PF* 的几何平均值
MCC (Matthews 相关系数)	$\frac{TP\times TN-FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	实际分类和预测分类之间的相关系数
Balance	$1-\sqrt{\frac{(0-PF)^2+(1-PD)^2}{2}}$	PF 和 PD 之间的平衡
Clean precision (正例精度)	$TN/(TN+FN)$	无缺陷模块中被预测为无缺陷的比例
Clean F_1 (正例调和平均值)	$\frac{2\times cleanRecall\times cleanPrecison}{cleanRecall+cleanPrecison}$	无缺陷模块的 F_1, 即 *cleanRecall* 和 *cleanPrecision* 的调和平均值

6.1.2 评估排序性能的指标

表 11 对排序性能评估指标的计算方法和使用情况进行了汇总. 从表 11 中可以看出: cost-effectiveness curve 是排序场景中的一个重要的性能指标, 一系列的指标是基于它构造的.

表 11 评估 SDP 模型的排序指标汇总

评估指标	计算公式	说明
Cost-effectivenesscurve	累计工作量(*x*-轴)和累计缺陷数的(*y*-轴)构成的曲线	整个排序的成本效益曲线
AUCEC	模型 *m* 的 Cost-effectiveness 曲线的曲线下面积	成本效益曲线的曲线下面积
CE_π	$\frac{Area_\pi(m)-Area_\pi(random)}{Area_\pi(optimal)-Area_\pi(random)}$	使用 $Area_\pi(random)$和 $Area_\pi(optimal)$进行了归一化的 AUCEC. 其中, 随机(random)模型以 50%的概率判定模块有缺陷. 最优(optimal)模型代表取得最佳 cost-effectivenes 曲线的理想模型
$Popt_\pi$	$1-\Delta opt_\pi$	Δopt_π表示在检查前 π% *SLOC* 的模块时, 被评估模型与最优模型的 cost effectiveness 曲线间的面积, 等于 $Area_\pi(optimal)-Area_\pi(random)$
Normalized $Popt_\pi$	$\frac{Area_\pi(m)-Area_\pi(worst)}{Area_\pi(optimal)-Area_\pi(worst)}$	使用最优模型和最差模型进行归一化的 $Popt_\pi$. 最差模型表示在成本效益曲线上表现最差的模型
ER	$\frac{Effort(random)-Effort(m)}{Effort(random)}$	与需要实现与预测模型 *m* 相同的召回率的随机模型相比, 模型 *m* 实现的工作量的减少程度
AUC	ROC 曲线下的面积	模型将随机选择的有缺陷的模块排在随机选择的无缺陷模块前面的概率. ROC 曲线下的面积. ROC 曲线是指随着分类阈值变大, 以 Recall 为 *y* 轴、以 FAR 为 *x* 轴所形成的曲线
PofB20	–	排序中占前 20%*SLOC* 的模块中有缺陷模块的比例
NofB20	–	排序中占前 20%*SLOC* 的模块中有缺陷模块的个数
Brier score	$Brier=\frac{1}{N}\sum_{i=1}^{N}(p_i-y_i)^2$	预测概率与实际标签之间的距离之和
Calibration slope	逻辑回归模型的斜率	预测概率的方向和分布
LogLoss	$-\frac{1}{N}\sum_{i=1}^{n}(y_i\log(p_i)+(1-y_i)\log(1-p_i))$	分类损失函数[17]
Top-10-accuracy	–	检查前 10 个模块的准确率
Median percentile rank	–	我们的模型排序的已部署推送请求批次中, 已恢复推送请求的中位数百分比排名

为了绘制成本效益曲线, 首先按模型预测的缺陷可能性给目标数据集中的模块降序排序; 然后, 模块的测试/检查成本的累积百分比为 *x* 轴, 从所测试/检查的模块中发现的缺陷的累积百分比为 *y* 轴作图. 通常, 模块的测试/检查成本是通过源代码行数(SLOC)来衡量的, 这样的成本效益曲线被称为"基于 SLOC 的阿尔伯格图"[28]. 我们在文献综述中发现了许多成本效益曲线的变体, 它们的定义略有不同. 例如: Ostrand 等人[53]使用 *x* 轴为文件数目的累积百分比、*y* 轴为发现的缺陷类的累积百分比(即累积召回)的曲线来评价模型; Zhou 等人[62]构建了一条成本效益曲线, 其中, *x* 轴为模块数的累积百分比, *y* 轴为发现的缺陷类的累积百分比(即累积召回). 尽管 *x* 轴和 *y* 轴的含义略有不同, 但这些成本效益曲线总体上反映了已识别缺陷的百分比如何随着测试/检查工作量的增加而发生变化的.

在一些文献中, 成本效益曲线直接用于评估缺陷预测模型的性能[53,153]. 然而, 通过比较成本效益曲线来确定哪个缺陷预测模型性能更好并不总是显而易见的. 只有当一条曲线完全高于另一条曲线时, 才能说一个模型肯定比另一个好. 因此, 需要将成本效益曲线缩小到一个数值, 以便于进行性能比较. 几种基于成本效益曲线的数值指标已被用于缺陷预测领域. AUCEC 是指成本效益曲线下的面积, 更具体地说, 是基于 SLOC 的 Alberg diagram 下的面积. 常用的 AUCEC 与评估预测模型比较的特殊模型有 3 种: 最优模型、最差模型和随机模型. 最优模型按其缺陷密度(缺陷数/*SLOC*)的降序预测测试版本中模块的缺陷倾向, 其他预测模型的所有成本效益曲线都不会出现在这条曲线之上, 因为没有任何预测模型可以给出比这个理想排名更好的测试模块排名. 最差模型按缺陷密度的升序预测测试版本中模块的缺陷倾向, 其他预测模型的所有成本效益曲线都不会出现在这条曲线之下, 因为没有任何预测模型可以给出比它更差的测试模块排名. 随机模型以 0.5 的概率随机预测一个模块有缺陷. 预计任何提出的缺陷预测模型至少会比随机模型表现得更好. 从表 11 可以看出: 基于评估模型 *m*、最优模型、最差模型和随机模型下的面积, 当给定 π%的测试/检查工作量, 则有 CE_π是 π的

最优模型和随机模型归一化的 AUCEC; $Popt_{\pi}$为 $1-\Delta_{opt}$ (最优模型与模型 m 之间的面积); Normalized $Popt_{\pi}$是由最优模型和最差模型归一化的 AUCEC.

6.2 评价指标使用情况

图 8 和图 9 分别给出了本文的 SDP 文献回顾范围内中分类与排序指标的使用频次. 从频次图中我们得到如下观察.

- 在 66 个 SDP 研究中, 共有多达 30 个指标被使用, 被用于模型评估的评价指标种类繁多, 文献的选择偏好各不相同. 尽管相比于其他指标, Recall、AUC、Precision、F_1 和 Accuracy 这几个指标被用于缺陷模型评估的频次更高, 但是几乎不存在一个指标受到研究者们的一致认同;
- 图 8 和图 9 中全部的指标加起来被使用了 200 次, 平均一篇文献使用大约 3 个指标进行评估. 也就是说, 现有的 SDP 研究倾向于使用多种评价指标共同评估预测性能, 而表中的指标大部分互相之间并不存在一致趋势. 比如说, 过高的 Recall 往往暗示着对 Precision 的牺牲, 一个模型有高 Precision 不一定有高 Accuracy, 等等;
- 使用次数最高的前 5 个评价指标分别是 Recall、AUC、Precision、F_1 和 Accuray. 总的来说, 分类指标比排序指标的使用频次更高. 在使用次数最多的 10 个评价指标中, 排序指标只占 3 个. 在分类指标中, 最被广泛使用的指标是 Recall、Precision 和 F_1. 在排序指标中, 应用最多的是 AUC.

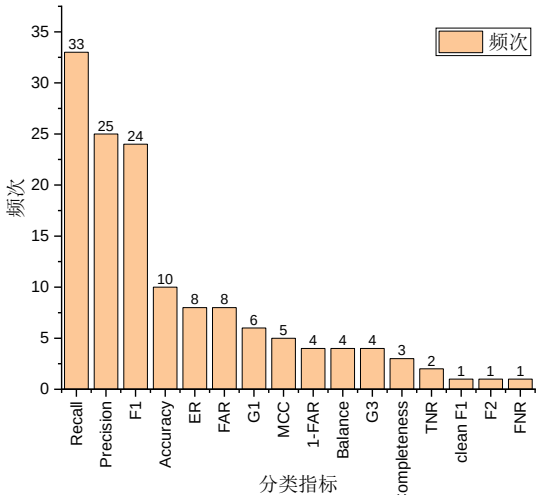


图 8 分类指标的使用频次

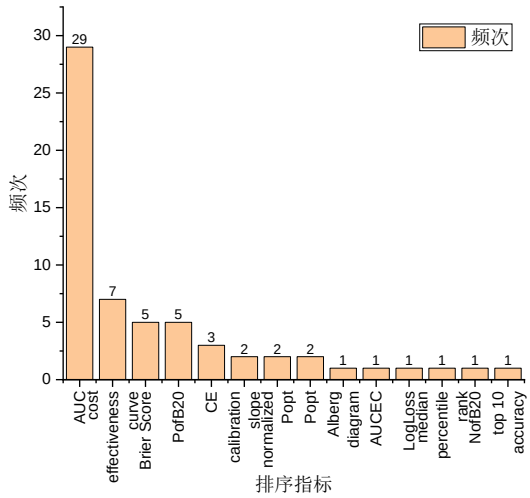


图 9 排序指标的使用频次

6.3 评价指标的使用变迁

尽管总的来说, 分类指标更多地应用于 SDP 模型之间的比较中, 但实际上, 在软件缺陷预测的早期, 模型间的比较就已经使用了排序指标. 早在 1996 年, Ohlsson 和 Alberg[153]就使用了 Alberg diagram 来比较他们所提出的缺陷预测模型与理想的最优模型之间的差距. 2005 年, Ostrand 等人[53]使用了 cost-effectiveness curve 来比较他们提出的多个模型. 在同一年, Gyimóthy 等人[45]也使用了 cost-effectiveness curve 来比较基于线性回归的预测模型和基于逻辑回归的预测模型. 在 2006 年, Zhou 等人[62]同样使用 cost-effectiveness curve 来比较多变量线性回归模型、朴素贝叶斯、随机森林和 NNge 在缺陷预测上的效果.

