小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems
DOI: 10.20009/j.cnki.21-1106/TP.2021-0964
2022 年 12 月 第 12 期
Vol. 43 No. 12 2022

LightGBM 融合 CFS 的开发者感知代码异味强度预测模型研究

宇 通,高建华

(上海师范大学 计算机科学与技术系,上海 200234)
E-mail: 342017424@qq.com

摘 要: 准确地对代码异味强度进行预测可使高危险性的代码问题得到优先处理,从而减少软件项目的维护开销. 目前针对异味强度的研究较少且基于传统手工和单一算法的异味强度识别方法不能保证检测的精确性与效率. 对此,本文提出一种基于 LightGBM 融合 CFS 的开发者感知代码异味强度预测模型,该模型利用经相关性特征选择后的代码度量指标,考虑基于开发者感知的代码异味严重性,使用 LightGBM 算法,对含 4 种代码异味的实例进行异味强度预测并划分强度等级. 本文从统计角度验证了所考虑的各项代码度量指标与异味严重性之间存在强相关关系. 实验表明,本文模型在精确率、召回率、F1 值、MCC 和 AUC 等多项指标上均优于原有性能最佳的随机森林(RF)模型,其中 F1 值最高达 90.0%,最多提升 3.7%; AUC 值最高达 94.2%,最多提升 3.8%; 且相比 RF 模型预测时间可缩短 76.1%.

关 键 词: 代码异味; LightGBM; CFS; 强度预测; 机器学习
中图分类号: TP311 **文献标识码:** A **文 章 编 号:** 1000-1220(2022)12-2667-08

Research on Developer Perceived Code Smell Intensity Prediction Model Based on LightGBM and CFS

YU Tong, GAO Jian-hua
(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234, China)

Abstract: Accurate prediction of code smell intensity can give priority to high-risk code problems, so as to reduce the maintenance cost of software projects. At present, there are few researches on smell intensity, and the smell intensity recognition method based on traditional manual and single algorithm can not ensure the accuracy and efficiency of detection. To solve this problem, this paper proposes a developer perceived code smell intensity prediction model based on LightGBM and CFS. The model utilizes the code metrics after correlation feature selection, considers the severity of code smell perceived by developers, and uses LightGBM algorithm to predict the smell intensity of four code smells instances and classify the intensity grade. From the statistical point of view, this paper verifies that there is a strong correlation between the considered code metrics and the severity of smell. Experiments show that the proposed model is superior to the original random forest (RF) model with the best performance in many indexes such as accuracy, recall, F1 value, MCC and AUC. The F1 value is up to 90.0% and increases by 3.7%; The AUC value was up to 94.2% and increases by 3.8%; Compared with RF model, the prediction time can be shortened by 76.1%.

Key words: code smell; LightGBM; CFS; intensity prediction; machine learning

1 引 言

代码异味是程序源代码中任何可能表明问题更深层次的特征,通常由不合理的代码设计和不恰当的开发操作引起,即如果软件系统包含代码异味,则通常表明代码质量存在问题. 传统的代码异味识别方法带有极大的主观性,并且因开发者、开发语言、开发方法的不同而异.

相关研究人员将常见的代码异味按照其不同特点进行了分类. Fowler 等人[1] 定义了 22 种发现的代码异味,针对如何检测代码异味,Moha 等人[2] 提出 DéCOR 的检测方法以及 DETEX 一种实例化该方法的检测技术,但该方法不能检测代码异味的强度.

代码异味作为软件技术债务[3] 的直接体现,会随时间推移而在软件系统中堆积[4],因为软件在开发与使用阶段经常会因需求变更而进行扩展[5],增加新需求或修改原有需求. 在此过程中由于开发时间或其他条件限制或通常会使源代码复杂度提高[6],从而降低了软件质量与可维护性. 代码异味问题应尽快得到解决,正如 Macia[7] 在实证分析中得出的结论,如果开发早期阶段在代码中引入了异味,如不尽快消除便可能在代码和体系结构级别上导致更为严重的问题.

为了解决异味问题引入了重构[1],重构是在不改变软件外部行为的条件下,改善代码内部结构的技术. 进行代码重构是提升软件质量的常用做法.

Zhang 等人[8] 对代码异味与重构相关的研究现状做了系

收稿日期: 2021-12-02 收修改稿日期: 2022-01-23 基金项目: 国家自然科学基金项目(61672355)资助. 作者简介: 宇 通,男,1997 年生,硕士研究生,研究方向为代码异味、软件测试和机器学习; 高建华,男,1963 年生,博士,教授,CCF 会员,研究方向为软件重构、软件测试、可信软件和可靠性模型设计等.

统的阐述,他们发现现有的大多数研究集中于开发自动检测代码异味的工具或方法,而少有关于代码异味的实证性研究.

Basili 等人[9] 提出使用面向对象的一系列度量指标(CK)来预测软件项目是否存在缺陷,但并未对软件中存在的代码异味进行预测.

选择重构代码的顺序至关重要,Yamashita 等人[10] 发现并非所有代码异味的危害程度都相同. 有些异味程度较轻而不会对软件质量造成缺陷,因此应优先重构异味较严重即异味强度大的代码. 选择合适的方法对代码异味强度进行预测有利于对代码异味的处理进行优先级排序,可使异味强度大的代码得到优先处理,以节省人力物力成本.

在重构的先后顺序上,Zhang 等人[11] 提出重构优先级概念,即根据不同代码异味的严重性确定重构的先后次序,但只定性分析了软件缺陷与代码异味的关系,没有进行深入研究.

