软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(12):3694−3713 [doi: 10.13328/j.cnki.jos.005604]

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

基于文件粒度的多目标软件缺陷预测方法实证研究*

陈 翔[1,2,3], 赵英全[1], 顾 庆[2], 倪 超[2], 王 赞[4]

[1](南通大学 信息科学技术学院,江苏 南通 226019)

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

[3](广西可信软件重点实验室(桂林电子科技大学),广西 桂林 541004)

[4](天津大学 软件学院,天津 300072)

通讯作者: 陈翔, E-mail: xchencs@ntu.edu.cn

摘 要: 软件缺陷预测技术通过挖掘和分析软件库训练出软件缺陷预测模型,随后利用该模型来预测出被测软件项目内的缺陷程序模块,因此可以有效地优化测试资源的分配.在基于代价感知的评测指标下,有监督学习方法与无监督学习方法之间的预测性能比较是最近的一个热门研究话题.其中在基于文件粒度的缺陷预测问题中,Yan 等人最近对 Yang 等人考虑的无监督学习方法和有监督学习方法展开了大规模实证研究,结果表明存在一些无监督学习方法,其性能要优于有监督方法.基于来自开源社区的 10 个项目展开了实证研究.结果表明:在同项目缺陷预测场景中,若基于 ACC 评测指标,MULTI 方法与最好的无监督方法和有监督方法相比,其预测性能平均有 105.81%和 123.84%的提高;若基于 P_{OPT} 评测指标,MULTI 方法与最好的无监督方法和有监督方法相比,其预测性能平均有 35.61%和 38.70%的提高.在跨项目缺陷预测场景中,若基于 ACC 评测指标,MULTI 方法与最好的无监督方法和有监督方法相比,其预测性能平均有 22.42%和 34.95%的提高.若基于 P_{OPT} 评测指标,MULTI 方法与最好的无监督方法和有监督方法相比,其预测性能平均有 11.45%和 17.92%的提高.同时,基于 Huang 等人提出的 PMI 和 IFA 评测指标,MULTI 方法的表现与代价感知的指标相比存在一定的折衷问题,但仍好于在 ACC 和 P_{OPT} 评测指标下表现最好的两种无监督学习方法.除此之外,将 MULTI 方法与最新提出的 OneWay 和 CBS 方法进行了比较,结果表明,MULTI 方法在性能上仍然可以显著优于这两种方法.同时,基于 *F*1 评测指标的结果也验证了 MULTI 方法在预测性能上的显著优越性.最后,通过分析模型构建的时间开销,表明 MULTI 方法的模型构建开销对开发人员来说处于可接受的范围之内.

关键词: 软件质量保障;软件缺陷预测;有监督学习;无监督学习;多目标优化

中图法分类号: TP311

中文引用格式: 陈翔,赵英全,顾庆,倪超,王赞.基于文件粒度的多目标软件缺陷预测方法实证研究.软件学报,2019,30(12):3694−3713. http://www.jos.org.cn/1000-9825/5604.htm

英文引用格式: Chen X, Zhao YQ, Gu Q, Ni C, Wang Z. Empirical studies on multi-objective file-level software defect prediction method. Ruan Jian Xue Bao/Journal of Software, 2019,30(12):3694−3713 (in Chinese). http://www.jos.org.cn/1000-9825/5604.htm

* 基金项目: 国家自然科学基金(61702041, 61602267, 61202006); 南京大学计算机软件新技术国家重点实验室开放课题(KFKT2019B14); 广西可信软件重点实验室研究课题(kx201610); 南通市应用研究计划(JC2018134); 江苏省政府留学奖学金

Foundation item: National Natural Science Foundation of China (61702041, 61602267, 61202006); Open Program of the State Key Laboratory for Novel Software Technology (Nanjing University) (KFKT2019B14); Guangxi Key Laboratory of Trusted Software (kx201610); Nantong Application Research Plan (JC2018134); Jiangsu Government Scholarship for Overseas Studies

收稿时间: 2017-12-18; 修改时间: 2018-03-26, 2018-05-06; 采用时间: 2018-05-22

Empirical Studies on Multi-objective File-level Software Defect Prediction Method

CHEN Xiang[1,2,3], ZHAO Ying-Quan[1], GU Qing[2], NI Chao[2], WANG Zan[4]

[1](School of Information Science and Technology, Nantong University, Nantong 226019, China)

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

[3](Guangxi Key Laboratory of Trusted Software (Guilin University of Electronic Technology), Guilin 541004, China)

[4](School of Computer Software, Tianjin University, Tianjin 300072, China)

Abstract: By mining software repositories, software defect prediction can construct models to predict potential defective modules of projects under testing in advance and then optimize the allocation of test resources. When considering effort-aware performance measures, the performance comparison between supervised methods and unsupervised methods has been a recent hot topic. In the recent study for file-level defect prediction problem, Yan *et al.* conducted empirical studies by using unsupervised and supervised methods considered by Yang *et al*. and obtained the conclusion that some unsupervised methods can outperform the supervised methods. The empirical studies based on 10 projects from the open source community were conducted. Final results show that under the within-project defect prediction scenario, MULTI method can improve 105.81% and 123.84% respectively on average when compared to the best unsupervised method and the best supervised method based on ACC performance measure. While MULTI method can improve 35.61% and 38.70% respectively on average when compared to the best unsupervised method and the best supervised method based on P_{OPT} performance measure. Under the cross- project defect prediction scenario, MULTI method can improve 22.42% and 34.95% respectively on average when compared to the best unsupervised method and the best supervised method based on ACC performance measure. While MULTI method can improve 11.45% and 17.92% respectively on average when compared to the best unsupervised method and the best supervised method based on P_{OPT} performance measure. Based on PMI and IFA performance measures proposed by Huang *et al*., it is found that MULTI method has the issue of trade-off, but it is still better than the best two unsupervised methods when considering ACC and P_{OPT} performance measures. Besides, MULTI method is compared with the recently proposed OneWay and CBS methods. The results show that MULTI performs significantly better than these two methods. Based on *F*1 performance measure, MULTI method also shows the superiority. Finally, the analysis on the time cost of the model construction shows that the overhead of MULTI method is acceptable.

Key words: software quality assurance; software defect prediction; supervised learning; unsupervised learning; multi-objective optimization

软件产品在软件开发生命周期内(例如需求分析、软件设计和软件实现)都存在引入缺陷的可能性,而含有缺陷的软件产品在部署上线后可能会产生预期之外的行为,在严重的时候甚至会给企业带来巨额经济损失或威胁到产品用户的生命安全.已有的项目管理经验表明:在软件开发生命周期中,检测出项目内缺陷的时间越晚,修复该缺陷的代价也越高并且修复代价呈指数级增长,因此,测试人员希望能够在软件部署上线前尽可能早地预先识别出所有的缺陷程序模块,并对这些缺陷模块投入更多的软件测试资源(即针对这些缺陷模块设计更为充分的测试用例或展开更多的代码审查工作等)[1].软件缺陷预测(software defect prediction)[1−3]是可以预先识别出缺陷程序模块的一种静态分析方法,其通过挖掘和分析软件库(例如项目托管的缺陷跟踪系统和版本控制系统),可以训练出缺陷预测模型,并以此来提前识别出被测项目内的缺陷程序模块.因此该方法可以有效优化测试资源的分配,从而有助于提高最终部署上线的软件产品的质量.

在实际软件测试时,可以使用的测试资源是有限的,因此很难保障对被测项目内的所有程序模块进行充分的软件测试,因此软件质量保障团队希望构建出一个模型,其能够将被测项目内的所有程序模块,按照模块含有缺陷的概率,从高到低进行排序.随后开发人员可以根据该排序列表依次完成对程序模块的代码审查或者软件测试.这样,在测试资源存在约束的时候,软件质量保障团队可以完成对更多缺陷的检测和修复.在上述测试场景中,研究人员常用代价感知(effort-aware)的评测指标对模型性能进行评估.基于代价感知的评测指标,有监督学习方法与无监督学习方法之间的性能比较是最近一个具有争议性的问题.在基于代码修改的软件缺陷预测(change-level software defect prediction,简称 CL-SDP)问题中,Yang 等人惊奇地发现:存在一些无监督学习方法,其预测性能要显著优于已有的经典有监督学习方法[4].随后,Fu 等人[5]以及 Huang 等人[6]对 Yang 等人的研究工作[4]进行了重现,并得出了一些新的发现.我们关注了基于多目标优化的有监督学习方法 MULTI,并在跨项目缺陷预测(cross-project defect prediction)场景、同项目缺陷预测(within-project defect prediction)场景和基于时序的

缺陷预测场景下,深入分析了 MULTI 方法与 Yang 等人考虑的无监督学习方法和有监督学习方法之间的性能差异.结果表明,MULTI 方法在这 3 个场景下均能取得更好的预测性能,从而说明有监督学习方法仍然值得研究人员进行关注[7].

与 CL-SDP 问题不同,基于文件粒度(在面向对象程序实现的项目中,文件粒度也可以被称为类粒度)的软件缺陷预测(file-level software defect prediction,简称 FL-SDP)问题将模块粒度设置为文件,因此模块内含有的代码行数更多,考虑的度量元也并不相同.最近,Yan 等人[8]针对 FL-SDP 问题,对 Yang 等人考虑的无监督学习方法和有监督学习方法进行了深入比较,并得到了相似的结论.但据我们所知:针对 FL-SDP 问题,在基于代价感知的评测指标下,并没有研究人员将基于多目标优化的 MULTI 方法[7]与 Yan 等人考虑的有监督学习方法和无监督学习方法的性能进行深入分析和比较.因此,我们针对上述问题设计并开展了大规模实证研究.基于代价感知的评测指标(例如 ACC 和 P_{OPT}),我们发现 MULTI 方法在同项目缺陷预测场景和跨项目缺陷预测场景下均要显著优于 Yan 等人考虑的无监督学习方法、有监督学习方法以及最新提出的 OneWay 方法[5]和 CBS 方法[6].基于 Huang 等人提出的 PMI 和 IFA 评测指标[6],我们发现 MULTI 方法与代价感知的指标相比存在一定的折衷,但仍好于在 ACC 和 P_{OPT} 评测指标下表现最好的两种无监督学习方法.除此之外,基于 $F1$ 评测指标的结果也进一步验证了 MULTI 方法的优越性.同时,通过分析模型构建的时间开销,我们认为 MULTI 方法在模型训练上花费的计算开销处于开发人员可以接受的范围之内.

