基于 SDL-LightGBM 集成学习的软件缺陷预测模型

谢华祥[1]，高建华[1+]，黄子杰[2]

（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：为提高软件缺陷预测准确性和预测模型的可解释性，提出一种 Spearman＋DE＋LIME＋LightGBM（SDL-LightGBM）集成学习的软件缺陷预测模型。使用混合特征选择方法 Spearman＋LightGBM 确定最佳特征子集，在保证模型预测性能的情况下降低模型复杂度；使用集成学习算法 LightGBM（light gradient boosting machine）对特征子集建立预测模型，并使用差分进化（differential evolution，DE）算法优化模型的重要超参数；使用局部可解释的模型无关技术（local interpretable model-agnostic explanations，LIME）对模型进行局部可解释分析。实验通过 12 个项目的 35 个版本的结果表明，SDL-LightGBM 算法优于现有的软件缺陷预测方法，F1 值平均提高 8.97%，AUC 值平均提高 11.42%，模型训练时间缩短 43.6%。

关键词：缺陷预测；机器学习；集成学习；特征选择；模型优化；模型解释；差分进化

中图法分类号：TP311 **文献标识号**：A **文章编号**：1000-7024（2024）03-0769-08

doi：10.16208/j.issn1000-7024.2024.03.018

Software defect prediction model based on SDL-LightGBM ensemble learning

XIE Hua-xiang[1]，GAO Jian-hua[1+]，HUANG Zi-jie[2]

（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：To improve the accuracy of software defect prediction and the interpretability of prediction model，a software defect prediction model based on Spearman＋DE＋LIME＋LightGBM（SDL-LighTGBM）ensemble learning was proposed. Spearman＋LightGBM hybrid feature selection method was used to determine the optimal feature subset，and model complexity was reduced while the prediction performance of the model was guaranteed. The ensemble learning algorithm LightGBM was used to build a prediction model for feature subsets，and differential evolution（DE）algorithm was used to optimize the important hyperparameters of the model. Local interpretable model-agnostic accessibility（LIME）was used for locally interpretable analysis of the model. Experimental results from 35 versions of 12 projects show that the proposed method is superior to the existing software defect prediction methods. The average increase of F1 value is 8.97%，the average increase of AUC value is 11.42%，and the model training time is shortened by 43.6%.

Key words：defect prediction；machine learning；ensemble learning；feature selection；model optimization；model interpretation；differential evolution

收稿日期：2022-07-20；**修订日期**：2024-02-20

基金项目：国家自然科学基金项目（61672355）

作者简介：谢华祥（1995－），男，安徽六安人，硕士研究生，研究方向为缺陷预测、代码异味和机器学习；＋通讯作者：高建华（1963－），男，浙江绍兴人，博士，教授，CCF 高级会员，研究方向为软件重构、软件测试、可信软件和可靠性模型设计；黄子杰（1994－），男，上海人，博士研究生，研究方向为代码异味、软件可靠性和实证软件工程。E-mail：13093626131@163.com

0 引 言

软件缺陷预测利用项目中已有的缺陷信息建立缺陷预测模型，并预测项目中可能出现缺陷的代码，以减少开发人员检查出错代码的时间和软件开发成本。相关研究[1]指出大型系统的维护工作占用软件开发总成本的 90%。软件缺陷预测技术可以根据软件的信息提取特征，使用分类器算法建立预测模型来确定软件具体模块是否包含缺陷。

在模型的使用方面，Erturk 等[2]使用 SVM（support vector machine）算法对 NASA 数据集建立缺陷预测模型，并取得不错效果。Feidu 等[3]使用多种机器学习算法和基于 LM（language model）的神经网络算法建立缺陷预测模型，实验结果显示基于 LM 的神经网络取得最高预测精度。

上述软件缺陷预测研究大多使用单个分类器，而单个分类器存在预测准确率不高，泛化能力弱等问题，相比单个分类器，集成学习具有更好的分类准确性和效率[4]。如陈丽琼等[5]使用 XGBoost 集成学习算法建立即时软件缺陷预测模型。

然而针对典型 Boosting 算法的不足，Ke 等[6]提出了 LightGBM 算法，大量实验结果表明 LightGBM 算法在性能和效率方面优于传统机器学习算法和其它集成学习算法。

在特征选择方面，实验数据特征间的多重共线性和大量无关特征会导致“维度灾难”，增加预测模型的复杂度和训练时间[7]，并可能导致模型过拟合[8]。合理的特征选择可以有效地实现特征降维[9]。

相比分类器的默认超参数，不同超参数组合对应模型性能有较大不同[14]，超参数优化主要有网格搜索、随机搜索以及贝叶斯优化。Shen 等[10]比较各种优化算法，发现 DE 算法优化结果最好。

