计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
2025,34(10):133−142 [doi: 10.15888/j.cnki.csa.009964] [CSTR: 32024.14.csa.009964]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

同构集成深度学习的代码异味检测①

张　苗[1], 黄子杰[2,3], 陈军华[1], 高建华[1]

[1](上海师范大学 信息与机电工程学院, 上海 201418)
[2](上海计算机软件技术开发中心, 上海 201112)
[3](上海市计算机软件评测重点实验室, 上海 201112)
通信作者: 黄子杰, E-mail: huangzj@sscenter.sh.cn

摘　要: 代码异味是软件源代码中存在的不良设计问题, 它影响着代码可维护性, 可读性以及可扩展性. 研究人员可以通过检测代码异味确定待重构的代码, 进而改善代码质量. 过去的研究广泛将机器学习技术应用于代码异味检测, 虽取得一定成果, 但在检测准确性、泛化能力方面仍存在局限. 本文构建两种模型: 直接同构集成 CNN 和 RNCNN (random forest-negative correlation learning-CNN), 来检测 4 种代码异味 (Data class, God class, Feature envy, Long method), 利用基于随机森林的特征选择技术提取更重要的特征, 利用深度学习技术对代码的结构特征进行建模, 再使用负相关学习同构集成深度学习模型方法训练模型, 进行异味检测. 与基线工作相比, 本研究的 *F*1 值最高提升 4.93%.

关键词: 代码异味; 特征选择; 深度学习; 负相关学习; 集成学习

引用格式: 张苗,黄子杰,陈军华,高建华.同构集成深度学习的代码异味检测.计算机系统应用,2025,34(10):133–142. http://www.c-s-a.org.cn/1003-3254/9964.html

Code Smell Detection Based on Homogeneous Ensemble Deep Learning

ZHANG Miao[1], HUANG Zi-Jie[2,3], CHEN Jun-Hua[1], GAO Jian-Hua[1]

[1](College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai 201418, China)
[2](Shanghai Development Center of Computer Software Technology, Shanghai 201112, China)
[3](Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai 201112, China)

Abstract: Code smells are undesirable design characteristics in software source code that negatively affect maintainability, readability, and scalability. By detecting code smells, it becomes possible to identify code segments requiring refactoring, thus enhancing code quality. Although machine learning techniques have been widely applied in previous studies for code smell detection, challenges persist regarding detection accuracy and generalization capability. In this study, we propose two models: a directly isomorphic ensemble model combining convolutional neural network (CNN) and a random forest-negative correlation learning-CNN (RNCNN), to detect four types of code smells: Data class, God class, Feature envy, and Long method. A feature selection technique based on random forests is employed to extract the most informative features, and deep learning techniques are utilized to model the structural characteristics of the code. The model is then trained using a homogeneous ensemble deep learning approach based on negative correlation learning. Experimental results demonstrate that the proposed method improves the *F*1 score by up to 4.93% compared to baseline approaches.

Key words: code smell; feature selection; deep learning; negative correlation learning; ensemble learning

代码异味[1]指软件源代码中存在的不良设计问题, 它会降低代码的可读性和运行效率, 增加维护成本. Fowler 等人[2]在 1999 年提出了代码异味的概念, 定义了 22 种类型代码异味, 并针对每种代码异味提出了重

① 基金项目: 中国博士后科学基金面上项目 (2024M761927); 上海市“科技创新行动”启明星项目 (扬帆专项) (24YF2719900)
收稿时间: 2025-02-26; 修改时间: 2025-03-18; 采用时间: 2025-04-07; csa 在线出版时间: 2025-08-26
CNKI 网络首发时间: 2025-08-27

构解决方法. 面对存在异味的程序, 即使能够正常运行并通过测试, 随着软件生命周期的延长, 这些代码异味会逐渐对程序的可维护性和可读性产生干扰. 如果不及时检测和处理代码异味, 将对开发人员的工作效率和软件质量产生不利影响[3,4]. 代码异味的检测需求催生了 PMD、SonarQube 等动静态程序分析工具, 应用于 IBM、阿里巴巴等大型企业, 支持其开发、维护和持续集成等阶段的软件质量保障. 上述方法已取得了一定的成效, 但仍然存在依赖经验性阈值、假阳性结果多等影响其应用的问题.

为了避免依赖阈值、进一步提升检测效果, 学者们采用了机器学习方法来检测代码异味[5], 包括有监督学习和无监督学习方法. 有监督学习方法[6]使用已标注的数据, 通过机器学习方法识别特定的代码异味. 无监督学习方法[7]则在数据未标注的情况下, 通过聚类等方法识别潜在的代码异味, 但其采用未标注的数据集, 根据真实结果对比预测结果的普遍评估标准则难以使用, 并且无监督学习无法充分捕捉特征与变量的复杂关系.

目前, 在模型选用方面, 许多研究人员采用机器学习模型进行研究, 使用决策树、SVM 等机器学习方法[8]对代码异味进行预测; 而单一的机器学习模型可能存在泛化能力不够. 于是集成机器学习方法逐步增加, 采用 Stacking 集成[9]或 Bagging 异构[10]集成机器学习等方法对代码异味进行检测. 然而, 机器学习在研究上能获得较好效果, 但机器学习对数据依赖性强, 模型迁移能力有限, 存在一定的缺陷[11].

在此背景下, 深度学习能够自动挖掘特征与变量之间的复杂关系, 适应能力强, 具备卓越的泛化能力, 同时在多个领域已取得显著成效. 并且在本文研究方向上, 同构集成深度学习的研究较少, 因此, 本文旨在探讨深度学习技术在代码异味检测中的应用, 采用两种集成策略: 直接集成深度学习模型和使用负相关学习训练的同构集成深度学习模型, 以此增加代码异味方面的集成深度学习研究. 直接集成简化模型结构, 提高检测效率; 而引入负相关学习思想, 通过在 CNN 集成中加入负相关损失项, 促进了模型之间的多样性, 从而提升集成效果.

