文章编号： 1006-3080(2026)02-0257-08 **DOI：** 10.14135/j.cnki.1006-3080.20250422002

基于代码提交信息的方法级软件缺陷预测

董　杭，　虞慧群
（华东理工大学计算机科学与工程系，上海 200237）

摘要：为了提升软件缺陷预测的准确性，本文提出了基于多维度提交特征的方法级缺陷预测框架，提出一组基于代码提交信息的新特征，并结合传统代码及历史特征构建更全面的多维度特征空间，以此构建的模型在17个开源项目上性能显著优于现有技术；通过SHAP(Shapley Additive Explanations)特征重要性分析，证实提交特征具有卓越的预测能力，增强了模型的可解释性；基于识别关键特征进一步简化模型，兼顾了效率与精度。实验结果表明，融合提交信息的模型在性能指标AUC(Area Under the Receiver Operating Characteristic Curve)、F1分数与MCC(Matthews Correlation Coefficient)上分别平均提升了4.3%、8.4%与17.7%。

关键词：软件缺陷预测；方法粒度；提交特征；特征重要性；可解释性

中图分类号：TP311.5　　**文献标志码：**A

在软件工程领域，软件质量保障是确保软件满足预期功能需求与非功能属性的核心环节。随着软件系统复杂性不断提升，软件开发中不可避免地引入的软件缺陷已成为影响软件质量与开发效率的关键因素。软件缺陷如若未能被及时识别修复，不仅可能引发数据泄露或功能失效，还可能导致软件安全隐患。软件缺陷预测基于代码特性与历史数据，可以预测易存在缺陷倾向的软件单元，以帮助开发者在项目早期识别缺陷，从而进一步优化软件测试资源分配并提高缺陷修复效率；有效的软件缺陷预测能够显著提高开发人员解决问题的效率[1]。

在过去的数十年里，研究者们相继提出了基于不同样本粒度的软件缺陷预测方法，包括类/文件粒度、包粒度和模块粒度等。例如，Gyimóthy等[2]基于从Mozilla的原始源代码中提取的代码特征评估软件类的缺陷可能；Schröter等[3]基于代码历史数据与组件交互模式在包粒度下预测容易出现缺陷的组件；而Yang等[4]提出无监督的线性阈值即时缺陷预测模型，基于代码变更粒度进行缺陷预测。然而，软件缺陷预测研究的实际意义近来受到审视与质疑，研究表明现有的缺陷预测可能无法满足任何程序员的现实需求[5]，尤其是软件缺陷预测的粒度。当前大多数研究都基于文件或类粒度预测缺陷，但这样粗糙的粒度在现实应用中是不切实际的：这些研究不能为从业者提供足够信息，使得代码审查仍需要大量时间与人力[6]。Kamei等[7]研究表明：粒度越粗，缺陷预测的实际价值越小；细粒度缺陷预测可以提供更详细具体的信息，从而能够精确地定位缺陷。因此，面向方法粒度的软件缺陷预测受到越来越多的重视。

尽管方法粒度缺陷预测可以通过更精确的信息帮助代码分析，但基于方法粒度的模型探索还相对较少。现有方法粒度缺陷预测主要基于代码特征及其历史变化，只关注传统代码特征而未纳入面向对象代码特征；仅关注源代码历史变化，而未将代码提交过程信息作为额外数据源。本文提出一组基于代码变更提交信息的方法粒度软件度量，并由此构建多维度的方法级缺陷预测模型，旨在显著增强方法粒度缺陷预测准确性与实用性，同时赋予可解释性。

收稿日期： 2025-04-22
基金项目： 国家自然科学基金(62372174)
作者简介： 董　杭(2000—)，男，硕士生，主要研究方向为人工智能与软件工程。E-mail：277397939@qq.com
通信联系人： 虞慧群，E-mail：yhq@ecust.edu.cn
引用本文： 董　杭，虞慧群. 基于代码提交信息的方法级软件缺陷预测 [J]. 华东理工大学学报(自然科学版)，2026，52(2)：257-264.
Citation： DONG Hang, YU Huiqun. Method-Level Bug Prediction Using Code Commit Information[J]. Journal of East China University of Science and Technology, 2026, 52(2): 257-264.

