融合加权代码异味强度因子软件缺陷预测模型

陈镜如[1]，黄子杰[2]，高建华[1]

（1. 上海师范大学 计算机科学与技术系，上海 200234；
2. 华东理工大学 计算机科学与工程系，上海 200237）

摘　要：现有代码异味强度检测方法未考虑度量对代码异味的影响力度。基于已有代码异味强度检测方法，分析度量对代码异味的影响力度，改进代码异味强度检测方法。应用随机森林得到影响代码异味的度量的特征重要性，将各度量的评估值作为各度量的权值，检测加权代码异味强度的值。评估检测策略时，对各个度量的相关性进行分析，发现用于检测的度量之间缺乏相关性，符合从不同角度衡量代码异味的思想。当加权代码异味强度作为缺陷模型预测因子时，可提高缺陷预测模型约2%的F-Measure值。

关键词：代码异味检测；代码异味强度；特征重要性；相关性分析；缺陷预测；开源软件；实证软件工程

中图法分类号：TP311　　**文献标识号**：A　　**文章编号**：1000-7024（2022）12-3356-09

doi：10.16208/j.issn1000-7024.2022.12.008

Software defect prediction model integrating weighted code smell intensity

CHEN Jing-ru[1], HUANG Zi-jie[2], GAO Jian-hua[1]

(1. Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234, China;
2. Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China)

Abstract: The existing code smell intensity detection methods fail to consider the impact of measurement on code smell. Based on the existing code smell intensity detection methods, the impact of measurement on code smells was analyzed, and the code smell intensity detection method was improved. The characteristic importance of metrics affecting code smell was calculated using random forest. The evaluation value of each measurement was used as the weight of each measurement to detect the value of weighted code smell intensity. It is found that there is a lack of correlation between the measures used for detections, which is in line with the idea of measuring code smell from different angles. When the weighted code smell intensity is used as the predictor of the defect model, the F-measure value of the defect prediction model can be increased by about 2%.

Key words: code smell detection; code smell intensity; feature importance; correlation analysis; defect prediction; open-source software; empirical software engineering

0　引　言

研究表明[1]，分析与源代码质量相关的结构度量来预测缺陷及其发生倾向，但使用单一的结构度量作为特征没有取得良好的预测效果。缺陷预测是一种有监督学习的任务，它将特征因子作为一组独立变量，将类的缺陷倾向性作为因变量，训练机器学习分类器，并给出预测结果。文献[2]指出在添加代码异味强度作为软件缺陷的预测因子时，可以为模型带来信息增益，并提高缺陷模型的预测能力。

代码异味已经成为软件系统在维护软件质量方面造成复杂性的一个标志[3]。代码异味用于识别要重构的代码部分，以提高软件的整体可维护性[4]。代码异味与软件缺陷成正相关，并能对缺陷检测模型的性能产生积极影响[5]。

收稿日期：2021-06-23；**修订日期**：2022-11-09

基金项目：国家自然科学基金项目（61672355）

作者简介：陈镜如（1996-），女，山东临沂人，硕士研究生，研究方向为软件测试技术；黄子杰（1994-），男，上海人，博士研究生，研究方向为软件工程、软件可靠性和软件重构；高建华（1963-），男，上海人，博士，教授，CCF高级会员，研究方向为软件重构、软件测试、可信软件、可靠性模型设计。E-mail：jrchen9615@163.com

使用文献[6]中提及的方法，将使用改进代码异味强度检测方法作为预测因子添加到预测模型中，分析预测模型性能。

机器学习技术在预测代码异味方面有很大的潜力，有助于检测这些异味并提高软件质量[7]。当进行特征选择和应用分类模型时，具有随机森林分类器的BFS森林分类器给出了最好的性能[8]。

Palomba等[9]认为与复杂/长源代码相关的异味通常被开发者视为重要威胁。代码异味一旦被引入，就难以被移除，因此应尽快将其重构[10]，否则将会在代码实现和软件架构的粒度上引发更严重的问题[11]，提升系统的易错性，进而导致软件缺陷[12,13]。研究表明，可以通过组合结构度量和阈值来检测代码异味[14]，并进一步衡量代码异味强度表示其严重程度。

1 相关术语

1.1 本文涉及的代码异味

本文主要考虑以下6种代码异味：God Class、Data Class、Brain Method、Shotgun Surgery、Dispersed Coupling和Message Chain，因为这6种代码异味不仅是最常见的异味[15]，而且与软件缺陷的相关性最强[5]。6种代码异味具体介绍如下。