在特征选择方面, 无论采用公开数据集或自行构建数据集, 都可能存在特征冗余问题, 冗余特征可能对模型预测代码异味的性能产生负面影响. 除了统计筛选特征方法, 还有采用基于决策树的特征选择等方法. 由于随机森林模型在单一机器学习模型中预测性能较好[6,8], 本文采用随机森林的特征选择方法进行重要性特征提取. 同时, 避免数据集存在数据不平衡问题, 本文采用过采样技术平衡数据集.

综上所述, 本文尝试融合特征选择、数据平衡技术、深度学习和负相关学习的方法进行代码异味检测. 采用 Fontana 等人[6]提供的 4 个带标签的代码异味数据集, 通过特征选择提取关键特征, 并将其输入到深度学习集成模型中, 最终通过评估指标分析其性能表现.

本文的贡献主要包括以下 3 个方面.

(1) 提出融合特征选择、数据平衡技术与深度学习集成学习的代码异味检测的思路, 并采用两种集成策略实现模型: ① 直接集成 CNN 的模型; ② 使用负相关学习训练集成多个 CNN 的 RNCNN. 随后, 经过十折交叉验证与基线工作进行比较.

(2) 两种模型在公开的 4 种数据集上进行训练测试, 并得到良好的结果, 证明模型的准确性.

(3) 通过实验探索了两种集成策略对代码异味检测的提升, 计算两种集成模型准确率、召回率、*F*1 值、精确率和 AUC 值, 进行比较分析. 同时本文所有实验代码已开源 (https://github.com/fly0-l/project1).

本文第 1 节介绍相关工作. 第 2 节介绍研究背景. 第 3 节提出两种集成深度学习的代码异味检测方法. 第 4 节给出对所提出的方法评估和分析讨论. 第 5 节提出结论与展望.

1 相关工作

早期的代码异味检测方法基于经验评价和主观因素, 例如 Travassos 等人[12]采用人工检查来识别代码异味. Mens 等人[13]提出了一种基于逻辑规则的异味检测算法, 该算法用于检测编译后的代码是否符合相应的规则. 在近期研究中, Prokić等人[14]设计了一种用于手动标注代码异味的规范化程序, 构建了可用于训练机器学习模型的高质量数据集. 除此之外, 研究者们还提出了不同的研究技术.

1.1 基于动态和静态程序分析的代码异味检测技术

在工具识别代码异味方面, 研究者利用源代码的结构属性或通过文本分析检测代码异味. 例如, Tsantalis 等人[15]提出了一种解决 Feature envy 问题的重构方法, 该方法通过度量方法与类之间的相关性和距离概念, 这一度量量化了实体在系统类中的布局效果. 近期研究

中, Yu 等人[16]利用静态代码分析工具来自动收集代码异味数据, 同时使用 GCN (图卷积网络) 和 Transformer 编码器对代码语义和代码度量进行融合检测代码异味. Prestat 等人[17]提出了一种名为 DynAMICS 的方法用于规范和动态监测 Android 行为代码异味, 并提出了一个实现该方法的工具.

1.2 基于机器学习的代码异味检测技术

基于度量或依赖工具的方法通常需要人工干预, 不同开发者选择的度量指标可能存在差异, 可能受到度量指标选择、映射阈值设定等主观因素影响. 相比之下, 采用机器学习方法可以减少预测过程中的主观性. Fontana 等人[6]在 4 种代码味异味 (数据类、过大类、依恋情结、过长函数) 和 74 个软件系统上实验了 16 种不同的机器学习算法, 比较了其检测性能. Gupta 等人[7]使用无监督机器学习方法有效检测异味. 同时, 集成学习是机器学习领域的重要研究方向. 集成模型相较于单个分类器能表现出更好的性能. Alazba 等人[9]采用堆叠异构集成多个机器学习分类器, 提高了分类性能. 黄晨峻等人[18]采用多模型软投票集成多个机器学习方法对代码异味进行检测.

1.3 基于深度学习代码异味检测技术

深度学习广泛利用于各个领域. Liu 等人[19]采用反向重构的方法, 自动标记大量训练数据, 使用卷积神经网络 (CNN) 模型训练度量数据, 使用长短期记忆网络 (LSTM) 模型对语义进行训练. Nizam 等人[20]从 GitHub 仓库收集代码并使用 SonarQube 工具检测代码异味创建了实验数据集, 他们应用长短期记忆网络 (LSTM) 和变换器 (Transformers) 实现了代码异味检测. 边奕心等人[21]使用 CNN 模型对 Android 中的缓慢循环代码异味进行检测, 相较于机器学习方法有明显提升. Zhang 等人[22]则利用 3 种经典的图神经网络 (GNN) 检测大类代码异味.

2 研究背景

2.1 代码异味介绍

本研究选择了 4 种常见的代码异味进行检测实验. 这些异味具有较高出现频率, 可能对软件质量造成较大负面影响, 并且已有相关的检测规则或工具实现. 代码异味详细描述如表 1 所示.

2.2 数据归一化技术

数据集在不同数据范围会影响模型效果, 导致模型过拟合或者欠拟合, 因此最好在数据预处理过程中, 将数据规范到同一范围. 本文采用了最小-最大归一化 (min-max normalization) 技术[23], 将数据缩放到 0–1 之间. 式 (1) 是最小-最大归一化, 其中 X 是原始数据, $X_{\min}$ 和 $X_{\max}$ 分别是数据的最小值和最大值, X' 是归一化后的数据.

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{1}$$

表 1 4 种代码异味

代码异味	级别	描述
Data class (数据类)	类级	当类没有所有适当方法
Feature envy (依恋情结)	方法级	当一种方法对其他类功能更感兴趣
God class (上帝类)	类级	当类有太多任务或者包含许多行代码
Long method (过长函数)	方法级	当类中方法具有过长的代码行数

2.3 特征选择

特征选择能够剔除不相关或者噪声特征, 有助于降低模型计算成本, 提高模型训练效率和模型预测性能. 特征选择[24]包括 Filter (过滤法)、Wrapper (包装法) 和 Embedded (嵌入法). 过去的研究表明, 随机森林在代码异味检测中的表现优越. 基于此, 本文采用了随机森林的特征提取方法.