本文的主要贡献可以简要总结如下:

(1) 我们基于代价感知的评测指标,深入分析了基于多目标优化的 MULTI 方法在基于文件粒度的缺陷预测问题上的预测性能.

(2) 本文在实证研究中考虑了 10 个开源项目,在模型性能评估上考虑了同项目缺陷预测场景和跨项目缺陷预测场景.通过将 MULTI 方法与 Yan 等人考虑的无监督学习方法和有监督学习方法[8]、Wu 和 Menzies 提出的 OneWay 方法[5]以及 Huang 等人提出的 CBS 方法[6]进行比较,发现 MULTI 方法在预测性能上均具有显著性优势,从而表明 MULTI 方法在 FL-SDP 问题上同样具有优势并值得关注.

本文第 1 节介绍软件缺陷研究背景,以及无监督学习方法和有监督学习方法在缺陷预测问题上的已有研究成果.第 2 节深入分析本文重点考察的基于多目标优化的 MULTI 方法以及 Yan 等人考虑的有监督学习方法和无监督学习方法.第 3 节给出本文的实验设计,包括本文考虑的评测对象、评测指标、模型性能评估场景、实验流程以及结果分析时使用的显著性检验方法等.第 4 节分别针对实证研究结果和有效性影响因素进行了深入分析.第 5 节总结全文,同时针对未来值得关注的多个研究点进行了展望.

1 软件缺陷研究背景和相关工作

1.1 研究背景

软件缺陷预测[1−3]通过分析和挖掘软件库(例如项目托管的缺陷跟踪系统和版本控制系统),训练出缺陷预测模型,并使用训练出的模型来提前识别出被测项目内的缺陷程序模块,随后开发人员可以针对这些识别出的缺陷程序模块展开更为充分的软件测试或投入更多的代码审查工作量,借助这种方式可以优化测试资源的分配并且有助于提高部署上线的软件产品质量.具体来说,首先可以通过分析软件模块的代码复杂度或软件开发过程中的特点,设计出与软件缺陷存在一定相关度的度量元(metrics)[9,10].其中,软件模块的代码复杂度可以通过分析代码行数(line of code)、Halstead 科学度量、McCabe 环路复杂度或 CK 度量元等进行度量,而软件开发过程的特点则可以通过分析模块间的控制/数据依赖性、代码修改的特征、开发人员的编程经验和领域熟悉程度以及项目团队的组织构架等[1]进行度量.随后,通过分析软件库,抽取出所有的程序模块(其模块粒度可以根据应用场景设置为代码修改、文件或类等),借助手工设计的度量元可以对这些抽取出的模块进行度量,随后通过分析代码修改、修改日志信息和相关缺陷报告[11]可以对这些模块进行标记(即将模块类型标记为有缺陷或者无缺陷).基于对模块的度量和类型标记可以构造出用于模型训练的数据集.最后,借助某一分类方法(例如神经网络、支持向量机、朴素贝叶斯或随机森林等[12])来构建出缺陷预测模型.

1.2 无监督学习方法与有监督学习方法间的性能比较

近年来,研究人员提出了大量的缺陷预测建模方法[1],无监督方法和有监督方法是软件缺陷预测研究中常见的两类方法.其中,大部分方法属于有监督学习方法,但这类方法需要基于已标记的训练数据集来构建模型,因此需要花费大量的人力和物力来完成标记数据集的搜集.而无监督学习方法则不需要搜集已标记的训练数据集,可以直接对测试集进行预测,因此可以缓解上述不足,并更容易与企业实际开发流程相结合.

在基于代码修改的软件缺陷预测问题(在一些文献中,该问题也被称为 just-in-time 软件缺陷预测[13−16])中,其具有模块粒度细、可以迅速找到缺陷相关开发人员等优点.Kamei 等人[13]从代码修改的规模、目的、分散度、修改历史以及开发人员经验等角度对抽取出的代码修改进行度量,并提出了基于有监督的 EALR(effort-aware linear regression)方法.Yang 等人[4]基于代价感知的评测指标,在性能评估时意外地发现:存在一些简单的无监督方法,其比已有的有监督方法(包括 Kamei 等人提出的 EALR 方法[13])可以取得更好的性能.Fu 等人[5]对 Yang 等人的实证研究进行了重现,他们发现:并不是所有的无监督方法的预测性能都可以显著优于有监督方法.因此,他们提出了 OneWay 方法,其可以自动选出更好的无监督学习方法.随后,Huang 等人[6]发现:在给定相同的测试成本时,有监督学习方法需要审查更多的代码修改.Liu 等人[17]则提出一种新的无监督学习方法 CCUM(code churn based unsupervised method),该方法将代码修改涉及到的删除代码量和新增代码量相加,构成 CHURN 度量元,并根据该度量元完成对代码修改的排序.结果表明,CCUM 方法[17]相比于 Yang 等人考虑的方法,可以取得更好的预测性能.

与 CL-SDP 问题不同,基于文件粒度的软件缺陷预测问题将模块粒度设置为文件,因此模块内含有的代码量更大.Yan 等人在最近的一个研究工作中[8]针对 FL-SDP 问题,对 Yang 等人考虑的无监督学习方法和有监督学习方法的性能进行了比较和分析,并得到了相似的结论.

在我们之前针对 CL-SDP 的研究工作[7]中发现:基于多目标优化的方法 MULTI 在 3 种不同的场景(即同项目缺陷预测场景、跨项目缺陷预测场景和基于时序的缺陷预测场景)下,都要显著优于 Yang 等人考虑的有监督学习方法和无监督学习方法.基于上述研究工作,我们希望继续分析基于多目标优化的 MULTI 方法在 FL-SDP 问题中,其性能是否能够显著优于 Yan 等人考虑的有监督学习方法和无监督学习方法.

2 基于文件粒度的软件缺陷预测方法

该节首先简要分析本文考虑的基于多目标优化的 MULTI 方法,随后依次介绍需要与 MULTI 方法比较的基准方法[8],这些基准方法可分为两类:有监督学习方法和无监督学习方法.

2.1 基于多目标优化的MULTI方法

MULTI 方法针对 FL-SDP 问题设置了两个不同的优化目标:其中第 1 个优化目标被设置为模型需要尽可能多地识别出缺陷模块数,第 2 个优化目标被设置为模型需要尽可能地减少代码审查工作量.不难看出,模型设置的这两个优化目标之间存在一定程度的折衷关系.即如果训练出的模型需要识别出更多的缺陷模块数,则需要投入更多的代码审查工作量;与此相反,若需要减少代码审查工作量,则训练出的模型有可能会遗漏掉一些缺陷模块[7].

本文借助 Logistic 回归方法来训练预测模型,假设程序模块用 n 个特征(即软件缺陷预测的度量元)进行度量,则预测模型的系数可以表示为一维向量 $w=\{w_1,w_2,\ldots,w_n\}$.给定预测模型的系数向量 w 和需要预测的程序模块 m_i,其中,程序模块 m_i 的第 j 个特征取值用 $v_{i,j}$ 表示,则可以使用公式(1)计算出该程序模块包含缺陷的概率为

$$y(m_i,w)=\frac{1}{1+\mathrm{e}^{-(w_0+w_1v_{i,1}+\ldots+w_nv_{i,n})}} \tag{1}$$

本文将 FL-SDP 问题建模为经典的二元分类问题,同时将模块分类阈值设置为 0.5,即若预测出的含有缺陷的概率取值大于 0.5,则将该程序模块预测为有缺陷;否则,将该程序模块预测为无缺陷.其计算公式如下所示.

$$Y(m_i,w)=\begin{cases}1, & \text{if } y(m_i,w)>0.5\\ 0, & \text{if } y(m_i,w)\leqslant 0.5\end{cases} \tag{2}$$

随后依次给出两个优化目标的计算公式.假设需要进行预测的模块用集合 M 来表示,候选解用 w 来表示,则针对模型设置的第 1 个优化目标,可将其计算公式定义如下.

$$benefit(w)=\sum_{m_i\in M} Y(m_i,w)\times buggy(m_i) \tag{3}$$

其中,$buggy(m_i)$用于表示程序模块 m_i 是否包含缺陷:如果该模块内含有缺陷,其取值为 1;否则,其取值为 0.

针对模型设置的第 2 个优化目标,可将其计算公式定义如下.

$$cost(w)=\sum_{m_i\in M} Y(m_i,w)\times LOC(m_i) \tag{4}$$

其中,$LOC(m_i)$表示文件模块内含有的代码行数,在该研究工作中,我们假设模块含有的代码行数越多,则需要投入的代码审查量越大(即需要投入更多的软件测试资源).

我们通过一个虚拟实例对这两个优化目标的具体计算过程进行解释.表 1 展示了训练集中含有的文件和各个文件在某个模型下对应的预测类型 $Y(\cdot)$、实际类型 $buggy(\cdot)$以及代码审查代价 $LOC(\cdot)$.则第 1 个优化目标的值可借助公式(0×1+1×0+...+1×1)计算出,第 2 个优化目标的值可借助公式(1×50+0×80+...+1×20)计算出.

Table 1 A synthesized instance

表 1 虚拟实例

文件	实际类型	预测类型	审查代价
f_1	0	1	50
f_2	1	0	80
...	...	...	...
f_n	1	1	20

基于上述分析,本文将 FL-SDP 问题建模为经典的双目标优化问题,为了方便后续描述,我们首先对与多目标优化相关的概念进行定义.

定义 1(Pareto 支配关系). 假设用于训练 FL-SDP 模型的两个候选解分别用 w_a 和 w_b 进行表示,则 w_aPareto 支配 w_b,当且仅当:

($benefit(w_a)>benefit(w_b)$ and $cost(w_a)\leqslant cost(w_b)$) or ($benefit(w_a)\geqslant benefit(w_b)$ and $cost(w_a)<cost(w_b)$).

定义 2(Pareto 最优解). 一个候选解 w 被认为是 Pareto 最优解,当且仅当不存在其他候选解 w^*,该解能够 Pareto 支配 w.

定义 3(Pareto 最优解集). 所有 Pareto 最优解可以构成 Pareto 最优解集.

定义 4(Pareto 前沿). 所有 Pareto 最优解对应的不同优化目标值在空间内形成的曲面被称为 Pareto 前沿.