1 缺陷预测方法

软件缺陷预测的具体步骤通常包括数据收集与预处理、特征提取与选择、模型构建与训练、模型性能评估以及后续模型的优化。软件缺陷预测常用的度量特征有代码特征、历史特征、过程特征与代码异味特征等;而常用的分类算法有决策树(DT)、支持向量机(SVM)、逻辑回归(LR)、朴素贝叶斯、随机森林(RF)、梯度提升等机器学习与集成学习方法。为了提升软件缺陷预测模型决策过程的透明度与可信度,开始有研究将可解释性机器学习(XAI)引入软件缺陷预测研究中。XAI能够帮助软件缺陷预测研究实现从黑盒模型到透明决策的前进,XAI中常用的模型不可知方法有LIME(Local Interpretable Model-agnostic Explanations)与SHAP(Shapley Additive Explanations)等。

本文聚焦于方法粒度软件缺陷预测,选取不同领域与不同规模的17个Java开源项目作为研究对象,整体流程框架如图1所示(其中,RQ为研究问题)。首先,采用FCHE(Fine-Grained Code Metrics and History Measures Extractor)[1]、CK以及PyDriller等软件数据挖掘技术,从开源项目源代码中直接生成或间接计算得出包括新提出的提交特征在内的多维度方法粒度软件特征,并以此构建可供分类器学习与训练的方法粒度缺陷预测数据集;其次,通过RF等机器学习分类算法与交叉验证技术构建软件缺陷预测模型进行系统训练,以预测软件代码片段的缺陷倾向性;为评估模型表现,研究将基于指标AUC(Area Under the Receiver Operating Characteristic Curve)、$F1$分数以及MCC(Matthews Correlation Coefficient)[1]进行性能评价,并通过SK-ESD(Scott-Knott Effect Size Difference Test)统计学意义测试以确保实验结果的可靠性与显著性;最后,利用SHAP模型的解释技术分析缺陷预测模型的特征重要性,以解释缺陷预测模型,并依据SHAP技术所识别出的关键特征探索模型的简化,以此兼顾模型的精度和效率。

2 缺陷数据与特征提取

2.1 缺陷预测数据集及其预处理

2.1.1 *数据来源* 为了保证数据源的多样性与代表性,进而体现研究结果的普适性与说服力,本文基于涵盖不同技术领域的17个Java开源项目展开软件缺陷预测实证研究。开源项目均基于稍早版本而非最新版本,以保证所有方法都有足够历史提交信息。表1示出了所有开源项目的详细信息,包括项目版本、提交次数、方法样本数与缺陷数量。

2.1.2 *数据标注* 为了评估软件代码的缺陷倾向,研究者将所有方法样本分为两类:易出错方法与不易出错方法。与过往研究[8]定义类似,如果一个方法曾因修复缺陷而被修改过至少一次,研究就将其标记为易出错方法;反之则是不易出错方法。为了确定方法是否曾因缺陷修复而被变更,研究采用Śliwerski

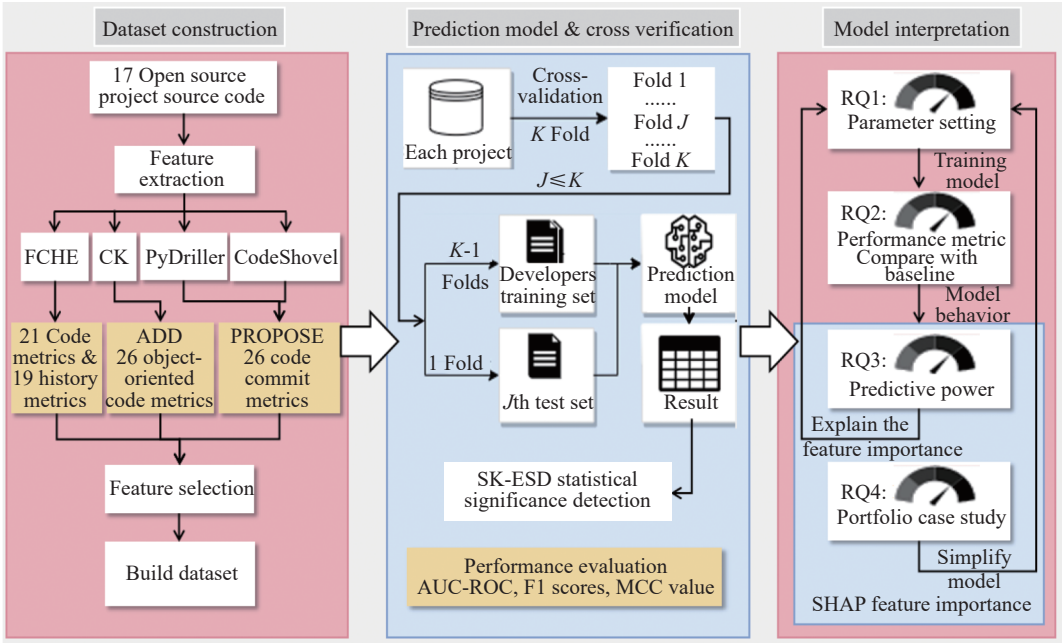

图1 方法粒度缺陷预测模型框架

Fig. 1 Method level bug prediction model framework

表 1　研究项目

Table 1　Research projects

Project	Version	Number		
		Commit	Method	Bug
ActiveMQ	5.16.0	5 920	3 660	1 526
Avro	1.9.1	860	613	223
Calcite	1.24.0	2,971	863	407
Cassandra	3.11.6	10 631	1 309	587
CXF	3.3.7	10 635	6 120	1 248
Drill	1.15.0	2 682	1924	953
Flink	1.8.0	9 950	1 713	444
Flume	1.8.0	1 060	378	266
HBase	3.6.3	3 740	877	420
Ignite	0.94.8	12 009	1 610	200
Kafka	2.6.0	3 338	812	199
Maven	2.5.1	5 591	1 063	211
Nutch	1.17	1 714	617	227
PDFBox	2.0.20	6 883	3 597	1 627
Struts	2.5.20	2 773	763	311
Wicket	8.8.0	14 777	2 112	663
Zookeeper	3.5.8	879	653	358