随机森林 (random forest, RF) 是一种常见的机器学习算法. 本次研究采用基于随机森林的特征选择, 随机森林在分裂节点会计算特征重要性, 计算策略有根据分裂节点时的基尼不纯度 (Gini impurity) 或者信息增益 (entropy)[25], 而本次选择的是根据基尼不纯度来判断特征的重要性. 式 (2) 如下:

$$Gini(t) = 1 - \sum_{i=1}^{C} P_i^2 \tag{2}$$

其中, C 是类别数量, P_i 是第 i 类在节点中的比例. 每次分裂时, 算法会计算分裂后的基尼不纯度的减少量, 并选择使基尼不纯度最大幅度减少的分裂点.

2.4 过采样平衡技术

过采样是一种处理不平衡数据集的技术, 通过增加少数类别的样本数量, 使各类别在训练中样本量均衡, 从而提高模型准确性. 过采样技术具有随机过采样、SMOTE 等方法. 本文采用随机过采样方法平衡数据集.

2.5 CNN 介绍

CNN (convolutional neural network, 卷积神经网络)[26]是人工神经网络模型中一种经典模型, 因其高效

主动学习和特征提取能力, 常被用于分类、预测任务. CNN 通过卷积层提取层次化特征, 全连接层进行有效的特征融合, 集成多个 CNN 能够减少单一模型可能由于数据不足出现的过拟合问题, 提高整体训练效果和泛化能力. 而其他机器学习模型在处理复杂的高维问题时可能无法提供同样的特征提取和自适应能力.

2.6 负相关学习方法

负相关学习 (negative correlation learning, NCL) 是一种集成学习技术, 1999 年, Liu 等人[27]第 1 次提出一种负相关学习集成神经网络, 用于回归和分类问题, 并取得了显著提升. 2018 年, Shi 等人[28]提出了一种深度负相关模型 (D-ConvNet) 计算人群数量, 在深度学习架构中引入负相关学习技术, 并得到了有效结果. 本文引入负相关学习 (NCL) 思想, 旨在引入负相关损失函数训练集成的每个模型. 相较于传统损失函数专注于减小模型与真实标签之间的误差, 该损失函数在训练过程中引入已训练模型预测结果的均值作为参考, 通过增加一个负相关项来惩罚当前模型与集成预测之间的相似性, 模型按顺序逐个训练, 每个新模型的损失不仅考虑自身预测误差, 也考虑与前面所有模型预测均值的差异, 从而鼓励预测结果的多样性. 该策略能够有效减少模型间的预测冗余, 增强集成模型的多样性.

$$f_{\text{ens}}(x_n)=\frac{1}{M}\sum_{i=1}^{M}f_i(x_n) \tag{3}$$

其中, M 为训练的基模型数量, x_n 为输入数据, $f_i(x_n)$ 是输入数据与预测结果的映射, 最终取的是平均值[29].

本研究负相关损失函数通过惩罚模型之间的相关性增强深度学习的多样性, 总损失如式 (4):

$$Loss=MSE+\lambda_{\text{ncl}}\times Correlation\ Penalty \tag{4}$$

其中, λ_{ncl} 是相关项强度, 基础损失 (均方误差 *MSE*) 的计算公式如下:

$$MSE=\frac{1}{N}\sum_{i=1}^{N}\left(y_{\text{true},i}-y_{\text{pred},i}\right)^2 \tag{5}$$

其中, y_{true} 为实值, y_{pred} 为预测值, N 为样本数量, 负相关惩罚项 (*Correlation Penalty*) 计算主模型预测与其他集成模型预测之间的平均平方误差, 计算公式如下:

$$Correlation\ Penalty=-\frac{1}{N}\sum_{i=1}^{N}\left(f_{\text{ens}}(x_n)-y_{\text{pred},i}\right)^2 \tag{6}$$

其中, n 为集成模型的数量, $f_{\text{ens}}(x_n)$ 表示现有模型的平均预测值.

2.7 十折交叉验证

十折交叉验证通过将数据集划分为 10 个子集, 抽取其中一个子集作测试集, 其余作训练集, 重复 10 次, 能有效评估模型性能, 减少数据划分的偏差. 特别在数据量较小的情况下, 它能提高评估的稳定性和泛化能力, 避免偶然性影响. 本文所有实验均采用十折交叉验证.

3 集成深度学习模型设计

目前许多代码异味研究使用机器学习模型进行预测, 其中也包括多种集成学习方法, 例如利用 Stacking 思想集成多种模型输出, 或利用 Bagging 训练多个基模型进行投票或者平均得到最终结果. 然而, 集成深度学习网络模型的研究比较少, 于是本次研究探究两种集成策略: 直接集成和采用负相关学习同构集成训练多个 CNN 模型获取平均预测结果. 直接集成深度学习策略能够减少模型复杂度, 提高模型检测效率, 负相关学习策略在训练单个基模型过程中会与其他基模型形成负相关误差, 增强整体模型的多样性. 同时负相关学习集成方法也被指出在提升神经网络模型性能方面具有积极影响[27].

实验流程中, 首先对数据集进行预处理, 使用随机森林进行特征选择并归一化, 再采用过采样技术进行数据平衡, 随后采用十折交叉验证划分训练集和测试集, 训练集用于模型训练, 测试集用作模型评估. 实验整体流程如图 1.

3.1 特征提取

数据预处理阶段采用随机森林方法对特征进行重要性评估. 随机森林通过基于节点分裂的指标评估特征的重要性, 从而确定关键特征的贡献度并进行选择, 以提升模型性能. 相对于信息增益计算方式, Gini 不纯度识别重要特征更加准确, 同时信息增益偏向高分支变量, 消耗计算更大[9], 相反 Gini 计算效率更加高.

将 4 种代码异味数据集输入到特征提取算法中, 筛选后保留的特征如表 2.

3.2 数据平衡

本文采用随机过采样技术对数据集进行平衡. 与 SMOTE 等方法不同, 随机过采样通过简单地复制现有的少数类别样本来实现平衡, 无需生成新样本或额外特征, 从而避免了潜在的噪声或与实际数据不符的问题. 此外, 随机过采样的计算开销较小, 执行效率较高.

