第 46 卷 第 10 期 计 算 机 工 程 2020 年 10 月
Vol. 46 No. 10 Computer Engineering October 2020

·体系结构与软件技术· 文章编号：1000-3428(2020)10-0216-07 文献标志码：A 中图分类号：TP311

基于 BP 神经网络的代码坏味检测

王曙燕，张一权，孙家泽

（西安邮电大学 计算机学院，西安 710000）

摘 要： 程序中若出现代码坏味将严重影响其质量且难以对软件维护提供保障。针对机器学习算法在代码坏味检测中准确度较低以及数据集仅存在单一类型代码坏味的问题，提出一种基于 BP 神经网络的代码坏味检测方法。考虑软件实际开发过程中会存在不同类型的坏味，对数据类、上帝类、长方法和特征依恋 4 种代码坏味进行研究并将其合并为方法级别和类级别的 2 种坏味数据集，根据数据集中的标签信息进行有监督深度学习，进而构建代码坏味的真假阳性检测模型。实验结果表明，相比基于机器学习和基于度量的代码坏味检测方法，该方法的平均准确度提高 15.19%，平均 F1 值提高 58.39%。

关键词： 代码坏味；软件维护；BP 神经网络；深度学习；检测模型

开放科学（资源服务）标志码（OSID）：

中文引用格式： 王曙燕，张一权，孙家泽. 基于 BP 神经网络的代码坏味检测[J]. 计算机工程，2020，46(10)：216-222，230.

英文引用格式： WANG Shuyan，ZHANG Yiquan，SUN Jiaze. Detection of bad smell in code based on BP neural network[J]. Computer Engineering，2020，46(10)：216-222，230.

Detection of Bad Smell in Code Based on BP Neural Network

WANG Shuyan，ZHANG Yiquan，SUN Jiaze

（School of Computer Science and Technology，Xi'an University of Posts and Telecommunications，Xi'an 710000，China）

【Abstract】 Bad smells in code seriously affect the quality of software and its maintenance. To address the low accuracy of machine learning algorithms in bad smell detection and the single type of bad smell dataset，this paper proposes a detection method for bad smells in code based on BP Neural Network（BPNN）. Considering that there are different types of bad smells in the actual development of software，four types of bad smells，Data class，God class，Long method，and Feature envy，are studied and merged into method-level and class-level code smell datasets. Based on the label information in the dataset，supervised deep learning is implemented to build a true and false positive prediction model for bad smells. The experimental results show that compared with the bad smell detection methods based on machine learning and metric，the proposed method improves the average accuracy by 15.19% and the average F1 value by 58.39%.

【Key words】 bad smell in code；software maintenance；BP Neural Network（BPNN）；deep learning；detection model

DOI： 10.19678/j.issn.1000-3428.0055862

0 概述

FOWLER 等人[1]在 1999 年提出了“代码坏味”的概念，其指软件开发过程中可能会出现的代码问题。代码坏味检测是通过重构步骤解决源代码（或设计）问题的既定方法，其目的是提高软件质量和维护效果。FOWLER 等人一共提出了 22 种代码坏味，包括克隆代码、特征依恋和长方法等。代码坏味及其检测方法极大地推动了自动化软件重构的应用和发展，成为软件重构领域的研究热点之一。

目前，研究人员设计出很多代码坏味检测工具，具有代表性的包括 DECOR、IPlasma、InFusion、JDeodorant、PMD 以及 Checkstyle，这些工具基于一组度量和规则来检测特定代码坏味，如众所周知的面向对象的度量或者为检测特定坏味而临时定义的度量。根据检测规则，用于通过不同工具检测代码坏味的度量可以是

基金项目： 陕西省科技厅工业公关项目“基于搜索的程序并行测试数据优化关键技术”（2018GY-014）；西安市科技计划项目“基于群体智能的多目标软件测试优化关键技术研究”（GXYD17.10）。