Ouni 等人[12] 利用软件开发历史信息,运用多目标优化的方法确定重构操作的最佳顺序以减少系统中的异味数量,提高了重构效率. Vidal 等人[13] 开发了 SPIRIT 工具,根据代码异味类型、过去做出的修改、系统修改可能性的评估,3 个重要度量标准对代码异味进行排序. 但此类方法均未综合考虑代码的各项度量指标.

在异味严重性检测方法上,Fontana 等人[14] 提出使用机器学习技术对代码异味严重性进行检测,运用了机器学习分类与回归模型,使用软件多种度量指标作为模型输入进行预测.

Pecorelli 等人[15] 在进行代码异味处理优先级排序时比较了 RF、LR(逻辑回归)、朴素贝叶斯等机器学习算法,并验证了 RF 具有最佳效果,但未经特征筛选过程且模型精度与运行效率都有待提高.

在特征选择方面,通常原始数据集中存在着冗余和不相关特征,不经过处理可能会导致过拟合,引发“维度灾难”[16],Hall[17] 提出使用相关性分析研究机器学习中的特征选择问题,以提高机器学习模型性能,取得了良好的效果.

Ke 等人[18] 提出了 LightGBM,一种实现 GBDT(梯度提升决策树)算法的框架,在保证高精度的同时具有更快的训练速度,也实现了更低的内存消耗. LightGBM 已被运用在金融领域如加密货币价格趋势预测[19] 以及医学领域如针对乳腺癌患者的 miRNA 分类[20],均取得了良好的效果,但目前还没有使用 LightGBM 进行代码异味强度预测的相关研究.

对此,本文提出了一种基于 LightGBM 融合 CFS 的开发者感知代码异味强度预测模型,该模型综合考虑代码产品与过程度量指标,使用经相关性分析筛选后的高相关特征,运用 LightGBM 算法对数据集中所考虑的 4 种代码异味实例进行异味强度预测.

本文的主要贡献有如下 3 个方面:

1) 在统计层面分析了选取的代码度量指标与异味强度的相关关系,并选用软件代码的多个不同层面的度量指标,综合构建代码异味强度预测模型.

2) 提出了一种基于 LightGBM 融合 CFS 的代码异味强度预测模型,结合相关性分析进行特征选择,并对 LightGBM 模型参数进行调整,根据模型评价指标对模型进行优化.

3) 基于开发者感知的代码异味数据集,分别对软件项目中 4 种不同类型代码异味进行研究并将本文提出的基于 LightGBM 融合 CFS 模型与 RF 模型进行多方面对比,验证了本文提出的模型相比 RF 模型在预测精度以及效率等各方面性能上均有较大提升.

2 相关术语

2.1 代码异味

代码异味也称设计缺陷或设计异常,是代码在次优设计下的产物,其存在严重影响了软件系统的可靠性与可维护性,同一段代码可能会受多种代码异味影响. 代码重构可以解决异味问题. 本文聚焦以下 4 种代码异味,如表 1 所示.

表 1 代码异味及其特征

Table 1 Code smells and characteristics

Code Smell	特征
Blob Class (BC)	类中代码行数过多,内聚性低,占用过多系统资源
Complex Class (CC)	类内结构复杂
Spaghetti Code (SC)	没有定义良好结构的类,通常类内有许多长方法
Shotgun Surgery (SS)	类中的修改会牵扯到其他类

选择以上 4 种类级别的代码异味进行研究,原因为:

Blob Class 是内聚性较差的类所存在的代码异味,对代码质量造成了严重影响,且根据最近的研究,此种气味对软件项目及开发人员来说最为关键.

Complex Class 表明类结构过于复杂,由此导致对这些类的测试工作较为困难,开发者通常能识别此种异味并对其严重性进行评估.

Spaghetti Code 在过去研究中被深入调查,此种异味导致开发人员对源代码理解能力的下降,从而增加了代码维护工作量,开发人员也能对其关键性进行准确评估.

Shotgun Surgery 是对某类进行修改时需要一并修改其他类,开发人员可根据触发其他类修改的数量来评估此种代码异味的强度.

以上 4 种代码异味已被证明在研究软件项目中大范围分布,其存在会对软件系统的可维护性、可理解性、可测试性造成严重的负面影响,且由于异味检测工具的限制,需在针对 Java 语言编写的程序中进行异味研究. 综上所述,本文聚焦以上 4 种类级别的代码异味,对这 4 种代码异味进行分析研究也有助于更好地解决软件开发中的潜在问题.

2.2 代码异味强度

根据异味严重性数值划分不同等级得到代码异味强度,其可用于衡量异味的严重程度. 准确地对代码异味强度进行预测可使高危险性的代码问题得到优先处理,从而在极大程

表 2 代码异味强度等级划分

Table 2 Code smell intensity classification

代码异味强度等级	严重性数值
NON-SEVERE	1 ~ 2
MEDIUM	3
SEVERE	4 ~ 5

度上减少软件项目的维护开销. 本文考虑基于开发者感知的代码异味严重性 Criticality,该指标是开发者对代码异味严重程度的直接评价,并按程度不同分为 1 ~ 5,1 为最轻微,5 为最严重. 代码异味强度相应地分为 3 个不同等级: NON-SEVERE(轻微)、MEDIUM(中度)、SEVERE(严重),以表征代码异味的严重程度,如表 2 所示.

