EasyChair Preprint

№ 3918

A Code Naturalness Based Defect Prediction Method at Slice Level

Xian Zhang, Ke-Rong Ben and Jie Zeng

July 21, 2020

软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于代码自然性的切片粒度缺陷预测方法*

张 献, 贲可荣, 曾 杰

(海军工程大学 电子工程学院,湖北 武汉 430033)

通讯作者: 张献, E-mail: tomtomzx@foxmail.com

摘 要: 软件缺陷预测是软件质量保障领域的一个活跃话题,它可以帮助开发人员发现潜在的缺陷并更好地利用资源.如何为预测系统设计更具判别力的度量元,并兼顾性能与可解释性,是人们致力于的研究方向.针对这一挑战,提出了一种基于代码自然性特征的缺陷预测方法——CNDePor.该方法通过正逆双向度量代码和利用质量信息对样本加权的方式改进语言模型,提高了模型所得交叉熵(CE)类度量元的缺陷判别力.针对粗粒度缺陷预测存在难以聚焦缺陷区域、代码审查成本高的不足,研究了一种新的细粒度缺陷预测问题——面向语句的切片级缺陷预测.针对这一问题设计了 4 种度量元,并在两类安全缺陷数据集上验证了度量元和 CNDePor 方法的有效性.实验结果表明:CE 类度量元具有可学习性,它们蕴涵了语言模型从语料库中学习到的相关知识;改进的 CE 类度量元的判别力明显优于原始度量元和传统规模度量元;CNDePor 方法较传统缺陷预测方法和已有的基于代码自然性的方法有显著优势,较先进的基于深度学习的方法具有可比性性能和更强的可解释性.

关键词: 软件质量保障;缺陷预测;代码自然性;切片粒度;语言模型;交叉熵;深度学习

中图法分类号: TP311

中文引用格式: 张献,贲可荣,曾杰.基于代码自然性的切片粒度缺陷预测方法.软件学报. http://www.jos.org.cn/1000-9825/0000.htm

英文引用格式: Zhang X, Ben KR, Zeng J. A code naturalness based defect prediction method at slice level. Ruan Jian Xue Bao/Journal of Software, 2021(in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

A Code Naturalness Based Defect Prediction Method at Slice Level

ZHANG Xian, BEN Ke-Rong, ZENG Jie

(School of Electronic Engineering, Naval University of Engineering, Wuhan 430033, China)

Abstract: Software defect prediction is an active research topic in the domain of software quality assurance. It can help developers find potential defects and make better use of resources. How to design more discriminative metrics for the prediction system, taking into account performance and interpretability, is a research direction that people devote to. Aiming at this challenge, a code naturalness feature based defect predictor method (CNDePor) is proposed. This method improves the language model by taking advantage of the bidirectional code-sequence measurement and weighting the samples by using the quality information, so as to increase the defect discrimination of the cross-entropy (CE) type metrics obtained from the model. Aiming at the shortcomings of coarse-grained defect prediction (e.g. difficulties in focusing on defect areas and high cost of code reviews), a new fine-grained defect prediction problem, statement-oriented slice level defect prediction, is studied. Four metrics are designed for this problem, and the effectiveness of these metrics and CNDePor are verified on two types of security defect datasets. The experimental results show that: CE-type metrics are learnable, which contain the relevant knowledge learned from the corpus by language model; the improved CE metrics are significantly better than the original metrics and traditional size metrics; the CNDePor method has significant advantages over the traditional defect prediction methods and an existing

* 基金项目: 国家安全重大基础研究计划项目(613315)
Foundation item: National Security Program on Key Basic Research Project of China (613315)
收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00
CNKI 在线出版时间: 0000-00-00

method based on code naturalness, and own comparable performance and stronger interpretability than a state-of-the-art mothed based on deep learning.

Key words: software quality assurance; defect prediction; code naturalness; slice granularity; language model; cross-entropy; deep learning

随着信息技术的快速变革,软件已全面渗透至现代社会中,人类正在步入“软件无处不在、软件定义一切、软件使能一切”的时代[1].然而随着系统规模与复杂性的快速增长、不同新软件模式的广泛运用,软件系统内外各种的非确定性不断增强,软件质量问题日渐突出.当软件缺陷(defects)一旦在航天、交通、军事等安全攸关领域触发时,所导致的后果和损失是巨大的.由于软件系统的机理及复杂性决定,软件缺陷常常不可避免.为此,如何有效地预测、发现并排除缺陷,提高软件质量及可靠性已成为人们关注的重要问题.

鉴于此,软件工程领域研究者提出了软件缺陷预测(software defect prediction)技术[2-3],其目的在于提前预测可能存在缺陷的软件模块(如文件和函数).该技术通过从软件历史仓库中提取相关度量元(metrics,也称作特征)来表征被预测的软件模块,然后将这些度量元输入给预测模型(如分类器、回归器等)进行训练,进而对新产生的软件模块预测其存在缺陷的可能性[4].其研究意义在于[5]:及时发现缺陷,提高软件质量;优化资源分配,节省维护成本.从软件缺陷预测的一般过程来看,影响系统实际应用效果的关键因素包括以下三个方面[2]:度量元的设计、预测模型的构建方法、数据集的相关问题.为增强缺陷预测系统的性能与能力,不少学者致力于从这些关键因素出发进行突破与创新.从调研情况看,近年来该领域研究成果呈增长趋势,相关研究热度持续升高[6].然而现有工作还存在一些不足,亟待解决的挑战有:

挑战 1: 设计更具判别力的度量元,并兼顾性能与可解释性.为缺陷预测模型设计有效的度量元一直是研究者关注的核心问题[3,7].近年来迅速兴起的深度学习方法已开始在软件缺陷预测领域发挥重要作用,其往往能够带来远超传统方法的性能.但深层网络结构自动生成的语义度量元缺乏物理含义、可解释性差,且大多数“免模型(model free)”的深度学习算法仍被视作“黑箱”.鉴于此,不少学者呼吁兼顾模型性能与可解释性[8-9].

挑战 2: 更多关注细粒度缺陷预测.由于细粒度缺陷预测关注的模块规模更小,因此其更利于定位故障、缩小测试和审查范围.但从已发表的文献来看,现有工作主要针对粗粒度软件实体[2,5],如包级或文件/类级,而对细粒度缺陷预测问题的研究较少.鉴于此,学术界和产业界均对细粒度缺陷预测提出了更多呼声[2,5-6,10-11].