God Class：类承担太多不同的职责，使得代码耦合性增强，内聚性降低。

Data Class：类的函数没有行为，仅用于存取类的属性。

Brain Method：集中实现多个函数功能的过大方法。

Shotgun Surgery：每一次子方法发生变化，都会触发其它几个类许多的变化的一个类。

Dispersed Coupling：与其它的类发生太多耦合的类。

Message Chains：过度耦合的函数调用链，在执行某个功能时，需要调用连续的多个方法。

上述代码异味的检测策略将在2.1节详述。

1.2 随机森林

随机森林是一种基于Bagging的集成学习方法，其核心思想是将Bootstrap方法应用到Cart算法中。

本文用VIM表示变量重要性的得分，用GI来表示Gini指数。若有m个特征X_1，X_2，……X_m，本文需要计算出每个特征X_j的Gini指数评分$VIM_j^{(Gini)}$，即第j个特征在RF所有决策树中节点分裂不纯度的平均改变量。

Gini指数的计算如式（1）所示

$$GI_m = 1 - \sum_{k=1}^{|K|} p_{mk}^2 \tag{1}$$

式中：K表示K个类别，p_{mk}表示节点m中类别k所占的比例。

特征X_j在节点m的重要性，即节点m分枝前后的Gini指数变化量如式（2）所示

$$VIM_{jm}^{Gini} = GI_m - GI_l - GI_r \tag{2}$$

式中：GI_l和GI_r分别表示分枝后两个新节点的Gini指数。

特征X_j在决策树i中出现的节点在集合M中，那么X_j在第i棵树的重要性，如式（3）所示

$$VIM_{ij}^{(Gini)} = \sum_{m \in M} VIM_{jm}^{(Gini)} \tag{3}$$

假设RF有n棵树，则其$VIM_j^{(Gini)}$如式（4）所示

$$VIM_j^{(Gini)} = \sum_{i=1}^{n} VIM_{ij}^{(Gini)} \tag{4}$$

为了能够更方便的对特征重要性评分作比较，对其进行归一化处理，如式（5）所示

$$VIM_j = \frac{VIM_j}{\sum_{i=1}^{c} VIM_i} \tag{5}$$

2 基于随机森林加权的代码异味强度

本文提出了一种基于随机森林的代码异味强度检测方法，对Fontana等[15]提出的代码异味强度计算公式进行了改进，给每个度量赋予相应的权值。本文提出的算法整体框架流程如图1所示：

（1）获取软件系统源文件。

（2）使用JCodeOdor进行代码异味强度检测，导出数据。

（3）不同的检测策略对度量的选择各有侧重，且度量的区分度不同，依照JCodeOdor中检测策略中的度量整理数据，无需进行特征提取。

（4）将检测策略中的各个度量作为特征，代码异味强度作为标签，使用随机森林评估特征重要度。

（5）将特征重要度作为各个特征的权值，计算加权代码异味强度。本文在计算代码异味强度时使用检测策略中的度量。

本文提出的加权代码异味强度，体现了各个度量对异味强度的不同影响，求得的代码异味强度更为精确。

2.1 验证代码异味检测策略

JCodeOdor是一个代码异味检测器，它依赖于度量和阈值组成的检测策略来检测代码异味强度。其度量详情见文献[15]，检测策略见表1。

不同的检测策略对度量的选择各有侧重，且度量的区分度不同，所以不同的检测规则结果存在一定的差异[16]。不仅如此，度量指标之间存在的相关性[17]可能会影响规则的检测效果。

为了验证规则中的度量对本文数据集的适切程度，本文衡量其相关性。若规则中存在高度相关的度量，说明规则可能会部分失效，且这些度量会造成多重共线性问题(multicollinearity)。通过因变量R，即Pearson系数的高低，可以计算自变量x_1，x_2，……x_n，$n \in [1,20]$之间的