在可解释方面，LIME 是 Ribeiro 等[11]提出的局部替代模型。该模型使用可解释的机器学习模型（即线性回归、决策树等）局部模拟黑盒模型的预测。因此，可以使用局部代理模型来解释单个实例。

基于以上研究，本文提出一种基于 SDL-LightGBM 集成学习的软件缺陷预测模型。

1 相关工作

1.1 LightGBM 算法

集成学习是组合多个基分类器来建立一个强分类器，即使每个基分类器是弱分类器，通过集成学习后也能建立一个强分类器。它能有效避免传统分类器的过拟合问题，同时能获得更强的泛化能力。由 Ke 等[6]提出的 LightGBM 算法是集成学习的一种，主要使用基于梯度的单边采样（gradient-based one-side sampling，GOSS）和互斥特征捆绑（exclusive feature bundling，EFB）这两种方法弥补 Boosting 算法的不足[6]。

GOSS 是从减少样本的角度出发，排除大部分权重小的样本，仅用剩下的样本计算信息增益，它是一种在减少数据和保证精度上平衡的算法。而高纬度数据中很多特征是互斥的，特征很少同时出现非 0 值。EFB 的思想就是把这些特征捆绑在一起形成一个新的特征，以减少特征数量，提高训练速度。因此本文选择 LightGBM 算法构建基础模型。

1.2 特征选择

原始特征集可能存在无关特征和特征间的多重共线问题[5]。而 LightGBM 的特征重要性算法可以有效剔除无关特征。处理特征间的多重共线问题，本文选择 Spearman[13]定义的 Spearman 秩相关系数，定义如式（1）

$$\rho = 1 - \frac{6\sum(x_i - y_i)^2}{n(n^2-1)} \tag{1}$$

式中：x_i，y_i 分别是两个特征按大小（或优劣）排位的等级，n 是样本大小，ρ 代表 Spearman 秩相关系数。ρ 越大，代表两个特征之间共线性越强。

如艾成豪等[12]通过 ReliefF＋XGBoost＋Pearson 相关系数混合特征选择，融合所有特征的权重，删除融合权值较低的特征得到特征子集，有效地降低了模型的运行时间。

1.3 模型超参数优化

相比分类器的默认超参数，不同超参数组合对应模型性能有较大不同[14]。Shen 等[10]通过 4 个优化算法优化 6 个常用的机器学习分类器，使用 AUC 值作为模型性能度量，实验发现模型超参数优化可以显著提高代码异味检测的性能，差分进化（DE）算法可以比其它 3 种优化器获得更好的性能。因此本文使用 DE 算法优化 LightGBM 的超参数。

1.4 模型可解释相关研究

现有的软件缺陷预测重点关注模型的预测能力，忽略了模型的可解释性。大多数分类器都属于黑盒模型，其预测的结果无法使人信服。LIME 解释模型的原理通过简单模型来解释复杂模型。对样本数据变换得到一个新的数据集，用这个新数据集训练一个简单模型，即一个容易解释的模型。假设 f 是一个黑箱模型，x 是一个需要解释的实例，LIME 的损失函数如式（2）

$$\mathrm{E}(x) = \mathrm{argmin}L_{g\mathbf{z}_i}(f, g, \pi_x) + \Omega(g) \tag{2}$$

式中：L 反应黑箱模型（复杂模型）f 与简单模型 g 之间预测结果相似程度，G 表示简单模型 g 的算法集，π 表示定义在实例 x 周围采样时的域范围，$\Omega(g)$ 表示可解释模型 g 的模型复杂性，本文采用 LIME 对模型进行解释性分析。

2 SDL-LightGBM 设计流程

机器学习算法可以有效地建立软件缺陷预测模型。然而相关工作存在特征冗余、分类器选择和模型解释的不足[3]，对此，本文提出一种基于 SDL-LightGBM 集成学习的软件缺陷预测模型，方法流程如图 1 所示。