针对多目标优化问题,研究人员已经提出了大量不同类型的算法,这些算法主要基于演化算法来构造出 Pareto 前沿.目前,多目标优化算法在软件工程问题中已经取得了较多的成功应用[7,18].本文考虑的 MULTI 方法[7]主要基于一种经典的多目标优化算法 NSGA-II[19],NSGA-II 算法本身具有一定的机制(即基于分层的快速非支配排序算法、染色体的拥挤度计算方法和精英保留机制)可以有效避免种群的过早收敛问题.具体来说:NSGA-II 算法在每次进行种群演化时,首先使用演化算子产生新的染色体,随后将这些新产生的染色体与上一轮种群内的染色体进行合并,并使用基于分层的快速非支配排序算法来完成染色体的排序,同时,对于每个非支配层中的染色体依次计算出各个染色体的拥挤度,最后同时考虑非支配关系和染色体的拥挤度,选出指定数量的高质量染色体并形成新的种群..MULTI 方法将种群中的染色体编码为预测模型的系数向量,在训练集上可以借助染色体对应的系数向量来训练出预测模型,并随后分别使用 benefit 函数(见公式(3))和 cost 函数(见公式(4))依次计算出该预测模型设置的两个优化目标取值.其伪代码如算法 1 所示.

算法 1. MULTI 方法.

输入:种群规模 N,最大迭代次数 T;

输出:Pareto 最优解集.

```
1   i←0
2   P_i←initPop(N)
3   while i<T do
4     C_i←makeNewPop(P_i)
5     B_i←P_i∪C_i
6     F←fastNondominatedSort(B_i)
7     P_{i+1}←∅
8     j←1
9     while |P_{i+1}|+|F_j|≤N do
10      crowingDistanceAssign(F_j)
11      P_{i+1}←P_{i+1}∪F_j
12      j←j+1
13    end while
14    sort(F_j)   //according to crowding distance
15    P_{i+1}←P_{i+1}∪F_j[1:(N−|P_{i+1}|)]
16    i←i+1
17  end while
18  return P_i 内的所有 Pareto 最优解
```

MULTI 方法的具体执行过程可以总结如下:首先对种群进行初始化(步骤 2),染色体对应的向量内元素被随机赋值.随后,使用交叉算子和变异算子可以生成新的染色体(步骤 4).具体来说,变异算子会基于指定的变异概率随机挑选出一个染色体,对其进行变异并产生一个新的染色体.交叉算子会基于指定的交叉概率随机选出两个染色体,对其进行交叉并产生两个新的染色体.随后,MULTI 方法基于选择算子并通过分析 Pareto 支配关系,从当前种群内的染色体中选出一定数量的高质量染色体到下一轮种群.具体来说,首先将当前种群内的染色体和基于两种演化算子生成的新染色体合并到集合 B_t 中(步骤 5).然后使用 fastNondominateSort 函数计算出 B_t 中的每个染色体的 NDR(non-dominated rank)取值(步骤 6).该计算过程总结如下,首先从 B_t 中选出所有不能被 Pareto 支配的染色体,将它们的 NDR 值设置为 1,同时将它们从集合 B_t 移到集合 F_1.随后,继续从 B_t 中选出所有不能被 Pareto 支配的染色体,将它们的 NDR 值设置为 2,同时将它们从集合 B_t 移到集合 F_2.重复执行上述过程,直到集合 B_t 不再含有任何染色体.不难看出,基于染色体的 NDR 取值,MULTI 方法会优先选择 NDR 取值更小的染色体到下一轮种群(步骤 7~步骤 15).除此之外,MULTI 方法还通过拥挤距离[19]来避免选出高相似度染色体,从而确保种群内的染色体具有一定的多样性.当达到指定的最大迭代次数 *MaxT* 后,MULTI 方法将返回当前种群中的所有 Pareto 最优解.这里值的注意的是,MULTI 方法仅基于训练集来计算出 Pareto 最优解集,随后基于该集合内的每个最优解训练出缺陷预测模型,并在测试集上计算出该对应模型的预测性能.

2.2 基准方法

2.2.1 有监督学习方法

Ghotra 等人[12]最近深入分析了大量有监督学习方法的性能在软件缺陷预测上是否存在显著性差异.本文考虑的 31 种无监督学习方法主要来自于 Ghotra 等人[12]在实证研究中考虑的方法,具体见表 2.

Table 2 Overview of supervised methods

表 2 本文考虑的有监督学习方法

方法类型	有监督学习方法极其简称
Function	Linear regression (EALR), simple logistic (SL), radial basis functions network (RBFNet), sequential minimal optimization (SMO)
Lazy	*K*-nearest neighbor (IBK)
Rule	Propositional rule (JRIP), ripple down rules (Ridor)
Bayes	Naive Bayes (NB)
Tree	J48(J48), logistic model tree (LMT), random forest (RF)
Ensemble (bagging)	BG+LMT,BG+NB,BG+SL,BG+SMO,BG+J48
Ensemble (adaboost)	AB+LMT,AB+NB,AB+SL,AB+SMO,AB+J48
Ensemble (rotation forest)	RF+LMT,RF+NB,RF+SL,RF+SMO,RF+J48
Ensembel (random subspace)	RS+LMT,RS+NB,RS+SL,RS+SMO,RS+J48

这些方法也被广泛用于其他文献中[4,6−8].这 31 种无监督学习方法可以细分为 6 类.具体来说,function 类型包括 4 种方法,Lazy 类型包括 1 种方法,Rule 类型包括 2 种方法,Bayes 类型包括 1 种方法,Tree 类型包括 3 种方法.集成学习类方法根据集成方式的不同又细分为 4 个子类(bagging 对应的简称为 BG、adaboost 对应的简称为 AB、rotation forest 对应的简称为 RF、random subspace 对应的简称为 RS),每个子类各包括 5 种方法.以 BG+LMT 为例,其表示该方法采用 LMT 方法作为基方法,以 BG 作为集成学习方法.不难看出,这些方法基本上可以覆盖目前机器学习领域中比较经典的有监督方法.

2.2.2 无监督学习方法

与有监督学习方法相比,无监督学习方法具有不需要已标记训练数据、计算开销小以及实现简单等的优点,因此成为当前软件缺陷预测研究中的一个关注热点.Koru 等人[20,21]发现,模块的代码规模与其含有缺陷的概率呈正比,因此,代码规模越小的模块,越需要优先给他分配测试资源.这样才能在给定的测试资源下,检测出更多的缺陷.Menzies 等人[22]基于 Koru 等人的发现提出了 ManualUp 模型,随后,Zhou 等人[23]在他们的实证研究中对 ManualUp 模型的有效性进行了验证.本文重点考虑了 Yan 等人[8]提出的无监督方法,具体来说,对某个度量元 M,其相应的建模方法为 $R(c)=1/M(c)$,其中,c 表示文件模块,R 为预测的缺陷概率.不难看出,在该方法中,度量值较小的模块将排在更前面.

3 实验设计

本文在实证研究中重点分析如下实验问题.

- RQ1:在同项目缺陷预测场景下,基于多目标优化的 MULTI 方法的预测性能是否好于基准方法?
- RQ2:在跨项目缺陷预测场景下,基于多目标优化的 MULTI 方法的预测性能是否好于基准方法?
- RQ3:若基于 PMI 和 FIA 指标,基于多目标优化的 MULTI 方法与基准方法相比,表现如何?

前两个实验问题基于代价感知的评测指标来比较多目标优化方法 MULTI 与 Yan 等人考虑的基准方法的性能差异.后两个指标则由 Huang 等人[6]提出,重点从测试的模块数以及模块测试时的误报问题这两个角度进行比较,其具体含义见第 3.2 节.

3.1 评测对象

本文考虑的评测对象来自 PROMISE 数据集[24],这些评测对象来自于开源项目,具有代码规模大、项目较为有名、项目活跃时间长等特点.除此之外,这些项目也覆盖了不同类型的应用领域,因此具有一定的典型性.如 Ant 属于程序构建工具、Tomcat 属于 Web 服务器.这些评测对象在度量文件模块时考虑的度量元的类型、名称和具体含义件表 3.

基于这些评测对象搜集到的数据集的统计特征见表 4,包括项目的名称和版本号、含有的文件数、缺陷文件数以及缺陷文件数所占的比例.不难看出,这些数据集都存在一定的类不平衡问题,缺陷文件占所有文件的比例介于 6.6%~34.1%之间.

Table 3 Metrics used by experimental subjects

表 3 评测对象考虑的度量元

类型	度量元名称	度量元含义
Complexity	LOC	Lines of code
	WMC	Weighted methods per class
	NPM	Number of public methods
	AMC	Average method complexity
	Max_cc	Max value of CC among methods of the investigated class
	Avg_cc	Average value of CC among methods of the investigated class
	MOA	Measure of aggregation
Coupling	CBO	Coupling between object classes
	RFC	Response for a class
	CA	Afferent couplings
	CE	Efferent couplings
	IC	Inheritance coupling
	CBM	Coupling between methods
Cohesion	LCOM	Lack of cohesion in methods
	LCOM3	Lack of cohesion in methods
	CAM	Cohesion among methods of class
Abstraction	DIT	Depth of inheritance tree
	NOC	Number of children
	MFA	Measure of functional abstraction
Encapsulation	DAM	Data access metric

Table 4 Statistics of datasets

表 4 数据集的统计特征

项目名称及版本号	文件数	缺陷文件数	缺陷文件所占比例(%)
Ant-1.7	745	166	22.3
Camel-1.6	965	188	19.5
Ivy-1.4	241	16	6.6
Jedit-4.0	306	75	24.5
Log4j-1.0	135	34	25.2
Velocity-1.6	229	78	34.1
POI-2.0	314	37	11.8
Tomcat-6.0	858	77	9.0
Xlan-2.4	723	110	15.2
Xerces-1.3	453	69	15.2

3.2 评测指标

本文重点考察测试代价感知的评测指标,Mende 等人[25]首次将测试代价引入到缺陷预测的建模过程,Kamei 等人[26]则将传统指标下的实证研究结论在代价感知的评测指标下进行了重新验证.与已有研究工作[4,13]保持一致,本文重点考虑了两个评测指标:ACC 和 P_{OPT}.这两个指标均考虑了软件测试代价,并且都是取值越大,对应的模型性能就越好.本文将程序模块的规模视为测试代价,其中,ACC 指标计算的是当使用了指定比例(一般该比例被设置为 20%)的测试资源后,模型针对缺陷模块的查全率;而 P_{OPT} 指标则可以认为针对基于测试开销的指标进行了归一化处理,其计算示意图如图 1 所示,图中共有 3 条曲线,分别对应预测模型 *m*、最优模型 *optimal* 和最差模型 *worst*.