尽管上述早期 SDP 研究中使用了 CE 图, 也就是从工作量感知的视角来进行 SDP 模型间的比较, 但是整体来说, SDP 模型间的比较还是偏好使用分类指标, 这其中也包括 Accuracy. Accuracy 仅次于 Recall、Precision 和 F_1, 是使用频次第四高的分类指标、使用频次第五高的评价指标. 然而, Accuracy 被认为并不适合用来评估缺陷预测这种类不平衡问题(缺陷模块远远少于无缺陷模块). 在缺陷预测问题中, 高 Accuracy 并不意味着好

的预测效果, 因为在缺陷数据集中, 仅仅把全部模块都预测为无缺陷就能得到很高的 Accuracy. 另外, Recall 受阈值影响很大, 把全部模块都预测为缺陷模块就可以达到最大的召回率.

越来越多的研究者已经意识到, 选取不同的性能指标会对模型评价造成影响. 为了能够更真实地反映模型的性能, 越来越多的指标被提出并应用于 SDP. 在十几年的发展之后, 一个回顾总结惊讶地发现, 多年来并没有提升多少. 近年来, 大量的研究为了避免阈值选择对模型的评价产生偏差, 而使用阈值独立的评价指标来防止不同模型间阈值不对齐导致的不公平比较. AUC 这一阈值独立的指标格外受到 SDP 研究者们的青睐. 另外, 研究者们也提出了工作量感知的概念, 希望能在更现实的场景下对模型进行比较. 因此, IFA 和 PofB20 等工作量感知指标近年来更多地被用于 SDP 模型间的比较.

6.4 评价指标的优缺点

在 SDP 模型的比较中, 研究者们积累了一系列对评价指标优缺点的认知. 其中一些认知逐渐得到公认, 从而影响了后续研究者对于评价指标的选取. 其中一些认知互相冲突, 并且迄今未达成共识. 本节列举部分文献综述中研究者们对于 SDP 模型评价指标优缺点的评价, 为后续研究者提供参考. 表 12 总结了部分对评价指标优缺点的认知.

表 12 缺陷预测模型的常见评价指标优缺点的认知汇总

指标	优点	缺点
Recall	SDP 领域的传统指标[20,73,75,76]	在类不平衡数据集上不稳定[77]
Precision	SDP 领域的传统指标[20,73,75,76]	在类不平衡数据集上不稳定[68,77]
Accuracy	–	在类不平衡数据集上不稳定[12]
F_1	SDP 领域的传统指标[20,75,76]	在类不平衡数据集上不稳定[68,77], 计算中没有考虑到 TN[12]
AUC	独立于分类阈值[1,17] SDP 领域的传统指标[73] 不受类不平衡问题的影响[20]	未考虑 FP 和 FN 的相对成本[12]
MCC	反映了 TP、FP、TN 和 FN 这 4 个基本的分类输出结果[12], 过往研究的推荐[17]	–
G_1	在类不平衡的情况下仍然保持稳定[17]	尚未获得广泛接受[73]
G_2	–	计算中没有考虑到 TN[12]
G_3	在类不平衡的情况下仍然保持稳定[17]	根据模型在每个类上的精度进行对比并不合理[12]
Balance	过往研究的推荐[17]	尚未获得广泛接受[73]
FAR	SDP 领域的传统指标[73] 过往研究的推荐[17]	–
Brier score	独立于分类阈值[17]	–

在 2008 年, Lessmann 等人[1]在模型间的比较中使用了独立于分类阈值的 AUC 作为评价指标, 以避免选择分类阈值.

在 2015 年, Tantithamthavorn 等人[75]在研究误标签对缺陷预测模型的性能和解释的影响时, 使用的评价指标是 Precision、Recall 和 F_1, 为此, 他们给出的理由是, 因为“它们是缺陷预测领域的传统评价指标”. 类似地, 在 2015 年, Tan 等人[76]在研究不平衡数据的在线缺陷预测时, 也使用了 Precision、Recall 和 F_1 作为评价指标, 并且给出了类似的理由: “它们是缺陷预测领域的常用评价指标”. 然而, 同年发表的对于 CPDP 的隐私保护数据共享策略的研究中, Peter 等人[77]认为: 对于缺陷数据集这样的类不平衡数据集来说, Precision、Recall 和 F_1 作为评价指标的性能并不好, 他们的理由是: 这样的数据集上, Precision 的值不稳定.

在 2018 年, Bennin 等人[68]在对于缺陷预测的类不平衡问题的研究中提到: “众所周知, 对于在高度不平衡的数据集上训练的模型, Precision 是一个不稳定的指标”. 因此, 他们决定不使用 Precision 以及基于 Precision 的 F_1 作为评价指标. 同年, Agrawal 等人[73]认为: Balance 和 G_1 尚未获得广泛接受, 因而不应当被用作 SDP 模型的评价指标; 而 Recall、Precision、FAR 和 AUC 由于在文献中得到了广泛使用, 因此应当被用作评价指标.

在 2019 年, Tantithamthavorn 等人[17]在研究自动调参技术对缺陷预测模型影响的工作中, 使用了多种不同类型的评价指标用于模型间比较. 他们认为: Precision、Recall 和 F_1 是缺陷预测领域的传统指标, 因此应当被

使用; G_1、G_3 和 TNR 的优点是能够在类不平衡的情况下仍然保持稳定; MCC、FAR 和 Balance 是过往研究推荐过的评价指标. AUC、Brier score 和 LogLoss 这 3 个阈值独立指标的优点是独立于分类阈值, 因此避免了选择分类阈值对比较结果的影响以及对类不平衡问题不敏感.

Song 等人[12]在对不平衡学习在软件缺陷预测中作用的综合研究中, 指出了他们所认为的一系列常用的评价指标的缺点: 首先, 更高的 AUC 并不一定意味着更好的性能, 因为在 AUC 的计算中未考虑 FP 和 FN 带给质量保障人员的相对成本; 对于 G_3, 根据模型在每个类上的精度进行对比并不合理; F_1 和 G_2 在计算中没有考虑到 TN, 但 TN 对质量保障人员来说很重要, 因为他们会想知道组件是否真的没有缺陷; Accuracy 对类不平衡太敏感. 由于以上提到的缺点, Song 等人[12]不使用这些评价指标, 而是使用了 MCC, 因为 MCC 包含了 TP、FP、TN 和 FN 这 4 个基本的分类输出结果.

Li 等人[20]在对于 HDP 的多源及隐私保护问题的研究中提到: Menzies 等人[130]认为, Precision 对于类不平衡的数据集来说是一个极不稳定的性能指标. 因此, 文献[130]不使用 Precision、Recall 和 F_1; 并且认为, AUC 是一个用于模型间比较的有效指标, 被广泛使用, 且具有不受类不平衡影响以及独立于分类阈值的两大优点, 因此使用 AUC 作为评价指标.

由此可见, 尽管缺陷预测的研究者对于模型比较中各种评价指标有了一定的认识, 但对于在模型比较中应当使用哪些指标以及使用哪些指标不合适, 仍存在较大分歧. 尤其是被缺陷领域广泛使用的 Recall、Precision 和 F_1, 大量研究出于它们是缺陷预测领域的传统/常用指标而使用它们进行模型间比较; 可又有不少研究认为, 它们对类不平衡数据集敏感而不推荐使用. 为了能够更有依据地为缺陷预测模型比较选择评价指标, 缺陷预测模型中的评价指标的性质有待进一步加以研究.

6.5 小 结

本节从评价指标计算、评价指标使用变迁和评价指标优缺点这 3 个方面详细介绍了在近年来缺陷预测模型间的比较所使用的评价指标的发展方向. 下面简述主要的发现.

(1) 现有的缺陷预测的评价指标可大致分为分类指标和排序指标. 其中, 使用频次更多的是分类指标, 尤其是 Precision、Recall 和 F_1 这 3 个指标. 排序指标中使用最多的是 AUC, 它作为排序指标不需要分类阈值, 因而近年来受到研究者们的青睐;

(2) 以往的缺陷预测评价往往分为排序性能和分类性能, 对工作量感知指标关注不多. 但近年来, PofB20 和 IFA 等工作量感知指标被用于缺陷预测模型间的比较中, 反映了近年来对于工作量感知的缺陷预测的关注度的提升;

(3) 目前, 不同研究者对于缺陷预测模型的比较中使用的评价指标的优缺点看法不一. 也就是说, 对于在模型比较中应当使用哪些评价指标没有公认的选择. 比如: 大量研究工作选择 Precision、Recall 和 F_1 作为评价指标, 而其他一些研究却认为, 它们由于对类不平衡敏感因而不适合在缺陷数据集上使用. 我们在第 8.2 节尝试提出了一种可行的评价指标评估方案, 希望在未来能有对评价指标的进一步讨论, 并最终在本领域建立起对于评价指标的共识.