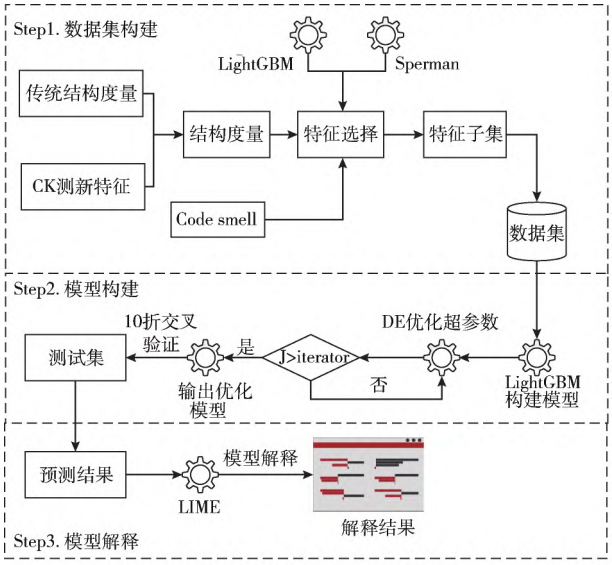

图 1　本文方法流程

2.1　建立特征集

实验的特征集主要有两类特征组合，一类为传统结构度量，另一类由 Fowler 等[16]定义的代码异味。

2.1.1　结构度量

产品度量测量源代码内在特征，如代码行数、复杂度等。本文在传统结构度量的基础上[16]利用 CK[17]软件测量 36 个结构度量新特征。通过类关键字匹配对两个数据集进行融合，供后面特征选择组合。具体特征描述分别见表 1 与表 2。

2.1.2　代码异味

本文采用由 Fontana 等[18]定义的代码异味强度指数（Intensity）量化代码异味。通过 JCodeOdor[18]工具检测 6 种类型的代码异味并计算得到一个 Intensity 值。6 种代码异味分别是：

（1）God Class：实现不同职责和集中大部分系统处理的大型类。对程序理解、软件可维护性有负面影响。

（2）Data Class：单纯用作数据存储的类，该类仅有一些字段（fields），以及读写这些字段的函数。

（3）Brain Method：实现多个函数的过大方法。

（4）Shotgun Surgery：一个类遇到某种变化，其它类需要被动作出修改。

（5）Dispersed Coupling：类与其它类具有太多耦合关系。

（6）Message Chains：过长的方法调用。

表 1　传统结构度量

特征	描述	特征	描述	特征	描述
wmc	加权方法统计	*lcom3*	方法 3 缺乏连贯性	*max_cc*	最大圈复杂度
dit	遗传树深度	*loc*	代码行数	*avg_cc*	平均圈复杂度
noc	子类数量	*dam*	数据访问度量	*fi-changes*	代码更改熵
cbo	类之间的耦合	*moa*	聚合度	*ostrand*	代码组件
rfc	类反应	*mfa*	功能抽象度量	*scattering*	注意力分散度
lcom	方法缺乏连贯性	*cam*	类方法间的内聚	*ana*	反模式平均数
ca	传入耦合	*ic*	继承耦合	*acm*	反模式复杂性
ce	传出耦合	*cbm*	方法间的耦合	*arl*	反模式递归长度
npm	公共方法数量	*amc*	平均方法复杂度	*acpd*	反模式累积差

表 2　CK 测量的 36 个结构度量

特征	描述	特征	描述	特征	描述
cboM	类依赖项数	FMQty	final 方法数	RQty	return 数量
FAN-IN	类输入依赖数	SCMQty	同步方法数	LQty	loop 表达式数
FAN-OUT	类输出依赖数	NOSI	静态方法调用次数	CPQty	比较表达式数
ACQty	匿名类数	TFQty	字段总数	TCQty	tryCatch 数
ICQty	内部类数	SFQty	静态字段数	PEQty	括号表达式数
TMQty	方法总数	PFQty	公共字段数	SLQty	重复字符串数
SMQty	静态方法数	PVFQty	私有字段数	NBQty	数字的数量
PVMQty	私有方法数	DFQty	默认字段数	MOQty	数学运算数
PTMQty	保护方法数	FFQty	final 字段数	VBQty	声明变量数
DMQty	默认方法数	SCFQty	同步字段数	MNBQty	代码块最大数
VMQty	可见方法数量	UWQty	独有单词数	LBDQty	lambdas 表达式
AMQty	抽象方法数	modifiers	修饰符数	LSMQty	日志行数

2.1.3 混合特征选择

本文为了更细化研究软件结构度量，使用 CK[17] 软件测量 36 个结构度量新特征，加上 27 个传统结构度量和 JCodeOdor[18] 工具计算得到的一个代码异味强度指数 Intensity 值共提取 64 个特征，并利用类名作为匹配键对其合并。然而合并后特征间的高度共线性和大量无关特征会导致“维度灾难”，增加预测模型的复杂度和训练时间。对此本文采用 LightGBM 特征重要性和 Spearman[13] 算法对特征选择与组合，并确定特征子集。通过实验比较得到 LightGBM 重要性阈值为 30 时，不会降低模型的预测能力。根据 Nucci 等[19] 的建议，当设置 Spearman 的相关度 ρ 的阈值为 0.8，即当 ρ 大于 0.8 时认为两个特征具有强相关，这时保留对模型贡献最大的特征。Spearman＋LightGBM 混合特征选择算法具体流程如算法 1 所示：