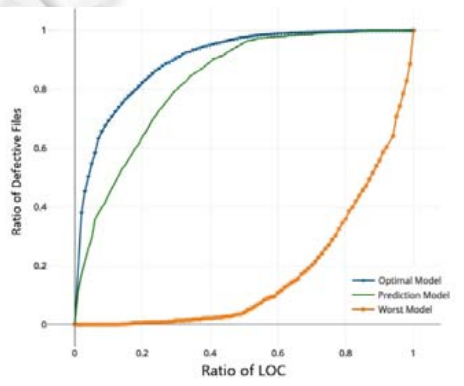


Fig.1 Illustration of P_{opt} performance metric

图 1 P_{opt} 指标的示意图

在最优模型中,模块按照其实际的缺陷密度从高到低进行排序,在最差模型中,模块则按照其实际的缺陷密度从低到高进行排序.*Area*(*optimal*),*Area*(*worst*)和

$Area(m)$分别表示模型 *optimal*,*worst* 以及 m 对应曲线下的面积.最终 P_{OPT} 指标的计算公式如下所示.

$$P_{opt}(m)=1-\frac{Area(optimal)-Area(m)}{Area(optimal)-Area(worst)} \tag{5}$$

除此之外,本文还考虑了 Huang 等人[6]提出的两种新评测指标.具体来说,PMI(proportion of modules inspected)指标计算在花费 20%的测试资源后,测试的模块所占比例.其取值越高,表示在相同的测试成本下进行测试的模块数越多,这意味着需要开发人员进行更多的上下文切换,并可能对他们的开发效率产生影响[27].IFA (number of initial false alarms)指标返回开发人员依次审查模块时,当遇到第 1 个真正缺陷模块之前需要测试的模块数.该指标在软件缺陷定位研究中[28]经常被用到,其取值越高,表示误报问题越严重,并可能会对开发人员的信心和耐心造成影响.

3.3 模型性能评估场景

与 Yan 等人的研究工作[8]保持一致,本文同样考虑了两种模型性能评估场景:同项目缺陷预测场景和跨项目缺陷预测场景.已有的研究工作大部分集中于同项目缺陷预测场景,即基于一个项目的部分数据来训练预测模型,并用该项目内的剩余数据来评估训练出的模型的预测性能.目前,一般采用 10 折交叉验证方式,即将数据集借助分层采样方法划分为 10 等份,轮流将其中的 9 份作为训练集,剩余 1 份作为测试集.重复上述过程 10 次,以确保每个模块都能被预测到 1 次.同时为了避免数据集中实例次序对预测结果的影响,在实验中,我们将 10 折交叉验证重复了 10 次,每次执行前使用不同的随机种子将数据集中的模块进行随机打乱,论文将上述模型性能验证方式称为 10×10 折交叉验证.

但在实际软件开发的时候,需要进行预测的项目(即目标项目)可能是一个全新启动项目,或这个项目已搜集的标记数据不多,一种解决方法是使用搜集自其他项目(即源项目)的标记数据来训练模型.本文将这种模型验证方式称为跨项目缺陷预测[29−31].假设数据集内含有 n 个项目,则总共可构成 $n\times(n-1)$个跨项目缺陷预测场景.由于本文共考虑了 10 个项目,因此最终可构成 90 个跨项目缺陷预测场景.

3.4 实验流程极其方法参数设定

根据多目标优化算法在数值问题上的参数设置经验[32],本文在实验时将 MULTI 方法的参数和具体取值分别设置如下:种群规模被设置为 200,系数向量内元素的有效取值区间被设置为[−10000,10000],在种群初始化时,系数向量内元素的有效取值区间被设置为[−10,10],最大迭代次数被设置为 400,变异概率被设置为 0.05,交叉概率被设置为 0.5.

考虑到 MULTI 方法内部存在多个随机因素,我们会独立运行 MULTI 方法 10 次并每次会设置不同的随机种子.每次运行时,MULTI 方法基于训练集会计算出一组 Pareto 最优解集.因此独立运行 10 次后,会累计计算出 10 组 Pareto 最优解集.最终,MULTI 方法会返回这 10 组 Pareto 最优解集中能够在测试集上取的最好性能的解.其具体运行过程如图 2 所示.

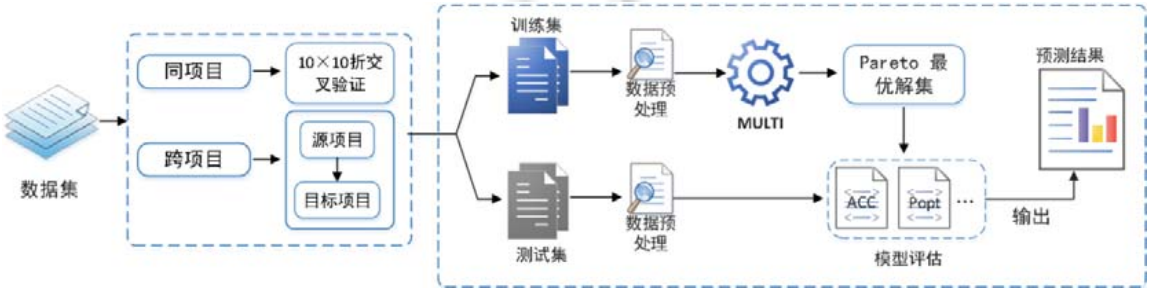


Fig.2 Experimental process of MULTI method

图 2 MULTI 方法的实验流程

对需要比较的有监督学习方法,其超参数取值的设定与Yan等人[8]保持一致.针对FL-SDP问题,本文主要考虑了19种无监督学习方法(每种度量元对应一种方法,未考虑LOC度量元的原因是因为本文需要使用LOC值来度量模块的测试代价).

除此之外,针对多目标优化方法和有监督学习方法,我们进行了如下数据预处理.

(1) 已有研究表明,数据集内含有的冗余特征会降低随后构建出的模型的预测性能[33].如果一个特征重复了其他单个或多个特征含有的信息,则称该特征为冗余特征.因此,借助Yan等人[8]考虑的特征选择方法移除具有高冗余性的特征.这样可以保证本文所提的方法与Yan等人考虑的方法进行公平的比较.

(2) 对所有数值型特征取值进行Log转换处理,以缓解特征取值的偏斜(skew)问题.

(3) 在训练集上应用随机欠采样方法来解决搜集的数据集内存在的类不平衡问题.该处理方式与文献[8]保持一致.即会在训练集上随机移除无缺陷程序模块,直至无缺陷程序模块的数量与有缺陷程序模块的数量相等.

除此之外,为了保证结果比较的公平性,针对无监督学习方法,我们仅基于测试集内的程序模块来训练预测模型.

3.5 显著性检验方法

本文借助Scott-Knott检验[34]为本文考虑的所有方法(总共51种)进行排序和分组.Scott-Knott检验尝试将这些不同的方法划分到具有显著性差异的秩中(α=0.05).具体来说,Scott-Knott检验使用分层聚类分析为每个方法设置不同的秩.其首先将所有方法基于平均性能(基于ACC或P_{OPT}指标)划分成两组.如果处在一组内的方法仍存在显著差异性,则其会迭代使用上述过程将该组内的方法继续分组,直至组内的方法之间不存在显著差异性为止.

为了分析基于多目标优化的MULTI方法相比其他基准方法是否具有显著性优势,本文考虑了Benjamini-Hochberg(BH)修正后的p值[35],并将其显著水平设置为0.05.如果MULTI方法具有显著优势,则进一步采用Cliff's δ来度量这种差异程度.其差异程度及建议取值范围与文献[35]保持一致,具体见表5.具体比较过程总结如下:如果MULTI方法显著优于/差于指定的基准方法,需要BH修正后的p值小于0.05,同时,其差异程度不是negligible的;如果MULTI方法与指定的基准方法不存在显著性差异,则BH修正后的p值不小于0.05,或者虽然BH修正后的p值小于0.05,但其差异程度是negligible的.

Table 5 Magnitude of effectiveness level of Cliff'S δ and its thresholds

表5 Cliff's δ的差异程度及其建议取值范围

差异程度	取值范围
Negligible	$\|\delta\|\leqslant 0.147$
Small	$0.147\leqslant\|\delta\|<0.33$
Medium	$0.33\leqslant\|\delta\|<0.474$
Large	$0.474\leqslant\|\delta\|$

4 实证研究结果的分析

首先,我们通过将MULTI方法与两种简单的搜索方法相比,来验证FL-SDP问题的求解难度.求解难度是指在问题对应的搜索空间内搜索到高质量解的难度.如果问题对应的搜索空间比较简单,则仅使用本文考虑的两种简单搜索方法就可以找到高质量解,这样就不需要求助更加复杂的搜索方法(例如本文提出的基于多目标优化的MULTI方法)了.第1种简单搜索方法是基于单目标优化的遗传算法(SIMPLE-GA).给定需要预测的程序模块集M和候选解w,其搜索优化目标可以通过如下公式进行计算.

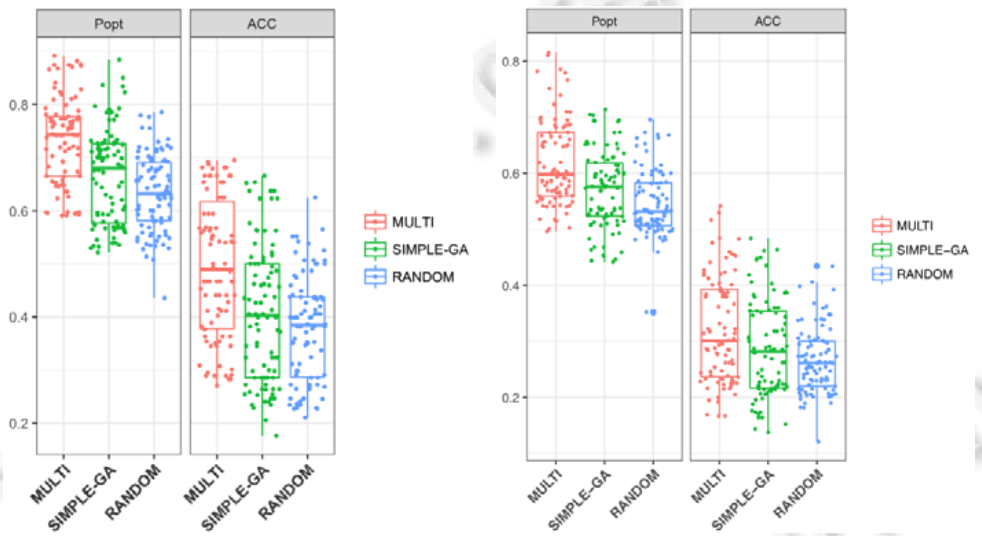
$$0.5\times\frac{benefit(w)}{|M|}+0.5\times\frac{cost(w)}{\sum_{m_i\in M}SQA(m_i)} \tag{6}$$

不难看出,该简单搜索方法将MULTI方法考虑的两个不同优化目标拟合成一个单一优化目标(这里仅简单

地将这两个优化目标的权重各自设置为 0.5),同时,在拟合时我们将两个不同优化目标的取值归一到同一取值区间[0,1]内.