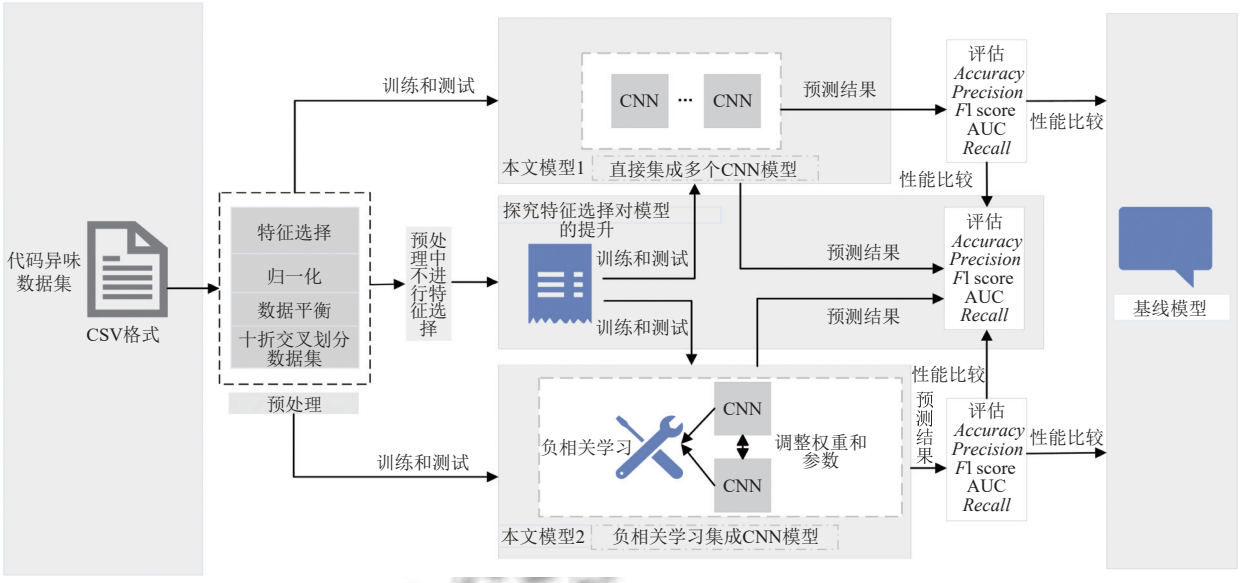

图 1 实验整体流程框架

表 2 特征选择之后保留的特征

代码异味	提取前数量	提取后数量	保留的特征类别
Data class	61	21	NOAM_type, NOM_type, FANOUT_type, NOA_type, LOCNAMM_type, CFNAMM_type, TCC_type, CBO_type, RFC_type, WMC_type, LCOM5_type, WOC_type, WMCNAMM_type, AMW_type, AMWNAMM_type, number_private_visibility_attributes, number_public_visibility_methods, number_private_visibility_methods, number_not_abstract_not_final_methods, number_not_final_not_static_methods, number_standard_design_methods, number_standard_design_methods
Feature envy	82	13	ATFD_method, FDP_method, MAXNESTING_method, CYCLO_method, NOLV_method, NOAV_method, LAA_method, FANOUT_method, CFNAMM_method, CINT_method, CDISP_method, ATFD_type, CFNAMM_type
God class	61	14	NOM_type, ATFD_type, FANOUT_type, NOMNAMM_type, LOC_type, LOCNAMM_type, CFNAMM_type, CBO_type, RFC_type, WMC_type, WMCNAMM_type, AMW_type, AMWNAMM_type, number_standard_design_methods
Long method	82	13	NOP_method, MAXNESTING_method, LOC_method, CYCLO_method, NOLV_method, NOAV_method, FANOUT_method, CFNAMM_method, CINT_method, LOC_type, WMC_type, AMW_type, AMWNAMM_type

3.3 CNN 模型构建

本文中的 CNN 模型构建根据 Liu 等人[19]的工作为基础做出修改. Liu 等人的 CNN 模型中一层卷积层的卷积核数量为 128, 卷积核大小为 1, 全连接层也是 128 个神经元, 最终输出一个神经元, 在原基础上加入正则化层, 丢弃一部分神经元来减少模型过拟合, 降低模型复杂性. CNN 结构如图 2 所示.

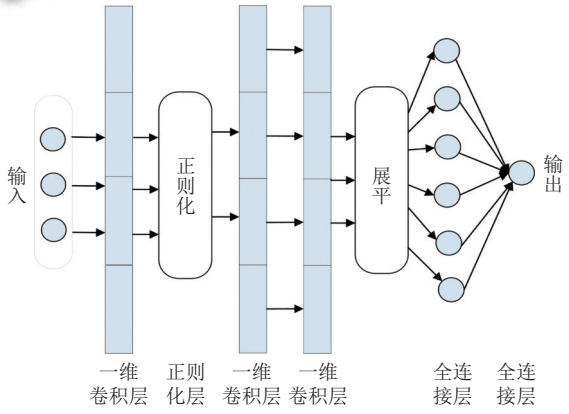

图 2 CNN 模型

3.4 直接集成训练 CNN

直接训练集成多个 CNN 主要步骤如下.

(1) 第 1 步构建多个分类器: 假设构建 N 个基模型 M, $N \in \{1,2,3,\cdots\}$.

(2) 第 2 步直接使用反向传播来训练基模型.

3.5 负相关学习集成训练 CNN

使用负相关学习训练多个模型主要经过 3 个步骤.

(1) 第 1 步构建多个分类器: 假设构建 N 个基模型 M, $N \in \{1,2,3,\cdots\}$.

(2) 第 2 步构建负相关损失函数: 要将负相关学习利用到模型训练中, 最关键是构建负相关损失函数, 假设惩罚项系数大小为 λ, 然后将其代入到式 (4) 中, 假设单个基模型预测损失为 X, 所有模型预测值与当前模型预测值之间的损失为 X', 则式 (4) 简化为: $Loss = X + \lambda \cdot X'$.

(3) 第 3 步根据损失更新基模型.

将以上步骤细化到 CNN 模型构建与训练过程中, 如算法 1.