7 SDP 模型比较的阈值设定

在第 6 节我们发现, SDP 模型间的比较往往使用分类指标. 预测模型在计算分类指标时需要用到分类阈值, 即模型中 $TP+FP$ 的值, 因此被比较模型的分类阈值的选取将会影响模型间的比较结果. 按照 SDP 模型比较时分类阈值的设定方法, 过往的 SDP 研究的做法可大致分为 3 类.

(1) 模型比较所使用的指标是分类阈值无关的. 例如 AUC(ROC 曲线下的面积), 这样的指标在整个被预测模块的完整排名上进行计算, 因而避免了对于阈值选取的讨论[1,17,20];

(2) 采用默认分类阈值进行比较. 不同模型可能有不同的选择分类阈值的方法, 如: 基于传统有监督机器学习算法的缺陷预测模型通常选择算法默认的分类阈值($score$>0.5 分类为有缺陷, 其余模块预测为无缺陷); ManualDown 的默认分类阈值是目标项目中总模块数量的 50%; ManualUp 的默认分类阈

值是目标项目中总代码行数的前 20%; 而另外一些缺陷预测模型基于硬分类算法, 其分类阈值不是人为可以设定的. 在比较实验中, 使用各个模型默认的分类阈值是目前常见的实验设置[9,15,18,40], 但是这种做法会导致模型间的比较在不对齐的工作量下进行. 具体来说, 使用缺陷预测模型来进行检查或测试所需的工作量大致可分为上下文切换工作量和代码审核工作量: 上下文切换工作量可以用缺陷预测模型所要求检查的模块数量来表示, 代码审核工作量可用缺陷预测模型所要求检查的代码行数来表示. 当比较实验中各个模型使用它们各自默认的分类阈值时, 这些被比较模型极有可能需要的是不同的上下文切换工作量和代码审核工作量. 当每个被比较模型的分类结果导致的各自所需的工作量不同时, 比较他们的分类指标结果是不公平的. 举例而言, 要求质量保障人员检查50%的模块的缺陷预测模型得到的20%的缺陷模块召回率和要求质量保障人员检查10%的模块的缺陷预测模型得到的20%的缺陷模块召回率的含金量不同;

(3) 工作量阈值对齐的比较. 模块之间在对齐上下文切换工作量或者对齐代码审核工作量的情况下进行了比较. 例如, 使用 NofB20 进行模型间的比较, 也就是比较检查 20% *SLOC* 时模型找到的缺陷数目[63], 对齐了代码审核工作量. 使用 top-10-accuracy, 比较检查前 10 个模块得到的 Accuracy, 对齐了上下文切换工作量.

表 13 对这 3 种分类阈值设置方式进行了汇总. 由于一次模型评估往往使用多个指标进行模型间的比较, 其中有的指标计算时可能未对被比较模型的工作量进行对齐, 有的指标计算时对工作量进行了对齐, 而有的指标可能是分类阈值无关的, 因此一次模型评估可能涉及到上述情况的一个或多个. 为了探究过往文献的阈值设定方式以及对模型间比较造成的潜在影响, 我们标注了被调研的 66 篇文献中模型比较的阈值设定方式.

表 13 分类阈值设置汇总

设置方式	优点	缺点
选择分类阈值无关的评价指标(不设置分类阈值)	避免了选择为每个被比较模型选择分类阈值	更好的分类阈值指标值、更好的指标值并不意味更大的实用价值
默认分类阈值	采用默认分类阈值简单, 无需额外的工作量对齐操作	使用模型各自的分类阈值则各模型所需的工作量不对齐
工作量对齐	模型在公平的设置下进行比较, 可以选出符合质量保障人员实际需求的模型	需要自行决定可用工作量的值

7.1 分类阈值无关的比较

一些论文使用分类阈值独立指标来评价和比较预测模型, 这样的论文不需要选择分类阈值[14,16,64]. 例如: AUC 是在整个模块缺陷可能性排序上进行计算的, 不需要利用分类结果, 因而其计算独立于分类阈值选择. 鉴于 AUC 仅次于 Recall 是文献回顾中使用频率第二高的评价指标, 因此可见大量缺陷预测研究都使用了默认分类阈值来进行模型间比较. 使用分类阈值独立指标进行模型间的比较实验, 可以避免为每个模型选择分类阈值, 对这一性质的重视也是 AUC 使用频率高的重要原因(参见第 6.4 节对于评价指标优缺点的观点汇总).

使用分类阈值无关的评价指标的问题是, 更好的指标值并不意味着更大的实用价值. 比如图 10 中展示了两个缺陷预测模型对一个拥有 4 个有缺陷模块(标记为 1)和 4 个无缺陷模块(标记为 0)的项目给出的预测结果的 ROC 曲线. 可以看出: 模型 *A* 的特点是在检查的中段模型表现较好; 模型 *B* 的特点是在检查的早期表现很好, 但在检查的中段假警报率很高. 尽管最终模型 *A* 和 *B* 的 AUC 值相同(ROC 曲线下面积都为 0.5), 然而这并不意味着 *A* 和 *B* 的预测性能在给定工作量下相同. 显然, 当质量保障人员仅来检查两个模块的资源时, 模型 *B* 的表现会远好于模型 *A*(100%的召回率对比 0%召回率). 也就是说, 分类阈值无关的比较方法在整个排序下进行模型间比较, 而忽略了模型在给定工作量下的表现, 违背了缺陷预测的实际应用情形(质量保障人员显然不能检查整个模块排序).

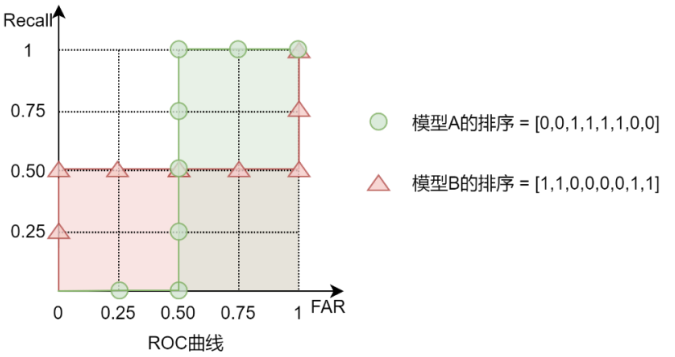


图 10　两个缺陷预测模型对相同项目排序下的 ROC 曲线

7.2 采用默认分类阈值进行比较.

在模型比较中使用分类器的默认分类阈值，会导致预测为有缺陷的模块数量、代码行数不同. 也就是说，带给质量保障人员的工作量不一致. 大部分的模型间比较都使用了各自分类器的默认分类阈值，并导致被比较模型的工作量不对齐[9,15,18,40]. 例如: 在 2020 年，在使用软件可视化和深度迁移学习进行有效软件缺陷预测的研究[9]中，各个基于深度学习的预测模型是用各自默认的分类阈值来计算并比较 F_1 的. 在 2020 年, Wang 等人[40]在研究缺陷预测模型的深度语义特征学习时，也使用了 ADtree、朴素贝叶斯等分类算法各自的默认分类阈值来计算并比较它们的 Recall、Precision 和 F_1. 2019 年，Gong 等人[19]在对比不同分类器的预测性能时，就使用了被比较分类器(NB、RF、SVM、LR 和 KNN)各自的默认分类阈值来计算分类指标 Balance 和 Recall.

在 2019 年，在对即时软件缺陷预测中的类不平衡演化和验证延迟的研究中，Cabral 等人[41]比较了不同的类不平衡进化学习方法在缺陷预测上的性能，在在线学习场景下计算并比较了不同方法的 G_3，Recall 和 1–*FAR*. 在 2020 年，在一项跨项目在线即时缺陷预测研究[72]中，比较了不同的训练数据选择方法对于预测性能的影响(G_3、Recall 和 1–*FAR*). 在线学习方法的场景决定了分类阈值无法被调节，因而不同的在线学习方法的工作量是不对齐的.

在 2018 年, Zhou 等人[28]在对 CPDP 研究的回顾中，使用 ManualDown 和 ManualUp 作为基线模型对过往的 CPDP 模型做了大规模比较实验，其中，过往模型使用的是过往文献中的数值结果，而不是重新复现. 也就是说，被比较的过往模型采用了它们原本的分类阈值，而基线模型 ManualDown 和 ManualUp 采用了 Zhou 等人另外设置的分类阈值(分别是 50%的总模块数量和 20%的总代码行数)，因此在这些比较中，被比较模型的工作量是不对齐的. 类似地，在 2019 年, Hosseini 等人[30]对 CPDP 研究进行了系统性文献综述与元分析，他们: (1) 比较了不同的特征集训练出的模型的性能; (2) 比较了不同 classifiers 训练出的模型的性能; (3) 比较了用于 CPDP 的数据预处理技术哪个有用; (4) 比较了 CPDP 技术和 WPDP 技术. 在这一系列元分析中，不同技术间的比较直接使用了过往文献的数值结果，而不是重新复现并进行实验，因此这些比较中的各模型工作量也是不对齐的.