算法 1：混合特征选择算法

输入：合并后的数据集

输出：特征子集

（1）使用 LightGBM 计算每个特征对预测结果的重要性权重向量 W，并按照从大到小进行排序

$$W=[\omega_1(x),\omega_2(x),\cdots,\omega_n(x)]$$

（2）设置重要性阈值 30，去除重要性低于阈值的特征，得到第一次选择的特征子集 X_1

$$X_1=[x_1,x_2,\cdots,x_n]$$

（3）使用 Spearman 测量特征子集 X_1 两两特征间的相关系数 ρ，得到相关系数矩阵 P

$$P=\begin{bmatrix}\rho_{11} & \rho_{12} & \cdots & \rho_{1n}\\ \rho_{21} & \rho_{22} & \cdots & \rho_{2n}\\ \cdots & \cdots & \cdots & \cdots\\ \rho_{n1} & \rho_{n2} & \cdots & \rho_{mn}\end{bmatrix}$$

（4）设置相关性阈值为 0.8，当两两特征阈值大于 0.8 时认为其具有高度相关，保留重要性最高的一个特征。依次去除高度相关特征，最后得到特征子集 X_2

$$X_2=[x_1,x_2,\cdots,x_n]$$

经过 Spearman＋LightGBM 混合特征选择后，从高度冗余的 64 个特征中选择 15 个组合成结构度量子集，去除大量无关特征和高度共线特征，进而实现降低模型复杂度。

2.2 模型优化

机器学习模型中有各种参数需要调整，这些参数可分为模型参数和模型超参数。模型参数是模型内部通过自动学习而得出的配置变量，如神经网络中的权重，逻辑回归中的系数等。模型超参数则需要从外部配置，模型训练之前需要手动设置的参数。如随机森林树的深度、神经网络中迭代次数等。不同超参数设置会有不同的模型性能。本文对 LightGBM 的 5 个关键超参数进行优化其具体描述见表 3。

表 3 本文优化的 LightGBM 超参数

超参数名称	默认值	取值范围	超参数描述
n_estimators	100	[10,500]	树的数量
max_depth	None	[10,200]	树最大深度
num_leaves	31	[2,200]	叶子节点个数
min_child_sample	20	[1,100]	叶子节点最小样本数
learning_rate	0.1	[0.05,0.3]	学习率

本文使用 DE 算法优化 LightGBM 的超参数，优化方法如算法 2 所示：

算法 2：DE 优化 LightGBM 超参数

输入：数据集和 LightGBM 分类器

输出：优化后的一组超参数

/＊将数据集划分为训练集和测试集＊/

$$x_{train},x_{test},y_{train},y_{test}\leftarrow D$$

/＊1. 10 折交叉验证＊/

/＊2. DE 种群初始化＊/

$$X_i(0)=(x_{i,1}(0),x_{i,2}(0),\cdots,x_{i,n}(0))$$

/＊3. 开始迭代＊/

$$H_i(g)=X_{p1}(g)+F\cdot(X_{p2}(g))-X_{p3}(g)$$

/＊4. 从种群中随机选择 3 个个体产生变异，F 是缩放因子取 0.5＊/

$$\text{if rand}(0,1)\leqslant CR \text{ or } j=j_{rand}$$

$$U_{j,i}(g)=H_{j,i}(g)$$

$$Otherwise$$

$$U_{j,i}(g)=X_{j,i}(g)$$

/＊5. 在变异操作后，对第 g 代种群 $\{X_i(g)\}$ 及其变异中间体 $\{H_i(g)\}$ 进行个体间交叉操作＊/

$$\text{if } f(U_i(g))\leqslant f(X_i(g))$$

$$X_i(g+1)=U_i(g)$$

$$Otherwise$$

$$X_i(g+1)=X_i(g)$$

/＊6. DE 算法采用贪婪算法来选择下一代的个体＊/

$$End\ for$$

/＊7. 迭代结束，输出最优超参数组合＊/

3 实验分析

为了验证基于 SDL-LightGBM 集成学习的软件缺陷预测模型的有效性，本文主要回答以下 4 个问题：

Q1：Spearman＋LightGBM 混合特征选择算法是否有效？

Q2：使用 DE 算法优化模型超参数是否能提高模型预测性能？

Q3：本文所提出的 SDL-LightGBM 方法与其它文献相比是否具有优势？

Q4：使用 LIME 分析什么特征对结果影响最大？

3.1　实验数据集

实验使用的数据集[16]包括 12 个开源系统的 35 个版本。表 4 给出了数据集的详细信息，包括系统名称、系统版本数量、系统类的数量（最小-最大）、系统代码千行数量（最小-最大）和缺陷类的百分比（最小-最大）