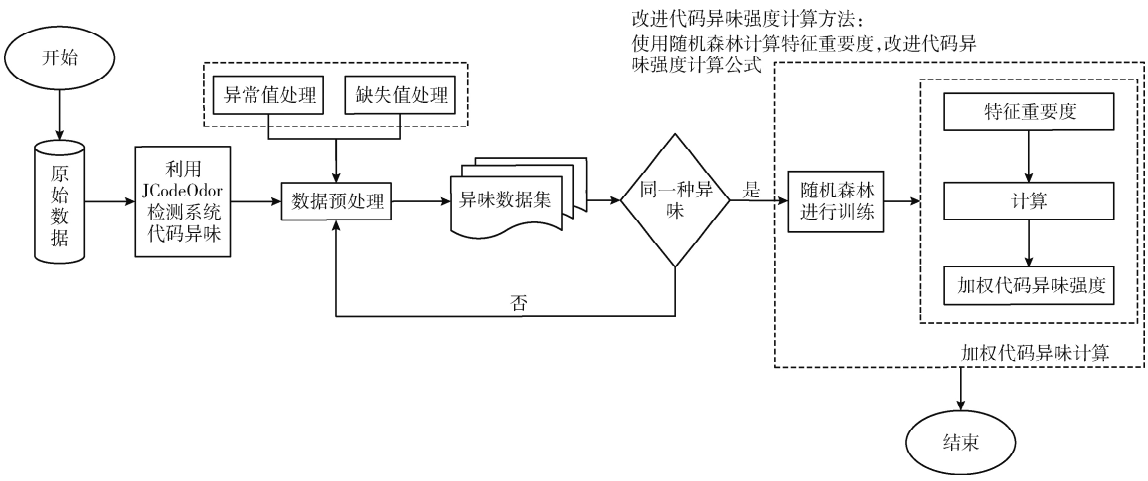

图 1　加权强度因子总体流程

表 1　代码异味检测策略

代码异味	检测策略：LABEL(n)→LABEL has value n for smell
God Class	LOCNAMM≥HIGH(176) ∧ WMCNAMM≥MEAN(22) ∧ NOMNAMM≥HIGH(18) ∧ TCC≤LOW(0.33) ∧ ATFD≥MEAN(6)
Data Class	WMCNAMM≤LOW(14) ∧ WOC≤LOW(0.33) ∧ NOAM≥MEAN(4) ∧ NOPA≥MEAN(3)
Brain Method	(LOC≥HIGH(33) ∧ CYCLO≥HIGH(7) ∧ MAXNESTING≥HIGH(6) ∨ (NOLV≥MEAN(6) ∧ ATLD≥MEAN(5))
Shotgun Surgery	CC≥HIGH(5) ∧ CM≥HIGH(6) ∧ FANOUT≥LOW(3)
Dispersed Coupling	CINT≥HIGH(8) ∧ CDISP≥HIGH(0.66)
Message Chains	MaMCL≥MEAN(3) ∨ (NMCS≥MEAN(3) ∧ MeMCL≥LOW(2))

相关性。两个变量之间的 Pearson 相关系数，用两个变量之间的协方差和标准差的商来定义。

样本 Pearson 相关系数如式（6）所示

$$r=\frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\overline{Y})^2}} \tag{6}$$

r 的相关程度判断标准如下：[0.00,0.20）极低相关，[0.02,0.40）低相关，[0.40,0.60）中等相关，[0.60,0.80）高度相关，[0.80,1.0）极高相关。

2.2　数据预处理

JCodeOdor 在检测代码异味强度时，对检测到的值进行标准化处理，将数值缩放到 1 到 10 之间便于比较，如式（7）所示

$$z=\left[\frac{x-\min(x)}{\max(x)-\min(x)}\right]\cdot 10 \tag{7}$$

式中：min 和 max 是在数据集[18]中包含的指标的统计分布中的最小值和最大值。JCodeOdor 利用 1 到 10 范围内的数值和提供强度语义描述的标签来呈现代码异味强度，如图 2 所示。

图 2　异味强度标签和范围

本文选用 JCodeOdor 作为检测工具来衡量代码异味强度因子，该工具基于度量进行检测。6 种异味的阈值见表 2[15]。

JCodeOdor 在检测异味时使用检测策略中所有的度量并且对这些度量赋予相同的权重，如式（8）所示

$$Int=\frac{\sum_{i=1}^{n}Metric_i}{n} \tag{8}$$

式中：Int 表示该代码异味的代码异味的强度，$Metric_i$ 表示应用 JCodeOdor 测得的第 i 个度量的值，n 为代码异味检测策略中所用到的度量的个数。

经对比各种分析计算方法，发现因子分析和主成分分析利用了数据信息的浓缩原理，利用方差解释率进行权值运算，可能会忽略部分因子，而随机森林计算权值可以解决这一问题。根据随机森林评估特征重要性来改进代码异味

表 2　6 种异味的度量及其默认阈值

	度量	非常低	低	平均	高	非常高
God Class	LOCNAMM	26	38	78	176	393
	WMCNAMM	11	14	22	41	81
	NOMNAMM	7	9	13	21	30
	TCC	0.25	0.33	0.5	0.66	0.75
	ATFD	2	4	6	11	21
Data Class	WMCNAMM	11	14	21	40	81
	WOC	0.25	0.33	0.5	0.66	0.75
	NOPA	1	2	3	5	12
	NOAM	2	3	4	7	13
Brain Method	LOC	11	13	19	33	59
	CYCLO	3	4	5	7	13
	MAXNESTING	3	4	5	6	7
	NOLV	4	5	6	8	12
	ATLD	3	4	5	6	11
Shotgun Surgery	CC	2	3	4	5	10
	CM	2	3	4	6	13
	FANOUT	2	3	4	5	6
Dispersed Coupling	CINT	3	4	5	8	12
	CDISP	0.25	0.33	0.5	0.66	0.75
Message Chains	MaMCL	2	3	3	4	7
	MeMCL	2	2	3	4	5
	NMCS	1	2	3	4	5