使用启发式规则等硬分类算法来构建缺陷预测模型的做法常见于早期的缺陷预测研究. 在 2006 年，Song 等人[58]在对软件缺陷关联挖掘的研究中，基于缺陷关联规则来预测缺陷，并对比了不同规则数目下的预测结果，基于预测结果对比了不同规则模型下的 FAR、FNR 和 Accuracy. 基于规则的方法是硬分类方法，无法进行工作量对齐. 在 2007 年，Kim 等人[83]提出了用启发式规则来维护缺陷缓冲池的缺陷预测方法 BugCache 和 FixCache，这两种方法是启发式方法，属于硬分类方法，分类阈值取决于规则在数据集上的执行结果，无法调节，不同规则在目标数据集上得到的分类阈值往往是不同的，因此难以通过人为划分分类阈值的方法对齐它们的工作量.

7.3 工作量对齐的比较

为了提升缺陷预测的实用价值, 近年来缺陷预测领域提升了在工作量感知场景下评价缺陷预测模型的重视程度. 因而, 越来越多的文献选择对齐被比较模型的工作量. 工作量可以分为代码审核工作量和上下文切换工作量: 对齐代码审核工作量对应的是对齐模型检查的代码行数, 对齐上下文切换工作量对应的是对齐模型检查的模块数. 在这两种阈值对齐方法中, 对齐模块检查的代码行数在现有的研究工作中更为常见. 这一定程度上是因为出现在文献中的工作量感知的评价指标中, 给定检查代码行数来评价模型的评价指标远多于给定检查模块数来评价模型的评价指标. 下面列举一些今年来在模型比较中对齐了代码审核工作量的文献.

在 2016 年, Yang 等人[11]在对基于依赖聚类的工作量感知缺陷预测的实证研究中, 提出了一个用以提升工作量感知性能的分段预测模型. 模型评估使用的指标是 Popt (20% *SLOC*)、ER (20% *SLOC* 和 100% *SLOC*)和 CE (20% *SLOC*). 这 3 个指标都对齐了模型的代码审核工作量.

在 2017 年, Fu 等人[3]回顾了 Yang 等人对于工作量感知的 JIT SDP 的结论, 并提出了一个 OneWay 方法: 先用无监督方法选出最好的那个度量, 然后用这个单一的度量训练一个监督模型. 选择了 EALR、KNN 和 J48 作为基线模型. 评价指标包含 Popt, 因此对齐了代码审核工作量.

在 2019 年, Yu 等人[60]在对实际应用中的并发缺陷预测的研究中, 使用了 PofB20 以及 Popt, 在代码审核工作量对齐的情况下, 比较了 ConPredictor 与基线模型的性能.

在 2020 年, Wang 等人[40]在对面向软件缺陷预测的深度语义特征学习的研究中, 对比了基于语义特征和基于传统特征的模型的 PofB20. PofB20 表示模型在审查前 20% *SLOC* 的 Recall, 因此在比较该指标时, 被比较模型的代码审核工作量是对齐的.

在 2021 年, 在使用类依赖网络中的 *K* 核分解来提升缺陷预测模型的工作量感知性能的研究中[4], 研究者将他们提出的基于 top-core 的缺陷预测模型与两个基线模型(分别基于 pageRank 和 betweenness 技术)相比较, 从而研究 top-core 相较于基线模型对目标数据集上的 Popt (*SLOC*=20%)、Popt (*SLOC*=30%)和 Popt (*SLOC*=40%)的提升. 实验的重点是从实用角度研究新模型的预测性能, 使用了工作量感知指标 *Popt_pi*=20, 30, 40 作为评价指标, 因而被比较模型的代码审核工作量是对齐的.

除了近年来常用的工作量感知指标 Popt 和 PofB20 之外, 更早的时间也有一些文献, 选择基于 cost effectiveness curve 的评价指标来比较模型, 同样对齐了代码审查工作量[5,42,67,81].

而在我们的文献综述中出现的对齐代码上下文切换工作量的指标只有 top-10-accuracy[16]和基于模块百分比的 cost-effectiveness curve[45,62,153].

7.4 小 结

在过往的预测模型的比较极少对齐被比较模型所需的工作量(在相同的上下文切换工作量下进行比较, 或者在相同的代码审核工作量下进行比较). 我们标注了被调研的 66 篇文献中模型比较的情况, 得到图 11 中的韦恩图.

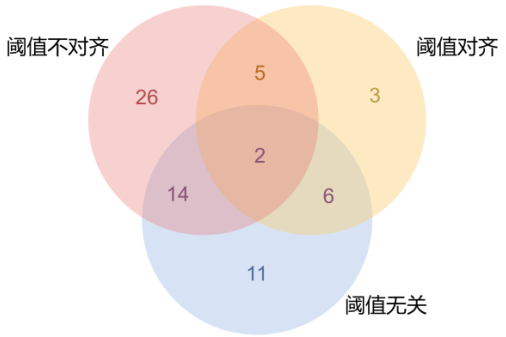


图 11 缺陷预测模型之间的比较中不同阈值处理方法次数的韦恩图

结合图 11 以及前文的分析, 可以得到如下结论.

(1) 高达 70.1%的文献在工作量不对齐的情况下进行了模型间的比较. 由于在比较时没有对齐被比较模型的工作量, 大量的缺陷预测模型评估得到的结论可能有潜在的误导性;

(2) 49.3%的文献使用了分类阈值无关的指标进行了模型间的比较, 这是当前文献中减轻或者避免分类阈值的选取不同带来的评价偏差的首选做法; 16.4%的文献完全使用分类阈值无关的指标进行了模型间的比较, 从而避免了对于分类阈值选择的讨论;

(3) 仅有 4.5%的文献完全在工作量对齐的情况下进行模型间的比较, 这表明工作量对齐对于公平比较的重要性并未引起研究者们的重视;

(4) 在涉及到使用工作量对齐的指标进行模型间比较的总共 16 篇文献中, 12 篇对齐了代码审查工作量, 仅有 4 篇对齐了上下文切换工作量, 所以目前对于测试/审查工作量的研究主要关注代码审查工作量, 而忽视了上下文切换工作量. 与此同时, 没有相关的研究工作同时考虑这两个工作量. 综上所述, 大量的模型间的比较实验是在不公平的阈值设置下进行的, 因此可能会导致新提出的模型的有效性评估存在偏差.

8 现有挑战及研究方向

8.1 SDP模型比较实验的现有挑战

从文献回顾的结果来看, 当前缺陷预测模型的比较框架中还存在着若干需要解决的挑战. 本节分别从缺陷数据集以及数据集划分方法、基线模型选择、评价指标选择以及模型阈值对齐来介绍缺陷预测模型比较框架所面临的挑战.

(1) 缺陷数据集的挑战. 通过汇总分析用于缺陷预测模型间的比较实验的 12 个公开的缺陷数据集, 我们发现这些缺陷数据集在收集和使用过程中存在一些可能会影响缺陷预测模型间的比较实验的公平性的问题, 包括涵盖的编程语言不够全面[9,18,54,55,59,68,69,73,77,80]、数据集质量各异[10,108]以及比较实验常在不充分的数据集上进行[13,50,82], 等等.

(2) 数据集划分的挑战. 比较实验中, 数据集划分是重要的实验设置, 不同划分方式的选择对应着在不同场景下验证模型的性能, 需要谨慎选择. 通过文献回顾我们发现, 当前缺陷预测模型间的比较实验中的数据集划分存在如下问题.

- 大规模使用了不符合实际场景的交叉验证方式[1,5,11,12,42,43,45,47,50,54,56–58,68,71,73,77–79,81,82], 交叉验证用某个版本中的一部分模块来预测另一部分模块很有可能会违反时序原则;
- 即便给定场景下的模型验证也缺乏统一的划分方式, 同时对划分方式也缺乏一致的认知. 如: CPDP 场景下, 有些实验选择 C-one[69], 而另外一些实验选择 C-all[69]; WPDP 场景下, 有些实验选择 SPV[4], 而有些实验选择 APV[34].

(3) 基线模型选择的挑战. 缺陷预测研究领域对于哪个缺陷预测模型在何种情况下适合比较实验的基线模型缺乏统一认知, 文献回顾中, 大量的缺陷预测研究各自选择多个不同的基线模型(有监督基线模型、无监督基线模型以及 CPDP 场景下的基线模型等)进行比较[64]. 缺陷预测领域缺乏一个统一的基线模型被持续用来衡量新模型对缺陷预测性能的提升.

(4) 评价指标选择的挑战. 研究人员对于缺陷预测模型间的比较应该使用哪个或者哪些指标并没有一致的看法. 进一步说, 研究人员对于这些评价的优缺点认知不统一. 例如: 针对被频繁用于模型评估的 AUC 指标, Li 等人[20]选择用它进行模型评估是因为 AUC“被称为比较不同模型的有用度量, 并且不受类别不平衡的影响以及独立于预测阈值”; 然而, Song 等人[12]却不推荐使用 AUC 进行模型评估, 因为“高 AUC 并不一定意味着好表现, 鉴于 FP 和 FN 的相对花销是未知的”. 再比如: 尽管 Li 等人[20]、Bennin 等人[68]以及 Peters 等人[77]都认为, 对于有缺陷模块只占低百分比的缺陷数据集, Precision、Recall、Accuracy 和 F_1 等指标是非常不稳定的性能指标, 因此不推荐使用这些指标进行

模型评估. 但是大量的研究无视了这个观点: 从第 6.2 节对于评价指标使用频次的调研中可以看出, 这 4 个指标仍然几乎是最常用的模型评价指标.