另一种简单搜索方法是基于多目标优化的随机搜索算法(RANDOM),与 MULTI 方法相比,其使用随机搜索代替了其中的遗传算法.

由于这两种简单搜索方法的内部均含有随机因素,因此我们同样独立运行 10 次,并返回其中最好的结果和平均的结果.同时,将这两种简单搜索方法的参数取值(例如种群规模、最大迭代次数、系数向量成员的有效取值区间)与 MULTI 方法保持一致.基于跨项目缺陷预测场景下的比较结果如图 3 所示.不难看出,MULTI 方法相对于两种简单搜索方法在 ACC 和 P_{OPT} 指标上均可以取的更好的预测性能.除此之外,我们在同项目缺陷预测场景下也得到了相似的结论,由此验证了 FL-SDP 问题的求解难度.



(a) 取最好结果 (b) 取平均结果

Fig.3 Comparision between MULTI with two simple search methods in cross-project defect prediction scenario

图 3 跨项目缺陷预测场景下 MULTI 方法与两种简单搜索方法的比较

随后,我们使用 hypervolume(HV)指标来比较 MULTI 方法与 RANDOM 方法生成的 Pareto 前沿的质量.HV 指标是多目标优化算法评估中常用的一种经典指标[36],其可以很好地衡量 Pareto 前沿覆盖的目标空间的容量,因此 HV 取值越大,表示对应的 Pareto 前沿质量越高.我们同样基于跨项目缺陷预测场景,将 MULTI 方法与 RANDOM 方法生成的 Pareto 前沿基于 HV 值进行了比较,最终结果如图 4 所示.通过图 4 不难看出,MULTI 方法生成的 Pareto 前沿质量更高.

4.1 针对RQ1的结果分析

同项目缺陷预测场景下,不同方法的 Scott-Knott 检验结果如图 5 所示.其中,虚线用于分隔不同的分组,所有方法按照他们的秩进行排序.蓝色表示无监督学习方法,黑色表示有监督学习方法.从图中不难看出:无论是基于 ACC 指标还是基于 P_{OPT} 指标,MULTI 方法均显著优于 Yan 等人考虑的无监督学习方法和有监督学习方法.

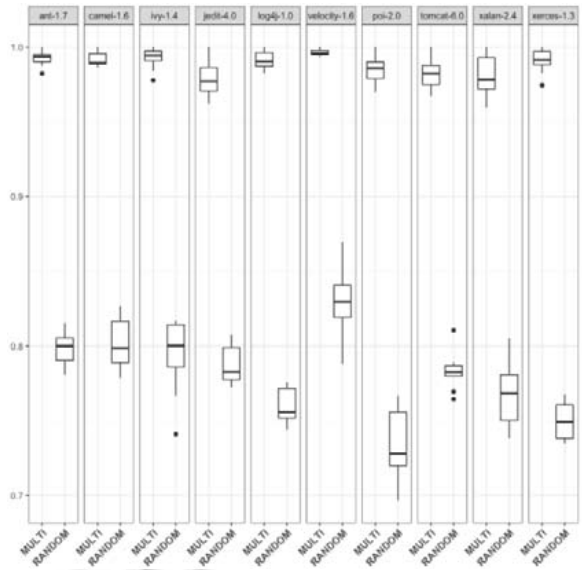
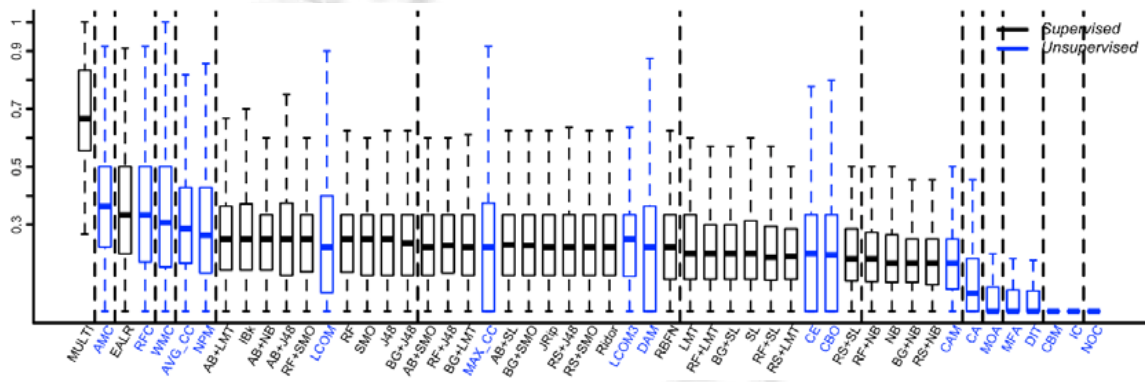
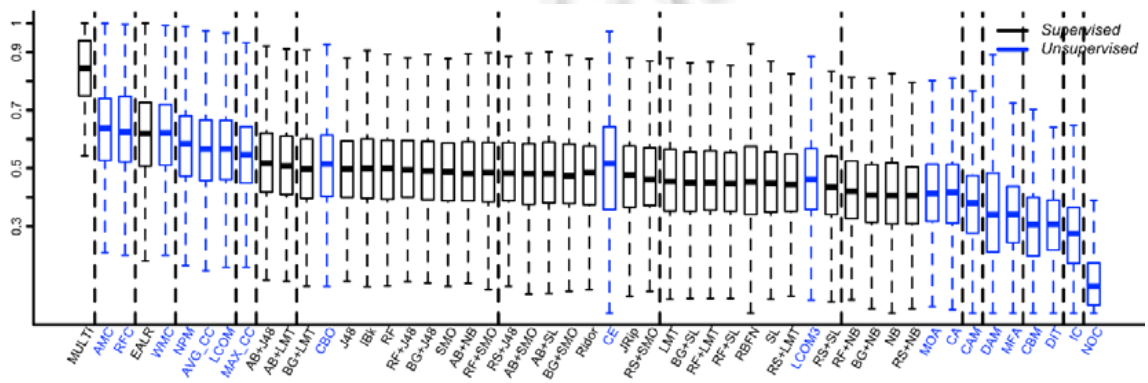


Fig.4 Comparision between MULTI with RANDOM based on HV in cross-project defect prediction scenario

图 4 跨项目缺陷预测场景下 MULTI 方法与 RANDOM 方法在 HV 值上的比较



(a) ACC



(b) P_{OPT}

Fig.5 Result of Scott-Knott test in within-project defect prediction scenario

图 5 同项目缺陷预测场景下的 Scott-Knott 检验结果

随后,我们基于每个项目来将 MULTI 方法与基准方法进行比较.我们从基准方法中选出性能最好的两种有监督方法和两种无监督方法.最终结果见表 6 和表 7.

Table 6 MULTI vs. top two supervised and unsupervised methods in within-project defect prediction scenario using ACC

表 6 同项目缺陷预测场景下 MULTI 方法与最好的两种有监督方法和无监督方法的比较(基于 ACC)

项目名称	无监督方法			有监督方法	
	MULTI	AMC	RFC	EALR	AB+LMT
Ant-1.7	0.412	0.235	0.205	0.209	0.200
Camel-1.6	0.682	0.400	0.500	0.455	0.263
Ivy-1.4	1.000	0.000	0.000	0.000	0.000
Jedit-4.0	0.625	0.444	0.369	0.286	0.286
Log4j-1.0	0.667	0.414	0.250	0.333	0.286
Velocity-1.6	0.778	0.500	0.500	0.500	0.300
POI-2.0	1.000	0.250	0.354	0.333	0.310
Tomcat-6.0	0.571	0.286	0.222	0.250	0.226
Xlan-2.4	0.551	0.308	0.250	0.304	0.200
Xerces-1.3	0.800	0.600	0.586	0.500	0.286
Average	0.708	0.344	0.324	0.317	0.236
W/D/L	–	10/0/0	10/0/0	10/0/0	10/0/0

Table 7 MULTI vs. top two supervised and unsupervised methods in within-project defect prediction scenario using P_{OPT}

表 7 同项目缺陷预测场景下 MULTI 方法与最好的两种有监督方法和无监督方法的比较(基于 P_{OPT})

项目名称	无监督方法			有监督方法	
	MULTI	AMC	RFC	EALR	AB+J48
Ant-1.7	0.676	0.550	0.556	0.540	0.477
Camel-1.6	0.819	0.666	0.740	0.702	0.550
Ivy-1.4	0.990	0.482	0.570	0.482	0.369
Jedit-4.0	0.840	0.653	0.671	0.597	0.509
Log4j-1.0	0.946	0.668	0.535	0.627	0.547
Velocity-1.6	0.921	0.755	0.746	0.748	0.561
POI-2.0	0.936	0.536	0.611	0.595	0.618
Tomcat-6.0	0.748	0.581	0.524	0.551	0.505
Xlan-2.4	0.750	0.619	0.599	0.614	0.502
Xerces-1.3	0.906	0.775	0.765	0.695	0.501
Average	0.853	0.629	0.632	0.615	0.514
W/D/L	–	10/0/0	10/0/0	10/0/0	10/0/0

表中记录的是对应方法在每个数据集上进行 10×10 折交叉验证时,所有预测值中的中位数.这两个表中的倒数第行表示各个方法在不同数据集上的均值,最后一行的 Win/Draw/Loss(简称 W/D/L)表示 MULTI 方法显著优于/相似/差于对应方法的数据集的数量.具体来说:

(1) 若基于 ACC 评测指标,平均来说,MULTI 方法与最好的无监督方法 AMC 和 RFC 相比,其性能有 105.81%和 118.52%的提高;MULTI 方法与最好的有监督方法 EALR 和 AB+LMT 相比,其性能有 123.34%和 200.00%的提高.

(2) 若基于 *Popt* 评测指标,平均来说,MULTI 方法与最好的无监督方法 AMC 和 RFC 相比,其性能有 35.61%和 34.97%的提高;MULTI 方法与最好的有监督方法 EALR 和 AB+J48 相比,其性能有 38.70%和 65.95%的提高.

W/D/L 分析结果也进一步确认了 MULTI 方法的优越性.