强度公式，加权代码异味强度的计算公式，如式（9）所示

$$WInt = \sum_{i=1}^{n} a_i \times Metric_i \tag{9}$$

式中：$WInt$ 表示该代码异味的加权代码异味的强度，a_i 为应用随机森林计算得出的特征重要度，$Metric_i$ 表示应用 JCodeOdor 测得的第 i 个度量的值。

以 Heritrix1.14.4 为例，本文首先利用 JCodeOdor 对其系统进行检测，测得该系统中只有 4 种代码异味，即 Brain Method、Disperse Coupling、Message Chain、Shotgun Surgery。部分代码异味度量值和代码异味强度的值（Intensity）的结果见表 3。

2.3　权值计算

根据上文中检测出的数据，以 Heritrix v1.14.4 中的 setAttribute（CrawlerSettings，Attribute）方法为例，它检测出具有 Shotgun Surgery 异味，其具有以下度量值：CC：8；CM：10；FANOUT：6。这些度量值满足 Shotgun Surgery 检测策略中定义的约束条件：CC $\geqslant$ HIGH(5) $\wedge$ CM $\geqslant$ HIGH(6) $\wedge$ FANOUT $\geqslant$ LOW(3)。

将得到的实际度量值与表 2 中这 5 个点的阈值进行比较，来定义检测规则中每个度量的强度标签，具体规则见式（10）

$$\arg \max_{x \in IntensityValues} \{MetricValue \geqslant x\} \tag{10}$$

由上文的强度阈值和划分出的强度标签和范围，如表 2 和图 2 所示，可知，在 Shotgun Surgery 代码异味检测策略中的度量与强度标签相对应如下：CC＝高，CM＝高，FANOUT＝非常高。

原代码异味强度计算公式为代码异味强度为其检测策略中各个度量强度的加和平均值，且每个强度标签与相应值范围的下限相关。

在本例中，CC 和 CM 度量取标签高即 7.75，FANOUT 度量取标签非常高即 10。将上述值带入式（8）求代码异味强度值，即式（11）所示

$$Int = \frac{\sum_{i=1}^{n} Metric_i}{n} = \frac{(7.75 + 7.75 + 10)}{3} = 8.5 \tag{11}$$

根据图 2 中的强度标签，代码异味强度因子值 8.5 与“高”强度标签相关，即在范围［7.75，10］内。

根据表 3 中的数据，应用随机森林评估特征重要度，并

表 3　Code Smell 度量及其异味值

	方法	CYCLO	MAXNESTING	LOC	ATLD	NOLV	Intensity
Brain Method 度量及异味值	processBdbLogs	13.0	6.0	51.0	1.0	12.0	8.3125
	load	18.0	6.0	85.0	6.0	10.0	7.75
	getByRegExpr	13.0	6.0	67.0	0.0	18.0	8.875
	getByRegExpr	13.0	6.0	85.0	0.0	17.0	8.875

	Method	CINT	CDISP	Intensity
Dispersed Coupling 度量及异味值	evaluate	10.0	0.8	8.875
	testFilter	8.0	0.875	6.625
	extract	9.0	0.6666	3.25

	Method	NMCS	MaMCL	MeMCL	Intensity
Message Chain 度量及异味值	kickUpdate	1.0	4.0	4.0	7.75
	initialTasks	2.0	3.0	3.0	7.75
	loadCheckpointSerialNumber	1.0	3.0	3.0	7.75
	getController	1.0	3.0	3.0	7.75

	Method	FANOUT	CM	CC	Intensity
Shotgun Surgery 度量及异味值	setAttribute(CrawlerSettings,Attribute)	6	10	8	8.5
	setAttribute	5.0	10.0	8.0	7.75
	addLabelValue	3.0	16.0	5.0	5.5

将得到的基尼系数作为度量的权值，故加权代码异味强度因子计算带入式（9），如式（12）所示

$$WInt = \sum_{i=1}^{n} a_i \times Metric_i = 0.329 \times 7.75 + 0.321 \times 7.75 + 0.35 \times 10 = 8.5 \quad (12)$$