(5) 模型阈值对齐的挑战. 被调研的 66 篇文献中, 70%以上的文献在阈值不对齐的情况下进行了模型间的比较[9,15,18,40]. 由于在比较时没有对齐被比较模型的工作量, 大量的缺陷预测模型评估得到的结论可能有潜在的误导性. 当前文献中, 减轻或者避免阈值的选取不同带来的评价偏差的首选做法是选取阈值无关指标(如 AUC)[14,16,64], 但 AUC 不能反映模型的工作量. 极少数的文献完全在阈值对齐的情况下进行模型间的比较, 这表明阈值对齐对于公平比较的重要性并未引起研究者们的重视. 并且, 目前对于测试/审查工作量的研究主要关注代码审查工作量[3,11,60], 而忽视了上下文切换工作量, 更鲜少有研究同时考虑这两个工作量. 总之, 大部分模型间的比较是在不公平的环境下进行的, 因此可能导致比较结论产生偏差.

8.2 SDP模型比较领域的研究方向

针对缺陷预测模型间的比较实验中存在的问题以及现有研究中的结论, 我们针对总结了实现公平的缺陷预测模型间进行比较可能需要得到研究者对以下研究方向给予重点关注.

(1) 提供统一的基线模型. 正如第 5.4 节中所说: 尽管每个模型都选择自己所认知的当前领域内的先进模型作为基线模型来证明自身的有效性看似是一种直观的有效性证明途径, 但在实际操作中, 不同作者偏好选择各自所认为的先进模型作为基线模型, 尽管每项研究都证明自己的性能超越了基线模型, 但是由于缺乏共同的基线模型, 我们无法确定这些新模型的相对有效性. 缺乏一个统一的基线模型作为全局参考点, 会导致我们无法确切衡量目前缺陷预测领域究竟在多大程度上提升了缺陷预测性能. 另外, 选择最新的先进模型进行比较时, 优于许多这样的模型是不开源的, 因此比较实验中不得不复现这些模型, 在复现中引入的微小偏差都有可能导致先进模型的性能退化, 从而报告出虚假的新模型的性能提升. 因而, 研究者需要探索一个简单且表现良好的基线模型, 新提出的模型只需要与这个基线模型作比较. 带来的好处是后续的研究者能够持续地了解新提出的模型们相比于这个基线模型给缺陷预测带来的提升. 为此, 我们建议研究者在设计缺陷预测比较实验中被持续使用的基线模型时, 要遵循基线模型的一些基本要求[154], 例如模型要足够简单, 简单到其他研究者在重复实现时几乎不太可能引入实现偏差; 再比如, 模型要有合理的预测性能, 足以作为基线模型去评估新提出的模型, 等等.

(2) 设计合理的工作量感知指标. 从第 6.2 节中可以看出, 缺陷预测模型间比较实验选择的主流评价指标是非工作量感知指标[20,75,76]. 近年来, 大量的工作量感知指标也得到使用[5,42,67,81]. 总体上说, 不同研究者对于在模型比较中应当使用哪些评价指标并无公论. 我们认为, 缺陷预测领域需要进一步讨论并最终建立起对于评价指标的共识. 基于这样的共识, 一套经过了合理设计的评价指标应当被持续地用于缺陷预测模型间的比较实验. 一种可行的评价指标评估方案是: 以不同的评价指标或评价指标的组合作为优化目标, 训练不同的有监督模型, 将这些有监督模型返回的缺陷预测结果交由工业界的质量保障人员评判, 从中选择使得质量保障人员最满意的优化目标, 即选出了最能帮助选出符合质量保障人员要求的模型的评价指标.

(3) 对齐模型间的工作量. 正如第 7 节所说, 现有的模型间比较存在严重的工作量不对齐问题[19,28,41,72], 这些模型通常使用它们各自的分类算法的默认分类阈值, 这样的实验设置会导致被比较模型各自所分类为有缺陷的模块在被审查的时候需要不同的代码审查工作量和上下文切换工作量. 在实际应用中, 质量保障人员需要的是在更少的工作量下能发现更多缺陷模块的缺陷预测模型. 因此, 在工作量不对齐的实验设置下进行模型间比较不仅是不公平的实验设置, 而且不能帮助质量保障人员从实用性的角度出发选择更优的模型. 使用工作量对齐方法来保证比较模型间的公平比较, 对发现有实用价值新缺陷预测模型至关重要. 一种可行的做法是: 在模型间比较时, 在获得每个模型对于目标项目的缺陷可能性排名后, 通过限定检查每个模型前 X%的总模块数或总代码行数并计算各

自的性能指标的方式, 来对齐被比较模型的上下文切换工作量或代码审查工作量.

(4) 关注上下文切换工作量. 缺陷预测模型所需的上下文切换工作量的多少, 决定了测试人员是否需要频繁地在待检查模块之间进行上下文切换, 而过大的上下文切换工作量会损害测试人员的耐心. 在对于 66 篇文献的回顾中, 仅有不到 6%的文献在模型间的对比实验中对齐了上下文切换工作量[16,45,62,153]. 上下文切换工作量对模型比较的影响目前并未得到足够的重视. 我们建议研究者们为对齐上下文切换工作量的缺陷预测模型间的比较实验投入更多的关注.

(5) 收集多元化的高质量数据集. 当前, 用于缺陷预测研究的缺陷数据集存在一些问题(参见第 6.2 节). 尽管研究领域已经发布了一系列公开的缺陷数据集, 但考虑到它们只涵盖了有限的编程语言[9,18,54,55,59,68,69,73,77,80]并且可能仍存在质量问题[10,108], 总体上, 样本数量不多[13,50,82]. 我们建议研究者继续收集多元化的高质量缺陷数据集, 这对进行公平的缺陷预测模型间的比较至关重要.

(6) 在多种数据集划分下全面评估缺陷预测模型. 目前, 缺陷预测模型间的比较实验中常见的实验设置是在单一的数据集划分方式下评估模型[13,14,18]. 正如第 4.2 节所说, 缺陷预测模型的建模与预测是独立于数据集划分方式的, 所以缺陷预测模型没有严格的使用场景限制. 为了全面评估缺陷预测模型, 模型间的比较实验应当在多种数据集划分下进行. 基于现有的预测场景, 我们建议研究者在比较实验中选择的数据集划分方式至少应当涵盖 CPDP 和 WPDP 这两种场景.

(7) 整合各类优化, 最终提供统一的公平比较框架. 不少缺陷预测研究公开了他们的实验程序和数据[4,9,14,20,64], 但这些程序大多并未把为后续研究者提供实验的统一框架作为设计目的. Herbold 等人[29]为 CPDP 实验提供了一个实验框架 Crosspare, 但截止到 2021 年 8 月, Crosspare 并不支持 CPDP 以外的实验配置, 如非监督缺陷预测或者 WPDP 数据划分方法. 据我们所知, 缺陷预测研究领域目前缺少一个用于配置缺陷预测模型间的比较实验的框架. 我们建议, 这个框架应当提供多种基线模型、全面的评价指标计算、公平的阈值设定、多元的缺陷数据集、完善的数据集划分方式等. 这样的框架提供流程化配置比较实验的方法, 以开源工具的形式发布, 将帮助其他研究者以统一规范的方式实现模型间的公平比较.

9 结束语

软件缺陷预测(SDP)领域的发展依赖于 SDP 模型性能的持续提升. 模型的性能提升往往通过模型间的比较实验加以验证, 因而可靠的预测性能提升依赖规范公平的模型间比较实验. 经过几十年的发展, 大量新颖的 SDP 模型被提出, 并在比较实验中展示出优于过往模型的性能. 而实际上, 缺陷预测模型在工业界的应用效果并不如在学术界的实验中所展示的那样有效, 其原因之一是, SDP 研究领域对如何规范公平进行 SDP 模型间的比较实验缺乏一致认知, 从而导致新模型的性能提升的验证是从有失规范公平的模型间比较实验中得出的. 据我们所知, 迄今尚未有工作对缺陷预测模型间的比较实验进行总结(截至 2021 年 8 月). 为了弥补这一研究空白, 本文大规模评估了 SDP 模型间的比较实验的现有做法, 分析现有做法中存在的挑战, 为建立规范公平的 SDP 模型间比较实验提供指导. 具体而言, 本文对 2005–2021 年间涉及到缺陷预测模型间比较的研究工作进行了梳理和总结, 从缺陷数据集、数据集划分、基线模型、评价指标和阈值设定这 5 个维度, 对过往发表在软件工程领域国际主流期刊或会议的相关工作总结分析 SDP 模型间比较实验的现有做法. 本文主要工作总结如下.

(1) 汇总分析了 SDP 模型间的比较实验中使用的缺陷数据集及存在的问题;

(2) 梳理了 SDP 模型间的比较实验的验证场景(即数据集划分方式), 包括交叉验证、CPDP 和 WPDP, 并分析了当前比较实验中采用的数据集划分方式存在的问题;

(3) 针对 SDP 模型间比较实验的基线模型, 分别介绍了有监督基线模型、无监督基线模型和 CPDP 专用的基线模型以及当前 SDP 模型间比较实验在基线模型选择上存在的问题;

(4) 针对 SDP 模型比较的评价指标, 从评价指标的计算、评价指标使用偏好变迁及评价指标的优缺点这

3 个方面总结了当前 SDP 模型间比较中评价指标选取上存在的问题;

(5) 分别根据阈值不对齐、阈值对齐及默认阈值这 3 类对 SDP 模型间比较实验中的工作量对齐情况进行分析总结, 指出比较中普遍存在的工作量不对齐问题对比较公平性的威胁.

