SBFS:基于搜索的软件缺陷预测特征选择框架*

陈 翔，陆凌姣，吉 人，魏世鑫
（南通大学 计算机科学与技术学院，江苏 南通 226019）

摘 要：在搜集缺陷预测数据集的时候，由于考虑了大量与代码复杂度或开发过程相关的度量元，造成数据集内存在维数灾难的问题。借助基于搜索的软件工程思想，提出一种新颖的基于搜索的包裹式特征选择框架 SBFS。该框架在实现时，首先借助 SMOTE 方法来缓解数据集内存在的类不平衡问题，随后借助基于遗传算法的特征选择方法，基于训练集选出最优特征子集。在实证研究中，以 NASA 数据集作为评测对象，以基于前向选择策略的包裹式特征选择方法 FW、基于后向选择策略的包裹式特征选择 BW、不进行特征选择的 Origin 作为基准方法。最终实证研究结果表明：SBFS 方法在 90% 的情况下，不差于 Origin 法；在 82.3% 的情况下，不差于 BW 法；在 69.3% 的情况下，不差于 FW 法。除此之外，若基于决策树分类器，则应用 SMOTE 方法后，可以在 71% 的情况下提高模型性能。而基于朴素贝叶斯和 Logistic 回归分类器，则应用 SMOTE 方法后，仅可以在 47% 和 43% 的情况下，提高模型的预测性能。

关键词：软件缺陷预测；特征选择；基于搜索的软件工程；类不平衡学习
中图分类号：TP311.5 **文献标志码**：A **文章编号**：1001-3695(2017)04-1105-04
doi:10.3969/j.issn.1001-3695.2017.04.034

SBFS:search based feature selection framework for software defect prediction

Chen Xiang, Lu Lingjiao, Ji Ren, Wei Shixin
(*School of Computer Science & Technology*, *Nantong University*, *Nantong Jiangsu* 226019, *China*)

Abstract: During the process of gathering defect prediction datasets, the issue of curse of dimensionality may exist in these datasets when considering different metrics based on code complexity or development process. Motivated by the idea of search based software engineering, this paper proposed a novel search based wrapper feature selection framework SBFS. In implementing this framework, it first used SMOTE approach to alleviate the issue of class imbalance, then used a genetic algorithm based feature selection method to select the optimal feature subset based on the training set. In empirical studies, it used NASA dataset as our subjects. Then it chose some classical baseline methods, such as forward search based wrapper feature selection method FW, backward search based wrapper feature selection method BW, and no feature selection method Origin. Finally results show that SBFS is no worse than Origin in 90% of cases, is no worse than BW in 82.3% of cases, and is no worse than FW in 69.3% of cases. Furthermore, when using decision tree classifier, using SMOTE can improve the model performance in 71% of cases. However when using Naive Bayes classifier or logistic regression classifier, using SMOTE can only improve the model performance in 47% and 43% of cases respectively.

Key words: software defect prediction; feature selection; search based software engineering; class imbalance learning

0 引言

软件缺陷预测[1~3]是当前软件工程数据挖掘领域[4]的一个研究热点。软件缺陷预测通过分析软件历史仓库，构建缺陷预测模型，来预先识别出被测项目内的潜在缺陷程序模块，通过将更多测试资源优先分配到这些程序模块，可以达到优化测试资源分配，提高软件产品质量的目的。但在搜集缺陷预测数据集时，若考虑多个度量元(metrics)，则容易造成数据集存在维数灾难(curse of dimensionality)问题，即数据集内会含有无关特征和冗余特征。特征选择[5,6]是一种有效方法，可以尝试识别并移除特征空间中的无关特征和冗余特征，最终达到降低数据集的维数、缩小训练集的规模、缩短训练时间以及提高分类器的性能。

特征选择方法在研究时主要存在两个难点：

a)特征交互问题。该问题是指特征之间存在两两交互、三三交互甚至更高强度的交互。一方面，一个特征可能与类标的关联度不大，但如果该特性与其他特征存在互补关系，则可以显著提升分类器的性能，因此，移除这类特征会造成选出的特征子集并不是最优的。另一方面，某一特征虽然与类标存在很强的相关性，但若与其他特征放在一起时，可能会具有一定冗余性，因此会造成分类器性能的下降。

b)搜索空间大。搜索空间会随着特征数的增加呈指数级增长(即相对于 n 个特征，其可能的特征子集数为 2^n 个)。在大部分情况下，搜索所有可能的子集是不可行的。

目前常见的特征选择方法可以简单分为包裹式方法和过滤式方法。其中包裹式方法借助预先指定的分类器的预测精

收稿日期：2016-05-05；**修回日期**：2016-06-24 **基金项目**：国家自然科学基金资助项目(61202006)；江苏省大学生创新训练计划项目(201610304090X)