算法 1. 负相关学习训练 CNN

输入: 训练数据 $\alpha=\{X_i,Y_i\}, i=1,\cdots,m, X\in R'_{(m,n,1)}, Y\in R'_m$.
输出: 一个训练好的基模型.

```
1) 初始化 CNN 基模型 M
2) 定义负相关学习损失函数 F
3) 对于 i 从 1 到 N:
4)     创建模型 M_i
5)     对于训练周期 epoch 从 1 到 E:
6)         进入训练周期
7)         对于每个批次 batch 从 1 到 B:
8)             批次训练
9)             调用 F, 计算损失
10)            更新模型
11) 保存当前模型 M_i
```

4 实验结果与分析

为了验证本文提出两种集成策略模型的性能和效果, 本次实验提出以下问题.

Q1: 基于随机森林的特征选择是否产生了良好的效果?

Q2: 与其他深度学习模型相比, 两种集成模型是否提高了代码异味检测的性能?

Q3: 与基线模型相比, 两种集成模型是否提高了代码异味检测的性能?

4.1 数据集介绍

本文使用的数据集来源 Fontana 等人[6]提供的代码异味数据集, 包含从 74 个软件系统收集的 6 种标记实例. 研究人员使用 5 种工具 (iPlasma、PMD、Fluid Tool、Anti-Pattern Scanner 和 Marinescu 检测规则) 识别代码异味. 同时 3 名研究生对结果进行了手动验证. 本次实验中仅使用了 4 种代码异味 (God class、Data class、Long method 和 Feature envy). 4 种代码异味数据集的总数均为 420 条, 其中包括异味样本 140 条、无异味样本 280 条.

4.2 实验环境

本实验的操作系统为 Windows 11, 处理器为 Intel 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz, 内存为 16 GB, 实现语言为 Python.

4.3 性能的衡量指标

在本次研究中, 将模型运用到每个代码异味数据集中, 同时利用多种衡量指标评估每次实验. 指标分别为: *F*1 (*F*1 值), *Precision* (精确率), *Recall* (召回率), *Accuracy* (准确率), AUC (AUC-ROC 曲线下的面积).

指标根据混淆矩阵计算, 混淆矩阵如表 3 所示.

表 3 混淆矩阵

真实值	预测值	
	有异味	无异味
有异味	*TP*	*FN*
无异味	*FP*	*TN*

TP、*FN*、*FP*、*TN* 的含义如下.

TP (true positive, 真阳性) 指正确预测有异味的样本.

FN (false negative, 假阴性) 指将有异味的样本预测为未有异味.

FP (false positive, 假阳性) 指将无异味的样本预测为有异味.

TN (true negative, 真阴性) 指正确预测无异味的样本.

根据混淆矩阵, 详细介绍各指标含义和计算公式.

(1) 召回率 (*Recall*): 召回率又称“查全率”, 表示实际有异味的样本中, 能有多少被正确预测, 衡量模型的覆盖率.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

(2) 精确率 (*Precision*): 精确率从预测结果判断, 表示预测为有异味的样本中, 有多少是真正的有异味样本.

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

(3) *F*1 值 (*F*1): *F*1 值是精确率和召回率的调和平均值, 用于衡量分类模型的精确性和全面性.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{9}$$

(4) 准确率 (*Accuracy*): 准确性指整个预测结果中, 正确预测有异味和无异味的样本数量占总样本的比例.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

(5) AUC-ROC (AUC): AUC 是衡量二分类模型性能的关键指标, 它表示 ROC 曲线下的面积. ROC 曲线通过比较不同分类阈值下的真阳性率 (*TPR*) 与假阳性率 (*FPR*) 来评估模型. 在 ROC 图中, *TPR* 为纵坐标, *FPR* 为横坐标. AUC 值范围在 0–1 之间, 越接近 1 表示模型性能越好, 能够更准确地区分正负类.

$$TPR = \frac{TP}{TP + FN} \tag{11}$$

$$FPR = \frac{FP}{FP + TN} \tag{12}$$

4.4 特征选择效果分析

本文使用随机森林进行特征选择, 剔除噪声和不相关特征, 保留与目标变量相关性强的特征, 并比较直接输入数据与特征选择后输入数据的模型性能, 评估基于随机森林的特征选择对模型性能的提升. 比较结果如表 4, 最优结果用加粗表示, 次优结果用下划线表示, 第三好结果用斜体表示.

通过表 4 评估指标结果比较可得, 直接集成 CNN 模型使用特征选择相较于不使用特征选择, 除了预测 God class 异味的评估数值有所下降, 其他代码异味预测结果皆有所提升, 其中 *F*1 值最大提升 24.06%, AUC 最大提升 14.79%, ACC 最大提升 15.71%, 精确度最大提升 26.18%, 召回率最大提升 20.15%. RNCNN 模型使用特征选择, 除预测 Long method 异味的评估数值有所下降, 但其他代码异味预测结果皆有所提升, 其中 *F*1 值最大提升 7.37%, AUC 最大提升 9.59%, ACC 最大提升 10.99%, 精确度最大提升 13.09%, 召回率最大提升 4.66%.

表 4 直接集成 CNN 模型有无随机森林的特征选择效果对比

模型	代码异味	使用特征选择					不使用特征选择				
		*F*1	AUC	ACC	P	R	*F*1	AUC	ACC	P	R
直接集成CNN	DC	**0.9859**	**0.9968**	**0.9857**	**0.9826**	*0.9899*	0.9714	0.9951	0.9714	0.9523	0.9925
	FE	**0.9731**	0.9899	**0.9732**	**0.9645**	0.9836	0.7325	0.8420	0.8161	0.7027	0.7821
	GC	0.9745	0.9955	0.9750	*0.9710*	*0.9786*	**0.9766**	**0.9968**	**0.9768**	**0.9747**	0.9788
	LM	**0.9909**	*0.9968*	**0.9911**	**0.9858**	*0.9963*	0.9892	**0.9975**	0.9893	0.9787	**1.0000**
RNCNN	DC	*0.9708*	*0.9946*	0.9714	*0.9476*	**0.9946**	0.9625	0.9927	*0.9625*	0.9413	0.9849
	FE	0.9680	**0.9901**	0.9679	0.9518	**0.9870**	*0.9335*	*0.9852*	*0.9339*	*0.9319*	0.9404
	GC	*0.9706*	*0.9949*	*0.9714*	0.9742	0.9676	0.8969	0.8990	0.8643	0.8433	**0.9856**
	LM	*0.9893*	0.9966	0.9893	*0.9790*	**1.0000**	0.9894	0.9970	0.9893	0.9825	0.9967

注: DC: Data class; FE: Feature envy; GC: God class; LM: Long method; P: *Precision*; R: *Recall*; ACC: *Accuracy*.