等[9] 提出的模式匹配方法, 分析项目修改历史与缺陷报告。与先前研究[1] 相同, 如果在代码提交信息中识别到缺陷跟踪报告的缺陷 ID(Bug Ticket ID), 则该提交被认为是为缺陷修复而进行的。鉴于此, 提取了所有缺陷修复提交中被修改的函数与方法, 并将其标记为易出错方法。

2.2　方法粒度的特征提取

2.2.1　*代码特征*　软件缺陷预测首先依赖衡量源代码特性的代码特征。代码特征通过量化代码规模等特性为预测提供输入数据, 而代码行数 (LOC) 等传统代码特征已不足以为模型提供足够信息; 因此在 FCHE 21 个传统代码特征基础上, 补充引入 26 个面向对象设计的 CK 代码特征, 具体见表 2。

需要说明的是, CK 能够生成 Java 中类粒度与方法粒度的代码特征且包含一组广泛的特征指标, 但并非所有特征都适用于方法粒度, 因此研究筛除少数只适用于类粒度而在方法粒度没有意义的代码特征; 与此同时, CK 特征中代表方法间耦合度的扇入 (Fan-in) 与扇出 (Fan-out) 已在 FCHE 代码特征集中涉及, 因此基于代码特征补充引入的实验目的, 本文不将其列在 26 个 CK 代码特征中。

2.2.2　*变更与提交特征*　软件缺陷预测同样依赖对代码修改历史的量化评估。代码变更特征能通过量化代码的修改频率和模式预测代码的潜在风险。研究在 FCHE[1] 的 19 个代码历史特征基础上, 设计提出一组基于变更提交信息的全新特征, 具体见表 3。其中, NF1、NF2、NF3、NF4 分别代表首次提交时间、最近提交时间、平均提交时间间隔以及提交时间间隔标准差。通常认为, 更早的首次提交时间与更长的提交间隔意味着方法的缺陷倾向更低, 而最近的提交更有可能导致缺陷; 若只用平均值衡量方法提交间隔并不能充分捕获提交时间间隔的全部信息, 因为其只与首次提交时间、最近提交时间以及提交次数有关, 因此本文引入标准差评估提交时间间隔。NF5~NF8 是关于代码提交变更类型的特征, 而变更类型包括添加、删除、修改与重命名。本文不仅关注各变更类型提交的次数统计, 还进一步推导出百分比统计 NF9~NF12, 并认为修改类型的变更提交更可能引入代码缺陷。而 NF13~NF14 则定义为方法在每次提交中同时变更的其他方法的平均数量, 其中 NF14 仅统计涉及修改的提交; 这一组提交规模特征在一定程度上能反映方法间耦合程度。过往研究[10] 中, 平均共变方法数 AVG_CS 已被证明与其缺陷倾向相关, 因此将其应用于方法粒度缺陷预测研究。

本文通过 PyDriller 分析项目 Git 仓库中的代码提交历史并提取提交的详细信息, 再间接计算得出上述这组方法粒度代码提交特征。

3　性能评价指标

软件缺陷数据集通常表现出易出错与不易出错间的类不平衡, 易出错方法样本作为预测正例往往只占所有样本的至多 20%; 鉴于此, 准确率等传统指标不适合评价缺陷数据集, 因此使用 AUC、*F*1 分数与 MCC 评估模型性能: AUC 能反映模型在不同分类阈值下区分正负样本的能力, *F*1 分数在类不平衡情境下能够综合考虑模型精确性和全面性但可能掩盖召回率问题, MCC 考虑所有 4 种混淆矩阵值而更具一致性与解释力。其计算公式如下:

$$\mathrm{TPR}=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{1}$$

$$\mathrm{FPR}=\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} \tag{2}$$

$$\mathrm{AUC}=\int_0^1 \mathrm{TPR}\,\mathrm{d}\,\mathrm{FPR} \tag{3}$$

$$\mathrm{Precision}=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{4}$$

表2 代码特征

Table 2 Code metrics

Index	Code metric	Index	Code metric
C1	LOC	CK4	WMC
C2	Number of comment lines	CK5	Number of returns
C3	Number of all lines	CK6	Number of loops
C4	Number of blank lines	CK7	Number of comparisons
C5	Number of declare lines	CK8	Number of try/catches
C6	Number of executable lines	CK9	Number of parenthesized expressions
C7	Number of parameters	CK10	Number of string literals
C8	Number of statements	CK11	Number of numbers
C9	Number of declare statements	CK12	Number of assignments
C10	Number of executable statements	CK13	Number of math operations
C11	Halstead-vocabulary	CK14	Number of variables
C12	Halstead-length	CK15	Max nested blocks
C13	Halstead-difficulty	CK16	Number of anonymous methods
C14	Halstead-volume	CK17	Number of inner methods
C15	Halstead-effort	CK18	Number of lambda expressions
C16	Halstead-defects	CK19	Number of unique words
C17	Cyclomatic complexity	CK20	Number of log statements
C18	Quantity of path	CK21	[Boolean] Has javadoc
C19	MaxNesting	CK22	Modifiers
C20	Fan-in	CK23	Number of methods invoked
C21	Fan-out	CK24	Number of methods invoked local
CK1	CBO	CK25	Number of methods invoked indirect local
CK2	CBO Modified	CK26	[Boolean] Constructor
CK3	RFC		

表3 变更历史与提交特征

Table 3 History & Commit metrics

Index	History metric	Index	Commit metric
H1	Added LOC	NF1	First commit time
H2	Deleted LOC	NF2	Last commit time
H3	Changed LOC	NF3	Average of commit time interval
H4	Number of changes	NF4	Std. deviation of commit time interval
H5	Number of authors	NF5	Number of added change commit
H6	Number of modified statements	NF6	Number of deleted change commit
H7	Number of modified expressions	NF7	Number of modified change commit
H8	Number of modified comments	NF8	Number of renamed change commit
H9	Number of modified return type	NF9	Percentage of added change commit
H10	Number of modified parameters	NF10	Percentage of deleted change commit
H11	Number of modified prefix	NF11	Percentage of modified change commit
H12	Added LOC / LOC	NF12	Percentage of renamed change commit
H13	Deleted LOC / LOC	NF13	Average commit size
H14	Added LOC / Deleted LOC	NF14	Average modified commit size
H15	Changed LOC / quantity of changes		
H16	Quantity of modified statements / quantity of statements		
H17	Quantity of modified expressions / quantity of statements		
H18	Quantity of modified comments / LOC		
H19	Quantity of modified parameters / quantity of parameters		

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{5}$$

$$F1 = \frac{2\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}} \tag{6}$$

$$\text{MCC} = \frac{\text{TP}\cdot\text{TN}-\text{FP}\cdot\text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}} \tag{7}$$

其中，TP、TN、FP与FN均是二元分类检验结果：TP是正确预测的正例，TN是正确预测的反例，FP是错误预测的正例，FN是错误预测的反例。

SK-ESD能基于比较多组均值确定组间差异的大小，还通过处理输入数据的非正态分布并根据效应大小系统去除非显著组。因此，本文还通过SK-ESD测试衡量模型性能变化的显著程度，并使用R语言ScottKnottESD包进行显著检测、分组排序和分离。

4 实验结果分析

4.1 参数设置与分类器选择

为了确定分类器的最佳参数设置，实验采用十折交叉验证穷举网格搜索机器学习分类器的超参数，而不是使用默认设置。超参数调整的粒度是在每个开源项目中单独进行参数调优：对于17个不同开源项目，为每个项目设置独特的超参数；而在特定一个项目中，使用相同的超参数集在不同的特征子集上进行实验。本文在选择最佳机器学习分类器上也遵循相同标准：为避免庞大的搜索空间，实验专注

于调整分类器中最具响应性的参数。例如, 在随机森林模型中, 实验主要聚焦于调整 *n*_estimators 与 max_depth 这两个关键参数。其中, 参数 *n*_estimators 的取值范围是从 10 到 300, 步长为 10; 而超参数 max_depth 的取值范围是从 3 到 21, 步长为 1。

不同分类器基于 AUC 的实验结果如图 2 所示(图中, LDA 为线性数据分析, KNN 为 k 邻近算法, MLP 为多层感知器)。图 2 直观地展示了 RF 在所有 17 个项目中都能够表现出最强的预测能力; 在多轮训练后, 确定 RF 为后续实验的算法模型。

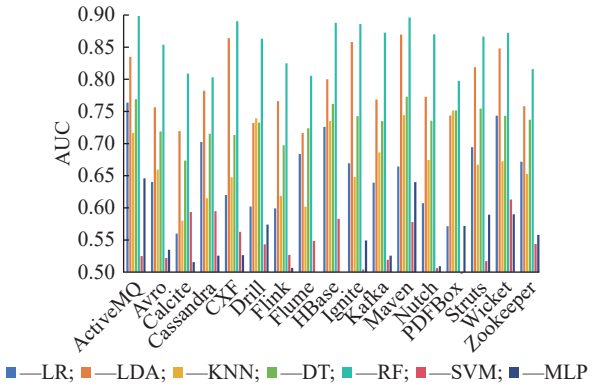

图 2　不同分类器的 AUC 值

Fig. 2　AUC values of different classifiers

4.2　模型性能

为了验证本文提出方法的有效性, 将实验分为 4 组: 第 1 组基于现有研究[1]的 21 个代码特征与 19 个历史特征进行缺陷预测, 记为 FCHE(Base), 这也是基线模型; 第 2 组基于基线模型 Base 引入了 26 个面向对象的方法级代码特征 CK, 即表 1 中的额外特征, 记为 FCHE+CK; 第 3 组在基线模型 Base 基础上添加本文新提出的 14 个提交特征 NF, 即表 2 中的额外特征, 记为 FCHE+NF; 第 4 组是基于集成以上全部的 80 个方法粒度特征的综合模型, 记为 FCHE+CK+NF。