近年来兴起的基于语言模型的代码自然性(code naturalness)分析技术[12]为**挑战 1** 提供了一种新思路.语言模型源自统计自然语言处理[13],其可通过语料库学习捕获语言发生规律和使用模式.借助这种能力,语言模型可以挖掘软件语料库,衡量代码的自然性,通常有缺陷的代码更加“不自然”[14].代码自然性一般由语言模型度量的交叉熵(cross-entropy,简称 CE)值表征[15-16],受语料库及模型影响,代码的自然性特征具有不确定性.受已有工作启发,本文提出了一种基于代码自然性特征的缺陷预测方法(a Code Naturalness feature based Defect Predictor method,简称 CNDePor).该方法针对现有语言模型仅对代码进行单向度量和未能利用缺陷信息的不足构建了双向加权神经语言模型(neural language model,简称 NLM).该模型一方面利用软件质量信息对样本加权,增强/抑制了模型对无/有缺陷代码的学习强度,提升了 CE 类度量元的缺陷判别力;另一方面还实现了对输入序列的正逆双向学习,得到的两种改进度量元更全面地刻画了代码的自然性.基于此模型,CNDePor 方法将缺陷预测阶段训练样本的标签信息共享给 NLM 学习,然后将语言模型度量的自然性特征与传统软件度量元相结合一同输入给预测模型,改善了缺陷预测性能.

程序切片(program slicing)是一种分析和理解程序的技术[17],它善于从被测程序中抽取出满足准则要求的有关语句,忽略许多与此无关的语句.为此,借鉴程序切片的优势来研究缺陷预测技术可为**挑战 2** 提供一种新视角.本文在面向语句的切片级粒度上实现了度量元设计和缺陷预测方法应用.这一预测粒度分析的软件模块是由一个或多个面向语句的程序切片构成,其中切片准则依据具体缺陷类型设计.具体地,设计了 4 种度量元,包括 2 种代码自然性特征和 2 种规模特征.利用 CNDePor 方法,在两类切片粒度缺陷数据集(缓冲区错误和资源管理

错误)上进行了缺陷预测应用及方法验证.经统计,使用的切片粒度模块的平均代码行为 9.16(远小于常用的文件粒度模块),这非常利于缺陷的进一步查找与确认、降低代码审查/测试成本.

本文的主要贡献可总结如下:

(1) 提出了一种基于代码自然性特征的缺陷预测方法(CNDePor).该方法将代码正序序列和逆序序列的 CE 值作为一类新的度量元引入到缺陷预测问题中,同时利用质量信息对样本加权学习,改进了原有语言模型,提升了 CE 类度量元的判别力.

(2) 设计了两种新的代码度量元:修正的代码交叉熵度量元(M-CE)和修正的代码逆序序列交叉熵度量元(M-CE-Inv).这两种度量元从不同角度刻画了软件模块的自然性并具有可学习性.

(3) 研究了一种新的细粒度缺陷预测问题.在面向语句的切片级粒度上初步探讨了度量元设计和缺陷预测方法应用,所做工作具有一定参考意义.

本文第 1 节介绍代码自然性和软件缺陷预测方面的研究背景和相关工作.第 2 节介绍基于代码自然性特征的缺陷预测方法的设计与实现.第 3 节介绍面向语句的切片粒度模块生成方法.第 4 节是实验设计,介绍研究问题、实验数据集、评价指标、实验环境与参数设置.第 5 节对研究结果进行详细分析.最后总结全文并提及未来工作.

1 相关工作

1.1 代码自然性分析

自然语言无疑是复杂的、丰富的、强大的.但在实际使用中,受认知限制和日常生活的紧迫性,大多数人类话语都十分简单、颇具重复性和可预测性[15-16].受此事实启发,2012 年 Hindle 等人[15]开始猜想:大多数程序语言编写的软件代码也是“自然的”,因为它们是由人类在工作中创造的,因此与自然语言话语一样,它们也可能是重复的和可预测的.同时它们具有有用的可预测统计特性,可以被统计语言模型捕获并用于软件工程任务.这便是他们提出的**自然性假设**(naturalness hypothesis).针对这一假设,Hindle 等人基于经典的 n-gram 语言模型给出了经验性论据.研究发现,在测试语料库上程序语言具有远比自然语言显著的“自然性”,即重复性、可预测性更强.在此基础上,该团队围绕代码自然性先后发表了一系列代表性成果[14,18-21].

语言序列或语料库的自然性一般由语言模型度量的**交叉熵**(CE)值表征[15-16](受语料库及模型影响,代码的自然性特征具有不确定性),如图 1 所示.若 CE 值越低,则表明序列的发生概率越高(在平均词汇规模意义上),也即序列更加“自然”.本质上,语言模型可以视作是一个语言学习者,通过语料库的学习,其可捕获语言发生规律和使用模式.在挖掘语料库的过程中,语言模型会对常见的序列赋予更高的自然性,即更低的 CE 值,而罕见的序列则反之.通常那些罕见的序列是令人惊奇的,甚至是不合理的.因此换言之,CE 值可理解为是一种序列异常度或“惊奇度(surprisingness)”[15]的度量,有助于帮助缺陷预测、缺陷检测等工作.

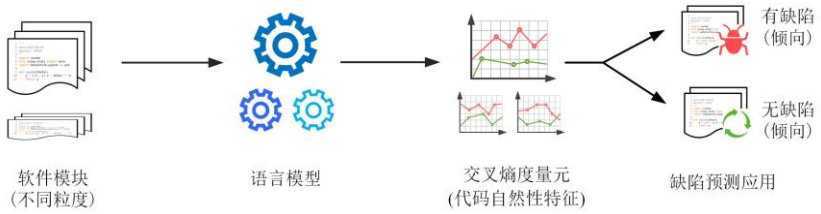


Fig.1 Measurement of code naturalness and its application in defect prediction

图 1 代码自然性特征的度量及其缺陷预测应用

考虑到编译器错误消息通常无法精确定位语法错误的实际位置(如错误的分号或大括号可能会导致文件的许多位置被报告为错误),Campbell 等人[21]利用 *n*-gram 语言模型实现了一个自动 Java 语法错误定位器,以增强编译器的语法错误定位效果.经实验分析发现,语法错误拥有较高的 CE 值,即是“不自然的”.进一步地,2016 年

Ray 等人[14]关注了有缺陷代码的自然性(不仅仅是语法错误),并提出了**不自然性假设**:“不自然代码更有可能是有缺陷的”.作者利用改进的 *n*-gram 模型[19]对 10 个 Java 项目的 7139 个 bug-fix 提交单进行了统计分析,得出了如下发现:有缺陷的代码倾向于拥有更高的 CE 值(即不自然),随着错误的修复代码的 CE 值降低.受上述工作启发,Jimenez[22]探讨了代码自然性在文件级安全缺陷预测中的应用.面向 3 个知名开源系统,他们发现 *n*-gram 模型度量的自然性特征(通过调整模型参数等方法获得一组特征值[23])的预测性能不佳,但通过特征组合可以为基于传统代码度量元的方法带来性能改进.他们还分析了 20 个 Java 开源项目源文件的自然性[23],并发现有缺陷文件的 CE 值高于无缺陷文件,这一结果与 Ray 等人相吻合.Allamanis 等人[12]综述了近几年代码自然性分析方面的相关工作,可以看到这一方向的研究热度持续上升.