作者简介：陈翔(1980-)，男，副教授，博士，主要研究方向为软件缺陷预测、软件缺陷定位和回归测试等；陆凌姣(1994-)，女(通信作者)，本科生，主要研究方向为软件缺陷预测(382243067@qq.com)；吉人(1994-)，男，本科生，主要研究方向为软件缺陷预测；魏世鑫(1996-)，男，本科生，主要研究方向为软件缺陷预测.

度来决定选择出的特征子集,虽然可以取得更高的预测精度,但以增加计算开销为代价。而过滤式方法则根据数据集的特征完成特征的选择,因此与选择的分类器无关,通用性更好且计算开销较小,但精确度并不保证。

研究人员对软件缺陷预测特征选择方法展开了初步的探索。Menzies 等人[7] 和 Song 等人[8] 提出的通用缺陷预测框架中,均包含了包裹式特征选择方法,该方法在特征子集搜索时考虑了两种不同的搜索策略,即前向选择策略和后向选择策略。Gao 等人[9] 针对一个大规模遗留电信软件系统,将特征选择方法应用到缺陷预测中,并考虑了多种过滤式特征选择方法。Kim 等人[10] 对基于代码修改的软件缺陷预测问题展开研究,通过分析源代码、代码修改和修改日志,抽取出大量与代码修改相关的特征。但 Shivaji 等人[11] 发现:上述方法构造出的数据集含有过多的特征(实证研究对象包含的特征数为 6 127 ~41 942 个),即存在明显的维数灾难问题。因此他们考虑了多种不同的过滤式特征选择方法来缓解上述问题。Liu 等人[12] 提出了一种基于聚类分析的过滤式特征选择方法 FECAR。该方法首先基于特征间的相关性将原有特征集进行聚类分析。随后根据特征与类间的相关性从每个聚类中选出相关性高的特征。他们[13,14] 接着进一步提出一种两阶段数据预处理框架,以通过同时解决维数灾难和类不平衡问题来提高缺陷预测数据集质量。该框架同时考虑了基于聚类分析的过滤式特征选择方法和随机欠采样的类不平衡学习方法。

与上述研究工作不同,本文拟借助基于搜索的软件工程(search based software engineering,SBSE)思想[15],提出一种有效的基于搜索的包裹式缺陷预测特征选择框架 SBFS。SBSE[15] 思想最早由 Harman 等人提出,并逐渐成为软件工程领域中的一个热点研究方向。该方向借助基于搜索的方法(即元启发式搜索算法)对软件工程中的组合优化问题进行求解。目前 SBSE 已经成功应用于软件项目开发的各个阶段,包括需求分析、项目规划、项目维护和逆向工程等。SBSE 日益得到越来越多的研究人员的关注,因为在搜索空间规模较大,甚至存在多个相互竞争/冲突目标的情况下,SBSE 能够提供自动化/半自动化的解决方案。

本文提出了一种新颖的基于搜索的包裹式缺陷预测特征选择框架 SBFS;基于来自实际项目的缺陷预测数据集,以 AUC 作为性能评测指标,以决策树、朴素贝叶斯和 Logistic 回归这三种分类器作为模型的构建方法。

1 SBFS 框架及实现

图 1 是 SBFS 的整体执行过程。该方法首先基于训练集尝试选出一个最优特征子集。随后根据选出的特征子集,对训练集和测试集进行预处理(即仅保留选中的特征),并形成预处理后的训练集 train′和预处理后的测试集 test′。最后借助特定的分类器(例如决策树),基于训练集 train′训练出模型 M,并基于测试集 test′得到模型的预测性能。

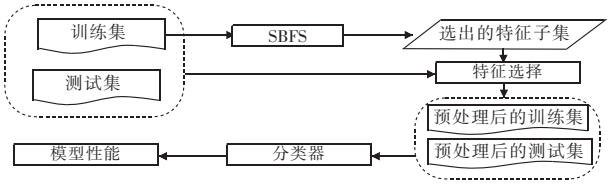

图 1 SBFS 的整体执行过程

由于软件缺陷在被测项目内部的分布存在类不平衡问题,即大部分缺陷都集中存在于少数程序模块之内,所以搜集到的缺陷预测数据集存在明显的类不平衡问题,即缺陷模块(少数类)的数量要远远少于无缺陷模块(多数类)的数量。所以在本文提出的 SBFS 框架中考虑了类不平衡学习方法[16]。其具体框架如图 2 所示。该框架包括两个阶段,首先在第一阶段,借助类不平衡学习方法对训练集进行预处理,随后在第二阶段,借助基于遗传算法的特征选择方法来尝试选出最优特征子集。该框架具有一定的扩展性,在第一阶段,可以考虑其他类不平衡学习方法,如欠采样法、过采样法等。在第二阶段,则可以考虑其他元启发式搜索算法,如粒子群优化、蚁群算法等。