作者简介： 王曙燕（1964—），女，教授、博士，主研方向为软件测试、数据挖掘、智能信息处理；张一权（通信作者），硕士研究生；孙家泽，教授、博士。

收稿日期： 2019-08-30 **修回日期：** 2019-11-06 **E-mail：** 15029600502@163.com

不同的,不同的工具对同一种代码坏味的检测结果也不同。此外,即使度量标准相同,度量标准的阈值也可能发生变化,从而使得检测到的坏味数量相应地改变。检测工具的另外一个问题是检测准确性普遍较低,会检测到许多假阳性坏味。上述问题推动了研究人员对代码坏味检测方法进行分析并提出了一系列的自动或半自动改进方法,以从代码中检测坏味[2]。其中,JDeodorant[3]是由 TSANTALIS 等人设计的代码坏味检测工具,该工具利用杰卡德距离衡量 2 个代码实体之间的相似性。文献[4-6]研究表明,在不同的代码坏味检测算法中,特征度量和阈值存在差异,不同的方法选取的度量值和阈值也有所不同,这就导致在代码坏味检测中没有一个具体的标准[2]。学者们为了解决该问题,提出基于各类机器学习算法的代码坏味检测方法,但是这些方法也存在一定的局限性。NUCCI 等人[7]通过大量实验,指出在机器学习算法的检测过程中,代码坏味数据集普遍仅存在一种类型的坏味,这并不符合实际情况,因为在软件开发中代码坏味不可能只存在单一类型,通过实验可知在数据集中存在不同类型的代码坏味,机器学习算法在对其检测时准确率较低,说明了不同类型的代码坏味分布在数据集中时减弱了机器学习算法对代码坏味的检测能力。

本文将 BP 神经网络(BPNN)与常见的软件度量项相结合,以对代码坏味进行检测。合并不同类型的代码坏味,使数据集中存在不同的坏味类型,将常见的软件度量特征与代码坏味实例进行相对应的编码,根据数据集中的标签信息实现有监督深度学习。利用神经网络在自动选择原始数据特征方面的优势,提取出这些度量项之间的相互关系,使度量特征项与代码坏味实例完成映射建模,同时模型学习输入的代码坏味特征中的复杂关系规则,从而对代码坏味实例进行分类。由于有监督的深度学习通常需要大量的标记数据作为训练样本,因此本文借助 2 种代码坏味检测工具针对 4 种类型的代码坏味实例进行提取,检测程序主要包括 Apache 中的 7 个 Java 开源项目。

1　相关工作

目前的代码坏味检测方法主要分为两大类,一类为基于规则的方法,这些方法主要依赖于度量,在某些情况下还有与代码结构和命名相关的其他规则;另一类方法使用机器学习技术,这些技术主要基于度量进行代码坏味检测。

基于规则的方法较复杂,因为它们使用的信息来源多于基于度量的机器学习方法,如命名规则[8]、结构规则[8]以及软件版本历史[9]。章晓芳等人[10]考虑到代码坏味与软件演化中的源文件操作关系,探究包含坏味的文件在软件版本历史中的不同特征,研究结果表明,有几种特定的坏味对文件的修改产生了显著影响。另一方面,基于规则的方法依赖于人为手动创建的规则。例如,DECOR 要求规则为特定的语言形式,并且此规范过程必须由领域专家、工程师完成[11]。然而,基于度量的机器学习方法是否比基于规则的方法需要更少的人为干预不得而知,它取决于 2 个因素:1)基于规则的方法需要什么复杂程度的规则;2)基于度量的机器学习方法需要多少训练样本[12]。基于度量的机器学习方法的明显优势是减少了开发人员的工作压力,基于规则的方法要求工程师创建定义每种气味的特定规则。对于基于度量的机器学习方法,由机器学习算法创建规则,要求工程师仅提供一段代码是否有气味的信息。

研究人员提出了基于各类机器学习算法的代码坏味检测方法。KREIMER[13]提出了一种自适应检测方法,该方法将基于度量与学习决策树的基础方法相结合,对过大类和长方法的 2 种代码坏味类型进行检测,并且在 IYC 系统和 WEKA 工具上完成分析。该方法适用于特定的场景,但在识别不同的代码坏味时会因为度量规则而存在性能差异。KHOMH 等人[14]提出了 BDTEX 方法,其为一种目标问题度量方法,根据反模式的定义构建贝叶斯信任网络,并在 2 个开源程序上使用 Blob 反模式、功能分解和 Spaghetti Code 反模式验证 BDTEX 方法。MAIGA 等人[15]利用一种基于支持向量机的检测方法在 3 个开源程序中进行反模式检测,该方法的F1 值达到 80%左右,但是准确度较低。YANG 等人[16]通过在克隆代码上应用机器学习算法研究开发人员对代码坏味的判断。PALOMBA 等人[17]提出一种基于信息检索技术,利用程序中的文本信息进行坏味检测。FONTANA 等人[12]将几种常见的机器学习算法应用在各类代码坏味检测中,并总结了 J48、随机森林以及贝叶斯网络等几种表现较好的机器学习算法。但是,在利用机器学习算法对代码坏味进行检测的过程中,数据集包含的有坏味与无坏味实例之间的度量分布不同,实例的选择可能会导致机器学习算法对坏味的检测性能下降。刘丽倩等人[18]针对数据不平衡对机器学习算法的影响问题,将决策树算法与代价敏感学习理论相结合以对长方法进行检测,研究结果表明,其能提高长方法的检测查准率和查全率。卜依凡等人[19]将代码中的文本信息和软件度量相结合,通过一种基于深度学习技术的方法对上帝类代码坏味进行检测,实验结果表明,该方法对