1.2 切片粒度软件缺陷预测

软件缺陷预测技术虽然已有 40 多年的历史,但现有研究工作主要针对粗粒度的软件实体[2,5].考虑到粗粒度方法难以精确聚焦缺陷区域,致使在实际中开发人员需要投入过多的时间和精力审查/测试可疑模块,实用价值受到影响,为此不少研究者开始呼吁向细粒度缺陷预测投入更多精力[5-6,10].

程序切片是一种分析和理解程序的技术,它通过寻找程序内部的相关性来分解程序,再通过对分解所得程序切片的分析达到对整个程序的理解.程序切片以切片准则(slicing criterion)为标准,从被测程序中抽取出满足准则要求的有关语句,忽略许多与此无关的语句.因此该技术有利于故障定位、程序调试、软件测试等任务[17,24].鉴于程序切片的优势,已有一些学者尝试将这一技术用于软件缺陷预测问题[25-29].其中主要成果[25-28]是将基于切片的软件度量元用于指导缺陷预测,而并没有将程序切片得到的模块本身作为分析对象进行缺陷预测研究.与此不同,王俊[29]提出了一种新的缺陷预测粒度——切片粒度.该粒度是通过对程序中的函数按照其在程序调用图中的距离进行聚类得到的,即待测软件模块是距离近的函数重新组合形成的切片.因此,本质上王俊[29]分析的软件模块是一种面向函数的粗粒度切片,这种切片包含多个函数,所以该工作依然属于粗粒度缺陷预测范畴.在安全缺陷检测领域,李珍等[30]利用程序切片技术抽取出与漏洞敏感库/API 函数调用相关的语句,构建了一个包含两类常见缺陷的程序切片模块数据集,然后为识别软件模块中是否含有漏洞提出了一种基于深度学习的自动检测方法 VulDeePecker,其中他们使用的切片为面向语句的细粒度切片[24].受上述工作启发,本文将在一种新的细粒度上,即面向语句的切片粒度上讨论度量元设计和缺陷预测方法.

1.3 基于深度学习的软件缺陷预测

预测模型的构建方法是影响软件缺陷预测系统性能的关键因素之一.为了建立有效的预测模型,研究者已尝试使用了众多机器学习算法[3,31],包括逻辑回归[32]、支持向量机(support vector machine,简称 SVM)[33]、随机森林(random forest,简称 RF)[34]、字典学习[35]、迁移学习[36]和判别分析[37],以及近年来迅速兴起的深度神经网络(deep neural network,简称 DNN)模型等.下面着重对基于深度学习的缺陷预测工作进行叙述.

为获得更具表达力的特征,Yang 等[38]将深度信念网络(deep belief network,简称 DBN)模型视作一个特征融合器用于 14 种变更度量元的融合与变换,然后基于新特征及逻辑回归分类器进行缺陷预测.实验表明,在即时缺陷预测问题上,上述方法发现的缺陷数比 Kamei 等人的方法[11]多出 32.2%.考虑到现有工作忽视了变更与变更间的演化信息,Wen 等[39]利用基于长短期记忆(long short-term memory,简称 LSTM)单元的循环神经网络(recurrent neural network,简称 RNN)对变更序列进行缺陷倾向性预测,其中每个变更由 6 类手工设计度量元表征.实验表明,该工作较传统方法 F1 指标提升幅度超过 31.6%.面向软件安全缺陷预测,Clemente 等[40]收集了 3 类软件度量元并对比了多层感知机(multilayer perceptron,简称 MLP)模型与其他 4 种传统分类器的预测性能,实验结果表明 DNN 模型表现最佳.

另一种引入深度学习方法的动机是为减少先验知识依赖、降低人工成本.这类工作主要利用深度学习模型自动生成软件代码的语义特征,从而避免了由人类专家手工设计特征带来的局限性.Wang 等人[41-42]首次提出利用 DBN,一种无监督深度学习模型,从源代码中学习出程序的语义表示,然后输入给分类器进行缺陷预测.通过大量的验证实验,结果表明自动学习出的语义特征较传统手工设计特征显著改善了缺陷预测性能.基于相似思

路,Dam 等[43]提出了一种用于自动学习代码(abstract syntax tree,简称 AST)节点表示的树结构 LSTM 网络,然后基于这些学习出的实值特征预测软件文件的缺陷倾向性.上述工作均将 DNN 用作无监督的学习模型,与他们不同,Li 等人[44]提出了基于卷积神经网络(convolutional neural network,简称 CNN)的缺陷预测方法,并证实自动学习出的特征与传统手工设计特征相结合,可以产生更多的性能提升.因此,这意味着手工设计特征和自动生成特征之间存在互补性.Phan 等[45]针对代码的汇编指令序列,构建了一种基于 CNN 的预测方法,用于特征学习及缺陷预测,并在 4 个真实数据集上进行了应用.实验结果说明,该方法可有效检测语义错误.为识别 Android 二进制可执行文件中的缺陷,董枫等[46-47]依据关键指令集对 apk 反编译的 samli 文件进行序列化,然后将序列化数据输入给 MLP 进行特征学习与缺陷预测.实验表明相比浅层模型,DNN 性能提升明显.另一篇相关工作是李珍等人的缺陷检测研究[30],他们为检测软件漏洞,利用程序切片技术抽取出与漏洞敏感库函数调用/API 函数调用相关的语句,然后基于双向 LSTM 对语句的 token 序列进行分析,以检测软件模块是否含有特定类型的安全缺陷.

Table 1 Taxonomy of software defect prediction works based on deep learning

表 1 基于深度学习的软件缺陷预测工作分类

深度学习模型的作用	是否利用缺陷标签训练神经网络	相关工作	深度学习模型	技术特点
特征融合器	无监督	Yang 等[38]	DBN	DNN 面向手工设计特征进行融合与变换;有监督类型中 DNN 与分类器有机融合,一同训练;手工设计特征的可解释性强.
	有监督	Wen 等[39]	LSTM	
		Clemente 等[40]	MLP	
特征生成器	无监督	Wang 等[41-42]	DBN	DNN 面向软件代码(源码、AST、二进制码等)自动生成语义特征;有监督类型中 DNN 与分类器有机融合,一同训练;自动生成的特征可解释性差.
		Dam 等[43]	Tree-LSTM	
	有监督	Li 等[44]	CNN	
		Phan 等[45]	CNN	
		董枫等[46-47]	MLP	
		李珍等[30]	Bi-LSTM	
语言模型建模	无监督	本文作者[48-49]	神经语言模型	利用 DNN 构建语言模型;基于语言模型生成表征代码自然性特征的交叉熵(CE)值;CE 类度量元的可解释性强.
	有监督	**本文**	加权神经语言模型	

综上所述,目前已有的基于深度学习方法的软件缺陷预测工作可以分为两大类(如表 1 所示):一类是将 DNN 视作特征融合器,即“手工设计特征+深度学习”模式,其中 DNN 用于特征的融合与变换;另一类是将 DNN 视作特征生成器,即“原始数据+深度学习”模式,其中 DNN 分析的对象并非人类专家设计的度量元数据,而是(向量化后的)原始软件代码,如实数编码的 token 序列、AST 节点序列等.在这类工作中,DNN 可从原始数据中自动学习出高阶语义特征,进而完成缺陷预测任务;但另一方面,学习出的自动特征缺乏物理含义,面临可解释性差问题.与上述工作不同,本文是利用 DNN 构建语言模型,然后基于语言模型度量的代码 CE 类度量元完成缺陷预测任务.因此,无论从模型角度还是度量元角度,我们的工作[50]均具有理论基础,可解释性更强.