轻微等级代表代码中存在较弱的代码异味强度,危害较弱且不易被察觉. 中度等级代表代码中已有较明显的异味,通常可被开发者直接感知. 严重等级代表代码中已存在危害性较高的代码异味,需要开发者尽快处理,防止后续发展为更严重的代码问题.

2.3　LightGBM 算法

LightGBM,全称为 Light Gradient Boosting Machine,该算法基于梯度增强的机器学习框架,利用了决策树算法. LightGBM 与其他树类算法不同,不逐行增长树,而按叶增长树,其选择它认为将产生最大损失减少的叶子. LightGBM 与 XGBoost 或其他模型所使用基于排序的决策树算法不同,LightGBM 实现了一种高度优化的基于直方图的决策树学习算法,这种算法与传统算法相比在效率与内存消耗方面具有巨大优势,LightGBM 算法采用两种新技术 GOSS(基于梯度的单边采样)与 EFB(互斥特征捆绑),使得算法运行速度更快且能保持极高的准确度[18]. LightGBM 具有许多 XGBoost 的优点,包括稀疏优化、并行训练等方面.

LightGBM 的基于梯度的单边采样方法如算法 1 所示.

算法 1. GOSS(基于梯度的单边采样)

输入: 训练数据 M、迭代次数 I、大梯度数据采样率 a、小梯度数据采样率 b、损失函数 loss、弱学习器 L.

输出: 训练完成的强学习器

1. $models \leftarrow \{\}$, $w \leftarrow \frac{1-a}{b}$
2. $topN \leftarrow len(M) * a$ /* 提取大梯度样本* /;
3. $randN \leftarrow len(M) * b$ /* 从剩余数据随机选取 $b \times 100\%$ 小梯度样本* /;
4. *for* $i = 1$ to I do /* 迭代循环直到达到规定迭代次数* /;
5. preds←models. predict(M) /* 模型预测得到样本预测值 preds* /;
6. $k \leftarrow loss(M, preds)$, $s \leftarrow \{1, 1, \cdots\}$ /* 由 preds 计算 loss,再计算得样本梯度,样本初始权重 s 均为 1* /;
7. sorted←SortedData(abs(k)) /* 由梯度绝对值降序排序得到样本索引数组 sorted* /;
8. topSet←sorted[1: topN] /* 在大梯度样本中选择 topN 数据得到索引数组 topset* /;
9. randSet←RandPick(sorted[topN: len(M)], randN) /* 从剩余数据中随机选取 b* 100% 的数据,生成小梯度样本 randSet* /;
10. usedSet←topset + randSet /* 大梯度样本与小梯度样本进行合并* /;
11. s[randSet] = w* s[randSet] /* 小梯度样本权重乘以权重系数 $\frac{1-a}{b}$ 得到新的样本权重 s* /;
12. newmodel←L(M[usedSet], k[usedSet], s[used Set]) /* 由 usedSet 中的样本 M、梯度 k 与权重 s 得到新的弱学习器* /;
13. models. append(newModel) /* 将新的弱学习器加入模型* /;

LightGBM 中的互斥特征捆绑方法(EFB)包括两项内容: 特征捆绑和互斥特征合并,如算法 2 与算法 3 所示.

算法 2. 特征捆绑

输入: 特征 F、最大冲突数 C、图 G;

输出: 特征捆绑集合 bundles;

1. searchOrder←G. sortByDegree() /* 构造具有加权边的图 G,图的顶点为特征,将不互斥的特征顶点相连,边的权值对应特征同时不为 0 的样本数,按照顶点的度按降序排序特征* /;
2. bundles←{}, bundlesConflict←{}
3. for i in searchOrder do
4. needNew←True
5. for $j = 1$ to $len(bundles)$ do /* 对有序表中每个特征,在特征捆绑簇中进行遍历* /;
6. cnt←ConflictCnt(bundles[j], F[i])
7. if $cnt + bundlesConflict[j] \leqslant C$ then /* C 为设置的最大冲突阈值,若该特征加入特征簇中后冲突数不超过 C,则加入该特征到这个簇中* /;
8. bundles[j]. add($F[i]$), needNew←False /* 将特征分配给具有小冲突的现有 bundle* /;
9. break
10. if *needNew* then
11. Add $F[i]$ as a new bundle to *bundles* /* 否则创建新特征簇,并将该特征加入* /;

算法 3. 合并互斥特征

输入: 数据数量 *numData*、一个互斥特征簇 F;

输出: *newBin*、*binRanges*;

1. binRanges←{0}, totalBin←0 /* LightGBM 中直方图算法对特征值进行了分桶(bin)* /;
2. for f *in* F do /* binRanges 为偏移常数数组* /;
3. totalBin += f. numBin
4. binRanges. append(totalBin)
5. newBin←new Bin(numData)
6. for $i = 1$ to *numData* do
7. newBin[i] ←0
8. for $j = 1$ to $len(F)$ do
9. if $F[j]$. *bin*[i] ≠ 0 then
10. newBin[i] ←$F[j]$. bin[i] + binRanges[j]

2.4　CFS

基于相关性的特征选择(Correlation-based feature selection,CFS). 从源代码中提取到的代码度量部分可能与异味强度无关,即特征与异味强度之间的相关性较小,直接使用原始数据集中高维的数据特征通常会导致算法模型运算速度变慢,还可能降低识别精度. 因此提出 CFS 根据相关性分析对代码度量指标进行筛选,计算各度量指标与基于开发者感知的代码异味严重性数值间的相关系数,衡量其之间的相关关系. 采取过滤的方法,将相关性较低的度量指标滤去,保留高

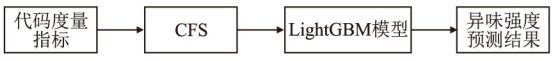

图 1　代码度量指标 CFS 过程

Fig. 1　Code metrics CFS process

相关度量指标来作为后续代码异味强度预测模型的输入，如图 1 所示.