使用最佳分类器 RF 构建方法粒度软件缺陷预测模型, 并采用十折交叉验证以保证实验结果的可靠性。实验模型表现基于 AUC、F1 分数与 MCC 进行多维度性能评价; 并将所有实验结果通过 SK-ESD 统计学显著性测试, 以保证实验模型的性能提升在统计学意义上是显著的。

各组实验模型基于 AUC、F1 分数与 MCC 的预测性能表现分别如图 3、图 4 与图 5 所示。可以得出: 引入 CK 代码特征在一定程度上提升了模型的预测准确性, AUC 平均提高 1.3%; 本文提出的提交特征 NF 显著优化了模型性能, AUC 平均提高 4.0%; 而综合模型 FCHE+CK+NF展现出强大的最佳性能表现, 在 AUC、F1 分数与 MCC 上分别平均提升 4.3%、8.4% 与 17.7%, 且经 SK-ESD 统计检验验证这种增强在统计上是显著的。实验结果充分证实了引入 CK 特征的积极作用与提出提交特征对方法粒度缺陷预测的显著贡献, 同时也印证了静态代码固有属性与动态开发提交特征的协同效应对缺陷预测具有系统性的增强作用。

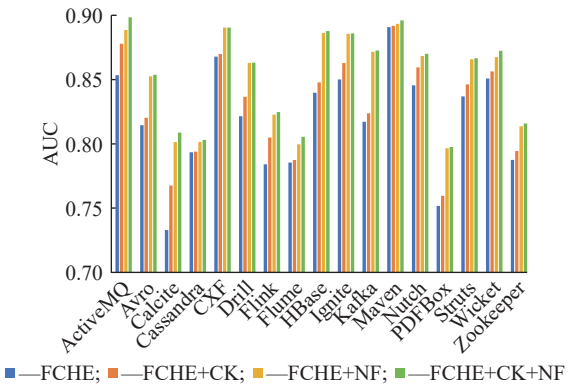

图 3　不同特征集的 AUC 值

Fig. 3　AUC values of different feature subsets

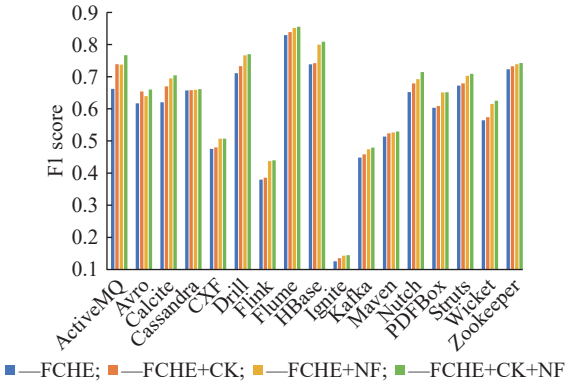

图 4　不同特征集的 F1 分数

Fig. 4　F1 scores of different feature subsets

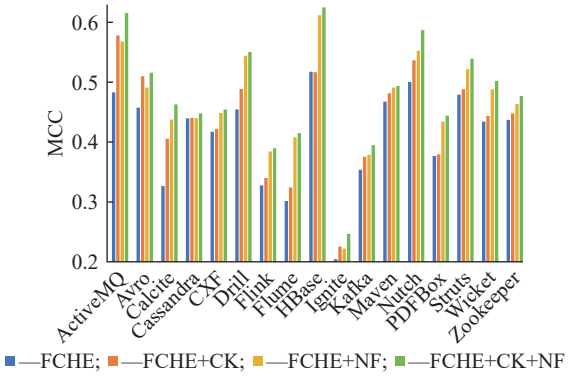

图 5　不同特征集的 MCC 值

Fig. 5　MCC values of different feature subsets

在项目 Ignite 中所有实验模型的 F1 分数都不理想, 分析认为, 这是在面对类不平衡数据集时预测模型过拟合所致, 高精确率与低召回率的失衡使得 F1 分数被限制在较低的范围内, 后续研究中将通过