2 基于代码自然性特征的缺陷预测方法

2.1 方法框架

本文提出的 CNDePor 方法框架如图 2 所示.该方法首先利用包含质量信息的代码语料库训练一个双向神经语言模型;然后基于此训练好的模型完成软件模块修正的交叉熵值(M-CE)和修正的逆序序列交叉熵值(M-CE-Inv)的自动计算,其中这两个特征从不同角度刻画了软件模块的自然性;最后将此得到的新度量元与其他特征相结合,一同执行软件缺陷预测任务,其中 CNDePor 的预测目标为软件模块的缺陷倾向性.

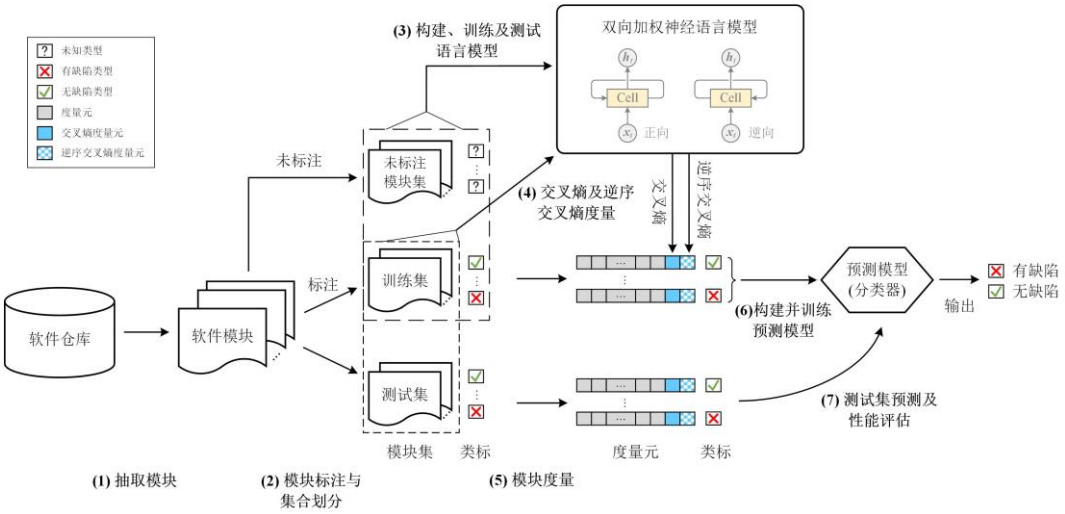


Fig.2 Framework of CNDePor method

图 2 CNDePor 方法框架

CNDePor 方法主要包含如下 7 个步骤:

步骤1: 抽取软件模块.从软件仓库中提取感兴趣的软件模块构成集合 $C_0=\{S_k\}_k^{N_C}$,这里模块的粒度依据缺陷预测任务而定,可以是包级、文件级或更细粒度等.

步骤2: 软件模块标注与集合划分.根据每个模块 S_k 包含的缺陷数标记其质量属性 γ_k,即有缺陷或无缺陷类型,同时对未标注模块的 γ_k 赋予未知类型.然后依据模块的标签将集合 C_0 划分为 3 个互不相交的子集合,即未标注模块集 C_{unkown}、训练集 C_{train} 和测试集 C_{test}.其中 C_{train} 和 C_{test} 是对已标注模块集的一种随机划分.

步骤3: 构建、训练及测试语言模型.利用 C_{unkown} 和 C_{train} 集合内模块的代码数据和质量信息训练双向加权语言模型 $\overrightarrow{\tilde{M}}$ 和 $\overleftarrow{\tilde{M}}$,然后利用 C_{test} 集合进行性能测试,其中测试阶段不使用样本的标签信息.

步骤4: 交叉熵及逆序交叉熵度量.为语言模型加载训练好的(准)最优参数 $\boldsymbol{\theta}^*$,并分别利用 $\overrightarrow{\tilde{M}}^*$ 和 $\overleftarrow{\tilde{M}}^*$ 模型度量 C_{train} 和 C_{test} 中软件模块的 M-CE 值和 M-CE-Inv 值,其中熵值计算未使用模块的类标.

步骤5: 软件模块度量.利用不同度量机制收集每个待测软件模块 S_k 对应的一组度量元 $\boldsymbol{\chi}_k$,例如规模度量、内聚度量等,其中已得到的 M-CE 值和 M-CE-Inv 值将作为新的度量元融入 $\boldsymbol{\chi}_k$ 中.

步骤6: 构建并训练缺陷预测模型.首先利用软件模块的度量元及类标数据创建缺陷预测训练集 D_{train} 和测试集 D_{test}.然后构建缺陷预测模型 F_{DP},并使用 D_{train} 数据训练模型 F_{DP}.

步骤7: 测试集预测及模型性能评估.使用训练好的模型 F_{DP}^* 对测试集 D_{test} 软件模块进行预测,然后依据结果评估模型的缺陷预测性能.

由上述过程可以看出,相比我们的先前工作 DefectLearner[48](该方法未有效利用样本的质量标签数据,就缺陷预测任务而言,语言模型的训练是无监督的),语言模型的训练与缺陷预测任务耦合得更加紧密,其中语言模型利用了部分缺陷预测样本的标签信息(保证测试集样本标签不可见),使得模型度量的修正代码 CE 值和逆序 CE 值更具判别力.从而在面向缺陷预测任务时,这两种新度量元将较原有 CE 度量元带来更大贡献.为便于分析,将方法的上述运行过程划分为 3 个阶段:语料库学习阶段(阶段 I)、交叉熵度量阶段(阶段 II)和缺陷预测阶段(阶段 III),其中阶段 I 涉及步骤 1~3,阶段 II 涉及步骤 2 和步骤 4,阶段 III 涉及步骤 1~2 及步骤 5~7.下面简述这 3 个阶段的主要任务.

阶段 I(语料库学习阶段):该阶段目标是构建并训练一个神经语言模型(NLM),通过 NLM 对软件仓库的学

习,刻画编程语言的常见模式或使用规律.该阶段首先对收集到的软件项目进行数据预处理[48],例如删除注释、词化等;然后采用词嵌入方法,将词化后的代码序列表示为一组实值矢量;利用多层栈式 LSTM 网络从输入的词向量序列中学习出隐含的语义特征,进而用于序列下一词的预测;最后构建好的NLM可以对输入序列的发生概率进行估计.