上帝类代码坏味的检测性能优于代码坏味检测工具JDeodorant,尤其是在查全率上优势明显。

上述基于机器学习的方法在进行代码坏味检测时取得了较好的检测效果。但是,此类方法仅考虑数据集中包含受单一类型坏味影响的实例的情景,并不符合软件开发过程中出现各种类型代码坏味的实际情况。本文基于 BP 神经网络,对同一种数据集中存在不同类型代码坏味和无坏味的情况进行代码坏味检测。

2 相关知识

BP 神经网络具有高效非线性数据函数映射逼近功能,其为一种功能强大的数据建模工具,能够捕获和表示复杂的输入与输出关系[20]。

2.1 网络初始化

神经网络模型主要由输入层、隐藏层和输出层组成,代码坏味的特征作为输入层的输入,隐藏层用于接收输入层的数据。输入层与隐藏层之间的关系可以描述为:输入层神经元为 i,输入层的输入为 I_i,输入层神经元 i 到隐藏层神经元 j 的权重为 W_{ji}^1,隐藏层神经元 j 的阈值为 θ_j^1,隐藏层神经元 j 的输出 H_j 与输入层的输入 I_i 之间的映射关系表达式如式(1)所示:

$$H_j = f\left(\sum W_{ji}^1 I_i + \theta_j^1\right) \tag{1}$$

其中,$f(\cdot)$ 为激活函数,在输出层输出过程中使用 Softmax 激活函数,将其设置为 3 个单元表示输出代码坏味特征类型。Softmax 激活函数可以将多个神经元的输出映射到(0,1)区间,选取概率最大的分类作为代码坏味预测结果。

2.2 输出值计算

在 BP 神经网络模型中,层与层之间存在计算线性关系,在网络初始化完成之后,需要为每层选择合适的激活函数,目的是为了尽可能多地得到神经网络中每层之间学习输入数据的线性变换集合空间,从而充分利用多层表示的优势。在网络中,隐藏层学习输入层数据的线性变换(网络中间一层使用 ReLU 激活函数),ReLU 激活函数用于隐藏层神经元输出,当输入 $x<0$ 时,输出为 0,当 $x>0$ 时,输出为 x,该激活函数使神经网络更快收敛,计算公式如式(2)所示:

$$\phi(x) = \max(0, x) \tag{2}$$

根据隐藏层神经元 j 的输出 H_j 和隐藏层神经元 j 到输出层的权重 W_j^2 以及输出层的阈值 θ^2,可以获得输出层的输出值 Y_{pred}(代码坏味预测值),表达式如式(3)所示:

$$Y_{\text{pred}} = f\left(\sum W_j^2 \cdot H_j + \theta^2\right) \tag{3}$$

2.3 模型训练

为了使 BP 神经网络对代码坏味进行可靠的预测,必须对 BP 神经网络进行适当训练。在训练过程中,BP 神经网络算法利用梯度下降法寻找最优解,在输出层和隐藏层之间,通过误差值按权重比例进行分割,计算出每条链接相关的特定误差值,通过重组这些误差值得到隐藏层神经元节点相关联的误差值,再次将这些误差值按照输入层和隐藏层之间的权重进行分割,完成误差的反向传播。其中,最主要的步骤是更新层与层之间的权重值和阈值,更新规则表达式如式(4)~式(7)所示:

$$W_j^2 = W_j^2 + \alpha \cdot (Y_{\text{pred}} - Y_{\text{real}}) \cdot Y_{\text{pred}} \cdot (1 - Y_{\text{pred}}) \cdot H_j \tag{4}$$

$$\theta^2 = \theta^2 + \beta \cdot (Y_{\text{pred}} - Y_{\text{real}}) \cdot Y_{\text{pred}} \cdot (1 - Y_{\text{pred}}) \tag{5}$$

$$W_{ji}^1 = W_{ji}^1 + \alpha \cdot (Y_{\text{pred}} - Y_{\text{real}}) \cdot W_j^2 \cdot H_j \cdot (1 - H_j) \cdot I_i \tag{6}$$

$$\theta_j^1 = \theta_j^1 + \beta \cdot (Y_{\text{pred}} - Y_{\text{real}}) \cdot W_j^2 \cdot H_j \cdot (1 - H_j) \tag{7}$$