3 代码异味强度预测模型

实际情况下代码之中存在的异味问题按危害程度有轻重之分，而现有的异味检测方法仍存在较大的局限性，或只依靠手动设置代码的单一结构度量阈值检测指标，如设置行数阈值范围来检测长方法，但缺点首先是指标的选取较为灵活，缺乏开发经验的人员往往不能准确定位某种代码异味的外在度量指标，其次是单一某种或简单几种综合的代码度量指标并不能充分反映代码质量.

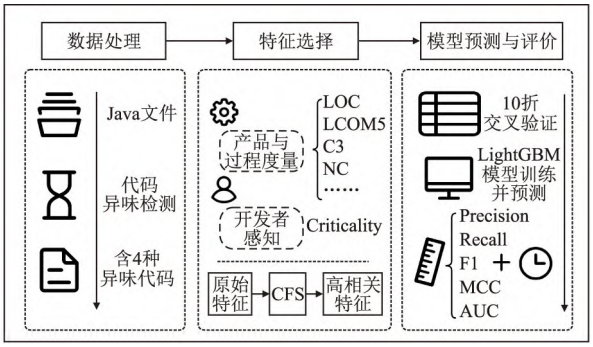

图 2 基于 LightGBM 融合 CFS 的代码异味强度预测模型

Fig. 2 Prediction model of code smell intensity based on LightGBM and CFS

基于此，本文考虑基于 LightGBM 融合 CFS 模型进行代码异味强度预测，先进行数据筛选过程，再提取相应代码度量指标并通过 CFS 进行特征选择，最后运用 LightGBM 进行预测并对预测效果及模型进行评价，如图 2 所示.

3.1 数据统计

本文考虑所研究的 4 种类级别代码异味，挖掘 Apache 与 Eclipse 系统中 9 个开源项目中存在异味的代码，使用 Décor [2] 和 HIST [21] 进行异味检测，提取分别包含 4 种异味的代码. 数据集中包含的代码异味数量情况如表 3 所示.

表 3 代码异味数量统计

Table 3 Code smell quantity statistics

Code Smell	数量
BC	341
CC	349
SC	313
SS	329

3.2 代码度量指标

选取产品度量、过程度量结合开发者度量，综合构建代码异味强度预测模型. 本文研究所考虑的度量指标如表 4 所示.

3.3 模型预测与评价

本文采用基于 LightGBM 融合 CFS 模型，运用 10 折交叉验证方式对数据集中代码的异味强度进行预测，如图 3 所示，依次将每折数据作为测试集，其余数据作为训练集，合并每折预测结果从而得到全体样本的代码异味强度预测结果. 共选取 6 个评价指标对实验进行评估，其中，基于实例的评价指标共 5 个，包括 Precision，Recall，F1，MCC，AUC. 基于模型效率的评价指标 1 个，分析基于 LightGBM 融合 CFS 模型的性能表现，并与 Pecorelli [15] 等人提出的 RF 模型方法进行对比.

表 4 代码度量指标

Table 4 Code metrics

	度量指标	英文缩写	描述
产品度量	代码行数	LOC	类的代码行数(不包括空格和注释)
	方法内聚缺乏度	LCOM5	类中没有公共属性引用的方法数
	类概念内聚度	C3	类的所有方法对之间的平均余弦相似性
	类之间的耦合度	CBO	系统中调用类的方法或访问类的属性的类数
	信息传递耦合度	MPC	类对系统外部类进行的方法调用数
	类内方法数	RFC	类的方法总和
	类权值方法	WMC	在类的所有方法上计算的 McCabe 圈复杂度之和
	可读性指数	Read.	代码可读性度量值
过程度量	共同修改的平均类数	AVG-CS	在涉及类的提交中共同更改的平均类数
	类更改次数	NC	执行类更改操作的次数
	Bug 修复数	NF	对类执行 Bug 修复行为的数量
	提交者数	NCOM	对类执行提交的开发者数
	开发者更改数	DSC	类提交者更改不同子系统的平均数
	持久度	Per.	异味影响类的后续版本数
开发者度量	异味严重程度(Criticality)	Criti.	开发者感知的代码异味严重性数值

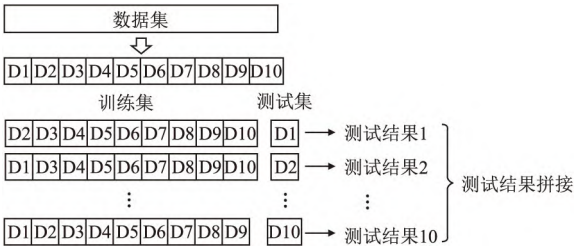

图 3 10 折交叉验证示意图

Fig. 3 10-fold cross-validation diagram

4 实验研究

本文设计 3 个实验，主要寻求解决以下 3 个问题:

Q1: 代码相关度量指标是否和基于开发者感知的代码异味强度间存在相关关系?