阶段 II(交叉熵度量阶段):经过训练和测试后,阶段 I 设计的 NLM 将被用于度量待测软件模块的自然性,即生成用于缺陷预测的 CE 值.这一过程是度量阶段的主要任务,其中实际被度量的是模块对应的源代码序列.因此,我们需要从软件仓库中提取出模块对应的代码区域,并执行与阶段 I 相同的数据预处理工作.然后,将预处理后的代码序列输入给训练好的 NLM,进行软件模块的 CE 值计算.这里新生成的 CE 值刻画了软件模块的自然性,是对代码的一种抽象,我们把它作为一类新的度量元.

阶段 III(缺陷预测阶段):新生成的CE类度量元被引入到典型的缺陷预测问题中.一般地,除CE外还需使用其他度量方法抽取待测软件模块的不同特征,例如代码行、圈复杂度等.之后,依据代码区域是否包含缺陷对每个模块进行标注,即为它们分配一个“有缺陷”或“无缺陷”标签.最后我们将 CE 类度量元与其他特征结合起来,一同输入给预测模型(即一个分类器)进行缺陷预测.

下面重点阐述阶段 I 和阶段 II 的关键技术,阶段 III 为经典的缺陷预测过程[2-3],不再赘述.

2.2 加权神经语言模型

在本文方法阶段 I,我们提出了一种面向软件代码的加权神经语言模型(a Weighted Neural Language Model for software code,简称 W-NLM),其是阶段 I 的核心构成.需要说明的是 CNDePor 方法中的双向语言模型是指分别独立训练两个 W-NLM,一个用于学习代码的正序词序列,一个用于学习代码的逆序词序列,这里仅以前者为例进行介绍.图 3 展示了 W-NLM 模型的整体架构.对比已有的 NLM 模型[48,51]不难看出,本文方法的主要改进在于模型优化.原有的优化方法基于随机梯度下降(stochastic gradient descent,简称 SGD)算法,其构建的损失函数 J 只与预测目标和预测结果相关,因此所有样本的权重是相同的.而在 W-NLM 模型中,我们还向损失函数 J 引入了与代码相关的质量信息,用于样本加权.不同的权重信息将通过梯度 ∇J 反传,引导模型更新参数,实现针对性学习.

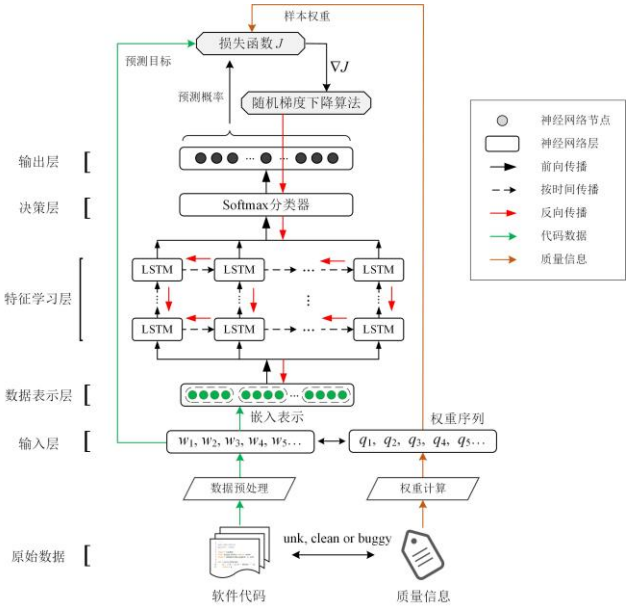


Fig.3 W-NLM model

图 3 W-NLM 模型

由图 3 展示的模型架构可知,W-NLM 采用了基于多层栈式 LSTM 单元的 RNN 进行程序语言建模.在代码表示方面,使用了词嵌入技术,分布式的词向量可以进行有效的语义表示和计算.最终,模型由一个 Softmax 多分类器预测输入序列的下一词,并以输出的概率分布来刻画整段代码的发生规律.通过挖掘代码样本语料库,模型可以学习到蕴涵于序列中的发生规律和使用模式,进而可用于代码自然性度量.特别地,W-NLM 首先在语料库构建阶段除需收集待测样本外,还需收集样本对应的质量信息.本文质量信息是指代码的质量类型,包括未知(unknown)、无缺陷(clean)和有缺陷(buggy)共 3 种.然后,经数据预处理阶段[48,50],生成待测样本的代码序列 S,并依据样本的质量信息生成对应的权重序列 Q.最后,模型输出的预测概率分布、预测目标及样本权重共 3 方面信息将一同输入给损失函数 J 用于模型优化.

(1) 模型形式化

这里重点阐述 W-NLM 的改进部分.形式化地,给定长度为 L 的代码序列 $S=w_1\cdots w_t\cdots w_L$,其中 $w_t\in\Omega$, $S\in\Omega^*$, Ω 为最小语言单元集合(即词汇表), Ω^* 为所有语言序列集合.代码序列 S 的质量类型记作 γ, $\gamma\in\{-1,0,1\}$,其中"-1"代表未知类型,"0"代表无缺陷类型,"1"代表有缺陷类型.则 W-NLM 模型使用的改进损失函数 $\tilde{J}$ 定义为(对于序列 S):

$$\begin{aligned}\tilde{J}(\boldsymbol{\theta})&=C\tilde{E}_{\tilde{M}(\boldsymbol{\theta})}(S;Q)\\&=\frac{1}{L}\sum_{t=1}^{L}C\tilde{E}\left(\boldsymbol{P}_t,\hat{\boldsymbol{P}}_{t;\tilde{M}(\boldsymbol{\theta})},q_t\right)\\&=\frac{1}{L}\sum_{t=1}^{L}\left(q_t\cdot\sum_{j=1}^{|\Omega|}\boldsymbol{P}_t^{(j)}\cdot\log_2\frac{1}{\hat{\boldsymbol{P}}_{t;\tilde{M}(\boldsymbol{\theta})}^{(j)}}\right)\\&=-\frac{1}{L}\sum_{t=1}^{L}\sum_{j=1}^{|\Omega|}\left[q_t\cdot P\left(w=w^{(j)}|w_1,\cdots,w_{t\text{-}1}\right)\cdot\log_2\hat{P}_{\tilde{M}}\left(w=w^{(j)}|w_1,\cdots,w_{t\text{-}1};\boldsymbol{\theta}\right)\right]\\&=-\frac{1}{L}\sum_{t=1}^{L}\left[q_t\cdot\log_2\hat{P}_{\tilde{M}}\left(w=w_t|w_1,\cdots,w_{t\text{-}1};\boldsymbol{\theta}\right)\right]\end{aligned}\tag{1}$$