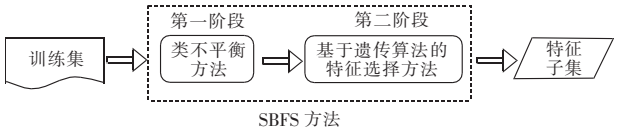

图 2 SBFS 的框架图

1.1 SBFS 的第一阶段

类不平衡学习[16] 是目前学术界和工业界的一个关注热点。存在类不平衡问题的场景包括搜索引擎的点击预测、电子商务领域的商品推荐、信用卡欺诈检测、网络攻击识别等。

已有解决方法可简单分为两类:一类是从调整数据集中的类分布角度出发,经典方法包括随机过采样、随机欠采样和 SMOTE(synthetic minority oversampling technique)方法[17];另一类从修改学习算法的角度出发,即通过修改算法的训练过程,使得学习算法可以针对少数类取得更好的预测精度,典型方法包括代价敏感的学习方法。本文在实现 SBFS 的第一阶段时,考虑了 SMOTE 方法[17],该方法通过对训练集的少数类的实例进行插值,来产生额外的少数类实例。

1.2 SBFS 的第二阶段

本文在实现 SBFS 的第二阶段时,考虑采用遗传算法这一经典的元启发式搜索方法。针对软件缺陷预测特征选择这一问题,其方法流程如图 3 所示。首先初始化种群,随后读取每个染色体对应的特征子集,并根据适应值函数计算出每个染色体的适应值,然后依次执行选择算子(selection operator)、交叉算子(crossover operator)和变异算子(mutation operator)后,不断基于上一个种群形成新的种群。若满足种群演化的终止准则,则返回当前种群中的最优解。种群在演化时,若达到指定的代数或种群提前收敛,则满足终止准则并停止种群演化。

在种群初始化的时候,假设种群规模为 popNum(即含有 popNum 个染色体),其中每个染色体为 n 比特串(假设数据集含有 n 个特征)。若第 i 个比特取值为 1,则表示对应的第 i 个特征被选择,否则比特取值为 0。假设有 5 个特征 $\{f_1, f_2, f_3, f_4, f_5\}$,则初始种群可能为{00100,10010,10110}。其含义是该初始种群共包括三个染色体,这三个染色体对应的特征子集分别为 $\{f_3\}$,$\{f_1, f_4\}$,$\{f_1, f_3, f_4\}$。本文在选择算子的设置时,每一次会从上一个种群中选出适应值最高的染色体,并重复一份放入下一个种群中。交叉算子在设置时,会每次从上一种群中,随机选择两个染色体,并随机确定交叉点,然后进行交叉并形成两个新的染色体。假设交叉算子随机选出两个染色体,分别是 00100 和 01101,并随机确定第三位为交叉点并完成交叉操作,最终会形成两个新的染色体,分别是 00101 和 01100。变异算子在设置时,会随机选择一个染色体,并随机确定变异点,然后进行变异(即将 1 变异为 0,或者将 0 变异为 1)。假设变异算子随机选出一个染色体为 01110,并随机确定第三位为变异