其中,α 和 β 为学习率,在训练过程中,需要获取 BP 神经网络的目标输出,Y_{real} 为代码坏味真实值,将其作为最终的目标输出。如果目标输出 Y_{real} 与预测目标 Y_{pred} 之间的误差小于当前设定的阈值,或者训练迭代轮数达到预设的阈值,则完成模型训练。

3 基于 BP 神经网络的代码坏味检测方法

机器学习算法在对代码坏味进行检测的过程中,数据集中只包含一种类型的代码坏味,并不能反映软件设计过程中存在的实际问题。本文提出基于 BP 神经网络的代码坏味检测方法,针对数据类(Data class)、上帝类(God class)、长方法(Long method)和特征依恋(Feature envy)4 种坏味类型进行分类,并将 4 种类型合并为方法级别和类级别2 种类型的数据集,使数据集中包含不同的坏味类型。表 1 所示为 4 种类型的代码坏味描述。

表 1 代码坏味描述

Table 1 Description of bad smell in code

坏味类型	描述
数据类	类中只提供公开成员变量或操作函数
上帝类	类中存在大量的属性和方法
长方法	方法中存在大量的函数
特征依恋	方法中大量使用其他类中的成员

利用神经网络对代码坏味进行检测,是将度量特征信息与标签信息组成的向量形式作为神经网络输入层的输入,通过网络得到特征并输入到目标函数的神经网络分类器中进行训练,分类器的预期输出为样本的标签,在经过多次迭代训练后,可以得到最终被训练好的神经网络分类器。图 1 所示为本文代码坏味检测方法的流程。

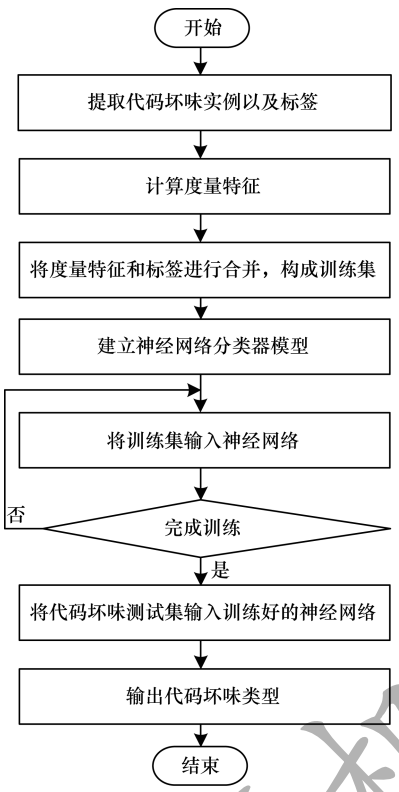

图 1　基于 BP 神经网络的代码坏味检测方法流程

Fig. 1　Procedure of detection method of bad smell in code based on BP neural network

3.1　数据集预处理

在对数据集进行预处理时需要提取代码坏味实例、度量特征以及标签,由于不同的研究人员对度量特征提取的结果不同,本文主要按以下度量项来提取代码坏味度量特征:

1)数据类:耦合强度(CINT)和类耦合类数(CBO)。

2)上帝类:访问外部数据数(ATFD)、类的圈复杂度(WMC)、类中通过访问相同属性而发生连接的方法对个数(TCC)。

3)特征依恋:访问外部数据数(ATFD)、属性访问的位置(LAA)、提供外部数据数(FDP)。

4)长方法:代码总行数(LOC)、方法数(NOM)、属性数(NOA)。

提取度量特征、代码坏味实例与标签的具体步骤如下:

步骤 1　针对 4 种代码坏味类型,使用代码坏味自动检测工具 iPlasma 和 Checkstyle 对开源软件系统进行检测,提取代码坏味实例和无坏味实例并对其进行标记,生成标签。

步骤 2　计算有代码坏味实例和无代码坏味实例的度量特征。使用浮点数序列对有代码坏味和无代码坏味实例度量特征进行编码表示,其中,0 代表某度量特征不是代码坏味影响因素,纯小数值代表某度量特征是影响代码坏味的因素。

步骤 3　将步骤 1 和步骤 2 中的度量特征和标签进行合并,生成方法级别和类级别的 2 种数据集,以此构成训练集。

利用度量特征和标签构成训练集的具体实现过程如下:

将得到的度量特征与标签信息进行合并,并将度量特征与标签信息转换为向量表示$\langle m_{11},m_{12},\cdots,p_{1n}\rangle$,$m$ 表示度量特征,p 表示标签。合并之后的2 种代码坏味训练集结构为:每一行代表代码坏味实例,每一列代表度量特征,最后一列为标签信息,以此形成一种矩阵数据 $\boldsymbol{M}$。

$$\boldsymbol{M}=\begin{bmatrix} m_{11} & m_{12} & \cdots & p_{1n} \\ m_{21} & m_{22} & \cdots & p_{2n} \\ \vdots & \vdots & & \vdots \\ m_{k1} & m_{k2} & \cdots & p_{kn} \end{bmatrix}$$