表 4　实验项目

系统名	版本数量	类数量	代码千行数	缺陷类百分比
Apache Ant	5	83-813	20-204	68-72
Apache Camel	4	315-571	70-108	30-38
Apache Forrest	2	112-628	18-193	37-39
Apache Ivy	1	349	58	29
JEdit	2	228-520	39-166	36-43
Apache Velocity	3	229-341	57-73	15-23
Apache Tomcat	1	858	301	6
Apache Log4j	3	338-2246	103-466	59-63
Apache Xalan	4	121-509	13-55	29-33
Apache POI	4	129-278	68-124	62-68
Apache Synapse	3	249-317	117-136	17-26
Apache Xerces	3	149-217	147-156	37-46

3.2　十折交叉验证

为了更好评估模型的性能，实验采用 10 折交叉验证策略，将数据集随机划分 10 个大小相等的数据子集，一个作为测试集，其余 9 个作为训练集，重复 10 次，使每个子集都恰好一次作为测试集，最后结果取 10 次操作的平均值。

3.3　评价指标

（1）F1 值：在分类预测中，常用精确度（Precision）和召回率（Recall）这两者的调和平均作为评价指标即 F 值（F-Score），如式（3）

$$\mathrm{F}=\frac{(\alpha^2+1)\,precision*recall}{\alpha^2(precision+recall)} \tag{3}$$

当 α 取 1 时，就是常见的 F1 值，如式（4）

$$\mathrm{F1}=2*\frac{precision*recall}{precision+recall} \tag{4}$$

precision 和 recall 的定义如式（5）、式（6）

$$precision=\frac{TP}{TP+FP} \tag{5}$$

$$recall=\frac{TP}{TP+FN} \tag{6}$$

其中，TP、FP 和 FN 的含义见表 5。

（2）AUC 值：AUC 值是 ROC 曲线下的面积，而 ROC 曲线的横轴是 FPRate，纵轴是 TPRate。当两者相等时，其表示的含义是对于正类和负类的预测概率为 1 的概率相等，即 TPRate＝FPRate＝0.5，此时 ROC 下的面积即 AUC 值为 0.5，分类器没有任何区分能力。而一个好的分类器要求 TPRate ＞＞ FPRate。而理想的情况为 TPRate ＝ 1，FPRate＝0。此时 ROC 下的面积即 AUC 值最大，分类器性能最好。

表 5　TP、TN、FP、FN 概念

	检查为正	检查为负
样本为正	TP 正判为正	FN 正判为负
样本为负	FP 负判为正	TN 负判为负

3.4　实验结果分析

Q1：为了验证本文 Spearman＋LightGBM 混合特征选择方法的有效性，本文分别使用选择前和选择后的特征数据集建立预测模型。首先利用 LightGBM 的特征重要性 importance 对 64 个特征进行排序。importance 越高，说明特征对预测结果越重要。排序结果见表 6。

importance 大于 30 的特征有 17 个，大于 10 的特征有 37 个，因此本文对 importance 阈值为 0、10 和 30 分别进行筛选比较，结果见表 7。

根据实验比较结果可以看出，当 importance 阈值设置为 30 时，能最大程度降低特征维度从而降低模型复杂度且不会影响模型的预测性能。因此利用 LightGBM 筛选结果保留 importance 大于 30 的 17 个特征，分别是 intensity、scattering、fi-changes、ce、ostrand、UWQty、loc、cbo、AMQty、cboM、VBQty、amc、rfc、FAN-OUT、acpd、TMQty 和 cam。

考虑到特征间可能具有多重共线性，本文利用 Spearman 秩相关系数计算 17 个特征间的相关系数 ρ。根据 Nucci 等[19]的建议，当设置 ρ 的阈值为 0.8，即当 ρ 大于 0.8 时认为两个特征具有强相关，这时保留对模型贡献最大的特征，剔除特征 loc 和 cam。

经过 Spearman＋LightGBM 混合特征选择后的特征子集共 15 个特征，分别是 intensity、scattering、fi-changes、ce、ostrand、UWQty、cbo、AMQty、cboM、VBQty、amc、rfc、FAN-OUT、acpd 和 TMQty。

特征选择前后对模型的影响比较结果见表 8，特征选择后的模型 F1 值提高 0.5%，AUC 值提高 0.4%，模型预测性能得到提高。特征选择前的模型模型训练时间为 30.36 s，特征选择后的模型训练时间为 21.12 s，模型训练时间缩短 43.6%。因此，本文使用的混合特征选择能有效提高模型预测性能和降低模型复杂度从而缩短模型训练时间，验证该方法的有效性。

Q2：根据 Shen 等[10]的建议，使用 DE 算法优化 LightGBM 的重要超参数。DE 算法的初始种群数量和算法迭代次数的取值会影响 DE 算法的性能，因此对种群数量取值