位,因为当前染色体的第三位取值为 1,则应用变异算子后,该值将变为 0,并形成一个新的染色体为 01010。

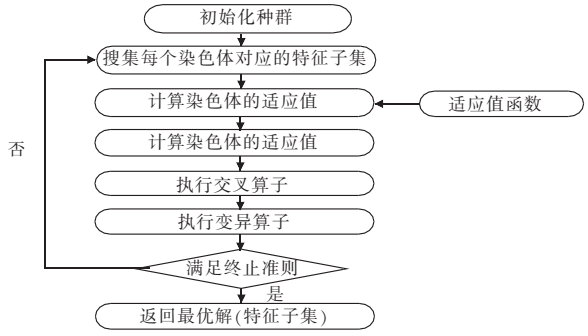

图 3　基于遗传算法的特征选择方法的执行过程

每个染色体的适应值计算过程如图 4 所示,首先读取染色体对应的特征子集,然后将数据集(即图 2 中借助类不平衡方法预处理后的训练集)划分为五份,其中四份构成训练集,剩余一份构成测试集,借助选出的特征子集对训练集和测试集进行预处理(即保留选中的特征),并基于预处理后的训练集,借助同一分类器构建模型,并在测试集上计算出模型的 AUC 值(3.2 节中将会对 AUC 值进行介绍)。上述过程重复五次,并确保每份数据都至少被用做测试集一次。最终计算出这五个 AUC 值的均值 AUC_{avg},并将 $1-AUC_{avg}$ 作为染色体对应的适应值,不难看出染色体的适应值越小,则表示其质量越高。

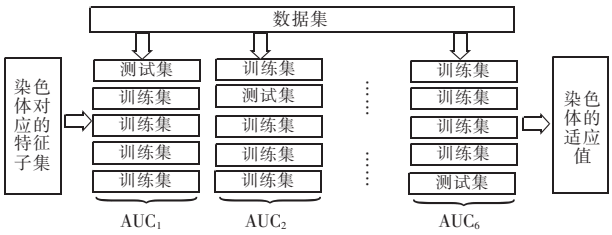

图 4　染色体适应值的计算过程

除此之外,针对软件缺陷预测问题,在种群最优染色体更新的时候,一方面考虑了染色体的适应值(即适应值越小越好),另一方面也考虑了染色体对应的特征子集的规模,即如果两个染色体具有相同的适应值,则更倾向于选择特征子集规模更小的染色体。

2　实证研究

为了评估 SBFS 的有效性,并为有效使用 SBFS 提供指导,本文在实证研究中,重点关注如下两个实验问题:

RQ1:本文提出的 SBFS 方法是否优于其他经典的基准方法?

RQ2:SBFS 方法在第一阶段考虑了类不平衡学习方法 SMOTE,是否可以提高 SBFS 方法的性能?

2.1　实证研究中采用的数据集

本文重点考虑了软件缺陷预测研究中经常使用的 NASA 数据集[7,8,12~14]。Shepperd 等人[17]发现:之前研究人员广泛使用的 NASA 数据集的内部存在一些噪声,并对已有实证研究中所得出的结论的有效性产生了一定的影响。因此他们对 NASA 数据集进行了一些清理,并显著提升了数据集的质量。本文选择了他们清理后的 NASA 数据集。NASA 项目的数据集可以从 PROMISE 数据库(http://openscience.us/repo/)中获取并且被广泛用于软件缺陷预测研究领域。NASA 数据集考虑的度量元与代码规模、McCabe 环路复杂度以及 Halstead 复杂度有关。表 1 总结了 NASA 数据集中不同数据的统计特征,包括数据名称、特征数、模块数、缺陷模块数以及缺陷模块所占的比例。不难看出这些数据的特征数为 36 ~ 39,包含的模块数为 125 ~ 1 988。同时数据集存在明显的类不平衡问题,其缺陷模块所占比例为 0.02 ~ 0.35。

表 1　NASA 数据集的特征统计

名称	特征数	模块数	缺陷模块数	缺陷模块所占比例
CM1	37	327	42	0.13
KC3	39	194	36	0.19
MC1	38	1 988	46	0.02
MC2	39	125	44	0.35
MW1	37	253	27	0.11
PC1	37	705	61	0.09
PC2	36	745	16	0.02
PC3	37	1 077	134	0.12
PC4	37	1 287	177	0.14
PC5	38	1 711	471	0.28

2.2　性能评测指标和评估流程

本文选择 AUC(area under ROC curve)作为模型的性能评测指标。由于查准率、查全率、*F*-measure 等指标需要一个阈值来区分正类和负类,而 AUC 值在计算时不需要阈值的设定,所以越来越多的研究人员倾向于选择 AUC 值作为性能评测指标。AUC 评测指标通过计算 ROC(receiver operating characteristic)曲线下的面积来获取。其取值越大,表示模型的性能越好。ROC 曲线的全称是受试者工作特征线。本文根据分类器的预测结果对实例进行排序,按此顺序逐个把实例作为正类进行预测,每次计算出两个值(真正例率和假正例率),分别以它们作为横坐标和纵坐标来作图,就可以得到 ROC 曲线。