综上所述, 采用特征选择方法对两种集成模型预测性能有部分提升, 产生了良好效果.

对 Q1 的回答: 基于随机森林的特征选择方式, 直接集成 CNN 模型检测 4 种代码异味数据集, 除 God class 异味, 其余 3 种异味检测性能有所提升; 而 RNCNN 模型检测 4 种代码异味数据集, 除 Long method 异味, 其余 3 种异味检测性能有所提升.

4.5 集成学习效果分析

本文探究两种集成策略, 构建两种集成学习模型来对代码异味进行检测. 通过比较两种集成模型与其他深度学习模型评估结果来判断本文集成模型是否提升了代码异味检测的性能. 本文选择 RNN 与 LSTM 模型, 这两种模型原理不同, 是经典的深度学习算法, 并且能够用有限的消费级算力实现, 所以这种对比在实践中具有实际意义. 比较结果如表 5–表 7 所示, 最优结果用加粗表示, 次优结果用下划线表示.

通过表 5 评估指标结果比较可得, 直接集成 CNN 模型与单个 CNN 模型相比, 各项指标皆有所提升, 其中 *F*1 值最大提升 1.63%, AUC 最大提升 0.08%, ACC 最大提升 1.61%, 精确率最大提升 3.32%, 召回率最大提升 0.68%. RNCNN 模型在预测 Data class 与 God class 异味上, 评估结果有所提升, 但在预测 Feature envy 上, 指标数值有所下降.

通过表 6 评估指标结果比较可得, 直接集成 CNN 模型与 RNN 模型相比整体预测结果较好, *F*1 值最大提升 12.85%, AUC 值最大提升 4.93%, ACC 值最大提升 14.28%, 精确度最大提升 18.57%, 召回率最大提升 3.77%. RNCNN 模型与 RNN 模型相比, 除了预测 Feature envy 异味上指标有所下降, 其余预测中 *F*1 值最大提升 11.34%, AUC 值最大提升 5.55%, ACC 值最大提升 12.85%, 精确度最大提升 15.07%, 召回率最大提升 4.24%.

通过表 7 评估指标结果比较可得, 直接集成 CNN 模型与 LSTM 模型相比, 整体结果有所提升, 其中 *F*1

值最大提升 10.35%, AUC 值最大提升 6.57%, ACC 值最大提升 11.61%, 精确度最大提升 16.58%, 召回率最大提升 6.77%. RNCNN 模型与 LSTM 模型相比, 整体结果有所提升, 其中 *F*1 值最大提升 9.84%, AUC 值最大提升 6.35%, ACC 值最大提升 10.18%, 精确度最大提升 13.08%, 召回率最大提升 7.11%.

表 5 直接集成 CNN 模型与单个 CNN 模型效果对比

代码异味	直接集成CNN					RNCNN					单个CNN模型				
	*F*1	AUC	ACC	P	R	*F*1	AUC	ACC	P	R	*F*1	AUC	ACC	P	R
DC	**0.9859**	**0.9968**	**0.9857**	**0.9826**	0.9899	0.9708	0.9946	0.9714	0.9476	**0.9946**	0.9696	0.9960	0.9696	0.9494	0.9923
FE	**0.9731**	0.9899	**0.9732**	**0.9645**	0.9836	0.9680	0.9901	0.9679	0.9518	**0.9870**	0.9709	**0.9915**	0.9714	0.9603	0.9834
GC	**0.9745**	**0.9955**	**0.9750**	0.9710	**0.9786**	0.9706	0.9949	0.9714	**0.9742**	0.9676	0.9678	0.9949	0.9679	0.9646	0.9718
LM	**0.9909**	**0.9968**	**0.9911**	**0.9858**	0.9963	0.9893	0.9966	0.9893	0.9790	**1.0000**	0.9888	0.9968	0.9893	0.9819	0.9960

注: DC: Data class; FE: Feature envy; GC: God class; LM: Long method; P: *Precision*; R: *Recall*; ACC: *Accuracy*.

表 6 直接集成 CNN 模型与 RNN 模型效果对比

代码异味	直接集成CNN					RNCNN					RNN模型				
	*F*1	AUC	ACC	P	R	*F*1	AUC	ACC	P	R	*F*1	AUC	ACC	P	R
DC	**0.9859**	**0.9968**	**0.9857**	**0.9826**	0.9899	0.9708	0.9946	0.9714	0.9476	**0.9946**	0.8574	0.9391	0.8429	0.7969	0.9522
FE	**0.9731**	0.9899	**0.9732**	**0.9645**	0.9836	0.9680	**0.9901**	0.9679	0.9518	0.9870	0.9696	0.9896	0.9696	0.9527	**0.9897**
GC	**0.9745**	**0.9955**	**0.9750**	0.9710	**0.9786**	0.9706	0.9949	0.9714	**0.9742**	0.9676	0.9652	0.9896	0.9661	0.9599	0.9714
LM	**0.9909**	0.9968	**0.9911**	**0.9858**	0.9963	0.9893	0.9966	0.9893	0.9790	**1.0000**	0.9769	0.9982	0.9768	0.9686	0.9863

注: DC: Data class; FE: Feature envy; GC: God class; LM: Long method; P: *Precision*; R: *Recall*; ACC: *Accuracy*.