其中, $C\tilde{E}$ 表示修正的交叉熵计算函数; $\tilde{M}(\boldsymbol{\theta})$ 表示参数化的 W-NLM 模型, $\boldsymbol{\theta}$ 为待估计参数; $\hat{\boldsymbol{P}}_{t;\tilde{M}(\boldsymbol{\theta})}$ 表示模型估计的序列下一词的概率分布,即模型的输出项 $\boldsymbol{y}$; $\boldsymbol{P}_t$ 表示序列下一词的真实概率分布,通常用 w_t 的 one-hot 编码 $\boldsymbol{w}_t$ 近似表示,为此上式推导的第四步被简化为最后形式; $w^{(j)}$ 表示词汇表 Ω 第 j 个词; Q 表示代码序列 S 对应的权重序列, $Q=q_1\cdots q_t\cdots q_L$,其中对于任意 $q_t=q^S\in\mathbb{R}^+$, q_t 表示每次交叉熵计算的权重, q^S 表示样本 S 的权重:

$$q^S=\begin{cases}q_{\text{unknown}}, & \gamma=-1\\ q_{\text{clean}}, & \gamma=0\\ q_{\text{buggy}}, & \gamma=1\end{cases}\tag{2}$$

式中, $q_{\text{clean}}\geq q_{\text{unknown}}\geq q_{\text{buggy}}\in\mathbb{R}^+$,表示不同质量类型代码序列的权重,其计算方法将在下文给出.

对比原始损失函数 J 可知,W-NLM 模型中引入的改进 $\tilde{J}$ 在每次评估 $\boldsymbol{P}_t$ 与 $\hat{\boldsymbol{P}}_{t;\tilde{M}(\boldsymbol{\theta})}$ 的交叉熵差异时进行了幅度为 q_t 的加权.因此从整体看,在利用梯度下降算法进行梯度反传时,有

$$\nabla_{\boldsymbol{\theta}}\tilde{J}(S;\boldsymbol{\theta})=q^S\cdot\nabla_{\boldsymbol{\theta}}J(S;\boldsymbol{\theta})\tag{3}$$

即对于序列 S 梯度幅度增加了 q^S 倍,进而参数的更新幅度也将增加 q^S 倍.这正是 W-NLM 的改进思路所在:模型重点关注高质量(即无缺陷)软件模块,同时降低对低质量(即有缺陷)软件模块的学习.类似情况也将出现于以小批量为单元的 SGD 算法.

更一般地,给定一个语料库(或任意序列集合) C:

$$C=\left\{\left(S_k,Q_k,\gamma_k\right)\right\}_{k=1}^{N}\tag{4}$$

其中,三元组 $\left(S_k,Q_k,\gamma_k\right)$ 描述了集合中第 k 个样本的序列表示 S_k、权重信息 Q_k 和质量类型 γ_k.则 W-NLM 模型的参数优化方法可描述为:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \tilde{J}(C;\boldsymbol{\theta}) \\ &= \arg\min_{\boldsymbol{\theta}} C\tilde{E}_{\hat{M}(\boldsymbol{\theta})}(C) \\ &= \arg\min_{\boldsymbol{\theta}} -\frac{1}{L_C}\sum_{k=1}^{N}\sum_{t=1}^{L_k}\left[q_{k,t}\cdot\log_2\hat{P}_M\left(w=w_{k,t}|w_{k,1},\cdots,w_{k,t-1};\boldsymbol{\theta}\right)\right]\end{aligned} \tag{5}$$

其中, N 表示序列总数, L_k 表示序列 S_k 的长度, L_C=$\sum_{k=1}^{N}L_k$ 表示语料库的词汇规模; $\boldsymbol{\theta}^*$ 表示模型参数 $\boldsymbol{\theta}$ 的(准)最优估计.

(2) 权重计算方法

由式(2)的定义可知,序列 S_k 的权重 q^{S_k} 依据样本质量类型 γ 取 $\{q_{\text{unknown}},q_{\text{clean}},q_{\text{buggy}}\}$ 中的某一值.为便于分析,本文以未知类型的代码样本为基准,令

$$q_{\text{unknown}}:q_{\text{clean}}:q_{\text{buggy}}=1:\rho:\frac{1}{\rho} \tag{6}$$

式中, $\rho\in[1,+\infty)$ 表示权重比例系数,例如 ρ =2 时表示未知类型样本的权重是有缺陷类型的 2 倍,是无缺陷类型的 1/2.因此 ρ 值越大,不同类型样本对模型优化(即梯度大小)的贡献差异就越明显(当 ρ =1 时,W-NLM 模型退化为原有 NLM 模型[48]).考虑到权值的引入会引起交叉熵计算结果的尺度发生变化,为此我们在训练语料库 $C=\{(S_k,Q_k,\gamma_k)\}_{k=1}^{N}$ 上进行如下正规化处理:

$$\frac{1}{L_C}\sum_{k=1}^{N}\sum_{t=1}^{L_k}q_{k,t}=\frac{1}{L_C}\sum_{k=1}^{N}\sum_{t=1}^{L_k}1=1 \tag{7}$$

即需满足

$$\sum_{k=1}^{N}L_k\cdot q^{S_k}=\sum_{k=1}^{N}L_k\cdot 1=L_C \tag{8}$$

这样的处理可以保证修正的 $C\tilde{E}_{\hat{M}}(C)$ 的尺度与原 $CE_M(C)$ 的尺度保持一致.那么,当给定语料库 C 和权重比例系数 ρ,我们便可以通过求解式(2)、式(4)、式(6)和式(7)联立的线性方程组求得 $q_{\text{unknown}},q_{\text{clean}},q_{\text{buggy}}$ 的具体取值,进而生成用于 W-NLM 模型计算的权重序列 $\{Q_k\}_{k=1}^{N}$.

2.3 代码交叉熵度量元

本文 CNDePor 方法阶段 II 的核心任务是利用已训练好的 NLM 来度量软件模块的 CE 值,生成阶段 III 所需的 CE 类度量元,其中该类度量元是对软件模块自然性的一种表征.形式化地,给定语言模型 M 及一个长度为 L 的语言序列 $S=w_1\cdots w_i\cdots w_L$,其中 $w_i\in\Omega$, $S\in\Omega^*$, Ω 为最小语言单元集合, Ω^* 为所有语言序列集合.则模型 M 对序列 S 的 CE 值[15-16]为:

$$\begin{aligned}CE_M(S)&=\frac{1}{L}\sum_{i=1}^{L}CE\left(\boldsymbol{P}_i,\hat{\boldsymbol{P}}_{i;M}\right)\\&=\frac{1}{L}\sum_{i=1}^{L}\left(\sum_{j=1}^{|\Omega|}\boldsymbol{P}_i^{(j)}\cdot\log_2\frac{1}{\hat{\boldsymbol{P}}_{i;M}^{(j)}}\right)\\&=-\frac{1}{L}\sum_{i=1}^{L}\sum_{j=1}^{|\Omega|}P\left(w=w^{(j)}|w_1,\cdots,w_{i-1}\right)\cdot\log_2\hat{P}_M\left(w=w^{(j)}|w_1,\cdots,w_{i-1}\right)\\&\approx-\frac{1}{L}\sum_{i=1}^{L}\log_2\hat{P}_M\left(w=w_i|w_1,\cdots,w_{i-1}\right)\end{aligned} \tag{9}$$

式中, $w^{(j)}$ 表示词汇表 Ω 中的第 j 个词; $\hat{\boldsymbol{P}}_{i;M}$ 表示模型 M 对序列 $w_1,\cdots,w_{i-1}$ 下一词概率分布的估计; $\boldsymbol{P}_i$ 表示序列下一词的真实概率分布.由于 $\boldsymbol{P}_i$ 一般不可知,所以通常采用序列实际下一词 w_i 的 one-hot 编码 $\boldsymbol{w}_i$ 近似表示,为此上式推导的第三步被简化为最后形式.最终,求交叉熵 $CE_M(S)$ 即等同于求模型 M 对序列 S 估计的负对数似然在序列词汇规模意义上的均值.