3.2　神经网络模型设计

AHSAN 等人[21]在肌电图(EMG)信号的基础上,利用人工神经网络检测不同的预定义手部运动(上、下、左、右),所设计的网络能够成功识别手部运动。王毅等人[22]将卷积神经网络和长短期记忆神经网络相结合,对用户伪装入侵模式的缺陷进行检测,其检测效果优于基准系统,从而验证了该方法的有效性。神经网络在分类方面具有优势,利用 BP 神经网络模型可以较好地预测代码坏味。BP 神经网络模型结构如图 2 所示。

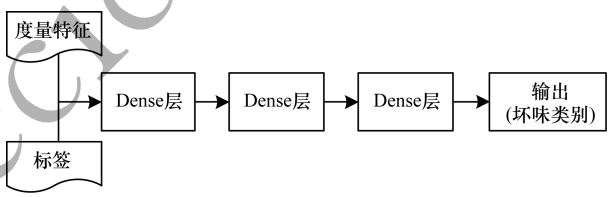

图 2　BP 神经网络分类模型结构

Fig. 2　Structure of BP neural network classification model

在对数据集进行预处理后,将数据集输入到神经网络模型中,代码坏味检测的具体步骤如下:

输入　带有标签的代码坏味矩阵数据样本 $\boldsymbol{M}$

输出　代码坏味类别

步骤 1　建立神经网络分类器模型,构建的神经网络结构采用全连接形式,第 1 层为一个输入层,第 2 层是隐藏层,网络的最后一层是输出层,输出层采用 Softmax 函数并输出代码坏味的类别。

步骤 2　将数据集预处理之后的代码坏味度量特征的矩阵数据 $\boldsymbol{M}$ 作为输入层的输入,将标签信息作为网络输出基准,表示为 Y_{real},输出层的输出值表示为 Y_{pred},如果 Y_{real}与 Y_{pred}之间的误差小于当前设定的阈值或者训练迭代轮数达到阈值,则完成神经网

络对代码坏味预测输出的训练,否则返回神经网络输入层进行模型训练。

步骤 3 将 FONTANA 等人提出的代码坏味公开数据集作为基准代码坏味测试集,并且按照数据集预处理过程中所述方式对测试数据进行合并与向量形式转换。将得到的测试集输入训练好的神经网络模型中,模型自动输出预测的代码坏味类别。

3.3 评估模型

在训练与测试阶段需要监控样本上的损失和精度,并以准确度(Accuracy)、F1 值以及 AUC 值评价指标评估神经网络模型的分类性能。准确度、F1 值具体计算公式如式(8)~式(11)所示:

$$\text{Accuracy}=\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}\times 100\% \tag{8}$$

$$\text{Precision}=\frac{\text{TP}}{\text{TP}+\text{FP}}\times 100\% \tag{9}$$

$$\text{Recall}=\frac{\text{TP}}{\text{TP}+\text{FN}}\times 100\% \tag{10}$$

$$\text{F1}=\frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}\times 100\% \tag{11}$$

其中,TP 代表样本正类,TN 代表样本负类,FP 为将错误的样本分类成正确样本的数量,FN 则是将正确样本分类成错误样本的数量。

F1 值可以看作模型精确率和召回率的一种加权平均值,取值范围为 0~1。在分类任务中,期望精确率和召回率都达到很高,但实际上不可能实现。因此,需要获取两者之间的平衡点,而 F1 值可以看作此平衡点,F1 值越高说明精确率和召回率同时达到较高且取得了平衡。

AUC 值被定义为 ROC 曲线下的面积,其不受阈值的影响。作为数值类型,AUC 值越大的分类器效果越好。

4 实验验证

NUCCI 等人对 FONTANA 提出的代码坏味公开数据集进行合并,使其中拥有不同的坏味类型。本文基于该数据集,在相同的条件下利用 Weka 工具提供的机器学习算法和本文方法分别进行代码坏味检测,以验证本文代码坏味检测方法的性能。

4.1 实验过程

本文实验在 Ubuntu14.04 LTS 环境下进行,模型代码基于 Keras 深度学习框架实现,使用 Tensorflow 作为计算后端引擎。神经网络模型采用全连接形式,以 batch size = 10 的形式组成小批量对网络进行梯度更新,网络的训练迭代轮数 epoch 设置为 500。