表 7 直接集成 CNN 模型与 LSTM 模型效果对比

代码异味	直接集成CNN					RNCNN					LSTM模型				
	*F*1	AUC	ACC	P	R	*F*1	AUC	ACC	P	R	*F*1	AUC	ACC	P	R
DC	**0.9859**	**0.9968**	**0.9857**	**0.9826**	**0.9899**	0.9708	0.9946	0.9714	0.9476	**0.9946**	0.8829	0.9311	0.8696	0.8168	0.9691
FE	**0.9731**	**0.9899**	**0.9732**	**0.9645**	**0.9836**	0.9680	**0.9901**	0.9679	0.9518	**0.9870**	0.8696	0.9593	0.8679	0.8305	0.9159
GC	**0.9745**	**0.9955**	**0.9750**	**0.9710**	**0.9786**	0.9706	0.9949	0.9714	**0.9742**	0.9676	0.9702	0.9924	0.9714	0.9669	0.9741
LM	**0.9909**	0.9968	**0.9911**	**0.9858**	**0.9963**	0.9893	0.9966	0.9893	0.9790	**1.0000**	0.9789	**0.9981**	0.9786	0.9722	0.9860

注: DC: Data class; FE: Feature envy; GC: God class; LM: Long method; P: *Precision*; R: *Recall*; ACC: *Accuracy*.

综合来看, 相较于单个 CNN 模型与其他深度学习模型, 两种集成方法使得模型性能有部分提升.

对 Q2 的回答: 与单个 CNN 相比, 两种集成策略检测 4 种代码异味, 其中 Data class、God class、Long method 异味检测性能有所提升. 而与其他深度学习模型相比, 两种集成方法整体有所提升.

4.6 集成模型与基线工作相比较

本文复现 Alazba 等人[9]提出的融合特征选择的堆栈集成机器学习模型, 该模型采用信息增益的特征选择方法, 使用堆栈集成各种机器学习技术, 其中堆栈集成逻辑回归技术评估结果最佳. 同时其结果皆经过十折交叉验证. 将本文集成模型评估结果与 Alazba 等人[9]最佳模型评估结果进行比较, 探究本文提出的集成深度学习模型是否提升代码异味检测性能. 比较结果如表 8.

通过表 8 评估指标结果可得, 相较 Alazba 等人[9]提出的集成方法, 直接集成 CNN 模型的评估指标 *F*1 值提升 4.93%–0.52%, AUC 值提升 0.19%–0.03%; RNCNN 模型的评估指标 *F*1 值提升 4.42%–0.36%, AUC 值提升 0.06% 与 0.05%.

表 8 提出的模型与基线工作对比

代码异味	Alazba等人[9]			本文方法			
				直接集成CNN		RNCNN	
	最佳算法	*F*1	AUC	*F*1	AUC	*F*1	AUC
DC	Stack-LR	0.9777	0.9949	**0.9859**	**0.9968**	0.9708	0.9946
FE	Stack-LR	0.9238	0.9895	**0.9731**	**0.9899**	**0.9680**	**0.9901**
GC	Stack-LR	0.9532	0.9944	**0.9745**	**0.9955**	**0.9706**	**0.9949**
LM	Stack-LR	0.9857	0.9971	**0.9909**	0.9968	**0.9893**	0.9966

注: DC: Data class; FE: Feature envy; GC: God class; LM: Long method.

对 Q3 的回答: 相较于基线工作, 本文提出的直接集成 CNN 模型检测代 4 种代码异味, 3 种代码异味检测性能皆提升, 而 Long method 异味的 *F*1 值有所提升; RNCNN 检测 4 种代码异味数据集, 3 种代码异味检测的 *F*1 值有所提升.

4.7 分析与讨论

本文提出的模型实验使用了两种集成策略, 而这两种集成策略评估结果也有所差异. 分析问题如下.

首先, 直接集成 CNN 与 RNCNN 模型的差异主要体现在训练过程中采用的方法不同. 然而, 直接集成 CNN 的评估指标通常高于 RNCNN 模型的评估指标, 可能原因如下: 第一, 负相关学习引入了额外的约束, 使得模型不仅关注最小化单个基模型的损失函数, 而是要平衡整体模型的负相关性, 这可能导致模型整体评估指标的下降; 第二, 本文采用的基模型之间的误差通过负相关学习不容易得到有效消除, 从而影响了负相关集成模型的性能.

其次, 在与单个 CNN 模型相比时, 引入负相关学习方法的模型在预测 Feature envy 异味性能上有所下降. 可能的原因是: 第一, 本文采用的数据集中, Feature envy 异味的数据特征通过负相关学习难以得到有效消除, 导致模型在该任务上的表现不如单个 CNN 模型.

5 结论与展望

本文采用了两种集成策略: 直接集成与负相关学习集成, 提出了将基于随机森林的特征选择、数据平衡技术、直接集成/负相关学习、CNN 相结合的集成学习模型. 通过基于随机森林的特征选择方法筛选重要性特征, 去除不相关、不重要的特征, 提升模型运行效率和性能. 使用过采样技术平衡数据集, 采用直接集成 CNN 方法提高模型预测效率, 同时集成多个 CNN 模型并加入负相关惩罚项, 降低模型过拟合, 增加模型的多样性, 最终得到模型的平均值. 最后在 4 种代码异味的数据集上训练测试其性能, 并且通过 5 种衡量指标 (*F*1 值、精确度、召回率、准确率和 AUC 值) 进行评估. 实验结果表明特征选择、数据平衡、直接集成/负相关学习与 CNN 集成方式的模型能够取得优异的性能.

后续的相关工作方向: 1) 本文使用公开数据集, 它规模较小, 且可能存在噪声较大的问题, 未来可以尝试构建更大规模、更高质量的数据集, 并引入数据清洗和增强技术以提高模型的鲁棒性. 2) 在模型设计方面, 未来可探索更先进的深度学习方法, 如注意力机制或图神经网络, 以更有效地捕捉代码结构和语义特征. 3) 扩展研究范围, 涵盖更多的代码异味类型和多种编程语言, 从而提升模型的通用性和适用性.

参考文献

1 Khomh F, di Penta M, Guéhéneuc YG, *et al*. An exploratory study of the impact of antipatterns on class change- and fault-proneness. Empirical Software Engineering, 2012, 17(3): 243–275. [doi: 10.1007/s10664-011-9171-y]

2 Fowler M, Beck K, Brant J, *et al*. Refactoring: Improving the Design of Existing Code. Boston: Addison-Wesley Longman Publishing Co. Inc., 1999.