集成学习的分类器堆叠技术对算法的准确程度与泛化能力进一步进行探索增强。

4.3 模型解释

为了解析方法粒度模型中各特征的贡献程度，同时增强模型的可解释性，本文采用SHAP算法对模型特征重要性及特征间交互进行分析。

SHAP模型解释技术的理论基础源自博弈论中的Shapley值，而每个特征的SHAP值能量化衡量该特征对模型预测结果的贡献程度，若SHAP为正值，则代表对预测结果有正面影响；负值则代表产生了负面影响。SHAP模型解释算法的数学形式可以表示为基于二元变量的线性函数：

$$\vartheta(z)=\varphi_0+\sum_{i=1}^{M}\varphi_i\cdot z_i \tag{8}$$

SHAP将实例特征转化为简单的二元特征作为输入，以构建线性解释模型ϑ。在SHAP模型中，特征向量$z\in\{0,1\}^M$代表简化特征，M为简化特征数量，向量z的每个元素z_i为二元指标：z_i=1表示第i个特征包含在模型中，z_i=0表示被排除在外；项φ_0为模型平均预测值，φ_i表示第i个特征对模型预测贡献的特征归约Shapley值，φ_i的正值越大说明第i个特征对模型的正向预测影响越显著。

针对每个项目，计算所有80个方法级特征SHAP值并降序排序；特征排名越高，代表提供的信息越多，意味预测缺陷时越重要。所有特征排名的跨项目分布箱线图如图6所示，根据图中信息可知以下规律：同一特征的SHAP排名在不同项目中存在波动，表明特征在缺陷预测中具有项目特异性；传统代码及历史特征箱线图分布较宽，说明其重要性高度依赖项目上下文而缺乏普适性；本文提出的代码提交特征NF在跨项目特征重要性评估中表现突出，箱线图分布集中于高位且范围紧凑，在众多开源项目中具有稳定且强大的预测能力。

表4示出了每个特征SHAP值的平均排名，根据表中数据可知：相较于基于FCHE特征预测能力表现分化，CK代码特征仅少数体现差强人意的预测能力；本文所提出的代码提交特征NF在排名中占据显著优势。其中，NF1位居榜首且NF1~NF4均位于排名前10%，这有效验证了代码提交时间[11]在方法粒度缺陷预测仍能有强大能力；关乎修改的NF7与NF11相对更有效，而NF13~NF14也贡献突出。这证明了代码提交特征NF的预测能力具有系统性优势，同时也验证了代码开发过程中的提交信息对软件缺陷预测的关键作用。

4.4 模型简化

模型简化对提高计算效率与可解释性至关重要，通过基于SHAP的关键特征识别与选择能够在降低模型复杂度的同时保留或提升预测性能。实验证实提出的代码提交特征对方法级缺陷预测具有显著作用，但现有模型依赖80个特征，可能导致工程实践中的复杂性。为验证是否能在减少特征数量的同时保留模型有效性，研究基于SHAP分析中排名前