本文在性能评估时,主要考虑的是五次 2 折交叉验证。其中 2 折交叉验证是指将数据集 D 划分为两个大小相近的互斥子集 D_1 和 D_2。其中 $D_1 \cup D_2 = D, D_1 \cap D_2 = \varnothing$。$D_1$ 和 D_2 在划分时,借助分层采样来确保这两个子集的数据分别与原有数据集 D 的分布保持一致。但将数据集划分为两个子集存在很多划分方式,为了减少因样本划分不同而引入的差别,本文将 2 折交叉验证重复五次。其具体流程如图 5 所示。

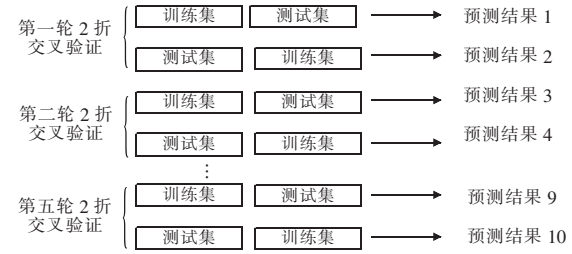

图 5　五次 2 折交叉验证的流程图

2.3　实验设计

本文主要考虑三种不同的分类器:决策树法(简称 J48)、朴素贝叶斯法(简称 NB)和 Logistic 回归法(简称 LR)。这三种方法也是目前软件缺陷预测研究中经常采用的分类方法[7~14]。本文基于 Weka 软件包实现了上述分类器,并将这些参数取值设置为缺省值。本文重点考虑了三种基准方法:a)基于前向选择策略的包裹式特征选择方法(简称 FW),该方法从空集开始,每次尝试选出一个特征,并加入到特征子集中,当新的特征子集的预测性能不如上一轮的特征子集的预测性能时,添加过程结束;b)基于后向选择策略的包裹式特征选择方法(简称 BW),该方法从考虑所有特征开始,每次尝试移除一个特征,当新的特征子集的预测性能不如上一轮特征子集的预测性能时,移除过程结束;c)不进行特征选择(简称 Origin),该方法不作特征选择,即保留已有特征。其中前两种方法在 Me-

nzies 等人[7]和 Song 等人[8]提出的通用缺陷预测框架中均使用过,因此具有一定的代表性。

SBFS 和基准方法均基于 Weka 软件包实现。SBFS 第一阶段中的 SMOTE 方法的参数取值设置如下:最近邻数设置为 5,需要虚拟创造出的少数类实例的比例为 100%(假设原有数据集中少数类实例的个数是 10,则基于上述参数,SMOTE 方法会额外虚拟创造出 10 个少数类实例)。SBFS 第二阶段中的遗传算法的参数取值设置如下:种群规模为 20,最大迭代次数为 20,变异概率为 0.7,交叉概率为 0.1。

2.4 实证研究的结果分析

2.4.1 对 RQ1 的分析

本文考虑的是五次 2 折交叉验证。在每一折的运行时,考虑到遗传算法内存在的随机因素,因此会独立运行 SBFS 方法五次,并最终选择其中的最优值。因为在 SBSE 研究领域中,研究人员[15]一般认为若能够获得更好的缺陷预测性能,多花一些时间在特征选择阶段上是值得的。最终基于 J48 分类器的结果如表 2 所示,基于 NB 分类器的结果如表 3 所示,基于 LR 分类器的结果如表 4 所示。由于每个数据集在不同方法下会有 10 个不同的执行结果(如图 5 所示),本文取其中的中位数并列在表中。本文对其中的最优结果进行了加粗表示。

表 2 基于 J48 分类器的 AUC 值

数据集	FW	BW	Origin	SBFS
CM1	0.564	0.577	0.559	0.677
KC3	0.629	0.6595	0.641	0.683 5
MC1	0.704	0.68	0.613	0.729
MC2	0.616	0.565	0.597 5	0.694
MW1	0.63	0.538	0.595	0.633 5
PC1	0.735 5	0.706 5	0.646	0.746 5
PC2	0.606	0.611	0.602 5	0.731
PC3	0.709 5	0.654 5	0.656 5	0.735 7
PC4	0.851	0.817	0.786 5	0.867 5
PC5	0.705	0.666	0.652	0.712

基于表 2 ~4,在大部分情况下,本文提出的 SBFS 方法要优于前向搜索方法 FW、后向搜索方法 BW 和未进行特征选择的方法 Origin。随后对上述结果进行 Win/Draw/Loss(简称 W/D/L)分析,其中比较方法是本文提出的 SBFS 法,而被比较的方法是 FW、BW 和 Origin。以表 2 为例,SBFS 方法与 FW 进行比较时,W/D/L 值为 10/0/0,其含义为在 10 个数据集上,若取中位数,SBFS 方法均优于 FW。