Q2: 本文所提出的基于 LightGBM 融合 CFS 模型与文献[15]的 RF 模型相比精度表现如何，是否有了提升?

Q3: 本文所提出的基于 LightGBM 融合 CFS 模型是否比文献[15]的 RF 模型速度更快，效率更高?

4.1 实验环境和数据集

本文实验环境如下: 操作系统为 Windows 10，处理器为 Intel(R) Core i5-6300HQ CPU @ 2.30GHz，内存 12GB，所用软件为 SPSS 和 PyCharm，开发语言为 Python.

为了评估本文模型的性能，实验使用基于开发者感知的

代码异味数据集,该数据集由开发者对代码气味实例进行评价得出,其中包含了开发者对感知到的代码异味严重程度的度量值.

表 5　实验所涉项目

Table 5　Projects for experiment

System	#Commits	#Dev.	#Classes	KLOCS
Apache Mahout	3054	55	813	204
Apache Cassandra	2026	128	586	111
Apache Lucene	3784	62	5506	142
Apache Cayenne	3472	21	2854	542
Apache Pig	2432	24	826	372
Apache Jackrabbit	2924	22	872	527
Apache Jena	1489	38	663	231
Eclipse CDT	5961	31	1415	249
Eclipse CFX	2276	21	655	106
总计	32889	436	5506	542

数据集包含 9 个开源 Java 项目,它们属于两个软件生态系统,分别为 Apache 与 Eclipse. 数据集中项目在代码量、作用领域等方面各不相同,具有一定的代表性. 所选项目的类数都超过了 500 个,变更历史超过 5 年,提交次数大于 1000 次,参与开发者数量都超过 20 人,可提供相应代码异味的实例以供研究. 表 5 总结了所选项目的基本信息与统计数据.

4.2　代码异味检测方法

数据集中包含从实验项目中筛选出的含 4 种代码异味的代码实例,其中 DÉCOR[9] 用来检测项目存在的 Blob Class、Complex Class 和 Spaghetti Code 异味,HIST[21] 用来检测 Shotgun Surgery 异味. 这两种工具已被代码异味研究学者广泛使用,能准确地识别出代码中的代码异味. 使用这两种工具进行检测从而避免了重新构造检测工具而带来的结果有效性威胁. 从备选实验项目中提取的含 4 种异味的数据集,均经过了以上两种方法的检验,并证实了各自所含的代码异味的存在. 用于实验的数据集中异味强度分布情况如表 6 所示.

表 6　数据集代码异味强度分布

Table 6　Code smell intensity distribution in dataset

Code Smell	Non-severe	Medium	Severe
BC	70	195	76
CC	69	190	90
SC	70	162	81
SS	96	152	81

4.3　相关性分析

从实验项目中选取代码度量指标,分别验证其是否与代码异味严重程度存在相关关系. 用 SPSS 统计软件进行实验. 这里引入 Spearman 相关系数.

$$\rho = \frac{\sum_i (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_i (x_i-\bar{x})^2 \sum_i (y_i-\bar{y})^2}} \tag{1}$$

其中 ρ 代表 Spearman 相关系数,它是衡量两个变量的依赖性的非参数指标,表明了 x(独立变量)与 y(依赖变量)的相关方向,这里 x 表示所考虑的各项代码度量指标,y 表示代码异味严重性. Spearman 相关系数取值范围为-1 ~ 1,0 代表变量间无相关性,即该代码度量指标和代码异味严重性不存在相关关系,Spearman 相关系数绝对值越接近 1 代表相关性越强,即该代码度量指标和代码异味严重性存在程度较强的相关关系,负值代表负相关,正值代表正相关.

由表 7 的统计结果表明,绝大多数考虑的度量指标和异味严重程度均有较强的相关性,Criti. 为基于开发者感知的代码异味强度值,Sig 意为显著性,在这里 Sig 代表判断异味严重程度和所考虑的代码度量指标之间具有相关关系的犯错几率,一般地在统计学中,若 Sig < 0.05 则说明所考虑变量间存在显著相关关系.(回答 Q1)

表 7　Criti. 与代码度量指标的相关性分析

Table 7　Correlation analysis between Criti. and code metrics

		LOC	LCOM5	C3	CBO	MPC	RFC	WMC
Criti.	相关系数	0.206	0.486	-0.396	0.184	0.007	0.196	0.242
	Sig	0.000	0.000	0.000	0.000	0.804	0.000	0.000
		Read.	AVG-CS	NC	NCOM	Pers.	NF	DSC
Criti.	相关系数	-0.310	0.138	0.190	0.034	0.121	0.304	0.275
	Sig	0.000	0.000	0.000	0.210	0.000	0.000	0.000

值得注意的是,由表 7 可观察到涉及表示代码变更的指标(如 NC、NF)均与开发者感知的代码异味严重性数值有着较强的相关性,且对代码修改的越频繁,代码异味严重程度也随之增加,呈明显正相关关系.(回答 Q1)

另外,MPC 的 Sig 值为 0.804,即判断 MPC 与 Criti. 相关的犯错概率高达 80.4%,且 Spearman 相关系数接近 0,说明 MPC 与 Criti. 无明显相关关系,故根据 CFS 的特征过滤方法,应将其删去. 本文所构建的异味强度预测模型具有一定通用性,若待测数据集中未提供 Criticality 值,通过本文 CFS 分析已筛选出与异味强度高度相关的代码度量指标,且高相关性的代码度量指标的选择一般不受不同代码数据集的影响,故只需采集待测数据集中相应代码度量指标代入模型即可进行代码异味强度的预测.