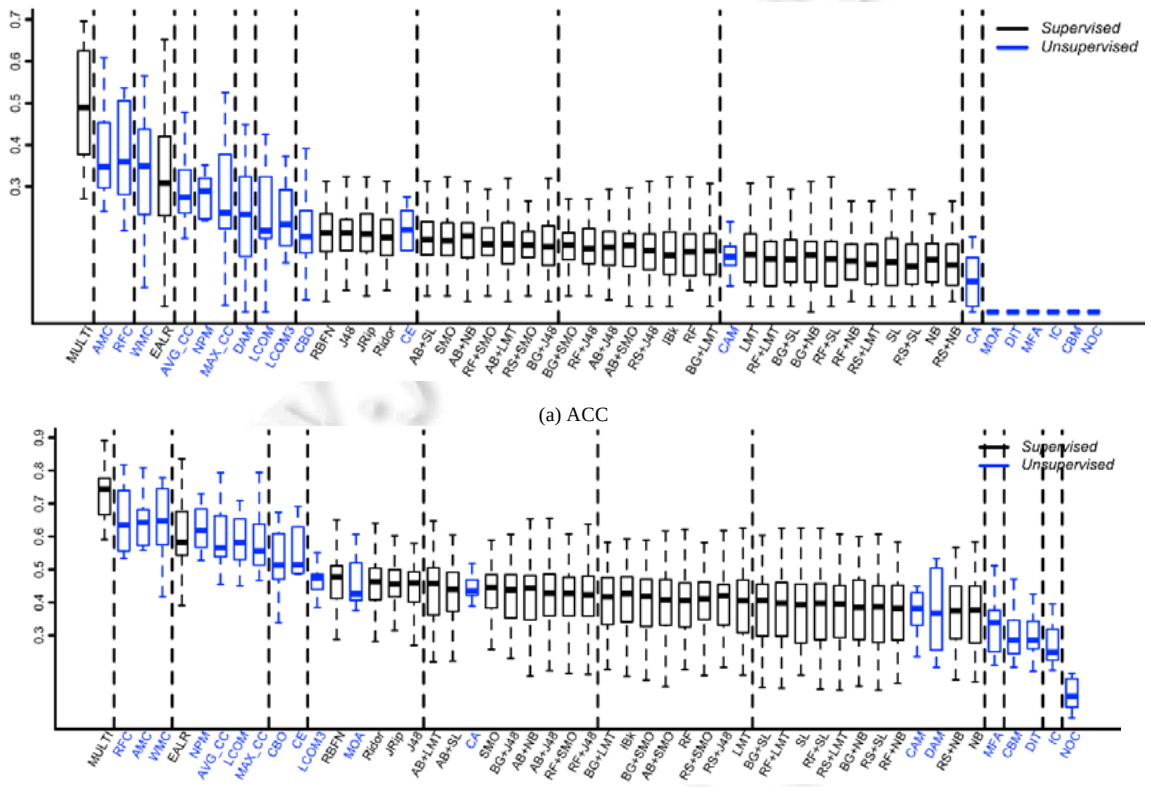
4.2 针对RQ2的结果分析

跨项目缺陷预测场景下,不同方法的 Scott-Knott 检验结果如图 6 所示.从图中不难看出:无论是基于 ACC 指标还是基于 P_{OPT} 指标,MULTI 方法均要显著优于 Yan 等人考虑的无监督学习方法和有监督学习方法.

随后,我们基于每个跨项目缺陷预测场景来将 MULTI 方法与基准方法进行比较.我们从基准方法中选出性能最好的两种有监督方法和两种无监督方法.由于篇幅所限,这里并不罗列出具体结果,总体来说:

(1) 若基于 ACC 评测指标,平均来说,MULTI 方法与最好的无监督方法 AMC 和 RFC 相比,其性能有 22.42%和 25.66%的提高;MULTI 方法与最好的有监督方法 EALR 和 RBFN 相比,其性能有 34.95%和 60.81%的提高.

(2) 若基于 P_{opt} 评测指标,平均来说,MULTI 方法与最好的无监督方法 AMC 和 RFC 相比,其性能有 11.45%和 10.80%的提高,MULTI 方法与最好的有监督方法 EALR 和 RBFN 相比,其性能有 17.92%和 37.28%的提高.



(a) ACC

(b) P_{OPT}

Fig.6 Result of Scott-Knott test in cross-project defect prediction scenario

图 6 跨项目缺陷预测场景下的 Scott-Knott 检验结果

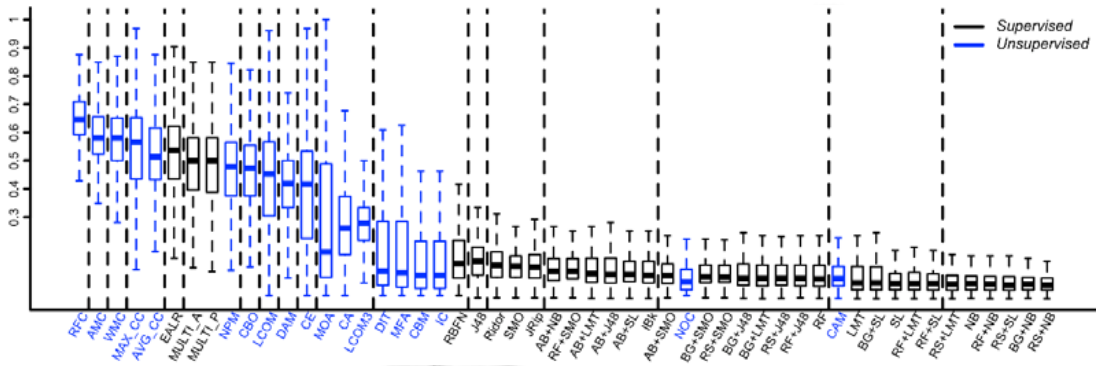
4.3 针对RQ3的结果分析

在该实验问题中,我们基于 PMI 和 IFA 评测指标,将 MULTI 方法与基准方法进行比较.由于基于 ACC 和 P_{OPT} 指标,可能 MULTI 方法返回的最优模型并不一样,所以本文用 MULTI_A 和 MULTI_P 分别表示基于 ACC 指标和 P_{OPT} 指标下得到的最优模型.最终,基于同项目缺陷预测场景和跨项目缺陷预测场景下的结果分别如图 7(a)和图 7(b)所示.不难看出,在考虑 20%的测试资源时,MULTI 方法需要审查的程序模块数较多,但在同项目缺陷预测场景中要低于无监督学习方法 RFC、AMC、WMC、MAX_cc 和 AVG_cc.在跨项目缺陷预测场景下,要低于无监督学习方法 RFC 和 AMC.

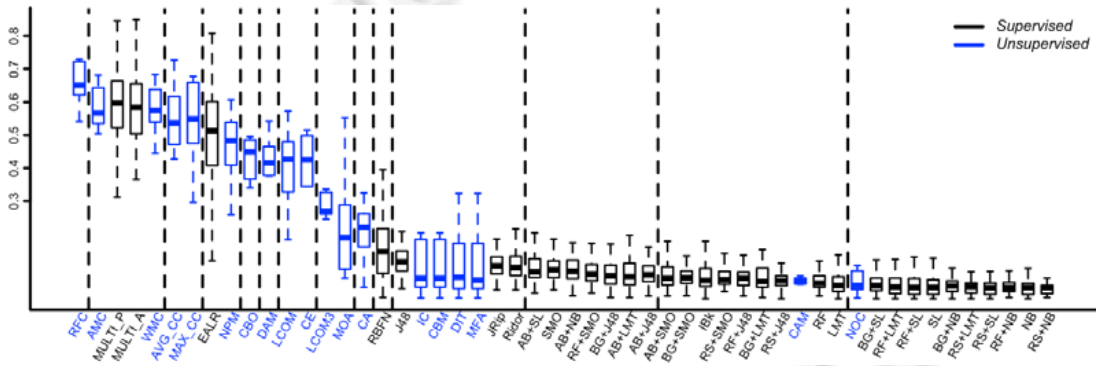
IFA 指标返回是模型在遇到第 1 个缺陷模块时,审查过的无缺陷模块的数量.最终,基于同项目缺陷预测场景和跨项目缺陷预测场景下的结果分别如图 8(a)和图 8(b)所示.在同项目缺陷预测场景中,MULTI_A 和 MULTI_P 分别排名第 5 和第 4.在跨项目缺陷预测场景中,MULTI_A 和 MULTI_P 分别排名第 3 和第 4.

综上所述,不难看出 ACC 指标和 P_{OPT} 指标与 Huang 等人提出的 PMI 指标和 IFA 指标之间存在一定的折

衷,即 MULTI 方法虽然在 ACC 指标和 P_{OPT} 指标上性能更好,但在 PMI 指标和 IFA 指标上则取值稍高,因此需要开发人员根据自己的实际需求做出合理的选择.但我们也发现:与最好的两种无监督学习方法(即 RFC 和 AMC)相比,MULTI 方法能够取得更好的 ACC 值和 P_{OPT} 值;同时,PMI 值和 IFA 值都要好于这两种无监督学习方法.因此,体现了 MULTI 方法具有一定的竞争性.