本次实验通过代码坏味检测工具对 7 个 Java 开源项目进行检测,以获取代码坏味实例,并将数据类、上帝类、特征依恋以及长方法的 4 种数据集进行合并,生成类级别和方法级别的 2 种类型数据集,使代码坏味数据集中包含不同坏味类型以及度量特征值。利用 FONTANA 等人[12]提出的公开代码坏味数据集作为基准测试集,对训练好的模型进行测试。具体过程如下:

1)对数据集中的度量特征、代码坏味实例以及标签进行预处理。本文利用代码坏味自动检测工具获取到代码坏味实例并进行标记生成标签,结合度量特征与标签信息得到代码坏味训练集。考虑到基准数据集为 Arff 格式,不利于输入 BP 神经网络,本文利用 Python 实现 CSV 数据集格式与 Arff 数据集格式的相互转换。为了验证代码坏味预测模型的实际能力,将上帝类和数据类的数据集合并为类级别的数据集,将特征依恋和长方法的数据集合并为方法级别的数据集。合并之后的 2 种数据集分别包含 840 种代码坏味实例。

2)神经网络模型训练。将预处理后的代码坏味数据集作为网络输入层的输入,标签值作为网络的预期输出,实现网络对代码坏味分类输出的训练,从而得到 BP 神经网络分类模型。

3)模型优化。在模型优化阶段,利用十折交叉验证法验证模型对代码坏味的预测精度,以避免在训练过程中出现过拟合现象。模型以分类交叉熵作为损失函数,降低 BP 神经网络对代码坏味的预测错误率,选择自适应学习率的 Adam 进行模型参数学习。

4.2 结果分析

实验从以下 3 个方面分析本文基于 BP 神经网络的代码坏味检测方法与 NUCCI 等人使用的机器学习算法在相同数据集上的代码坏味检测结果:

1)数据集中包含不同类型的代码坏味对于神经网络分类器的效果影响。为了探究数据集中存在不同类型的代码坏味对神经网络分类器检测效果的影响,将方法级别和类级别的代码坏味数据集分别作为神经网络的分类输入。各检测方法在不同数据集中的准确度、F1 值和 AUC 值结果分别如表 2~表 5 所示。

表 2 数据类数据集的测试结果

Table 2 Test results of Data class dataset

检测方法	评价指标		
	准确度/%	F1 值/%	AUC 值
J48	82.13	58.21	0.87
Random Forest	71.64	10.54	0.82
Naive Bayes	65.47	47.61	0.85
JDeodorant	73.80	11.52	0.75
本文方法	94.51	93.62	0.94

表 3　上帝类数据集的测试结果
Table 3　Test results of God class dataset

检测方法	评价指标		
	准确度/%	F1 值/%	AUC 值
J48	83.17	46.36	0.88
Random Forest	70.55	8.98	0.81
Naive Bayes	83.37	56.42	0.88
JDeodorant	71.69	11.67	0.79
本文方法	92.33	91.34	0.91

表 4　特征依恋数据集的测试结果
Table 4　Test results of Feature envy dataset

检测方法	评价指标		
	准确度/%	F1 值/%	AUC 值
J48	80.27	49.57	0.84
Random Forest	71.11	9.75	0.81
Naive Bayes	79.65	44.15	0.83
JDeodorant	75.67	15.21	0.71
本文方法	84.35	75.21	0.81

表 5　长方法数据集的测试结果
Table 5　Test results of Long method dataset

检测方法	评价指标		
	准确度/%	F1 值/%	AUC 值
J48	83.92	29.37	0.83
Random Forest	69.51	4.97	0.54
Naive Bayes	79.88	48.75	0.79
JDeodorant	81.27	14.79	0.70
本文方法	95.33	90.35	0.82

实验结果表明,本文方法在数据集存在不同坏味类型的情况下,平均准确度达到 91.63%,平均 F1 值达到 87.63%,在 AUC 值上与其他检测方法相比没有明显差别,但在分类整体效果上优于其他检测方法。以 J48 方法为例,本文方法在平均准确度上提高了 9.26%,平均 F1 值提高了 41.75%。对比基于度量和规则的代码坏味检测工具 JDeodorant,本文方法在平均准确度上提高了 16.03%,平均 F1 值提高了 74.33%。综合基于机器学习的代码坏味检测方法和基于度量的代码坏味检测工具 JDeodorant,本文方法在总体平均准确度上提升了 15.19%,平均 F1 值提升了 58.39%。

2)神经网络模型构建过程中的参数设置。在利用 BP 神经网络模型检测代码坏味的过程中,网络隐藏层神经元的数量选择尤为重要。根据 2 种数据集含有不同的代码坏味特征,本文对类级别数据集和方法级别数据集的模型神经元个数设置如表 6 所示。