4.4　实验模型

4.4.1　RF 模型构建

在 Pecorelli 等人[15] 的代码异味严重性预测实现上,其方法是在比较了多种经典机器学习算法包括随机森林、逻辑回归、空间向量机、朴素贝叶斯和多层感知器后,选择了效果最优的随机森林作为最后的预测模型.

随机森林是一种基于集成学习的监督式机器学习算法[22],用随机方式建立一个森林模型,该森林结合了多个相同类型的算法,由许多决策树组成. 决策树之间相互独立无关联,输入新样本时,该样本的取值为所有决策树共同决策得出.

随机森林随机性主要体现在了两方面,样本抽样随机性和特征抽样随机性. 在训练集中随机地抽取一定数量的样本,作为树的根节点样本建立每棵子树. 这两点随机性使得随机森林不容易陷入过拟合.

由于随机森林引入了两个随机性,也使得其具有了良好的抗噪能力. 基于树模型的树状结构与其他线性模型的区别,数值缩放不影响分裂点位置,因而无需对数据集进行标准化处理.

4.4.2　LightGBM 模型构建

LightGBM 是基于 GBDT 算法的一个具体实现框架,GB-

DT(Gradient Boosting Decision Tree)为了使模型的效果达到最优化,利用弱分类器(决策树)进行迭代训练,GBDT 广泛应用于多分类任务和点击率预估问题(CTR)的解决[23].

LightGBM 采用基于 Histogram 的决策树算法,特点是引入了单边梯度采样(GOSS)与互斥特征捆绑(EFB)[18].GOSS 的使用能有效降低小梯度数据在数据实例中所占比例,因此在计算信息增益时只利用剩余的高梯度数据即可,大大节省了在时间和空间上的消耗.EFB 可将许多互斥的特征进行捆绑,从而有效减少了特征维度.

另外,传统的决策树模型生长采用 Depth-wise 策略,即是按层进行的[24].而 LightGBM 决策树采取 Leaf-wise 生长策略,即是按叶进行的.每次分裂时从目前所有叶子之中寻找分裂增益最大的叶子进行分裂,如此循环.LightGBM 采用这种分裂方式的优点是使误差得到了有效的降低,有利于精度的提高,同时为了防止 Leaf-wise 生长策略带来的过拟合问题,LightGBM 在参数设置中引入了 max_depth 最大深度限制,有效地解决了该问题.

LightGBM 算法模型需要对参数进行调整已达到更好的训练与测试效果,LightGBM 在所考虑的4种代码异味数据集上表现出良好性能的关键优化参数及含义如表8所示.

表8 LightGBM 模型参数表

Table 8 LightGBM model parameter table

参 数	含 义	BC	CC	SC	SS
Boost	基学习器模型算法	gbdt	gbdt	gbdt	gbdt
Learning_rate	学习率	0.12	0.87	0.1	0.12
Bagging_fraction	不进行重采样的情况下随机选择数据比例	0.7	0.91	0.91	0.7
Feature_fraction	每次迭代随机选择特征比例	0.9	0.9	0.91	0.9
Max_depth	树最大深度	-1	-1	-1	-1
Min_data_in_leaf	叶子数据最小数量(每个叶节点的最少样本数量)	25	20	20	25
Min_sum_hessian_in_leaf	节点分裂最小海森值之和(叶节点样本权重之和的最小值)	12.5	6	20.1	12.5
Num_leaves	叶子节点数(一棵树上的叶子数)	80	80	80	80
Num_iterations	迭代次数	150	150	150	150
Lambda	正则化系数	0.3	0.32	0.32	0.3
Objective	任务类型	multiclass	multiclass	multiclass	multiclass
Num_class	多分类任务中类别数	3	3	3	3
Early_stopping_rounds	早停轮数	200	200	200	200

4.5 实验结果

根据 Pecorelli 等人[15]的 RF 模型与本文模型在4种代码异味数据集上的预测结果,分别构建混淆矩阵进行对比,如表9所示.

表9 本文模型与 RF 模型的混淆矩阵对比

Table 9 Comparison of confusion matrix between the proposed model and RF model

Model	Class/Smell	BC			CC			SC			SS		
		Non-severe	Medium	Severe	Non-severe	Medium	Severe	Non-severe	Medium	Severe	Non-severe	Medium	Severe
本文模型	Non-severe	57	6	7	50	19	21	57	11	2	47	36	13
	Medium	5	174	16	12	178	0	5	151	6	6	136	10
	Severe	0	8	68	7	2	60	0	7	74	6	15	60
RF	Non-severe	57	6	7	47	17	26	57	11	2	45	38	13
	Medium	5	174	16	15	175	0	6	150	6	11	131	10
	Severe	0	13	63	17	0	52	0	8	73	6	16	59

从表9中可看出,运用本文模型对4种代码异味强度进行预测,在 BC、CC、SC、SS 所有4种异味上预测效果均优于 RF 模型,在预测 Non-severe、Medium、Severe 3个异味级别中被错误预测的数量明显减少.采用本文模型预测效果可以在 RF 模型的基础之上再获得较大提升.(回答 Q2)

4.6 模型评价

本文选取6个评价指标对实验进行评估,其中,基于实例的评价指标共5个,基于模型效率的评价指标1个.

4.6.1 基于实例的评价指标

基于实例的评价指标是对模型在每种代码异味数据集上进行预测的结果进行评估,以验证模型区分正负样本的能力,相关指标如下:

1)精确率.