由图 2 中的强度标签来看，加权代码异味强度因子值 8.5 与“高”强度标签相关，即在范围［7.75,10］内。

3　实　验

将加权代码异味强度作为预测因子加入到现有的缺陷预测模型构建新模型，并将新模型与原本模型进行对比，评估代码异味强度的贡献。

本文提出的预测模型框架流程如图 3 所示：

（1）确定基本预测因子，构建基本软件缺陷预测模型；

（2）确定代码异味检测过程；

1）JCodeOdor 检测软件系统源文件；

2）随机森林特征重要度评估；

3）加权代码异味强度计算；

（3）在基本缺陷预测模型的基础上加入加权代码异味强度作为预测因子，构建新模型；

（4）确定用于分类的机器学习技术；

本文在一个由 Jureczko 等[19] 定义的 20 个软件度量组成的缺陷预测模型中测试代码异味强度作为预测因子的贡献。

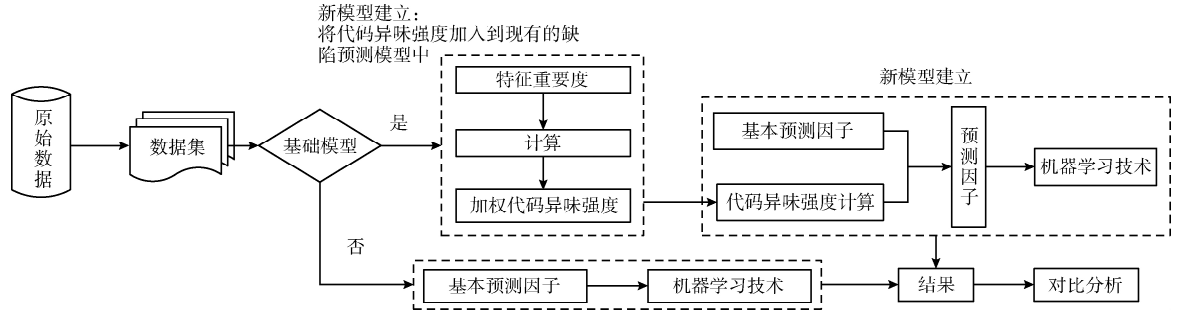

图 3　缺陷预测流程

3.1　实验环境

本文的实验在 Intel(R) Core(TM) i7-5500U、4 GB 内存的 Windows 10 环境下进行，实验的计算和分析使用 Python 完成。本文选取 PROMISE 数据集中的缺陷数据作为

实验对象，本文用到的数据集见表 4。

表 4　实验项目

项目	类	千行代码	%缺陷 CI．	%异味 CI.
Apache Ant	83	20	68	11
Apache Tomcat	858	301	6	4
Apache Lucene	338	103	59	10
Apache Velocity	229	57	15	7
Apache Xerces	588	141	74	5

本文对这 5 个软件系统进行实验，并回答以下问题：

Q1：本文使用的检测策略规则是否适用于数据集？

Q2：本文所建构的模型是否能提升缺陷预测的性能？

Q3：与 Fontana 等提出的代码异味强度因子相比，加权代码异味强度作为预测因子是否能为缺陷预测模型带来增益？可带来多少增益？

3.2　实验过程

本文使用以下 3 种预测因子组合构建新模型，分析预测因子对模型贡献的模型，以直观地显示预测因子带来的模型性能提升，这些组合包括：

（1）基本模型：

基于 Jureczko 等[19]定义的 20 个软件度量的软件缺陷预测模型；

（2）基本模型＋代码异味强度因子：

1）将 20 个软件度量和 Fontana 等提出的代码异味强度作为预测因子，对于没有异味的类，将其代码异味强度因子设为 0，构建软件缺陷预测模型；

2）将 20 个软件度量和本文提出的加权代码异味强度作为预测因子，对于没有异味的类，将其代码异味强度因子设为 0 构建软件缺陷预测模型。

本文对比了文献［2］中使用的常见分类器，包括决策树、逻辑回归和简单逻辑回归等，发现简单逻辑回归的效果最好。这一实验结果和文献［2］的结果相吻合。

3.3　实验结果与分析

为了回答 Q1，本文验证规则中的度量对数据集的适切程度，包括以下几个方面：①度量间是否存在高相关性；②度量是否包含多种衡量代码异味的角度。如代码异味常表现出的低内聚和高耦合，高复杂性等。因此，应考虑度量中是否包含内聚、耦合、复杂性和数据访问。

相关性分析是指对两个或多个具备相关性的变量元素进行分析，从而衡量两个变量因素的相关密切程度。若存在相关性较过高的度量意味着多重共线性，机器学习模型和信息增益算法可能无法有效区分共线的特征，因此它们需要被移除。本文使用 python 检测到的度量之间的相关性（Heritrix1. 14. 4）如图 4 所示，线条上的数字为它们之间的相关性。本文发现，检测策略中的度量无显著的相关性，不会造成多重共线问题。