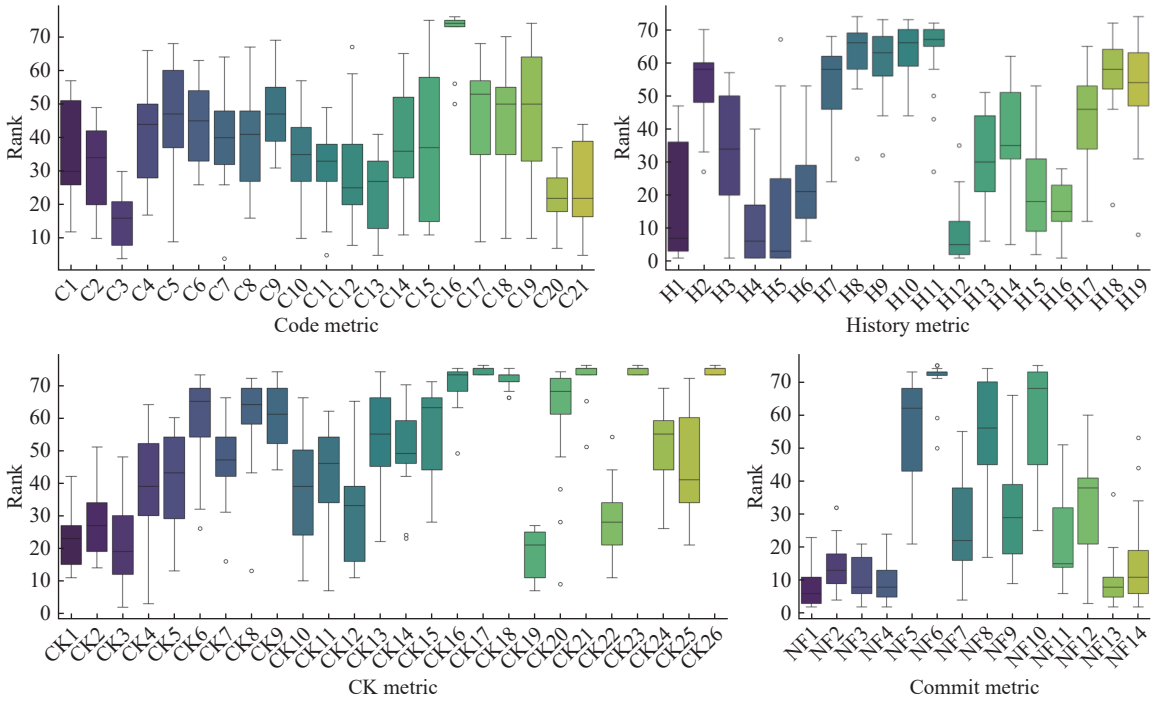

图6 特征排序分布箱线图

Fig. 6 Feature rank distribution

表 4　特征平均排名

Table 4　Average feature rank

Index	Rank	Index	Rank	Index	Rank	Index	Rank
NF1	8.1	NF7	26.4	C8	40.3	NF5	55.6
H12	9.0	C21	26.5	CK5	40.7	CK15	55.7
NF4	9.5	CK22	28.2	C4	40.8	H18	56.0
H4	9.7	CK2	28.5	H17	43.2	NF10	57.7
NF3	10.1	NF9	29.7	CK11	43.5	CK8	59.5
NF13	10.2	C15	29.8	C6	44.0	CK6	59.9
H5	14.8	C11	30.0	C18	44.7	CK20	59.9
NF2	15.0	H13	31.4	CK25	45.2	CK9	60.4
C3	15.2	C2	31.5	C17	45.3	H9	60.4
H16	16.4	C10	31.7	C9	45.8	H8	62.9
NF14	16.9	NF12	31.9	CK7	47.4	H11	62.9
H1	17.4	C12	32.0	C5	47.7	H10	63.4
CK19	17.7	CK12	32.2	CK14	50.1	CK16	69.8
C20	20.5	H3	33.4	CK24	51.7	NF6	70.8
H15	20.6	C1	34.8	NF8	51.8	CK18	71.5
CK3	21.1	C14	37.1	H19	52.0	CK21	72.5
CK1	22.4	CK10	37.2	CK13	52.8	CK23	74.4
NF11	22.8	H14	37.7	H2	53.1	CK17	74.4
H6	23.9	CK4	38.5	H7	53.2	CK26	74.4
C13	24.1	C7	40.0	C19	53.5	C16	74.4

5%、10%、20% 和 30% 的关键特征构建简化模型, 以降低数据收集和分析成本, 并使预测模型更有可能实际应用。

不同规格简化模型的性能表现如图 7 所示, 图中数据表明: 与原始模型相比, 基于前 5% 关键特征的简化模型可能因特征不足导致性能下降, 而基于前 10%~30% 特征的模型性能接近原始模型; 部分项目中, 基于前 30% 特征的简化模型可能因减少 CK 冗余特征而表现更优; 不同项目对关键特征比例的敏感度存在差异。需要强调的是, 模型简化与特征空间扩展并不冲突。扩展特征空间通过增加维度获取更多代码信息, 而简化模型通过特征选择在不牺牲性能的前提下降低数据收集和分析成本, 两者共同提升模型的实用性与效率。

综上, 实验证明基于 SHAP 识别的关键特征能够构建性能满意的软件缺陷预测简化模型。随着模型中特征的增多, 虽模型性能也随着提升, 但同时上升的还有计算成本, 因此综合考虑计算效率与性能平衡, 推荐优先使用基于前 10% 至前 20% 关键特征的模型。

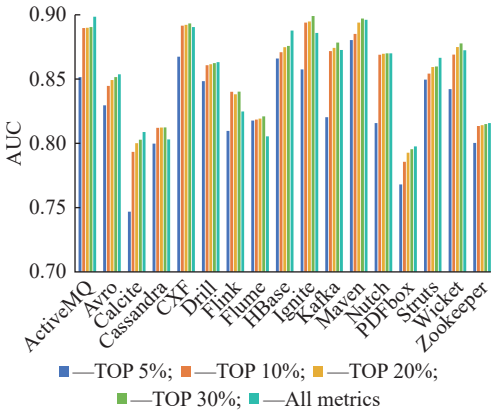

图 7　简化模型的 AUC 值

Fig. 7　AUC values using top features

4.5　性能对比

本文引入更多基线模型进行对比分析, 基线模型如下: (1) Menzies等[8] 基于代码行数、McCabe 复杂度和 Halstead 度量等静态属性构建的缺陷预测模型; (2) Giger 等[6] 分析代码与变更特征并结合 Eclipse 项目版本控制与问题追踪系统数据的技术, 以预测可能包含缺陷的方法。所有模型均采用最优分类器随机森林算法进行对比验证, 本文实验结果选取 FCHE+CK+NF 组。

上述模型的 AUC 值对比结果见图 8, 由图可见, 本文提出的研究方法其性能模型表现优势显著。相较于 Menzies 等[8] 的研究, 本文模型在 AUC 值上取得 9.6%~37.3% 的性能提升; 而相较于 Giger 等[6] 的研究, 在所有开源项目中, 本文模型在 AUC 值上有 3.9%~26.6% 的更好的性能表现。