真正正确占预测为正确的比例.计算公式如下:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

其中,TP 表示实际为正例且被分类器判定为正例的样本数;FP 表示实际为负例且被分类器判定为正例的样本数;FN 表示实际为正例但被分类器判定为负例的样本数;TN 表示实际为负例且被分类器判定为负例的样本数.

2)召回率.

真正正确占所有实际为正的比例.计算公式如下:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

3)F1 值.

F-Measure 计算公式如下.

$$F_\beta = \frac{(\beta^2+1) \times Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{4}$$

这里 β 取为 1，即为 F1 值.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5}$$

4) MCC 值(马修斯相关系数).

MCC 的本质描述了预测结果与实际结果间的相关系数. 计算公式如下:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{6}$$

MCC 取值范围为 [-1,1]，MCC 值为 1 表示预测结果与实际结果完全一致，-1 表示完全不一致.

5) AUC 值.

ROC 曲线下方的面积大小，其取值范围为 [0,1]，该值越大的分类器，表示分类的正确率越高.

以上基于实例的评价指标原是基于二分类问题所讨论的，但都根据本文所研究的多分类问题进行了重新调整，均利用基于 Python 的 Scikit-Learn 进行自动化计算. 精确率高表示模型要尽量在更有把握的情况下才会将样本预测为正样本，意味着精确率越高，模型对负样本区分能力越强. 召回率体现模型对于正样本的区分能力，召回率越高则模型对正样本的区分能力越强. 因此，对同一个模型来说，高精确率和高召回率通常不能同时达到，具有较高的精确率的模型往往具有较低的召回率. 选取精确率和召回率作为评价指标具有不同的参考意义.

另外，本文选取相应算法模型的程序运行时间作为基于模型效率的评价指标，即计算模型从开始训练到预测完毕的时间，以衡量模型运行效率的高低.

将 RF 模型与本文模型在 4 种异味数据集上分别进行实验，将 5 种基于实例的评价指标进行汇总如表 10 所示.

表 10　本文模型与 RF 模型的评价指标对比

Table 10　Comparison of evaluation indexes between the proposed model and RF model

Code Smell	Model	Prec.	Rec.	F1	MCC	AUC
BC	本文模型	88.5%	87.7%	87.8%	79.2%	89.8%
	RF	86.8%	86.2%	86.4%	76.4%	86.0%
CC	本文模型	82.0%	82.5%	81.9%	70.8%	87.2%
	RF	78.1%	78.5%	78.2%	64.2%	85.4%
SC	本文模型	90.2%	90.1%	90.0%	83.8%	94.2%
	RF	89.5%	89.5%	89.4%	82.7%	92.2%
RF	本文模型	74.6%	73.9%	72.8%	59.3%	77.5%
	RF	71.6%	71.4%	70.4%	55.2%	74.8%

由表 10 可知，Pecorelli 等人[15]所使用的 RF 模型的 F1 值在 70.4% 到 89.4% 之间，而采用本文的代码异味强度预测模型，其精确率、召回率、F1 值、MCC 和 AUC 在所有 4 种代码异味数据集上均较 RF 模型有了较大提升，其中预测精确率最高达 90.2%，相比 RF 模型最多提升了 3.9%；召回率最高达 90.1%，最多提升了 4.0%；F1 值最高达 90.0%，最多提升了 3.7%；MCC 值最高达 83.8%，最多提升了 6.6%；AUC 值最高达 94.2%，最多提升了 3.8%. 实验证明，采用基于 LightGBM 融合 CFS 的代码异味强度预测模型较 RF 模型优化效果明显. (回答 Q2)

4.6.2　基于模型效率的评价指标

为了比较两种模型运行效率的差异，运用本文基于 LightGBM 融合 CFS 的代码异味强度预测模型与 RF 模型对 4 种代码异味数据集进行异味强度预测并统计模型从开始训练至预测完毕的时间，作为衡量模型运行效率的指标. 每个模型在每种异味数据集上分别运行 3 次，记录每次所需时间如表 11 所示，表中数值单位均为毫秒(MS).

表 11　本文模型与 RF 模型运行时间对比

Table 11　Comparison of running time between the proposed model and RF model

Code Smell	RF			本文模型		
BC	1754	1694	1752	334	349	348
CC	1637	1649	1712	453	485	466
SC	1670	1632	1634	343	359	350
SS	1872	1768	1763	482	489	448

取两种模型分别在 4 种异味数据集上进行 3 次异味强度预测的运行时间平均值作为最终运行时间，得出两种模型的平均耗时对比图如图 4 所示.

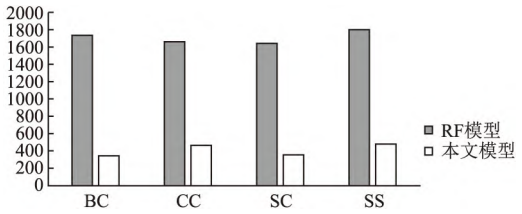

图 4　RF 模型与本文模型平均耗时对比图

Fig. 4　Comparison of average time consumption between RF model and the proposed model

如图 4 所示，本文所使用的基于 LightGBM 算法融合 CFS 模型在 4 种代码异味数据集上，运行耗时均远低于 RF 模型，运行时间缩短 76.1%. 相较于原模型，采用本文模型可使运行效率得到极大提升. (回答 Q3)

5　总结与期望

本文提出了一种基于 LightGBM 融合 CFS 的开发者感知代码异味强度预测模型，与文献[15]采用的 RF 预测模型相比，性能在考虑的全部 6 个评价指标上均得到了较大提升. 在提高了异味预测精度的同时加快了预测速度，降低了运行成本. 说明本文基于 LightGBM 融合 CFS 的开发者感知代码异味强度预测模型能较好地帮助开发者确定代码中存在异味的优先级，以便于优先解决异味严重程度高的代码，有助于降低软件开发与维护成本.