表 6　神经元数量设置
Table 6　Number setting of neurons

数据集	输入层	隐藏层	输出层
类级别数据集	8	16	3
方法级别数据集	10	25	3

在网络中,隐藏层采用 ReLU 激活函数,输出层采用 Softmax 激活函数。

3)代码坏味分类的准确度。为了能够形象描述本文模型在训练阶段与测试阶段的代码坏味分类准确度,记录样本在分类过程中的准确度,结果如图 3 ~ 图 6 所示。从中可以看出,神经网络分类器在迭代轮数为 400 ~ 500 时能达到最佳整体分类性能。

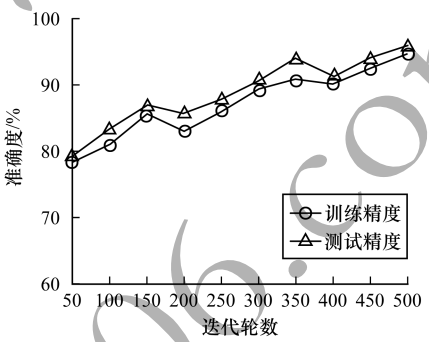

图 3　数据类数据集的准确度
Fig. 3　Accuracy of Data class dataset

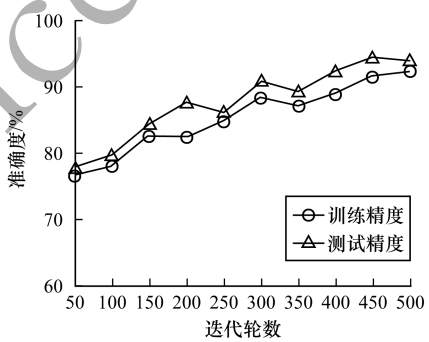

图 4　上帝类数据集的准确度
Fig. 4　Accuracy of God class dataset

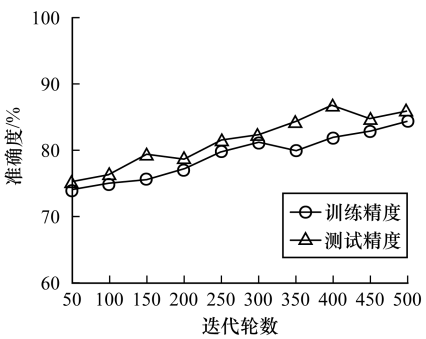

图 5　特征依恋数据集的准确度
Fig. 5　Accuracy of Feature envy dataset

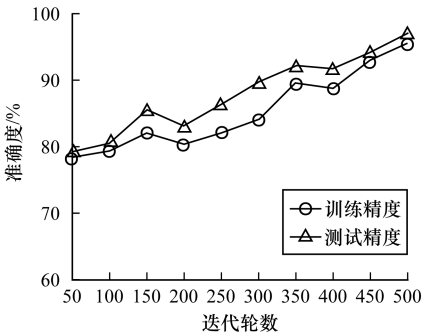

图 6 长方法数据集的准确度

Fig. 6 Accuracy of Long method dataset

5 结束语

代码坏味检测对程序质量具有重要影响,本文提出一种基于 BP 神经网络的代码坏味检测方法,将常见的软件度量特征项与代码坏味实例信息相结合,并将收集到的代码坏味类别合并成类级别和方法级别的 2 种类型数据集,以此作为神经网络模型的输入,模型输出代码坏味的分类类别。实验结果表明,与 J48、Random Forest 等代码坏味检测方法相比,该方法在测试集上的整体效果更优。

在实际的代码坏味检测中,收集到的相关数据集中正样本数量与负样本数量存在很大的不平衡性,影响了预测结果的准确度,为解决该问题,研究人员通常手动选取数据集的正负样本数量,以达到数据集的正负样本平衡,该方式会耗费大量的人力且难以保证检测结果的正确性。因此,下一步将基于生成式对抗神经网络对代码坏味进行检测,以解决数据集正负样本不平衡的问题。

参考文献

[1] FOWLER M. Refactoring: improving the design of existing code[M]//DON W, LAURIE W. Extreme programming and agile methods—XP/agile universe 2002. Berlin, Germany: Springer, 2002.

[2] ARCELLI FONTANA F, BRAIONE P, ZANONI M. Automatic detection of bad smells in code: an experimental assessment[J]. The Journal of Object Technology, 2012, 11(2): 1-38.