最终, 本文总结了当前 SDP 模型间的比较实验面临的问题挑战, 并针对性地指出了未来的研究方向.

致谢 感谢编辑以及匿名审稿专家提出的宝贵意见, 提高了这篇综述的质量.

References:

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

[2] Nam J, Kim S. Clami: Defect prediction on unlabeled datasets (t). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). 2015. 452−463. [doi: 10.1109/ASE.2015.56]

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

[4] Qu Y, Zheng QH, Chi JL, Jin YX, He AC, Cui D, Zhang HS, Liu T. Using *k*-core decomposition on class dependency networks to improve bug prediction model's practical performance. IEEE Trans. on Software Engineering, 2021, 47(2): 348−366. [doi: 10.1109/TSE.2019.2892959]

[5] Yang YB, Zhou YM, Lu HM, Chen L, Chen ZY, Xu B, Leung H, Zhang ZY. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. IEEE Trans. on Software Engineering, 2014, 41(4): 331−357. [doi: 10.1109/TSE.2014.2370048]

[6] Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In: Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. 2009. 91−100. [doi: 10.1145/1595696.1595713]

[7] Lewis C, Lin Z, Sadowski C, Zhu X, Ou R, Whitehead EJ. Does bug prediction support human developers? Findings from a Google case study. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). 2013. 372−381. [doi: 10.1109/ICSE.2013.6606583]

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

[9] Chen JY, Hu KY, Yu Y, Chen ZZ, Xuan Q, Liu Y, Filkov V. Software visualization and deep transfer learning for effective software defect prediction. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. 2020. 578−589. [doi: 10.1145/3377811.3380389]

[10] Shepperd M, Song Q, Sun Z, Mair C. Data quality: Some comments on the nasa software defect datasets. IEEE Trans. on Software Engineering, 2013, 39(9): 1208−1215. [doi: 10.1109/TSE.2013.11]

[11] Yang YB, Harman M, Krinke J, Islam S, Binkley D, Zhou YM, Xu BW. An empirical study on dependence clusters for effort-aware fault-proneness prediction. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). 2016. 296−307. [doi: 10.1145/2970276.2970353]

[12] Song QB, Guo YC, Shepperd M. A comprehensive investigation of the role of imbalanced learning for software defect prediction. IEEE Trans. on Software Engineering, 2018, 45(12): 1253−1269. [doi: 10.1109/TSE.2018.2836442]

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

[14] Li K, Xiang ZL, Chen T, Wang S, Tan KC. Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: An empirical study. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. 2020. 566−577. [doi: 10.1145/3377811.3380360]

[15] Tantithamthavorn C, Hassan AE, Matsumoto K. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Trans. on Software Engineering, 2020, 46(11): 1200−1219. [doi: 10.1109/tse.2018.2876537]

[16] Suh A. Adapting bug prediction models to predict reverted commits at wayfair. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. 2020. 1251−1262. [doi: 10.1145/3368089.3417062]

[17] Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. The impact of automated parameter optimization on defect prediction models. IEEE Trans. on Software Engineering, 2019, 45(7): 683−711. [doi: 10.1109/tse.2018.2794977]

[18] Wen M, Wu RX, Cheung SC. How well do change sequences predict defects? Sequence learning from software changes. IEEE Trans. on Software Engineering, 2018, 46(11): 1155−1175. [doi: 10.1109/TSE.2018.2876256]

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

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

[21] Wang Q, Wu SJ, Li MS. Software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2008, 19(7): 1565−1580 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1565.htm [doi: 10.3724/SP.J.1001.2008.01565]

[22] Catal C, Diri B. A systematic review of software fault prediction studies. Expert Systems with Applications, 2009, 36(4): 7346−7354. [doi: 10.1016/j.eswa.2008.10.027]

[23] Catal C. Software fault prediction: A literature review and current trends. Expert Systems with Applications, 2011, 38(4): 4626−4636. [doi: 10.1016/j.eswa.2010.10.024]

[24] Hall T, Beecham S, Bowes D, Gray D, Counsell S. A systematic literature review on fault prediction performance in software engineering. IEEE Trans. on Software Engineering, 2012, 38(6): 1276−1304. [doi: 10.1109/TSE.2011.103]

[25] Catal C. A comparison of semi-supervised classification approaches for software defect prediction. Journal of Intelligent Systems, 2013, 23(1): 75−82. [doi: 10.1515/jisys-2013-0030]

[26] Malhotra R. A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 2015, 27: 504−518. [doi: 10.1016/j.asoc.2014.11.023]

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

[28] Zhou YM, Yang YB, Lu HM, Chen L, Li YH, Zhao YY, Qian JY, Xu BW. How far we have progressed in the journey? An examination of cross-project defect prediction. ACM Trans. on Software Engineering and Methodology, 2018, 27(1): 1−51. [doi: 10.1145/3183339]

[29] Herbold S, Trautsch A, Grabowski J. A comparative study to benchmark cross-project defect prediction approaches. In: Proc. of the 40th Int'l Conf. on Software Engineering. 2018. 1063−1063. [doi: 10.1145/3180155.3182542]

[30] Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans. on Software Engineering, 2019, 45(2): 111−147. [doi: 10.1109/tse.2017.2770124]

[31] Cai L, Fan YR, Yan M, Xia X. Just-in-time software defect prediction: Literature review. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1288−1307 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5713.htm [doi: 10.13328/j.cnki.jos.005713]

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

[33] Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. on Software Engineering, 2016, 43(1): 1−18. [doi: 10.1109/TSE.2016.2584050]

[34] Amasaki S. Cross-Version defect prediction: Use historical data, cross-project data, or both? Empirical Software Engineering, 2020, 25(2): 1573−1595. [doi: 10.1007/s10664-019-09777-8]

[35] Li N, Shepperd M, Guo Y. A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology, 2020, 122: 106287. [doi: 10.1016/j.infsof.2020.106287]

[36] Chen X, Mu Y, Liu K, Cui Z, Ni C. Revisiting heterogeneous defect prediction methods: How far are we? Information and Software Technology, 2021, 130: 106441. [doi: 10.1016/j.infsof.2020.106441]

[37] Xu Z, Li L, Yan M, Liu J, Luo X, Grundy J, Zhang Y, Zhang X. A comprehensive comparative study of clustering-based unsupervised defect prediction models. Journal of Systems and Software, 2021, 172: 110862. [doi: 10.1016/j.jss.2020.110862]

[38] Ni C, Xia X, Lo D, Chen X, Gu Q. Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. IEEE Trans. on Software Engineering, 2020. [doi: 10.1109/TSE.2020.3001739]

[39] Liu YF, Zhang HH, Wu YC. Hard or soft classification? Large-margin unified machines. Journal of the American Statistical Association, 2011, 106(493): 166−177. [doi: 10.1198/jasa.2011.tm10319]

[40] Wang S, Liu TY, Nam J, Tan L. Deep semantic feature learning for software defect prediction. IEEE Trans. on Software Engineering, 2018, 46(12): 1267−1293. [doi: 10.1109/TSE.2018.2877612]

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

[42] 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): 1−51. [doi: 10.1145/2556777]

[43] Dejaeger K, Verbraken T, Baesens B. Toward comprehensible software fault prediction models using Bayesian network classifiers. IEEE Trans. on Software Engineering, 2013, 39(2): 237−257. [doi: 10.1109/tse.2012.20]

[44] Di Nucci D, Palomba F, De Rosa G, Bavota G, Oliveto R, De Lucia A. A developer centered bug prediction model. IEEE Trans. on Software Engineering, 2017, 44(1): 5−24. [doi: 10.1109/TSE.2017.2659747]

[45] Gyimothy T, Ferenc R, Siket I. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. on Software Engineering, 2005, 31(10): 897−910. [doi: 10.1109/tse.2005.112]

[46] Jing XY, Wu F, Dong XW, Xu BW. An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans. on Software Engineering, 2017, 43(4): 321−339. [doi: 10.1109/tse.2016.2597849]

[47] Lee T, Nam J, Han D, Kim S, In HP. Developer micro interaction metrics for software defect prediction. IEEE Trans. on Software Engineering, 2016, 42(11): 1015−1035. [doi: 10.1109/TSE.2016.2550458]

[48] Marcus A, Poshyvanyk D, Ferenc R. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Trans. on Software Engineering, 2008, 34(2): 287−300. [doi: 10.1109/TSE.2007.70768]

[49] McIntosh S, Kamei Y. Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. IEEE Trans. on Software Engineering, 2017, 44(5): 412−428. [doi: 10.1109/TSE.2017.2693980]

[50] Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. IEEE Trans. on Software Engineering, 2006, 33(1): 2−13. [doi: 10.1145/1368088.1368114]

[51] Nam J, Fu W, Kim S, Menzies T, Tan L. Heterogeneous defect prediction. IEEE Trans. on Software Engineering, 2017, 44(9): 874−896. [doi: 10.1109/TSE.2017.2720603]

[52] Olague HM, Etzkorn LH, Gholston S, Quattlebaum S. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Trans. on Software Engineering, 2007, 33(6): 402−419. [doi: 10.1109/TSE.2007.1015]

[53] Ostrand TJ, Weyuker EJ, Bell RM. Predicting the location and number of faults in large software systems. IEEE Trans. on Software Engineering, 2005, 31(4): 340−355. [doi: 10.1109/TSE.2005.49]

[54] Palomba F, Zanoni M, Fontana FA, De Lucia A, Oliveto R. Toward a smell-aware bug prediction model. IEEE Trans. on Software Engineering, 2017, 45(2): 194−218. [doi: 10.1109/TSE.2017.2770122]