表 3 基于 NB 分类器的 AUC 值

数据集	FW	BW	Origin	SBFS
CM1	0.6235	0.633	0.6685	0.662
KC3	0.697	0.707	0.732	0.732
MC1	0.777	0.7365	0.7055	0.755
MC2	0.686	0.729	0.727	0.761
MW1	0.709	0.701	0.716	0.741
PC1	0.812	0.8065	0.7885	0.812
PC2	0.7215	0.716	0.7355	0.7395
PC3	0.7865	0.8005	0.7445	0.8015
PC4	0.858	0.866	0.812	0.863
PC5	0.72	0.715	0.706	0.719

表 4 基于 LR 的 AUC 值

数据集	FW	BW	Origin	SBFS
CM1	0.6815	0.655	0.659	0.7105
KC3	0.677	0.5965	0.5865	0.6785
MC1	0.754	0.6945	0.664	0.773
MC2	0.701	0.703	0.708	0.7455
MW1	0.6475	0.552	0.5195	0.6555
PC1	0.8545	0.8095	0.829	0.8525
PC2	0.772	0.686	0.7015	0.7925
PC3	0.806	0.7955	0.788	0.8195
PC4	0.877	0.895	0.8935	0.901
PC5	0.733	0.7415	0.738	0.7445

基于 W/D/L 分析,笔者发现:a)若以 J48 分类器作为建模方法,本文提出的 SBFS 均要优于 FW、BW 以及 Origin,其 W/D/L 值依次为 10/0/0,10/0/0 和 10/0/0;b)若以 NB 分类器作为建模方法,本文提出的 SBFS 在绝大部分情况下要优于 FW、BW 以及 Origin,其 W/D/L 值依次为 7/1/2,9/0/1 和 8/1/1;c)若以 LR 分类器作为建模方法,本文提出的 SBFS 在绝大部分情况下要优于 FW,均要优于 BW 和 Origin,其 W/D/L 值依次为 9/0/1,10/0/0 和 10/0/0。

接下来,本文基于所有数据集,将 SBFS 与 FW、BW 和 Origin 进行比较。给定分类器和需要比较的某个基准方法,总共有 100 个执行结果。因为在每个数据集上均会采用五次 2 折交叉验证,因此会产生 10 个执行结果,同时实证研究共采用了 10 组数据。最终汇总结果如表 5 所示。表 5 中前三列是基于不同分类器,SBFS 与不同基准方法的 W/D/L 值。例如若选择 J48 分类器,则与 FW 比较时,有 72 次可以取得更好的预测性能。表 5 中的最后一列是汇总结果,其含义是:在 90% 的情况下,SBFS 的性能不差于 Origin。在 82.3% 的情况下,SBFS 的性能不差于 BW。在 69.3% 的情况下,SBFS 的性能不差于 FW。

表 5 基于不同分类器,SBFS 与基准方法的 W/D/L 分析

方法比较	基于 J48	基于 NB	基于 LR	汇总
Vs FW	72/0/28	63/2/35	71/0/29	206/2/92
Vs BW	82/1/17	72/2/26	89/1/10	243/4/53
Vs Origin	88/2/10	86/0/14	94/0/6	268/2/30

2.4.2 对 RQ2 的分析

在 RQ2 的分析中,本文重点分析 SBFS 框架中,在第一阶段考虑 SMOTE 方法和不考虑 SMOTE 方法对结果的影响。其基于每个数据集的分析结果如表 6 所示。结果表明:a)若基于 J48 分类器,考虑 SMOTE 的 SBFS 方法在 71% 的情况下,要优于未考虑 SMOTE 的 SBFS 方法;b)若基于 NB 分类器,考虑 SMOTE 的 SBFS 方法在 50% 的情况下,要优于或等于未考虑 SMOTE 的 SBFS 方法;c)若基于 LR 分类器,考虑 SMOTE 的 SBFS 方法在 43% 的情况下,要优于或等于未考虑 SMOTE 的 SBFS 方法。

表 6 考虑 SMOTE 方法与未考虑 SMOTE 方法的 W/D/L 分析

数据集	Vs 未考虑 SMOTE (基于 J48)	Vs 未考虑 SMOTE (基于 NB)	Vs 未考虑 SMOTE (基于 LR)
CM1	6/0/4	5/0/5	3/0/7
KC3	8/0/2	7/0/3	5/0/5
MC1	10/0/0	4/1/5	4/0/6
MC2	9/0/1	4/1/5	7/0/3
MW1	6/0/4	6/0/4	4/0/6
PC1	3/0/7	4/0/6	5/0/5
PC2	9/0/1	5/0/5	2/0/8
PC3	7/0/3	4/0/6	6/0/4
PC4	5/0/5	6/0/4	5/0/5
PC5	8/0/2	2/1/7	2/0/8
汇总	71/0/29	47/3/50	43/0/57