表 6 特征重要度排序

特征	importance	特征	importance	特征	importance	特征	importance
intensity	88	cam	30	acm	13	MNBQty	4
scattering	77	NBQty	23	MOQty	13	PFQty	4
fi-changes	74	SLQty	23	PEQty	10	TCQty	3
ce	66	RQty	22	DFQty	10	moa	2
ostrand	50	FAN-IN	21	dit	10	SMQty	2
UWQty	48	VMQty	21	TFQty	9	LQty	2
loc	48	mfa	19	arl	9	ic	1
cbo	47	avg_cc	18	FFQty	9	DMQty	1
AMQty	42	ca	17	max_cc	9	PVMQty	0
cboM	41	wmc	15	CPQty	8	noc	0
VBQty	40	nosi	15	LSMQty	6	FMQty	0
amc	37	lcom	14	modifiers	6	ICQty	0
rfc	35	npm	14	PTMQty	6	SCMQty	0
FAN-OUT	33	cbm	14	dam	5	ACQty	0
acpd	31	SFQty	13	ana	5	SCFQty	0
TMQty	30	lcom3	13	PVFQty	4	LBDQty	0

表 7 阈值筛选性能比较

阈值	F1 值	AUC 值	训练时间
>0	0.9463	0.9608	30.3599
>10	0.9522	0.9638	25.3245
>30	0.9563	0.9615	22.1347

表 8 特征选择前后模型对比

	F1 值	AUC 值	训练时间
特征选择前	0.9363	0.9408	30.3599
特征选择后	**0.9412**	**0.9548**	**21.1246**

[10,100]，迭代次数取值 [10,500]，以 AUC 值作为优化目标函数返回值分别进行实验，结果如图 2、图 3 所示。

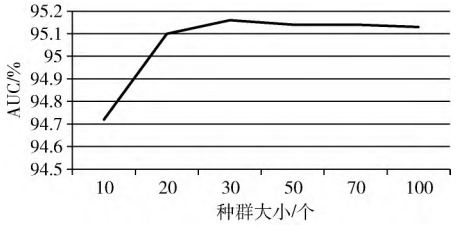

图 2 种群大小比较

从图 2、图 3 可以看出当种群大小和迭代次数分别设为 20 和 100 时 DE 算法可获得最佳性能。通过 DE 算法优化得到一组最优的 LightGBM 超参数组合，即 [n_estimators: 164,max_depth: 178,num_leaves: 63,min_child_sample: 42,learning_rate: 0.16]。通过表 9 可以看出，超参数调优后 F1 值提高 0.98%，AUC 值提高 0.92%。因此可验证通过 DE 算法优化 LightGBM 超参数可提高模型预测精度。

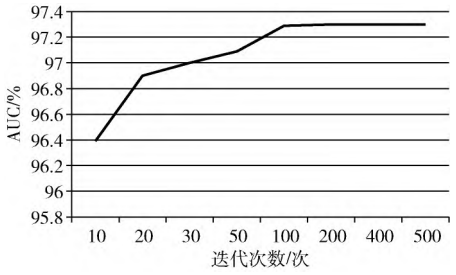

图 3 迭代次数比较

表 9 超参数优化前后性能对比

	F1 值	AUC 值
优化前	0.9412	0.9548
优化后	**0.9510**	**0.9640**

Q3：为了验证本文所提方法 SDL-LightGBM 的有效性，与近年同类论文进行对比。Palomba 等[16]使用逻辑回归建立软件缺陷预测模型。Pritam 等[20]使用多层感知机建立软件缺陷预测模型。对比实验将在相同项目数据集下进行，确保实验具有可比性。从表 10 可以看出，本文所提出的方法 SDL-LightGBM 相比文献 [16] 在 F1 值提高

12.28%，在 AUC 值上提高 18.63%，相比文献［20］在 F1 值上提高 5.67%，在 AUC 值上提高 4.22%。因此可证本文所提方法 SDL-LightGBM 的有效性。

表 10　缺陷预测方法对比

	F1 值	AUC 值
文献[16]	0.8282	0.7777
文献[20]	0.8943	0.9218
SDL-LightGBM	**0.9510**	**0.9640**

Q4：图 4 给出全部数据特征对模型输出结果的贡献度，可见在全局情况下，代码异味强度指数 intensity 对模型贡献度最高，这符合软件工程实际情况，即代码异味越严重越有可能出现缺陷。