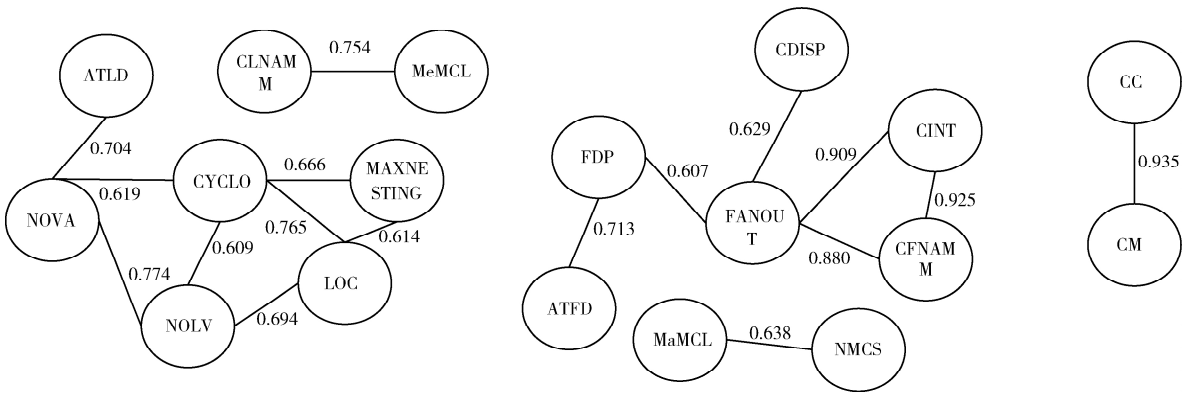

图 4　度量相关性网络

由相关性图可知，各个度量之间有着复杂且多元的关系。具有极高相关性的有｛FANOUT，CINT｝，｛CINT，CFNAMM｝，｛FANOUT，CFNAMM｝，｛CC，CM｝；具有高度相关性的有：｛ATLD，NOVA｝，｛NOVA，NOLV｝，｛CYCLO，LOC｝，｛FDP，ATFD｝；具有极低相关性的有：｛FANOUT，CM｝，｛FANOUT，CC｝。在同一异味下，检测策略之间的相关性多处于高度相关以下。本文发现，检测策略中的度量无显著的相关性，不会造成多重共线问题。

为了回答 Q2，需要评估模型的性能。本文采用 10 次 10 折交叉验证。实验结果见表 5、表 6，表 5 是原始缺陷预测模型测得有关模型各项数据，表 6 是添加代码异味强度作为预测因子和添加加权代码异味强度作为预测因子时模型的各项数据。

由表 6 可知，在基本预测模型中添加代码异味强度和加权代码异味强度作为预测因子后，能够实现高精度预测。对于 Apache Velocity，在分别添加代码异味强度和改进计算公式后求得的加权代码异味强度作为模型的预测因子后，其模型精确度较原始模型分别提高 25%和 28%。这一结果

表 5　原始模型结果

项目	模型	精度	精确度	召回率	F-值
Apache Ant	Basic	87	82	84	82
Apache Tomcat	Basic	56	9	16	12
Apache Lucene	Basic	78	75	76	75
Apache Velocity	Basic	67	18	28	22
Apache Xerces	Basic	94	93	94	93

表 6　改进模型结果

项目	模型	精度	精确度	召回率	F-值
Apache Ant	Basic+Int .	88	87	88	87
	Basic+WInt .	85	95	85	88
Apache Tomcat	Basic+Int .	79	20	31	24
	Basic+WInt .	91	89	92	90
Apache Lucene	Basic+Int .	80	80	80	80
	Basic+WInt .	80	82	80	80
Apache Velocity	Basic+Int .	92	31	46	37
	Basic+WInt .	95	97	95	97
Apache Xerces	Basic+Int .	95	96	95	95
	Basic+WInt .	96	90	96	90

意味着模型仅错误的分类了少部分数据。添加代码异味强度作为预测模型的预测因子时，可提高模型的精确度。图 5 描述了原软件缺陷预测模型与添加代码异味强度作为预测因子后的模型，模型的各项性能数据。值得注意的是，在基本模型运行良好的情况下，获得性能增量是相当困难的。尽管如此，在这种情况下，使用代码异味强度和加权代码异味强度作为预测因子时，依然能够提升模型的准确性。

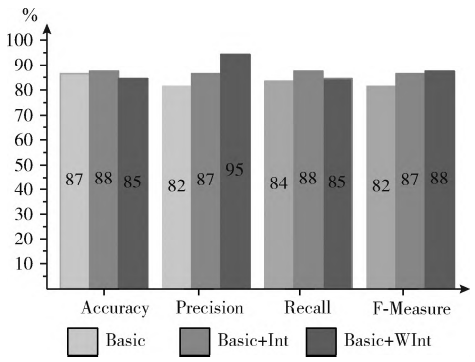

图 5　Apache Velocity 结果

在所有分析的项目中，增加代码异味强度和加权代码异味强度作为缺陷预测模型的预测因子时通常会提高基础缺陷预测模型的性能。

对于 Q2，在基本模型中分别添加代码异味强度作为预测因子和加权代码异味强度作为预测因子后，均能够提高原预测模型的分类精确度。添加代码异味强度作为预测因子模型的分类精确度可提高 25%，添加加权代码异味强度作为预测因子时比添加代码异味强度作为预测因子时，缺陷预测模型分类精确度可提升约 3%。添加加权代码异味强度作为预测因子的缺陷预测模型的 F-Measure 比添加代码强度作为预测因子的缺陷预测模型提升约 2%。