基于本文的实证研究,在使用 SBFS 时,若基于 J48 分类器,通过考虑 SMOTE 方法,可以在大部分情况下,进一步提升模型的性能。而若基于 NB 或 LR 分类器,则通过考虑 SMOTE 方法,则仅在一半情况下可以提升模型的性能。在下一步研究工作中,将继续考察更多的类不平衡学习方法(如随机欠采样、随机过采样、代价敏感的学习方法)对 SBFS 框架的影响。

3 结束语

本文提出并实现了一种基于搜索的包裹式缺陷预测特征选择框架 SBFS,基于 NASA 数据集上的实证研究,初步验证了 SBFS 的有效性,并为有效使用 SBFS 提供了指导。该研究工作仍存在很多值得进一步关注的研究点,主要包括:a)在 SBFS 的实现中,可以进一步尝试使用其他元启发式搜索算法(如粒子群优化算法、蚁群算法等),并验证其性能;b)本文主要考虑了 NASA 数据集,在下一步工作中还需要考虑更多数据集,来验证论文所得到的结论是否具有一般性。除此之外,还需要考虑其他分类方法(例如支持向量机、随机森林 (下转第 页)

用上均优于其他方法,这些在第 4 章的实现中均有说明。但是本文实验仍有不足:本文采用 Siemens 经典程序集进行实验已初步证明实验的有效性,但是实验中的程序都是单错误版本,因此在后续工作中将进一步加入不同缺陷数量的版本,同时优化 Bpfc 算法,使定位效果更加精确。

参考文献:

[1] Weyuker E J. The cost of data flow testing: an empirical study[J]. IEEE Trans on Software Engineering, 1990, 16(2): 121-128.

[2] Harrold M J, Soffa M L. Efficient computation of interprocedural definition-use chains[J]. ACM Trans on Programming Languages and Systems, 1994, 16(2): 175-204.

[3] Santelices R, Harrold M J. Efficiently monitoring data-flow test coverage[C]//Proc of the 22nd IEEE/ACM on Automated Software Engineering. 2007: 343-352.

[4] Wong W E, Qi Yu, Zhao Lei, *et al*. Effective fault localization using code coverage[C]//Proc of International Computer Software & Applications Conference. [S. l.]: IEEE Press, 2007: 449-456.

[5] Jones J A, Harrold M J. Empirical evaluation of the tarantula automatic fault-localization technique[C]//Proc of IEEE/ACM International Conference on Automated Software Engineering. 2005: 273-282.

[6] 陈翔,鞠小林,文万志,等. 基于程序频谱的动态缺陷定位方法研究[J]. 软件学报, 2015, 26(2): 390-412.

[7] Renieres M, Reiss S P. Fault localization with nearest neighbor queries[C]//Proc of the 18th IEEE International Conference on Automated Software Engineering. [S. l.]: IEEE Press, 2003: 30-39.

[8] Wang Tao, Roychoudhury A. Automated path generation for software fault localization[C]//Proc of IEEE/ACM International Conference on Automated Software Engineering. 2005: 347-351.

[9] Yousefi A, Wassyng A. A call graph mining and matching based defect localization technique[C]//Proc of the 6th IEEE International Conference on Software Testing, Verification and Validation Workshops. [S. l.]: IEEE Press, 2013: 86-95.

[10] Akbar M A, Lee J A. Self-repairing adder using fault localization[J]. Microelectronics Reliability, 2014, 54(6): 1443-1451.

[11] 丁晖,陈林,钱巨,等. 一种基于信息量的缺陷定位方法[J]. 软件学报, 2013, 24(7): 1484-1494.

[12] Liu Chao, Yan Xifeng, Fei Long, *et al*. SOBER(statistical model-based bug localization)[J]. Foundations of Software Engineering, 2005, 30(5): 286-295.

[13] Liblit B, Aiken A, Zheng A X, *et al*. Bug isolation via remote program sampling[J]. Programming Language Design and Implementation, 2003, 38(5): 141-154.

[14] 李伟,郑征,郝鹏,等. 基于谓词执行序列的软件缺陷定位算法[J]. 计算机学报, 2013, 36(12): 2406-2419.