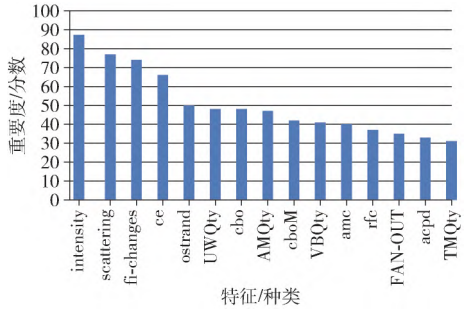

图 4　全局特征对模型贡献度

然而每条实例数据都有各自特点，具体问题要具体分析，因此不能用全局解释的结果来分析单个实例数据。因此本文采用 LIME 解释模型的单个实例。本文选择一条带缺陷的实例数据使用 LIME 解释，如图 5 所示，这条实例预测为有缺陷倾向的 3 个最重要的因素为｛fi-changes ＞ 61.00｝｛scattering ＞ 30.0｝｛intensity ＞ 1.05｝，对缺陷有影响的贡献依次是 0.19，0.19，0.11。而在全局解释中 intensity 对模型贡献度最高。这进一步说明全局解释结果不能解释单个实例。LIME 能计算单个实例数据中每个特征对模型影响的重要性和取值范围，这能够给测试人员分析具体缺陷类提供条件。

4　有效性威胁

本文有效性威胁分析主要从 3 个方面讨论，分别为建立有效性威胁、结论有效性威胁和外部有效性威胁。

建立有效性威胁主要与结构度量和代码异味的测量有关。本文通过 CK[17] 测量结构度量，依靠 JCodeOdor[18] 测量代码异味强度指数。为了验证结构度量和代码异味强度指数的有效性，采用 Spearman 秩相关性去除多重共线性影响，得到的 15 个特征相互之间的共线性均低于 0.8，表明它们之间的相关性较弱。因此，本文模型建立不受特征之

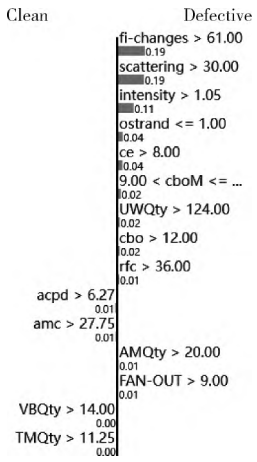

图 5　LIME 解释局部实例

间的多重共线性的威胁。

结论有效性威胁主要与评价指标有关，精确率含义是在所有被预测为正的样本中实际为正的样本的概率，召回率含义是在实际为正的样本中被预测为正样本的概率。前者偏向查准率，后者偏向查全率。为了综合两者优点，本文采用 F1 值，同时考虑查准率和查全率，是两者达到一个平衡。而为了评价模型的好坏，即模型的区分能力，本文引入了 ROC 曲线下的面积 AUC 值。AUC 值可以更全面的衡量一个模型的好坏。因此，本文结论不受评价指标的有效性威胁。

外部有效性威胁主要涉及到结果的泛化性。根据 Palomba 等[16] 的建议，删除缺陷比例高于 75% 的 7 个系统，保证数据的健壮性。同一种方法在不同的应用领域程序的缺陷预测中可能有不同的表现[22]，因此本文分析了来自不同应用领域、具有不同特征（大小、类数等）的 12 个软件系统的 35 个版本，从而提供本文数据的可靠性。

5　结束语

本文提出基于 SDL-LightGBM 集成学习的软件缺陷预测模型，为了避免特征冗余和多重共线的影响，根据 Spearman＋LightGBM 混合特征选择建立特征子集，避免无关特征和特征间的多种共线影响。采用集成学习算法 LightGBM 作为基础分类器。为了进一步提升分类器预测性能，利用 DE 算法优化 LightGBM 的重要超参数，得到一组最佳超参数组合。最后对模型实例进行解释，方便测试人员分析缺陷类。实验结果表明，本文提出的方法 SDL-LightGBM 与其它模型相比取得了更好的预测性能，F1 值平均提高 8.97%，AUC 值平均提高 11.42%。通过混合特征选择后，模型训练时间缩短 43.6%。同时使用 LIME 解释复杂的黑盒模型，生成可解释的特征重要性可视图，进而帮助测试人员更好理解软件缺陷预测模型。

未来的工作包括：①使用基于抽象语法树的神经网络(abstract syntax tree neural network，ASTNN)[21]的深度学习方法捕捉代码的上下文信息，并建构分类器；②进一步考虑其它特征对软件缺陷的影响；③探究本文方法在实际应用场景中的效度。

参考文献：

[1] ZHAO Min，GAO Jianhua. Method for code smell detection based on genetic programming and genetic algorithnm [J]. Journal of Chinese Computer Systems，2020，44 (11)：2434-2441 (in Chinese). [赵敏，高建华. 结合遗传规划和遗传算法的代码异味检测方法 [J]. 小型微型计算机系统，2020，44 (11)：2434-2441.]