后续的相关工作有两个方向: 1) 本文所选项目均基于 Java 语言，后续可以扩展到研究其他编程语言中的异味问题; 2) 本文只聚焦 4 种常见的代码异味，且均属类级别，可以扩展到其他种类的代码异味或研究粒度更高的方法级别代码异味问题.

References:

[1] Fowler M. Refactoring: improving the design of existing code [M].

Massachusetts: Addison-Wesley Longman Publishing Co. Inc. ,2018.

[2] Moha N,Guéhéneuc Y G,Duchien L,et al. Decor: a method for the specification and detection of code and design smells [J] . IEEE Transactions on Software Engineering,2009,36(1):20-36.

[3] Li Z,Avgeriou P,Liang P. A systematic mapping study on technical debt and its management [J] . Journal of Systems and Software,2015,101:193-220,doi:10.1016/j.jss.2014.12.027.

[4] Chatzigeorgiou A,Manakos A. Investigating the evolution of bad smells in object-oriented code [C] //Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology(QUATIC),2010:106-115.

[5] Lehman M M,Belady L A. Program evolution: processes of software change [M] . Massachusetts: Academic Press Professional,Inc. ,1985.

[6] Oliveira A,Sousa L,Oizumi W,et al. On the prioritization of design-relevant smelly elements: a mixed-method,multi-project study [C] //Proceedings of the XIII Brazilian Symposium on Software Components,Architectures,and Reuse(SBCARS),2019:83-92.

[7] Macia I. On the detection of architecturally-relevant code anomalies in software systems [D] . Rio de Janeiro: Pontifical Catholic University of Rio de Janeiro,2013.

[8] Zhang M,Hall T,Baddoo N. Code bad smells: a review of current knowledge [J] . Journal of Software Maintenance and Evolution: Research and Practice,2011,23(3):179-202.

[9] Basili V R,Briand L C,Melo W L. A validation of object-oriented design metrics as quality indicators [J] . IEEE Transactions on Software Engineering,1996,22(10):751-761.

[10] Yamashita A,Moonen L. To what extent can maintenance problems be predicted by code smell detection? -an empirical study [J] . Information and Software Technology,2013,55(12):2223-2242.

[11] Zhang M,Baddoo N,Wernick P,et al. Prioritising refactoring using code bad smells [C] //Proceedings of the IEEE 4th International Conference on Software Testing,Verification and Validation Workshops(ICSTW),2011:458-464.

[12] Ouni A,Kessentini M,Sahraoui H,et al. Improving multi-objective code-smells correction using development history [J] . Journal of Systems and Software,2015,105:18-39,doi:10.1016/j.jss.2015.03.040.

[13] Vidal S A,Marcos C,Díaz-Pace J A. An approach to prioritize code smells for refactoring [J] . Automated Software Engineering,2016,23(3):501-532.

[14] Fontana F A,Zanoni M. Code smell severity classification using machine learning techniques [J] . Knowledge-Based Systems,2017,128:43-58,doi:10.1016/j.knosys.2017.04.014.

[15] Pecorelli F,Palomba F,Khomh K,et al. Developer-driven code smell prioritization [C] //Proceedings of the 17th International Conference on Mining Software Repositories(MSR),2020:220-231.

[16] Jain S,Saha A. Rank-based univariate feature selection methods on machine learning classifiers for code smell detection [J] . Evolutionary Intelligence,2021,3:1-30,doi:10.1007/s12065-020-00536-Z.

[17] Hall M A. Correlation-based feature selection for machine learning [D] . Hamilton: The University of Waikato,1999.

[18] Ke G,Meng Q,Finley T,et al. Lightgbm: a highly efficient gradient boosting decision tree [C] //Advances in Neural Information Processing Systems(NIPS),2017:3146-3154.

[19] Sun X L,Liu M X,Sima Z Q. A novel cryptocurrency price trend forecasting model based on LightGBM [J] . Finance Research Letters,2020,32:101084,doi:10.1016/j.frl.2018.12.032.

[20] Wang D H,Zhang Y,Zhao Y. LightGBM: an effective miRNA classification method in breast cancer patients [C] //Proceedings of the International Conference on Computational Biology and Bioinformatics(ICCBB),2017:7-11.

[21] Palomba F,Bavota G,Di Penta M,et al. Mining version histories for detecting code smells [J] . IEEE Transactions on Software Engineering,2014,41(5):462-489.

[22] Svetnik V,Liaw A,Tong C,et al. Random forest: a classification and regression tool for compound classification and QSAR modeling [J] . Journal of Chemical Information and Computer Sciences,2003,43(6):1947-1958.

[23] Dave K S,Varma V. Learning the click-through rate for rare/new ads from similar ads [C] //Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval(SIGIR),2010:897-898.

[24] Myles A J,Feudale R N,Liu Y,et al. An introduction to decision tree modeling [J] . Journal of Chemometrics: A Journal of the Chemometrics Society,2004,18(6):275-285.