(1) 度量元的定义

定义 1:代码的交叉熵度量元(**CE**)是一种代码度量元,其值由语言模型度量软件模块的代码 token 序列生成.给定一个模块 k 和一个语言模型 M ,则模块的 CE 值由式(9)计算,即 $CE_M\left(S_k\right)$,其中 S_k 表示模块 k 的代码 token 序列, M 已在代码 token 序列的语料库上训练.

定义 2:代码逆序序列的交叉熵度量元(**CE-Inv**)是一种代码度量元,其值由语言模型度量软件模块的代码逆序序列生成.给定一个模块 k 和一个语言模型 $\overleftarrow{M}$,则模块的 CE-Inv 值由式(9)计算,即 $CE_{\overleftarrow{M}}\left(\overleftarrow{S}_k\right)$,其中 $\overleftarrow{S}_k$ 表示模块 k 的代码逆序序列, $\overleftarrow{M}$ 已在代码逆序 token 序列的语料库上训练.

定义 3:修正的代码交叉熵度量元(**M-CE**)是一种代码度量元,其值由加权语言模型度量软件模块的代码序列生成.给定一个模块 k 和一个加权语言模型 $\tilde{M}$,则模块的 M-CE 值由式(9)计算,即 $CE_{\tilde{M}}\left(S_k\right)$,这里 S_k 表示模块 k 的代码序列,且 $\tilde{M}$ 已在代码 token 序列的语料库上进行了加权训练,训练方法由式(5)定义.

定义 4:修正的代码逆序序列交叉熵度量元(**M-CE-Inv**)是一种代码度量元,其值由加权语言模型度量软件模块的代码逆序序列生成.给定一个模块 k 和一个加权语言模型 $\overleftarrow{\tilde{M}}$,则模块的 M-CE-Inv 值由式(9)计算,即 $CE_{\overleftarrow{\tilde{M}}}\left(\overleftarrow{\tilde{S}}_k\right)$,这里 $\overleftarrow{\tilde{S}}_k$ 表示模块 k 的代码逆序序列,且 $\overleftarrow{\tilde{M}}$ 已在代码逆序token序列的语料库上进行了加权训练,训练方法由式(5)定义.

(2) 度量元的生成方法

以 M-CE 度量元为例,阐述 CE 类度量元的生成方法.

步骤1: 从软件仓库中提取待测软件模块对应的源代码区域.

步骤2: 对源代码集进行数据预处理工作[48,50],如移除注释、代码词化等.

步骤3: 加载已训练的语言模型参数.在已标注质量类型的代码样本集上,按照 2.2 节方法训练加权语言模型 $\tilde{M}$,之后将加载了(准)最优参数的模型记作 $\tilde{M}^*$.

步骤4: 利用已训练的语言模型计算待测软件模块的 M-CE 值.按照 CE 计算公式(9),利用已训练的 $\tilde{M}^*$ 对软件模块的代码序列进行熵值计算.

3 切片粒度模块生成方法及度量元设计

事实上,本文提出的 CNDePor 方法适用于不同粒度缺陷预测问题.然而经典的粗粒度方法难以精确预测缺陷区域,例如文件级、类级,使得实际应用中软件开发人员需要付出高昂的代码审查/测试成本.为此我们将利用 CNDePor 方法在面向语句的切片级粒度上进行细粒度缺陷预测应用.下面主要介绍切片粒度模块的生成方法和面向此类模块的度量元设计.

3.1 模块生成方法

切片粒度软件模块的生成过程如图 4 所示.这一粒度分析的软件模块实质上是由一个或多个面向语句的切片构成的代码序列.针对特定缺陷类型设计相关的切片准则,再通过软件历史信息标注生成的切片模块,便可以构建出缺陷数据集.具体步骤如下[30]:

步骤1: 抽取库/API 函数调用.将库/API 函数调用分为两类:前向型和后向型.如果库/API 函数调用直接从外部接收一个或多个输入,例如命令行、外部程序等,则它属于前向型.如果库/API 函数调用不从程序运行环境中接收任何外部输入,则它属于后向型.

步骤2: 生成关于库/API 函数调用中参数的切片.针对前向库/API 函数调用中的每个参数,生成一个或多个前向切片;针对后向库/API 函数调用中的每个参数,生成一个或多个后向切片.需要注意,程序切片中的语句可以从属多个函数,即切片范围可以逾越切片准则所在的函数.

步骤3: 融合程序切片构成软件模块.针对已生成的切片集合,将从属于相同自定义函数的语句融合为一个代码段,次序按语句在自定义函数中的原有出现顺序.如果切片间存在重复语句,则删除.然后,将从属于不同自定义函数的代码段融合为一个整体,构成一个软件模块.如果两个代码段对应的函数在某个切

片中已有明确顺序,则顺序保留;否则代码段的顺序随机设置.

步骤4: 标注软件模块的质量类型.依据缺陷数据库信息及人工审核方法,为每个软件模块分配一个质量标签:“1”代表有缺陷;“0”代表无缺陷.需要注意的是,按照贡献者论文的叙述,实验数据集存在标签噪声,即相同的软件模块可能拥有不同的质量标签.对于这种冲突情况,本文将其列为未知类型.

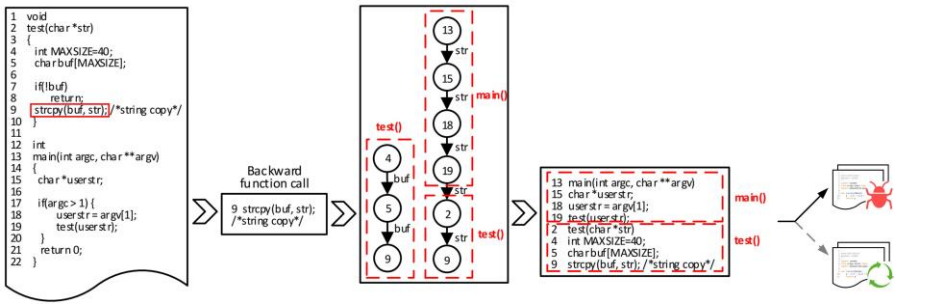


Fig.4 Generation process of slice-level software modules

图 4 切片粒度模块生成过程[30]

3.2 度量元设计

根据调研情况看,目前尚缺乏相关工作研究面向语句的切片粒度缺陷预测.为此,本文尝试设计了 4 种度量元用于验证实验:修正的代码 token 序列交叉熵值 M-CE、修正的代码 token 逆序序列交叉熵值 M-CE-Inv、代码行(lines of code,简称 LOC)和代码的 token 规模(token size,简称 TSize),如表 2 所示.其中,度量元 M-CE 和 M-CE-Inv 是本文 CNDePor 方法首次引入的;LOC 是一种经典的规模度量元,被较早地用于缺陷预测,这里 LOC 指切片粒度软件模块的 LOC;TSize 同样是一种规模度量元,它反映了软件模块词化后的 token 数量.