[55] Peters F, Menzies T, Gong L, Zhang H. Balancing privacy and utility in cross-company defect prediction. IEEE Trans. on Software Engineering, 2013, 39(8): 1054−1068. [doi: 10.1109/TSE.2013.6]

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

[57] Song QB, Jia ZH, Shepperd M, Ying S, Liu J. A general software defect-proneness prediction framework. IEEE Trans. on Software Engineering, 2010, 37(3): 356−370. [doi: 10.1109/TSE.2010.90]

[58] Song Q, Shepperd M, Cartwright M, Mair C. Software defect association mining and defect correction effort prediction. IEEE Trans. on Software Engineering, 2006, 32(2): 69−82. [doi: 10.1109/TSE.2006.1599417]

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

[60] Yu TT, Wen W, Han X, Hayes JH. Conpredictor: Concurrency defect prediction in real-world applications. IEEE Trans. on Software Engineering, 2018, 45(6): 558−575. [doi: 10.1109/TSE.2018.2791521]

[61] Zhang F, Hassan AE, McIntosh S, Zou Y. The use of summation to aggregate software metrics hinders the performance of defect prediction models. IEEE Trans. on Software Engineering, 2016, 43(5): 476−491. [doi: 10.1109/TSE.2016.2599161]

[62] 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]

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

[64] Li K, Xiang ZL, Chen T, Tan KC. Bilo-CPDP: Bi-level programming for automated model discovery in cross-project defect prediction. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). 2020. 573−584. [doi: 10.1145/3324884.3416617]

[65] Jing XY, Wu F, Dong XW, Qi FM, Xu BW. Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. 2015. 496−507. [doi: 10.1145/2786805.2786813]

[66] Nam J, Kim S. Heterogeneous defect prediction. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. 2015. 508−519. [doi: 10.1145/2786805.2786814]

[67] Rahman F, Posnett D, Herraiz I, Devanbu P. Sample size vs. Bias in defect prediction. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. 2013. 147−157. [doi: 10.1145/2491411.2491418]

[68] Bennin KE, Keung J, Phannachitta P, Monden A, Mensah S. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. In: Proc. of the 40th Int'l Conf. on Software Engineering. 2018. 699−699. [doi: 10.1145/3180155.3182520]

[69] Wang S, Liu TY, Tan L. Automatically learning semantic features for defect prediction. In: Proc. of the 38th Int'l Conf. on Software Engineering. 2016. 297−308. [doi: 10.1145/2884781.2884804]

[70] Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. Automated parameter optimization of classification techniques for defect prediction models. In: Proc. of the 38th Int'l Conf. on Software Engineering. 2016. 321−332. [doi: 10.1145/2884781.2884857]

[71] Zimmermann T, Nagappan N. Predicting defects using network analysis on dependency graphs. In: Proc. of the 30th Int'l Conf. on Software Engineering. 2008. 531−540. [doi: 10.1145/1368088.1368161]

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

[73] Agrawal A, Menzies T. Is "better data" better than "better data miners"? In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). 2018. 1050−1061. [doi: 10.1145/3180155.3180197]

[74] Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). 2016. 309−320. [doi: 10.1145/2884781.2884839]

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

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

[77] Peters F, Menzies T, Layman L. Lace2: Better privacy-preserving data sharing for cross project defect prediction. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. 2015. 801−811. [doi: 10.1109/ICSE.2015.92]

[78] Ghotra B, McIntosh S, Hassan AE. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. 2015. 789−800. [doi: 10.1109/ICSE.2015.91]

[79] Caglayan B, Turhan B, Bener A, Habayeb M, Miransky A, Cialini E. Merits of organizational metrics in defect prediction: An industrial replication. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. 2015. 89−98. [doi: 10.1109/ICSE.2015.138]

[80] Peters F, Menzies T. Privacy and utility for defect prediction: Experiments with morph. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). 2012. 189−199. [doi: 10.1109/ICSE.2012.6227194]

[81] Hata H, Mizuno O, Kikuno T. Bug prediction based on fine-grained module histories. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). 2012. 200−210. [doi: 10.1109/ICSE.2012.6227193]

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

[83] Kim S, Zimmermann T, Whitehead Jr EJ, Zeller A. Predicting faults from cached history. In: Proc. of the 29th Int'l Conf. on Software Engineering (ICSE 2007). 2007. 489−498. [doi: 10.1109/ICSE.2007.66]

[84] Lu H, Cukic B, Culp M. Software defect prediction using semi-supervised learning with dimension reduction. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2012. 314−317. [doi: 10.1145/2351676.2351734]

[85] Zhang ZW, Jing XY, Wang TJ. Label propagation based semi-supervised learning for software defect prediction. Automated Software Engineering, 2017, 24(1): 47−69. [doi: 10.1007/s10515-016-0194-x]

[86] Li M, Zhang H, Wu R, Zhou ZH. Sample-based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 2012, 19(2): 201−230. [doi: 10.1007/s10515-011-0092-1]

[87] Yan M, Xia X, Fan Y, Lo D, Hassan AE, Zhang X. Effort-aware just-in-time defect identification in practice: A case study at alibaba. In: Proc. of the Association for Computing Machinery. 2020. 1308−1319. [doi: 10.1145/3368089.3417048]

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

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

[90] Chapman M, Callis P, Jackson W. Metrics data program. NASA IV and V Facility, 2004. http://mdp.ivv.nasa.gov

[91] Cukic B. Guest editor's introduction: The promise of public software engineering data repositories. IEEE Software, 2005, 22(6): 20−22. [doi: 10.1109/MS.2005.153]

[92] Shirabad JS, Menzies TJ. The promise repository of software engineering databases. Technical Report, School of Information Technology and Engineering, University of Ottawa, 2005. 24

[93] Petrić J, Bowes D, Hall T, Christianson B, Baddoo N. The jinx on the NASA software defect data sets. In: Proc. of the 20th Int'l Conf. on Evaluation and Assessment in Software Engineering. 2016. Article 13. [doi: 10.1145/2915970.2916007]

[94] Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. In: Proc. of the 3rd Int'l Workshop on Predictor Models in Software Engineering (PROMISE 2007: ICSE Workshops 2007). 2007. 9. [doi: 10.1109/PROMISE.2007.10]

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

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

[97] Turhan B, Menzies T, Bener AB, Di Stefano J. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 2009, 14(5): 540−578.

[98] Ambros MD, 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). 2010. 31−41. [doi: 10.1109/MSR.2010.5463279]

[99] 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. 2010. 1−10. [doi: 10.1145/1868328.1868342]

[100] Jureczko M, Spinellis D. Using object-oriented design metrics to predict software defects. In: Proc. of the Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, 2010. 69−81.

[101] 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. 2011. 15−25. [doi: 10.1145/2025113.2025120]

[102] Bissyandé TF, Thung F, Wang S, Lo D, Jiang L, Reveillere L. Empirical evaluation of bug linking. In: Proc. of the 17th European Conf. on Software Maintenance and Reengineering. 2013. 89−98. [doi: 10.1109/CSMR.2013.19]

[103] 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). 2013. 118−127. [doi: 10.1109/ISSRE.2013.6698911]

[104] Hall T, Zhang M, Bowes D, Sun Y. Some code smells have a significant but small effect on faults. ACM Trans. on Software Engineering and Methodology (TOSEM), 2014, 23(4): 1−39. [doi: 10.1145/2629648]

[105] Tóth Z, Gyimesi P, Ferenc R. A public bug database of github projects and its application in bug prediction. In: Proc. of the Int'l Conf. on Computational Science and Its Applications. 2016. 625−638. [doi: 10.1007/978-3-319-42089-9_44]

[106] Shippey T, Hall T, Counsell S, Bowes D. So you need more method level datasets for your software defect prediction? Voilà! In: Proc. of the 10th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. 2016. 1−6. [doi: 10.1145/2961111.2962620]

[107] Ferenc R, Tóth Z, Ladányi G, Siket I, Gyimóthy T. A public unified bug dataset for Java. In: Proc. of the 14th Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. 2018. 12−21. [doi: 10.1145/3273934.3273936]

[108] Gray D, Bowes D, Davey N, Sun Y, Christianson B. The misuse of the NASA metrics data program data sets for automated software defect prediction. In: Proc. of the 15th Annual Conf. on Evaluation & Assessment in Software Engineering (EASE 2011). 2011. 96−103. [doi: 10.1049/ic.2011.0012]

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

[110] Liu S, Guo Z, Li Y, Wang C, Chen L, Sun Z, Zhou Y, Xu B. Inconsistent defect labels: Essence, causes, and influence. IEEE Trans. on Software Engineering, 2022. [doi: 10.1109/TSE.2022.3156787]

[111] Mende T. Replication of defect prediction studies: Problems, pitfalls and recommendations. In: Proc. of the 6th Int'l Conf. on Predictive Models in Software Engineering. 2010. 1−10. [doi: 10.1145/1868328.1868336]

[112] Yang Y, Yang J, Qian H. Defect prediction by using cluster ensembles. In: Proc. of the 10th Int'l Conf. on Advanced Computational Intelligence (ICACI). 2018. 631−636. [doi: 10.1109/ICACI.2018.8377533]

[113] Lewis DD. Naive (Bayes) at forty: The independence assumption in information retrieval. In: Proc. of the European Conf. on Machine Learning. 1998. 4−15. [doi: 10.1007/BFb0026666]

[114] Breiman L. Random forests. Machine Learning, 2001, 45(1): 5−32.