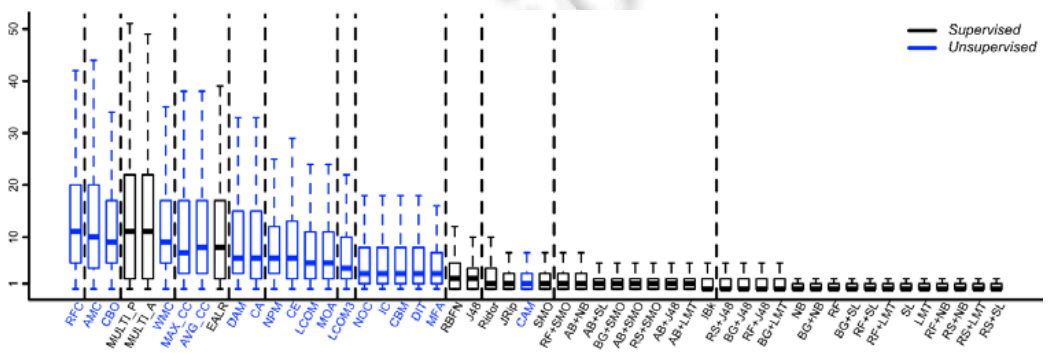
(a) 同项目缺陷预测场景



(b) 跨项目缺陷预测场景

Fig.7 Result of Scott-Knott test for PMI

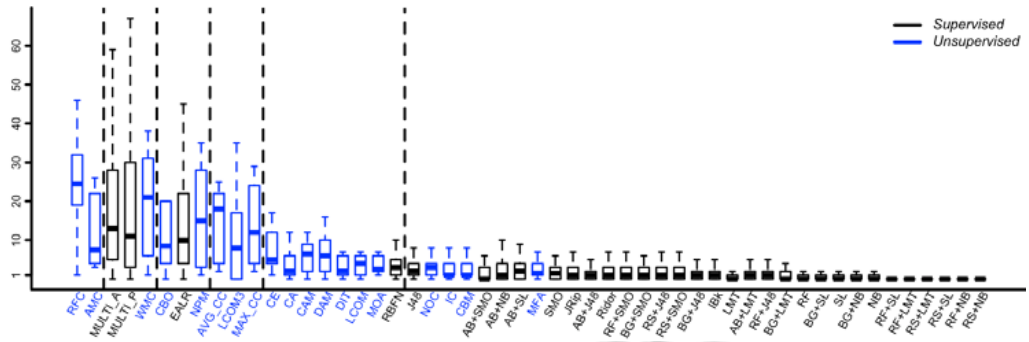
图 7 基于 PMI 指标的 Scott-Knott 检验结果



(a) 同项目缺陷预测场景

Fig.8 Result of Scott-Knott test for IFA

图 8 基于 IFA 指标的 Scott-Knott 检验结果



(b) 跨项目缺陷预测场景

Fig.8 Result of Scott-Knott test for IFA (Continued)

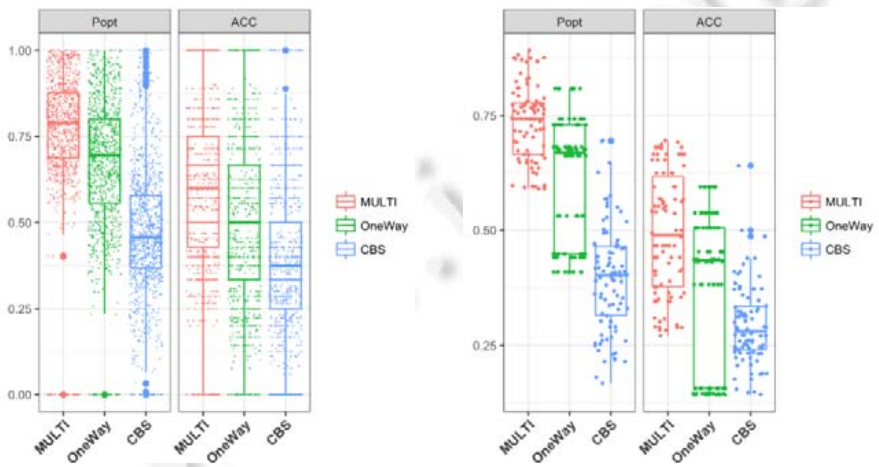
图 8 基于 IFA 指标的 Scott-Knott 检验结果(续)

4.4 进一步讨论

4.4.1 与 OneWay 方法和 CBS 方法的比较

OneWay[5]是 Fu 等人提出的最新的一种有监督学习方法,其基于训练集自动从已有的无监督方法中选出最好的方法.CBS(classify-before-sorting)方法[6]是 Huang 等人提出的最新的一种简单的改进型有监督学习方法,其首先对数据集进行一定的预处理(例如特征选择、类不平衡学习以及对特征取值进行 log 取值转化);随后,基于 Logistic 回归建模方法构建模型,在测试集上,根据模型的预测结果将模块分为两类:有缺陷的和无缺陷的;随后,根据模块代码规模将预测为有缺陷类中的模块按照代码规模从小到大进行排序.

和第 2.2 节提到的一系列基准方法一样,OneWay 方法和 CBS 方法均可用于 CL-SDP 问题和 FL-SDP 问题.除此之外,作为有监督方法,这两种方法均采用了第 3.4 节所示的同样的数据预处理方法.MULTI 方法与 OneWay 和 CBS 方法的比较结果如图 9 所示.从图中不难看出,无论是基于同项目缺陷预测场景还是基于跨项目缺陷预测场景,MULTI 方法在 ACC 评测指标和 P_{OPT} 评测指标上,其性能均要显著优于 OneWay 方法和 CBS 方法.



(a) 同项目缺陷预测场景 (b) 跨项目缺陷预测场景

Fig.9 Comparision among MULTI, OneWay and CBS

图 9 MULTI 方法与 OneWay 和 CBS 方法的比较

4.4.2 基于 *F*1 指标的比较结果

除了 ACC 和 P_{OPT} 评测指标以外,本文也考虑缺陷预测研究中经常使用的 *F*1 评测指标[1,31].具体来说,假设仅能使用 20%的测试资源,则使用 *precision* 表示审查到的模块中缺陷模块所占的比例,使用 *recall*(即 ACC)表示审查到的缺陷模块占所有缺陷模块的比例,*F*1 则是 *precision* 值和 *recall* 值的调和均值.以跨项目缺陷预测场景为例,本文将 MULTI 方法与 Yan 等人采用的方法[8]、OneWay 方法[5]以及 CBS 方法[6]进行了比较,结果如图 10 所示.不难看出,MULTI 方法均要显著优于这些基准方法.

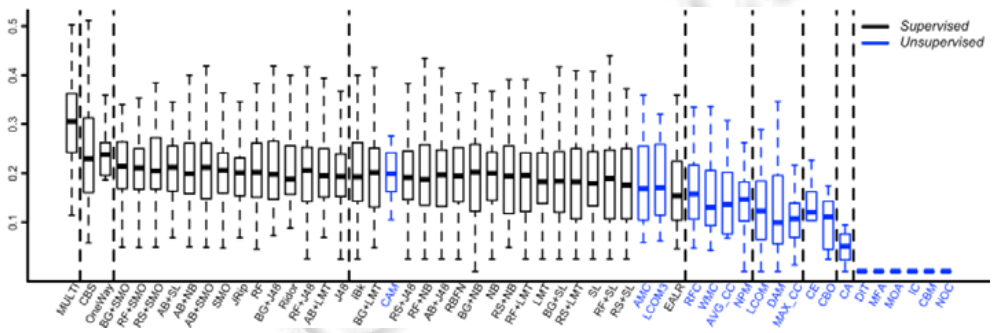


Fig.10 Result of Scott-Knott test in cross-project defect prediction scenario based on *F*1

图 10 跨项目缺陷预测场景下的 Scott-Knott 检验结果(基于 *F*1 评测指标)

4.4.3 时间开销分析

我们进一步分析 MULTI 方法的模型构建时间开销.由于无监督方法实现简单,因此其运行时间极短,本文主要将 MULTI 方法与不同的有监督学习方法(包括 Yan 等人考虑的方法、OneWay 方法和 CBS 方法)进行比较.这些方法运行在 MAC OS High Sierra 操作系统上(CPU 2.3 GHz Intel Core i5,内存 8GB),最终结果见表 8.

Table 8 Model construction time of different FL-SDP supervised methods (s)

表 8 不同 FL-SDP 有监督方法的构建时间 (秒)

方法	Ant	Camel	Ivy	Jedit	Log4j	Velocity	POI	Tomcat	Xlan	Xerces
MULTI	15.358	15.348	10.013	15.001	14.639	15.602	15.067	15.035	15.668	16.035
OneWay	0.069	0.069	0.058	0.068	0.065	0.062	0.130	0.066	0.110	0.080
CBS	0.479	0.382	0.372	0.549	0.312	0.290	0.249	0.335	0.326	0.473
EALR	0.311	0.250	0.115	0.188	0.184	0.209	0.143	0.305	0.199	0.316
NB	0.054	0.041	0.035	0.033	0.038	0.035	0.042	0.046	0.037	0.069
SL	0.066	0.084	0.032	0.043	0.061	0.046	0.036	0.062	0.053	0.077
LMT	0.111	0.376	0.036	0.069	0.055	0.109	0.050	0.094	0.069	0.110
RBFNet	0.055	0.054	0.041	0.031	0.042	0.034	0.032	0.070	0.034	0.050
JRIP	0.053	0.057	0.034	0.037	0.054	0.044	0.039	0.077	0.044	0.065
RF	0.144	0.228	0.049	0.077	0.070	0.102	0.058	0.086	0.110	0.178
SMO	0.043	0.048	0.038	0.036	0.051	0.043	0.037	0.034	0.034	0.116
J48	0.039	0.064	0.032	0.039	0.043	0.051	0.070	0.044	0.050	0.064
RIDOR	0.071	0.072	0.034	0.029	0.037	0.033	0.083	0.039	0.051	0.052
IBK	0.133	0.114	0.033	0.055	0.041	0.059	0.084	0.055	0.097	0.072
BG+SL	0.430	0.524	0.096	0.208	0.131	0.235	0.198	0.192	0.261	0.199
BG+LMT	0.962	2.583	0.142	0.684	0.244	0.782	0.382	0.541	0.758	0.596
BG+NB	0.191	0.130	0.096	0.102	0.110	0.146	0.110	0.169	0.103	0.096
BG+SMO	0.110	0.109	0.053	0.099	0.064	0.142	0.060	0.084	0.069	0.110
BG+J48	0.073	0.107	0.047	0.058	0.045	0.092	0.044	0.044	0.055	0.048
RF+SL	0.490	0.575	0.124	0.297	0.172	0.427	0.175	0.246	0.360	0.246
RF+LMT	1.110	1.497	0.151	0.440	0.222	0.718	0.292	0.435	0.613	0.434
RF+NB	0.280	0.280	0.142	0.178	0.149	0.232	0.165	0.254	0.240	0.203
RF+SMO	0.178	0.246	0.139	0.138	0.185	0.162	0.138	0.109	0.158	0.221
RF+J48	0.168	0.248	0.089	0.131	0.116	0.159	0.125	0.104	0.137	0.195
AB+SL	0.175	0.201	0.048	0.080	0.080	0.105	0.077	0.095	0.141	0.115
AB+LMT	1.751	2.499	0.106	0.666	0.307	0.733	0.396	0.691	1.413	0.840
AB+NB	0.181	0.154	0.105	0.145	0.160	0.111	0.174	0.076	0.135	0.113
AB+SMO	0.084	0.106	0.056	0.105	0.090	0.051	0.132	0.055	0.059	0.065

Table 8 Model construction time of different FL-SDP supervised methods (Continued) (s)

表 8 不同 FL-SDP 有监督方法的构建时间(续) (秒)

方法	Ant	Camel	Ivy	Jedit	Log4j	Velocity	POI	Tomcat	Xlan	Xerces
AB+J48	0.110	0.140	0.043	0.113	0.059	0.067	0.070	0.046	0.078	0.060
RS+SL	0.310	0.455	0.076	0.169	0.111	0.199	0.143	0.162	0.237	0.154
RS+LMT	0.640	1.415	0.082	0.318	0.160	0.356	0.186	0.366	0.541	0.332
RS+NB	0.089	0.103	0.060	0.076	0.061	0.068	0.064	0.068	0.100	0.078
RS+SMO	0.073	0.086	0.065	0.076	0.054	0.054	0.082	0.051	0.126	0.095
RS+J48	0.056	0.082	0.031	0.048	0.035	0.045	0.045	0.044	0.094	0.068

通过表 8 不难发现:MULTI 方法的模型构建时间虽然都高于基准方法,但处于可接受的范围之内(介于 10s~17s 之间),其主要计算开销集中在种群演化阶段,即每一轮中,种群内染色体适应值的计算以及常见演化算子的计算开销等.