为了验证代码异味强度在预测模型中所做的贡献 Q3，本文采用信息增益算法来量化代码异味强度作为预测因子时为预测模型提供的增益。信息增益见式（13）

$$InfoGain(M,p_i) = H(M) - H(M \mid p_i) \tag{13}$$

式中：M 为预测模型，$P = \{p_1, p_2 \cdots\cdots p_n\}$ 为模型的预测因子。函数 $H(M)$ 表示包含预测因子 p_i 的模型的熵，而函数 $H(M \mid p_i)$ 表示不包括预测因子 p_i 的熵，见式（14）

$$H(M) = -\sum_{i=1}^{n} prob(p_i)\log_2 prob(p_i) \tag{14}$$

图 6、图 7 展示了系统 Apache Xerces 和系统 Apache Lucene 排名前十的度量的信息增益。

代码异味强度和改进公式后得到的加权代码异味强度在 Apache Xerces 系统中提供的信息增益最少，分别是 0.1 和 0.15。在系统 Apache Lucene 系统中，与其它度量相比，代码异味强度因子和加权代码异味强度因子带来的信息增益最多，分别是 0.50 和 0.47。在 Apache Xerces 项目中，

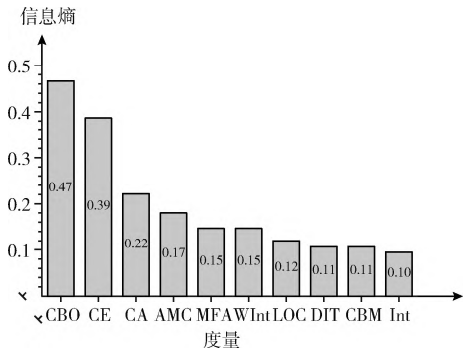

图 6 Apache Xerces 结果

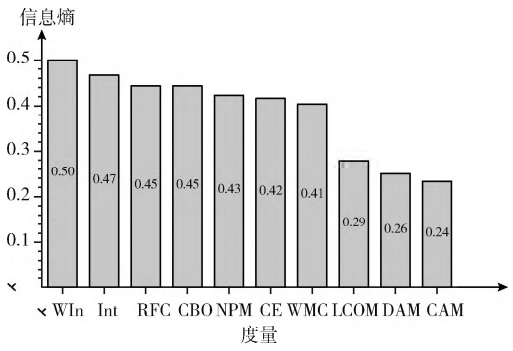

图 7 Apache Lucene 结果

基本模型的预测性能很高，获得增益较为困难，但当使用代码异味强度作为预测因子时可带来 0.1 的性能提升，而使用加权代码异味强度作为预测因子时可带来 0.11 的性能提升。在 Apache Lucene 项目中，代码异味强度作为预测因子时可带来 0.47 的性能提升，但用于计算代码异味强度的单个度量带来的性能提升却比它低。使用代码异味强度和加权代码异味强度作为预测因子相对比其衍生的单个度量具有更高的预测能力。

在所有研究项目中，代码异味强度和加权代码异味强度是模型最重要的预测因子之一。

对于 Q3，改进公式得到的加权代码异味强度提供的增益要大于代码异味强度。Fontana 等[14]提出的代码异味强度因子将影响代码异味的度量视为同等重要，不能精确地反映各个度量对代码异味强度的影响程度。本文提出的加权代码强度因子能够区分各个度量对代码异味强度的不同影响程度，测得的代码异味强度的值相对更为准确。综上所述，本文提出的加权代码异味强度在模型中作为预测因子时，相较于代码异味强度在模型中作为预测因子时，能够带来更多的增益。

4 结束语

代码异味强度可以评估代码异味的严重程度。Fontana 等[14]提出了一种计算代码异味强度的方法，它作为缺陷预测模型的预测因子时可提高模型的性能。然而，这种代码异味强度的计算方法没有区分各个度量对代码异味强度的不同影响，测得的代码异味强度的精确度有待提高。本文通过对各个度量赋予相应的权值来改善这一问题。研究发现，改进公式后测得的加权代码异味强度更为精确。将加权代码异味强度作为预测因子添加到缺陷预测模型中，与原始模型和使用未加权代码异味的强度作为预测因子的模型相比，可提高缺陷预测模型的准确性。加权代码异味强度作为预测因子比代码异味强度作为预测因子时，能够为模型带来更高的信息增益。