[115] Quinlan JR. Simplifying decision trees. Int'l Journal of Man-Machine Studies, 1987, 27(3): 221−234.

[116] Suykens JA, Vandewalle J. Least squares support vector machine classifiers. Neural Processing Letters, 1999, 9(3): 293−300. [doi: 10.1023/A:1018628609742]

[117] Hosmer Jr DW, Lemeshow S, Sturdivant RX. Applied Logistic Regression. Vol. 398, John Wiley & Sons, 2013.

[118] Cover T, Hart P. Nearest neighbor pattern classification. IEEE Trans. on Information Theory, 1967, 13(1): 21−27. [doi: 10.1109/TIT.1967.1053964]

[119] Hagan MT, Demuth HB, Beale M. Neural Network Design. PWS Publishing Co., 1997.

[120] Freund Y, Mason L. The alternating decision tree learning algorithm. In: Proc. of the ICML. 1999. 124−133.

[121] Friedman N, Geiger D, Goldszmidt M. Bayesian network classifiers. Machine Learning, 1997, 29(2): 131−163.

[122] Mathuria M. Decision tree analysis on j48 algorithm for data mining. Int'l Journal of Advanced Research in Computer Science and Soft-ware Engineering, 2013, 3(6):

[123] 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. 2009. 199−204. [doi: 10.1109/ITNG.2009.12]

[124] Zhong S, Khoshgoftaar TM, Seliya N. Unsupervised learning for expert-based software quality estimation. In: Proc. of the HASE. 2004. 149−155. [doi: 10.1109/HASE.2004.1281739]

[125] Landwehr N, Hall M, Frank E. Logistic model trees. Machine Learning, 2005, 59(1−2): 161−205.

[126] Quinlan JR. Improved use of continuous attributes in c4. 5. Journal of Artificial Intelligence Research, 1996, 4: 77−90.

[127] Cruz AEC, Ochimizu K. Towards logistic regression models for predicting fault-prone code across software projects. In: Proc. of the 3rd Int'l Symp. on Empirical Software Engineering and Measurement. 2009. 460−463. [doi: 10.1109/ESEM.2009. 5316002]

[128] Menzies T, Butcher A, Marcus A, Zimmermann T, Cok D. Local vs. global models for effort estimation and defect prediction. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2011). 2011. 343−351. [doi: 10.1109/ASE. 2011.6100072]

[129] Watanabe S, Kaiya H, Kaijiri K. Adapting a fault prediction model to allow inter languagereuse. In: Proc. of the 4th Int'l Workshop on Predictor Models in Software Engineering. 2008. 19−24. [doi: 10.1145/1370788.1370794]

[130] Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A. Defect prediction from static code features: Current results, limitations, new approaches. Automated Software Engineering, 2010, 17(4): 375−407. [doi: 10.1007/s10515-010-0069-5]

[131] Nam J, Pan SJ, Kim S. Transfer defect learning. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). 2013. 382−391. [doi: 10.1109/ICSE.2013.6606584]

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

[133] Herbold S, Trautsch A, Grabowski J. Correction of "a comparative study to benchmark cross-project defect prediction approaches". IEEE Trans. on Software Engineering, 2018, 45(6): 632−636. [doi: 10.1109/TSE.2018.2790413]

[134] Amasaki S, Kawata K, Yokogawa T. Improving cross-project defect prediction methods with data simplification. In: Proc. of the 41st Euromicro Conf. on Software Engineering and Advanced Applications. 2015. 96−103. [doi: 10.1109/SEAA.2015.25]

[135] Akiyama F. An example of software system debugging. In: Proc. of the IFIP Congress (1). 1971. 353−359.

[136] Koru AG, Zhang DS, El Emam K, Liu HF. An investigation into the functional form of the size-defect relationship for software modules. IEEE Trans. on Software Engineering, 2008, 35(2): 293−304. [doi: 10.1109/TSE.2008.90]

[137] Syer MD, Nagappan M, Adams B, Hassan AE. Replicating and re-evaluating the theory of relative defect-proneness. IEEE Trans. on Software Engineering, 2014, 41(2): 176−197. [doi: 10.1109/TSE.2014.2361131]

[138] Koru G, Liu HF, Zhang DS, El Emam K. Testing the theory of relative defect proneness for closed-source software. Empirical Software Engineering, 2010, 15(6): 577−598. [doi: 10.1007/s10664-010-9132-x]

[139] Koru AG, El Emam K, Zhang DS, Liu HF, Mathew D. Theory of relative defect proneness. Empirical Software Engineering, 2008, 13(5): 473−498. [doi: 10.1007/s10664-008-9080-x]

[140] Zhou YM, Xu BW, Leung H. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. Journal of Systems and Software, 2010, 83(4): 660−674. [doi: 10.1016/j.jss.2009.11.704]

[141] Catal C, Sevim U, Diri B. Metrics-driven software quality prediction without prior fault data. In: Proc. of the Int'l Conf. in Electronic Engineering and Computing Technology; World Congress on Engineering; WCE 2009. Springer, 2010. 189−199. [doi: 10.1007/978-90-481-8776-8_17]

[142] Jiang Y, Li M, Zhou ZH. Software defect detection with rocus. Journal of Computer Science and Technology, 2011, 26(2): 328−342. [doi: 10.1007/s11390-011-9439-0]

[143] Culp M, Michailidis G. An iterative algorithm for extending learners to a semi-supervised setting. Journal of Computational and Graphical Statistics, 2008, 17(3): 545−571. [doi: 10.1198/106186008X344748]

[144] Lu H, Cukic B, Culp M. An iterative semi-supervised approach to software fault prediction. In: Proc. of the 7th Int'l Conf. on Predictive Models in Software Engineering. 2011. 1−10. [doi: 10.1145/2020390.2020405]

[145] Li M, Zhou ZH. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. IEEE Trans. on Systems, Man, and Cybernetics—Part A: Systems and Humans, 2007, 37(6): 1088−1098. [doi: 10.1109/TSMCA.2007.904745]

[146] Chapelle O, Zien A. Semi-supervised classification by low density separation. In: Proc. of the Int'l Workshop on Artificial Intelligence and Statistics. 2005. 57−64.

[147] Joachims T. Making large-scale SVM learning practical. In: Proc. of the Advances in Kernel Methods-support Vector Learning. 1999. http://svmlight.joachims.org/

[148] Seliya N, Khoshgoftaar TM, Zhong S. Semi-supervised learning for software quality estimation. In: Proc. of the 16th IEEE Int'l Conf. on Tools with Artificial Intelligence. 2004. 183−190. [doi: 10.1109/ICTAI.2004.108]

[149] Zhu X, Ghahramani Z, Lafferty JD. Semi-supervised learning using gaussian fields and harmonic functions. In: Proc. of the 20th Int'l Conf. on Machine Learning (ICML 2003). 2003. 912−919.

[150] He Q, Shen B, Chen Y. Software defect prediction using semi-supervised learning with change burst information. In: Proc. of the 40th IEEE Annual Computer Software and Applications Conf. (COMPSAC). 2016. 113−122. [doi: 10.1109/COMPSAC.2016.193]

[151] Foyzur R, Premkumar D. How, and why, process metrics are better. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). 2013. 432−441. [doi: 10.1109/ICSE.2013.6606589]

[152] Raimund M, Witold P, Giancarlo S. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proc. of the Association for Computing Machinery. 2008. 181−190. [doi: 10.1145/1368088.1368114]

[153] Ohlsson N, Alberg H. Predicting fault-prone software modules in telephone switches. IEEE Trans. on Software Engineering, 1996, 22(12): 886−894. [doi: 10.1109/32.553637]

[154] Peter AW, Caitlin AO, Stephen GM. A baseline model for software effort estimation. ACM Trans. on Software Engineering and Methodology, 2015, 24(3): Article 20. [doi: 10.1145/2738037]

附中文参考文献:

[21] 王青, 伍书剑, 李明树. 软件缺陷预测技术. 软件学报, 2008, 19(7): 1565−1580. http://www.jos.org.cn/1000-9825/19/1565.htm [doi: 10.3724/SP.J.1001.2008.01565]

[27] 陈翔, 顾庆, 刘望舒, 刘树龙, 倪超. 静态软件缺陷预测方法研究. 软件学报, 2016, 27(1): 1−25. http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[31] 蔡亮, 范元瑞, 鄢萌, 夏鑫. 即时软件缺陷预测研究进展. 软件学报, 2019, 30(5): 1288−1307. http://www.jos.org.cn/1000-9825/5713.htm [doi: 10.13328/j.cnki.jos.005713]

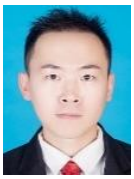
[32] 宫丽娜, 姜淑娟, 姜丽. 软件缺陷预测技术研究进展. 软件学报, 2019, 30(10): 3090−3114. http://www.jos.org.cn/1000-9825/5790.htm [doi: 10.13328/j.cnki.jos.005790]

刘旭同(1997－), 女, 博士生, 主要研究领域为软件分析与测试.

张鹏(1996－), 男, 博士生, 主要研究领域为软件分析与测试.

郭肇强(1994－), 男, 博士生, 主要研究领域为软件分析与测试.

卢红敏(1976－), 女, 博士, 高级工程师, CCF 专业会员, 主要研究领域为软件分析与测试.



刘释然(1990－), 男, 博士生, 主要研究领域为软件分析与测试.

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分析与测试.