3 Thakur PS, Jadeja M, Chouhan SS. CBReT: A cluster-based resampling technique for dealing with imbalanced data in code smell prediction. Knowledge-based Systems, 2024, 286: 111390. [doi: 10.1016/j.knosys.2024.111390]

4 Spadini D, Palomba F, Zaidman A, *et al*. On the relation of test smells to software code quality. Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution. Madrid: IEEE, 2018. 1–12. [doi: 10.1109/ICSME.2018.00010]

5 Singh S, Kaur S. A systematic literature review: Refactoring for disclosing code smells in object oriented software. Ain Shams Engineering Journal, 2018, 9(4): 2129–2151. [doi: 10.1016/j.asej.2017.03.002]

6 Fontana FA, Mäntylä MV, Zanoni M, *et al*. Comparing and experimenting machine learning techniques for code smell detection. Empirical Software Engineering, 2016, 21(3): 1143–1191. [doi: 10.1007/s10664-015-9378-4]

7 Gupta R, Kumar N, Kumar S, *et al*. Unsupervised machine learning for effective code smell detection: A novel method. Journal of Communications Software and Systems, 2024, 20(4): 307–316. [doi: 10.24138/jcomss-2024-0083]

8 Mhawish MY, Gupta M. Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. Journal of Computer Science and Technology, 2020, 35(6): 1428–1445. [doi: 10.1007/s11390-020-0323-7]

9 Alazba A, Aljamaan H. Code smell detection using feature selection and stacking ensemble: An empirical investigation. Information and Software Technology, 2021, 138: 106648. [doi: 10.1016/j.infsof.2021.106648]

10 吴海涛, 蔡咏琦, 高建华. Bagging 异构集成的代码异味检测与重构优先级划分. 计算机工程与应用, 2024, 60(3): 138–147. [doi: 10.3778/j.issn.1002-8331.2305-0218]

11 Di Nucci D, Palomba F, Tamburri DA, *et al*. Detecting code smells using machine learning techniques: Are we there yet? Proceedings of the 25th IEEE International Conference on Software Analysis, Evolution and Reengineering. Campobasso: IEEE, 2018. 612–621. [doi: 10.1109/SANER.2018.8330266]

12 Travassos G, Shull F, Fredericks M, *et al*. Detecting defects in object-oriented designs: Using reading techniques to

increase software quality. ACM SIGPLAN Notices, 1999, 34(10): 47–56. [doi: 10.1145/320385.320389]

13 Mens T, Tourwe T. A survey of software refactoring. IEEE Transactions on Software Engineering, 2004, 30(2): 126–139. [doi: 10.1109/TSE.2004.1265817]

14 Prokić S, Luburić N, Slivka J, *et al*. Prescriptive procedure for manual code smell annotation. Science of Computer Programming, 2024, 238: 103168. [doi: 10.1016/j.scico.2024.103168]

15 Tsantalis N, Chatzigeorgiou A. Identification of move method refactoring opportunities. IEEE Transactions on Software Engineering, 2009, 35(3): 347–367. [doi: 10.1109/TSE.2009.1]

16 Yu DJ, Yang QX, Chen X, *et al*. Actionable code smell identification with fusion learning of metrics and semantics. Science of Computer Programming, 2024, 236: 103110. [doi: 10.1016/j.scico.2024.103110]

17 Prestat D, Moha N, Villemaire R, *et al*. DynAMICS: A tool-based method for the specification and dynamic detection of android behavioral code smells. IEEE Transactions on Software Engineering, 2024, 50(4): 765–784. [doi: 10.1109/TSE.2024.3363223]

18 黄晨峻, 高建华. 融合特征选择与多模型软投票集成学习的代码异味检测方法. 小型微型计算机系统, 2025, 46(2): 504–512. [doi: 10.20009/j.cnki.21-1106/TP.2023-0468]

19 Liu H, Jin JH, Xu ZF, *et al*. Deep learning based code smell detection. IEEE Transactions on Software Engineering, 2021, 47(9): 1811–1837. [doi: 10.1109/TSE.2019.2936376]

20 Nizam A, Avar MY, Adaş ÖK, *et al*. Detecting code smell with a deep learning system. Proceedings of the 2023 Innovations in Intelligent Systems and Applications Conference. Sivas: IEEE, 2023. 1–5. [doi: 10.1109/ASYU58738.2023.10296577]

21 边奕心, 李禹齐, 张子恒, 等. 深度学习策略下缓慢循环异味检测方法. 小型微型计算机系统, 2024, 45(2): 490–497.

22 Zhang HY, Kishi T. Large class detection using GNNs: A graph based deep learning approach utilizing three typical GNN model architectures. IEICE Transactions on Information and Systems, 2024, E107.D(9): 1140–1150. [doi: 10.1587/transinf.2023EDP7192]

23 Ali PJM, Faraj RH. Data normalization and standardization: A technical report. Machine Learning Technical Reports, 2014, 1(1): 1–6. [doi: 10.13140/RG.2.2.28948.04489]

24 Chandrashekar G, Sahin F. A survey on feature selection methods. Computers & Electrical Engineering, 2014, 40(1): 16–28. [doi: 10.1016/j.compeleceng.2013.11.024]

25 Quinlan JR. Induction of decision trees. Machine Learning, 1986, 1(1): 81–106. [doi: 10.1007/BF00116251]

26 Alzubaidi L, Zhang JL, Humaidi AJ *et al*. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data, 2021, 8(1): 53. [doi: 10.1186/s40537-021-00444-8]

27 Liu Y, Yao X. Ensemble learning via negative correlation. Neural Networks, 1999, 12(10): 1399–1404. [doi: 10.1016/S0893-6080(99)00073-8]

28 Shi ZL, Zhang L, Liu Y, *et al*. Crowd counting with deep negative correlation learning. Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Salt Lake City: IEEE, 2018. 5382–5390. [doi: 10.1109/CVPR.2018.00564]

29 Chen HH, Jiang BB, Yao X. Semisupervised negative correlation learning. IEEE Transactions on Neural Networks and Learning Systems, 2018, 29(11): 5366–5379. [doi: 10.1109/TNNLS.2017.2784814]

(校对责编: 张重毅)