[2] Erturk E，Sezer EA. A comparison of some soft computing methods for software fault prediction [J]. Expert Systems with Applications，2015，42 (4)：1872-1879.

[3] Feidu A，Ermiyas B，Bahir S. A literature review study of software defect prediction using machine learning techniques [J]. IJERMT ISSN，2017，6 (6)：2278-9359.

[4] Caram FL，Rodrigues DO，Rafael B，et al. Machine learning techniques for code smells detection：A systematic mapping study [J]. International Journal of Software Engineering and Knowledge Engineering，2019，29 (2)：285-316.

[5] CHEN Liqiong，WANG Can，SONG Shilong. A just-in-time software defect prediction model and its interpretability research [EB/OL]. Journal of Chinese Computer Systems. [2022-03-01]. http://kns.cnki.net/kcms/detail/21.1106.TP.20210506.1049.014.html (in Chinese) [陈丽琼，王璨，宋士龙. 一种即时软件缺陷预测模型及其可解释性研究 [EB/OL]. 小型微型计算机系统. [2022-03-01]. http://kns.cnki.net/kcms/detail/21.1106.TP.20210506.1049.014.html.]

[6] Ke GL，Meng Q，Finely T，et al. LightGBM：A highly efficient gradient boosting decision tree [C] //Proceedings of Advances in Neural Information Systems，2017：3146-3154.

[7] Bashir K，Li T，Yohannese CW，et al. Smotefris-inffc：Handling the challenge of borderline and noisy examples in imbalanced learning for software defect prediction [J]. Journal of Intelligent & Fuzzy Systems，2020，38 (1)：917-933.

[8] Jain S，Saha A. Rank-based univariate feature selection methods on machine learning classifiers for code smell detection [EB/OL]. Evolutionary Intelligence. [2022-03-01]. https://link.springer.com/article/10.1007/s12065-020-00536-z.

[9] Palomba F，Tamburri DA. Predicting the emergence of community smells using socio-technical metrics：A machine-learning approach [J]. Journal of Systems and Software，2021，171 (1)：110847-110854.

[10] Shen L，Liu W，Chen X，et al. Improving machine learning-based code smell detection via hyper-parameter optimization [C] //Proceedings of the 27th Asia-Pacific Software Engineering Conference. Singapore，2020：276-285.

[11] Ribeiro MT，Singh S，Guestrin C. Why should I trust you?：Explaining the predictions of any classifier [C] //Proceedings of the 22nd ACM SIGKDD International Conference. USA，2016：1135-1144.

[12] AI Chenghao，GAO Jianhua，HUANG Zijie. Code smell detection driven by hybrid feature selection and ensemble learning [EB/OL]. Computer Engineering. [2022-03-01]. 10.19678/j.issn.1000-3428.0062165 (in Chinrese). [艾成豪，高建华，黄子杰. 混合特征选择和集成学习驱动的代码异味检测 [EB/OL]. 计算机工程. [2022-03-01]. 10.19678/j.issn.1000-3428.0062165.]

[13] Spearman C. The proof and measurement of association between two things [J]. American Journal of Psychology，1987，3 (4)：441-471.

[14] Tantithamthavorn C，Mcintosh S，Hassan AE，et al. The impact of automated parameter optimization on defect prediction models [J]. IEEE Transactions on Software Engineering，2019，45 (7)：683-711.

[15] Fowler M. Refactoring improving the design of existing code [M]. XIONG Jie，Trans. 1st ed. Beijing：Post & Telecom Press，2010：368-412 (in Chinese). [Fowler M. 重构改善既有代码的设计 [M]. 熊节，译. 1 版. 北京：人民邮电出版社，2010：368-412.]

[16] 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.

[17] Chidamber SR，Kemerer CF. A metrics suite for object oriented design [J]. IEEE Transactions on Software Engineering，1994，20 (6)：476-493.

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

[19] Nucci DD，Palomba F，Rosa GD，et al. A developer centered bug prediction model [J]. IEEE Transactions on Software Engineering，2017，44 (1)：5-24.

[20] Pritam N，Khari M，HSon L，et al. Assessment of code smell for predicting class change proneness using machine learning [EB/OL]. IEEE Access. [2022-03-01]. 10.1109/ACCESS.2019.2905133.

[21] Xu W，Zhang X. Multi-granularity code smell detection using deep learning method based on abstract syntax tree [C] //Proceedings of the 33rd International Conference on Software Engineering and Knowledge Engineering. USA，2021：503-510.

[22] Catolino G. Just-in-time bug prediction in mobile applications：The domain matters [C] //Proceedings of the 4th International Conference on Mobile Software Engineering and Systems，2017：201-202.