4.5 有效性影响因素分析

这一节主要分析可能影响到本文实证研究结论有效性的影响因素,具体说明如下.

(1) 内部有效性主要涉及到可能影响到实验结果正确性的内部因素.第一个有效性影响因素是代码实现是否正确,为了减少该影响因素的影响,我们参考了 Yang 等人[4]、Wu 和 Menzies[5]、Yan 等人[8]提供的代码,并确保与他们实证研究的结果保持一致.除此之外,我们使用了第三方提供的成熟框架,例如来自 Matlab 和 R 中的机器学习包.其次,在 ACC 指标和 PMI 指标设定时,我们假设可用的测试资源比例是 20%,该设定与已有研究工作保持一致[4,13].

(2) 外部有效性主要涉及到实验研究得到的结论是否具有一般性.为了确保实证研究结论的一般性,我们选择了 FL-SDP 问题研究中经常使用的 PROMISE 数据集,该数据集累计搜集了 10 个开源项目,选择的这些项目是开源项目中具有一定代表性的项目,同时,这些项目也覆盖了不同类型的应用领域,可以确保研究结论具有一定的代表性.

(3) 结论有效性主要涉及到使用的评测指标是否合理.本文重点考虑了测试代价感知的评测指标 P_{OPT}, ACC 和 $F1$ 指标;除此之外,我们也深入分析 Huang 等人[6]提出其他两个评测指标 PMI 和 IFA.

5 总结和展望

本文针对 FL-SDP 问题,将基于多目标优化的 MULTI 方法与 Yan 等人考虑的基准方法[8]、Wu 和 Menizes 提出的 OneWay 方法[5]以及 Huang 等人提出的 CBS 方法[6]进行了深入的分析和比较.结果表明:无论在同项目缺陷预测场景还是跨项目缺陷预测场景,若考虑代价敏感的评测指标,MULTI 方法的预测性能均要显著优于这些基准方法,从而表明 MULTI 方法在 FL-SDP 问题上同样值得关注.

本文仍存在很多值得探讨的下一步工作:首先,我们希望能够从开源项目和商业项目中搜集更多的数据集,来验证本文所得的实证研究结论是否具有一般性;其次,本文考虑的数据集预处理方法较为简单,未来可以考虑更为有效的特征选择方法或者类不平衡学习方法[33,37];接着,需要考虑与最新提出的 FL-SDP 方法进行比较,例如 Nam 等人提出的 CLA 和 CLAMI 方法[38]以及 Zhang 等人提出的基于谱聚类(spectral clustering)的方法[39]等;最后,本文仅简单使用模块规模来度量模块的测试开销,该假设较为简单[40],在下一步工作中,需要考虑更为合理的测试开销度量方法.

为了方便研究人员重现本文的实证研究,我们将相关代码和实验结果进行了共享,其访问网址是 https://github.com/Hecoz/FL-SDP.

References:

[1] 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]

[2] 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.

[3] Kamei K, Shihab E. Defect prediction: Accomplishements and future challenges. In: Proc. of the Int'l Conf. on Software Analysis, Evolution, and Reengineering. 2016. 33−45.

[4] Yang YB, Zhou YM, Liu J, Zhao Y, Lu H, Xu L, Xu BW, Leung H. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2016. 157−168.

[5] Fu W, Menzies T. Revisiting unsupervised learning for defect prediction. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. 2017. 72−83.

[6] Huang Q, Xia X, Lo D. Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2017. 159−170.

[7] Chen X, Zhao YQ, Wang QP, Yuan ZD. MULTI: Multi-objective effort-aware just-in-time software defect prediction. Information and Software Technology, 2018,93:1−13.

[8] Yan M, Fang YC, Lo D, Xia X, Zhang XH. File-level defect prediction: unsupervised vs. supervised models. In: Proc. of the Int'l Symp. on Empirical Software Engineering and Measurement. 2017. 344−353.

[9] Radjenovic D, Hericko M, Torkar R, Zivkovic A. Software fault prediction metrics: A systematic literature review. Information and Software Technology, 2013,55(8):1397−14.

[10] Yan M, Xia X, Zhang XH, Yang D, Xu L. Automating aggregation for software quality modeling. In: Proc. of the Int'l Conf. on Software Maintenance and Evolutionary. 2017. 529−533.

[11] Kim S, Zimmermann T, Pan K, Whitehead EJ. Automatic identification of bug-introducing changes. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2006. 81−90.

[12] Ghotra B, McIntosh S, Hassan AE. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proc. of the Int'l Conf. on Software Engineering. 2015. 789−800.

[13] 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.

[14] Mockus A, Weiss D. Predicting risk of software changes. Bell Labs Technical Journal, 2000,5(2):169−180.

[15] Yang X, Lo D, Xia X, Sun J. TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. Information and Software Technology, 2017,87:206−220.

[16] Kamei Y, Fukshima T, McIntosh S, Yamashita K, Ubayashi N, Hassan AE. Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering, 2016,21(5):2072−2106.

[17] Liu JP, Zhou YM, Yang YB, Lu HM, Xu BW. Code churn: A neglected metric in effort-aware just-in-time defect prediction. In: Proc. of the Int'l Symp. on Empirical Software Engineering and Measurement. 2017. 11−19.

[18] Xia X, Lo D, Wang XY, Yang XH. Collective personalized change classification with multiobjective search. IEEE Trans. on Reliability, 2016,65(4):1810−1829.

[19] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2002,6(2):182−197.

[20] Koru AG, Emam KE, Zhang D, Liu H, Mathew D. Theory of relative defect proneness. Empirical Software Engineering, 2008, 13(5):473−498.

[21] Koru AG, Liu H, Zhang D, Emam KE. Testing the theory of relative defect proneness for closed-source software. Empirical Software Engineering. 2010,15(6):577−598.

[22] 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.

[23] Zhou YM, Xu BW, Leung H, Chen L. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Trans. on Software Engineering and Methodology, 2014,23(1):10:1−10:51.

[24] Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction. In: Proc. of the Int'l Conf. on Predictive Models in Software Engineering. 2010. 9:1−9:10.

[25] Mende T, Koschke R. Effort-Aware defect prediction models. In: Proc. of the European Conf. on Software Maintenance and Reengineering. 2010. 107−116.

[26] Kamei Y, Matsumoto S, Monden A, Matsumoto KI, Adams B, Hassan AE. Revisiting common bug prediction findings using effort-aware models. In: Proc. of the Int'l Conf. on Software Maintenance. 2010. 1−10.

[27] Meyer A, Fritz T, Murphy G, Zimmermann T. Software developers' perceptions of productivity. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2014. 19−29.

[28] Parnin C, Orso A. Are automated debugging techniques actually helping programmers? In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2011. 199−209.

[29] 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.

[30] Zhang Y, Lo D, Xia X, Sun JL. An empirical study of classifier combination for cross-project defect prediction. In: Proc. of the IEEE Annual Computer Software and Applications Conf. 2015. 264−269.

[31] Chen X, Wang LP, Gu Q, Wang Z, Ni Chao, Liu WS, Wang QP. A survey on cross-project software defect prediction methods. Chinese Journal of Computers, 2018,41(1):254−274 (in Chinese with English abstract).

[32] Coello C, Lamont G, Veldhuizen D. Evolutionary Algorithms for Solving Multi-Objective Problems. 2nd ed., Springer-Verlag, 2007.

[33] Liu WS, Chen X, Gu Q, Liu SL, Chen DX. A cluster-analysis-based feature-selection method for software defect prediction. Scientia Sinica Informationis, 2016,46(9):1298−1320 (in Chinese with English abstract).

[34] Jelihovschi E, Faria J, Allaman I. ScottKnott: A package for performing the scott-knott clustering algorithm in *R*. Tendências em Matemática Aplicada e Computacional, 2014,15(1):3−17.

[35] Benjamini Y, Hochberg Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. Journal of the Royal Statistical Society, 1995,57(1):289−300.

[36] Wang S, Ali S, Yue T, Liaaen M. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software enginnering. In: Proc. of the Int'l Conf. on Software Engineering. 2016. 631−642.

[37] Liu WS, Liu SL, Gu Q, Chen JQ, Chen X, Chen DX. Empirical studies of a two-stage data preprocessing approach for software fault prediction. IEEE Trans. on Reliability, 2016,65(1):38−53.

[38] Nam J, KimS. CLAMI: Defect prediction on unlabeled datasets. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2015. 452−463.

[39] Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-Project defect prediction using a connectivity-based unsupervised classifier. In: Proc. of the Int'l Conf. on Software Engineering. 2016. 309−320.

[40] Shihab E, Kamei Y, Adams B, Hassan AE. Is lines of code a good measure of effort in effort-aware models? Information and Software Technology, 2013,55(11):1981−1993.

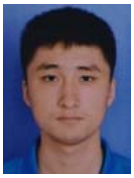
附中文参考文献:

[1] 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] 陈翔,王莉萍,顾庆,王赞,倪超,刘望舒,王秋萍.跨项目软件缺陷预测方法研究综述.计算机学报,2018,41(1):254−274.

[33] 刘望舒,陈翔,顾庆,刘树龙,陈道蓄.软件缺陷预测中基于聚类分析的特征选择方法.中国科学:信息科学,2016,46(9):1298−1320.

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



赵英全(1994－),男,硕士生,主要研究领域为软件缺陷预测.

顾庆(1972－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件质量保障,分布式计算.

倪超(1990－),男,博士生,主要研究领域为软件缺陷预测.

王赞(1979－),男,博士,副教授,CCF 专业会员,主要研究领域为软件测试优化,软件缺陷定位,软件缺陷修复.