[3] FOKAEFS M, TSANTALIS N, CHATZIGEORGIOU A. JDeodorant: identification and removal of Feature envy bad smells[C]//Proceedings of 2007 IEEE International Conference on Software Maintenance. Washington D. C., USA: IEEE Press, 2007: 519-520.

[4] JIANG Dexun, MA Peijun, SU Xiaohong, et al. Related work analysis of code bad smell detection and refactoring[J]. Intelligent Computer and Applications, 2014, 4(3): 23-27. (in Chinese)
姜德迅,马培军,苏小红,等. 代码坏味检测及重构的现状分析[J]. 智能计算机与应用,2014,4(3):23-27.

[5] 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.

[6] AL DALLAL J. Identifying refactoring opportunities in object-oriented code: a systematic literature review[J]. Information and Software Technology, 2015, 58: 231-249.

[7] DI NUCCI D, PALOMBA F, TAMBURRI D A, et al. Detecting code smells using machine learning techniques: are we there yet? [C]//Proceedings of 2018 IEEE International Conference on Software Analysis, Evolution and Reengineering. Washington D. C., USA: IEEE Press, 2018: 612-621.

[8] MÄNTYLÄ M V, LASSENIUS C. Subjective evaluation of software evolvability using code smells: an empirical study[J]. Empirical Software Engineering, 2007, 11(3): 395-431.

[9] MOHA N, GUEHENEUC 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, 2010, 36(1): 20-36.

[10] ZHANG Xiaofang, ZHU Can. Empirical study of code smell impact on software evolution[J]. Journal of Software, 2019, 30(5): 1422-1437. (in Chinese)
章晓芳,朱灿. 代码坏味对软件演化影响的实证研究[J]. 软件学报,2019,30(5):1422-1437.

[11] PALOMBA F, PANICHELLA A, DE LUCIAA, et al. A textual-based technique for smell detection[C]//Proceedings of 2016 IEEE International Conference on Program Comprehension. Washington D. C., USA: IEEE Press, 2016: 1-10.

[12] FONTANA F A, ZANONI M, MARINO A, et al. Code smell detection: towards a machine learning-based approach[C]//Proceedings of 2013 IEEE International Conference on Software Maintenance. Washington D. C., USA: IEEE Press, 2013: 159-168.

[13] KREIMER J. Adaptive detection of design flaws[J]. Electronic Notes in Theoretical Computer Science, 2005, 141(4): 117-136.

[14] KHOMH F, VAUCHER S, GUÉHÉNEUC Y G, et al. BDTEX: a GQM-based Bayesian approach for the detection of antipatterns[J]. Journal of Systems and Software, 2011, 84(4): 559-572.

[15] MAIGA A, ALI N, BHATTACHARYA N, et al. SMURF: a SVM-based incremental anti-pattern detection approach[C]//Proceedings of 2012 Working Conference on Reverse Engineering. Washington D. C., USA: IEEE Press, 2012: 466-475.

[16] YANG J C, HOTTA K, HIGO Y, et al. Filtering clones for individual user based on machine learning analysis[C]//Proceedings of 2012 International Workshop on Software Clones. Washington D. C., USA: IEEE Press, 2012: 455-469.

(下转第 230 页)

(上接第 222 页)

[17] PALOMBA F,BAVOTA G,PENTA M D,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.

[18] LIU Liqian,DONG Dong. Long method detection based on cost-sensitive integrated classifier[J]. Computer Science,2018,45(11A):497-500. (in Chinese)
刘丽倩,董东. 基于代价敏感集成分类器的长方法检测[J]. 计算机科学,2018,45(11A):497-500.

[19] BU Yifan,LIU Hui,LI Guangjie. A God class detection method based on deep learning[J]. Journal of Software,2019,30(5):161-176. (in Chinese)
卜依凡,刘辉,李光杰. 一种基于深度学习的上帝类检测方法[J]. 软件学报,2019,30(5):161-176.

[20] YI J Q,KUROGI S,MATSUOKA K. Back-propagation learning of neural networks for translation invariant pattern recognition[J]. Systems and Computers in Japan,1991,22(14):80-89.

[21] AHSAN M R,IBRAHIMY M I,KHALIFA O O. Neural network classifier for hand motion detection from EMG signal[M]. Berlin,Germany:Springer,2011.

[22] WANG Yi,FENG Xiaonian,QIAN Tieyun,et al. CNN and LSTM deep network based intrusion detection for malicious users[J]. Journal of Frontiers of Computer Science and Technology,2018,12(4):575-585. (in Chinese)
王毅,冯小年,钱铁云,等. 基于 CNN 和 LSTM 深度网络的伪装用户入侵检测[J]. 计算机科学与探索,2018,12(4):575-585.

编辑　吴云芳