[15] Ye Junmin, Wang Junjie, Dong Wei, *et al*. Alogrithm for generation of unconstrained edges based on the decision-to-decision graph[J]. Computer Science, 2009, 36(2): 296-298.

[16] Yu Jian, Cheng Qiansheng, Huang Houkuan, *et al*. Analysis of the weighting exponent in the FCM[J]. IEEE Trans on Systems, Man, and Cybernetics, 2004, 34(1): 634-639.

[17] Reiss S P, Renieris M. Encoding program executions[C]//Proc of the 23rd International Conference on Software Engineering. [S. l.]: IEEE Computer Society, 2001: 221-230.

[18] 肖满生,阳娣兰,张居武,等. 基于模糊相关度的模糊 C 均值聚类加权指数研究[J]. 计算机应用, 2010, 30(12): 3388-3390.

(上接第　　页)等),来验证方法的有效性。(c)本文主要考虑的是包裹式特征选择方法,虽然该方法的性能较好,但也存在计算开销大的问题,下一步需要进一步研究基于搜索的过滤式缺陷预测特征选择框架。

参考文献:

[1] 陈翔,顾庆,刘望舒,等. 静态软件缺陷预测方法研究[J]. 软件学报, 2016, 27(1): 1-25.

[2] 王青,伍书剑,李明树. 软件缺陷预测技术[J]. 软件学报, 2008, 19(7): 1565-1580.

[3] Hall T, Beecham S, Bowes D, *et al*. A systematic literature review on fault prediction performance in software engineering[J]. IEEE Trans on Software Engineering, 2012, 38(6): 1276-1304.

[4] 郁抒思,周水庚,关佶红. 软件工程数据挖掘研究进展[J]. 计算机科学与探索, 2012, 6(1): 1-31.

[5] Molina L C, Belanche L, Nebot A. Feature selection algorithms: a survey and experimental evaluation[C]//Proc of International Conference on Data Mining. 2002: 306-313.

[6] Dash M, Liu H. Feature selection for classification[J]. Intelligent Data Analysis, 1997, 1(3): 131-156.

[7] Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors[J]. IEEE Trans on Software Engineering, 2007, 32(11): 1-12.

[8] Song Qinbao, Jia Zihan, Shepperd M, *et al*. A general software defect-proneness prediction framework[J]. IEEE Trans on Software Engineering, 2011, 37(3): 356-370.

[9] Gao Kehan, Khoshgoftaar T, Wang Huanjing, *et al*. Choosing software metrics for defect prediction: an investigation on feature selection techniques[J]. Software-Practice and Experience, 2011, 41(5): 579-606.

[10] Kim S, Whitehead Jr, Zhang Yi. Classifying software changes: clean or buggy[J]. IEEE Trans on Software Engineering, 2008, 34(2): 181-196.

[11] Shivaji S, Whitehead Jr, Akella R, *et al*. Reducing features to improve code change-based bug prediction[J]. IEEE Trans on Software Engineering, 2013, 39(4): 552-569.

[12] Liu Shulong, Chen Xiang, Liu Wangshu, *et al*. FECAR: a feature selection framework for software defect prediction[C]//Proc of the Annual International Computers, Software and Applications Conference. 2014: 426-435.

[13] Liu Wangshu, Liu Shulong, Gu Qing, *et al*. Empirical studies of a two-stage data preprocessing approach for software fault prediction[J]. IEEE Trans on Reliability, 2016, 65(1): 38-53.

[14] Chen Jiaqiang, Liu Shulong, Liu Wangshu, *et al*. A two-stage data preprocessing approach for software fault prediction[C]//Proc of the International Conference on Software Security and Reliability. 2014: 20-29.

[15] Harman M, Mansouri S A, Zhang Yuanyuan. Search-based software engineering: Trends, techniques and applications[J]. ACM Computing Survey, 2012, 45(1): 1-61.

[16] He Haibo, Garcia E A. Learning from imbalanced data[J]. IEEE Trans on Knowledge and Data Engineering, 2009, 21(9): 1263-1284.

[17] Shepperd M, Song Qinbao, Sun Zhongbin, *et al*. Data quality: some comments on the NASA software defect datasets[J]. IEEE Trans on Software Engineering, 2013, 25(9): 1208-1215.

[18] Chawla N V, Bowyer K W, Hall L O, *et al*. SMOTE: synthetic minority over-sampling technique[J]. Journal of Artificial Intelligence Research, 2002, 16(1): 321-357.