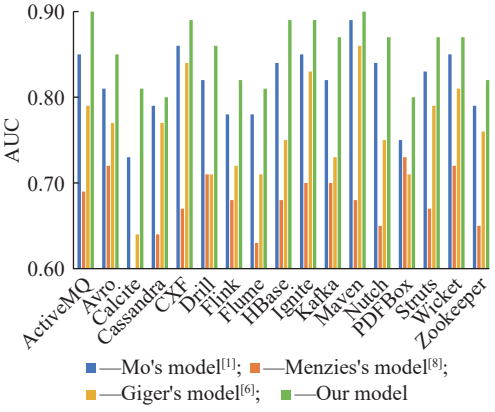

图 8　对比基线模型的 AUC 值

Fig. 8　AUC values with compared work

5　结束语

本文提出一种结合代码提交信息的方法粒度软

件缺陷预测方法。首先,基于代码变更的提交时间、提交变更类型以及提交规模大小3个子维度,设计并提出一组基于代码提交信息的方法粒度软件度量;接着从扩展度量空间与优化特征工程角度出发构建多维度预测模型,该模型在AUC、F1分数与MCC上的平均增强分别达到4.3%、8.4%和17.7%;最后,通过SHAP技术分析各个特征的预测贡献程度,以此解释并简化了预测模型。

参考文献:

[1] MO R, WEI S, FENG Q, *et al*. An exploratory study of bug prediction at the method level[J]. Information and Software Technology, 2022, 144: 106794.

[2] GYIMÓTHY T, FERENC R, SIKET I. Empirical validation of object-oriented metrics on open source software for fault prediction[J]. IEEE Transactions on Software Engineering, 2005, 31(10): 897-910.

[3] SCHRÖTER A, ZIMMERMANN T, ZELLER A. Predicting component failures at design time[C]//ACM/IEEE International Symposium on Empirical Software Engineering. China: ACM/IEEE, 2006: 18-27.

[4] YANG Y, ZHOU Y, LIU J, *et al*. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models[C]//ACM SIGSOFT International Symposium on Foundations of Software Engineering. USA: ACM, 2016: 157-168.

[5] PASCARELLA L, PALOMBA F, BACCHELLI A. On the performance of method-level bug prediction: A negative result[J]. The Journal of Systems and Software, 2020, 161: 110493.

[6] GIGER E, D'AMBROS M, PINZGER M, *et al*. Method-level bug prediction[C]//ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. Sweden: ACM, 2012: 171-180.

[7] KAMEI Y, SHIHAB E. Defect prediction: Accomplishments and future challenges[C]// IEEE International Conference on Software Analysis, Evolution, and Reengineering. Japan: IEEE, 2016, 5: 33-45.

[8] MENZIES T, GREENWALD J, FRANK A. Data mining static code attributes to learn defect predictors[J]. IEEE Transactions on Software Engineering, 2006, 33(1): 2-13.

[9] ŚLIWERSKI J, ZIMMERMANN T, ZELLER A. When do changes induce fixes?[J]. ACM Sigsoft Software Engineering Notes, 2005, 30(4): 1-5.

[10] CHEN X, XIA H, PEI W, *et al*. Boosting multi-objective just-in-time software defect prediction by fusing expert metrics and semantic metrics[J]. Journal of Systems and Software, 2023, 206: 111853.

[11] SHEHAB M A, KHREICH W, HAMOU-LHADJ A, *et al*. Commit-time defect prediction using one-class classification[J]. Journal of Systems and Software, 2024, 208: 111914.

Method-Level Bug Prediction Using Code Commit Information

DONG Hang, YU Huiqun

(Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China)

Abstract: Software bug prediction is a vital aspect of software quality assurance and has become a key research area in software engineering. However, current prediction technologies face two main challenges: First, coarse-grained bug prediction often fails to meet the practical needs of industry. Second, existing models have limited adaptability to dynamic development processes and rely heavily on static code features and historical data, making it difficult to effectively capture code changes and commit information. To tackle these issues, this paper presents a method-level bug prediction framework that utilizes multi-dimensional commit features to improve prediction accuracy. The primary innovation lies in introducing a novel set of features derived from code commit information, which are combined with traditional code and historical features to create a more comprehensive feature space. This model significantly outperforms existing technologies across 17 open-source projects. SHAP-based feature importance analysis further confirms that the commit features possess strong predictive capabilities while enhancing model interpretability. By identifying key features, the model is streamlined without compromising efficiency or accuracy. Experimental results show that incorporating code commit information increases AUC (Area Under the Receiver Operating Characteristic Curve) by an average of 4.3%, F1 score by 8.4%, and MCC (Matthews Correlation Coefficient) by 17.7%.

Key words: software bug prediction; method level; commit metric; feature importance; interpretability

(责任编辑:李娟)