本文为验证规则中的度量对本文数据集的适切程度，检验了各个度量之间的相关性，发现了 {CC，CM}、{FANOUT，CINT} 这两组度量呈高度相关关系，为对原始数据集进行处理时提供了依据。JCodeOdor 中所提供的检测策略中各个度量之间并无显著相关性，不会造成多重共线问题，符合从各个角度考虑代码异味的思想。今后的工作包括：进一步研究其它代码异味的强度，提高加权代码异味强度作为预测因子的普适性；选用开源软件以外的软件种类，验证本文模型的普适性；进一步研究其它检测策略中的度量之间的相关性，并将其结合到代码异味强度中；研究代码异味之间的相关性与软件缺陷倾向性之间的关系。

参考文献：

[1] CAI Liang，FAN Yuanrui，YAN Meng，et al. Just-in-time software defect prediction：Literature review [J]. Journal of Software，2019，30 (5)：1288-1307 (in Chinese). [蔡亮，范元瑞，鄢萌，等，即时软件缺陷预测研究进展 [J]. 软件学报，2019，30 (5)：1288-1307.]

[2] Palomba F，Zanoni M，Fontana FA，et al. Toward a smell-aware bug prediction model [J]. IEEE Transactions on Software Engineering，2017，45 (2)：194-218.

[3] Kaur A. A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes [J]. Archives of Computational Methods in Engineering，2020，27 (4)：1267-1296.

[4] Lafi M，Botros JW，Kafaween H，et al. Code smells analysis mechanisms，detection issues，and effect on software maintainability [C]//Proceedings of Jordan International Joint Conference on Electrical Engineering and Information Technology. Amman，Jordan，IEEE，2019：663-666.

[5] Kim DJ，Chen THP，Yang J. The secret life of test smells-an empirical study on test smell evolution and maintenance [J]. Empirical Software Engineering，2021，26 (5)：1-47.

[6] Palomba F，Zanoni M，Fontana FA，et al. Smells like teen spirit：Improving bug prediction performance using the intensity of code smells [C] //Proceedings of IEEE International Con-

ference on Software Maintenance and Evolution. Raleigh, NC, USA, IEEE, 2016: 244-255.

[7] Mhawish MY, Gupta M. Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics [J]. Journal of Computer Science and Technology, 2020, 35 (6): 1428-1445.

[8] Khadse VM, Mahalle PN, Shinde GR. Statistical study of machine learning algorithms using parametric and non-parametric tests: A comparative analysis and recommendations [J]. International Journal of Ambient Computing and Intelligence, 2020, 11 (3): 80-105.

[9] Palomba F, Bavota G, Di Penta M, et al. On the diffuseness and the impact on maintainability of code smells: A large scale empirical investigation [J]. Empirical Software Engineering, 2018, 23 (3): 1188-1221.

[10] Paiva T, Damasceno A, Figueiredo E, et al. On the evaluation of code smells and detection tools [J]. Journal of Software Engineering Research and Development, 2017, 5 (1): 1-28.

[11] Kim D, Hong JE, Yoon I, et al. Code refactoring techniques for reducing energy consumption in embedded computing environment [J]. Cluster Computing, 2018, 21 (1): 1079-1095.

[12] Eken B, Palma F, Ayşe B, et al. An empirical study on the effect of community smells on bug prediction [J]. Software Quality Journal, 2021, 29 (1): 159-194.

[13] Abidi M, Rahman MS, Openja M, et al. Are multi-language design smells fault-prone? An empirical study [J]. ACM Transactions on Software Engineering and Methodology, 2021, 30 (3): 1-56.

[14] Catolino G, Palomba F, Fontana FA, et al. Improving change prediction models with code smell-related information [J]. Empirical Software Engineering, 2020, 25 (1): 49-95.

[15] Fontana FA, Ferme V, Zanoni M, et al. Towards a prioritization of code debt: A code smell intensity index [C] //Proceedings of 7th International Workshop on Managing Technical Debt. Bremen, Germany, IEEE, 2015: 16-24.

[16] Sobrinho EVDP, Lucia AD, Maia MDA. A systematic literature review on bad smells—5 W's: Which, when, what, who, where [J]. IEEE Transactions on Software Engineering, 2018 (99): 1.

[17] Izadkhah H, Hooshyar M. Class cohesion metrics for software engineering: A critical review [J]. Computer Science Journal of Moldova, 2017, 73 (1): 44-74.

[18] Verhaeghe B, Fuhrman C, Guerrouj L, et al. Empirical study of programming to an interface [C] //Proceedings of 34th IEEE/ACM International Conference on Automated Software Engineering. San Diego: IEEE, 2019: 847-850.

[19] Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction [C] //Proceedings of the 6th International Conference on Predictive Models in Software Engineering. Timisoara Romania, ACM, 2010: 1-10.