Table 2 The designed metrics of slice-level software modules

表 2 设计的切片粒度软件模块度量元

度量元		度量元类型	度量元描述
M-CE	Modified CE	代码自然性	修正的代码交叉熵值(token 序列)
M-CE-Inv	Modified CE of Inverse Sequence	代码自然性	修正的代码逆序交叉熵值(token 序列)
LOC	Lines of Code	代码规模	模块的代码行
TSize	Token Size	代码规模	模块的代码 token 规模

4 实验设计

4.1 研究问题

为验证本文方法在切片粒度软件缺陷预测问题上的有效性,我们设计了以下 3 个研究问题(research question,简称 RQ):

RQ1: 在切片粒度缺陷数据集上,权重比例系数 ρ 如何影响 CE 类度量元?

RQ2: 在切片粒度缺陷数据集上,CE 类度量元是否具有判别力?

RQ3: 在相同切片粒度缺陷数据集上,本文方法与其他方法对比如何(性能与可解释性)?

4.2 数据集

本文使用的数据集[30]关注两类常见安全缺陷:缓冲区错误(buffer error,简称 BE)和资源管理错误(resource management error,简称 RME),它们均可由库/API 函数调用的不确当使用触发.因此,针对一些缺陷敏感的函数调

用类型,设计与之相关的切片准则,我们便可以收集到实验所需的数据集.

该数据集的原始样本源自美国国家标准与技术协会(national institute of standards and technology,简称 NIST)的两个漏洞数据源:美国国家漏洞数据库(national vulnerability database,简称 NVD)和软件保证参考数据集(software assurance reference dataset,简称 SARD).目前这两个数据源已被广泛应用于安全缺陷领域的研究工作.在 NVD 中,每个漏洞样本包含一个通用漏洞披露标识(common vulnerabilities and exposures identifier,简称 CVE ID)和一个通用缺陷列表标识(common weakness enumeration identifier,简称 CWE ID),它们指示了漏洞的相关类型.在 SARD 中,每个程序样本对应一个或多个 CWE ID,这是由于实例可能包含多种漏洞类型.

Table 3 Information of the slice-level defect datasets

表 3 切片粒度缺陷数据集信息

数据集名称	模块数	缺陷模块数	缺陷率	平均代码行
BE-ALL	39,753	10,440	26.26%	8.54
RME-ALL	21,885	7,285	33.29%	10.29
总计/平均	61,638	17,725	28.76%	9.16

数据集的基本信息如表 3 所示,其中软件模块总数为 61,638,无缺陷模块数为 43,913,缺陷模块数为 17,725,平均缺陷率为 28.76%,模块的平均代码行为 9.16.这些样本源自 NVD 库中 19 个知名 C/C++开源项目(如 Linux kernel、Firefox、Thunderbird 等)的 840 个源程序和 SARD 库中 9851 个 C/C++源程序.针对这些源程序和 6,045 种 C/C++库/API 函数调用,实验共抽取了 56,902 个函数调用实例[30],其中包括 7,255 个前向函数调用和 49,647 个后向函数调用.其中,数据集中共有 10,440 个模块对应 BE 缺陷,7,285 个模块对应 RME 缺陷.进一步观察,数据集中软件模块的平均代码行仅为 9.16,远小于常用的文件粒度模块的代码规模[34,41,44,52-53].换言之,当缺陷预测模型将某个新模块判别为“有缺陷类型”时,软件开发人员只需要重点审查大约 10 行代码(平均意义上)即可,这样极大地节约了成本、提高了效率.

4.3 评价指标

为充分验证本文方法的有效性,实验将考察 CE 类度量元的判别力和缺陷预测模型的分类性能.考虑到不同评价机制存在不同“偏好”,实验将选取多种评价指标来减少偏差.

(1) 度量元判别力评价指标

判别力指标旨在评价度量元在区分样本类别方面的能力,本质上也是在解决特征与类标间的相关程度度量问题.一般情况下,度量元的判别力越强,模型就越有可能得到好的预测结果.我们选取了三种常用的过滤式特征选择方法[54],用来衡量不同度量元的判别力:Pearson 相关性、Fisher 准则和信息增益,这些方法均是效益型度量指标,目前已在软件缺陷预测中得到了广泛应用[34,55-56].

Pearson 相关性是一种简单且常用的相关性度量指标,它反映了两个变量间的线性相关程度:

$$PC(X,Y)=\left|\mathrm{cov}(X,Y)\right|/\sqrt{\mathrm{var}(X)\cdot\mathrm{var}(Y)}$$

其中 $PC\in[0,1]$, X 和 Y 表示两个随机变量, $\mathrm{cov}(X,Y)$ 和 $\mathrm{var}(X)$ 分别表示协方差函数和方差函数.

Fisher 准则可由经典的线性判别分析诱导而来,其旨在衡量不同类别样本间的易区分度:

$$FC(X_+,X_-)=\frac{S_b}{S_w}=\frac{\left(\bar{X}_+-\bar{X}_-\right)^2}{S_+^2+S_-^2}$$

其中 $FC\in[0,+\infty)$,'+'和'-'表示两种类别, $\bar{X}$ 表示样本特征的均值; S_b 和 S_w 分别表示类间散度和类内散度; $S_{+/-}^2=\sum_{x_i\in X_{+/-}}\left(x_i-\bar{X}_{+/-}\right)^2$ 表示'+'类/'-'类的类散度.

信息增益是一种基于熵的度量指标,其度量观测特征 x 后类标 y 减少的不确定性,即增加的信息量:

$$IG(X,Y)=H(Y)-H(Y\mid X)$$

其中 $IG\in[0,+\infty)$,这里 X 和 Y 表示的是特征 x 和类标 y 的取值集合, $H(Y)$ 和 $H(Y|X)$ 表示香农熵和条件熵.

(2) 缺陷预测性能评价指标

本文关注的软件缺陷倾向性预测问题在本质上可视为一个典型的二分类问题,即将软件模块分为有缺陷

类或无缺陷类.在二分类问题中,一个待测样本可被预测为四种情况,即真阳性(true positive,简称 TP)、假阳性(false positive,简称 FP)、真阴性(true negative,简称 TN)或假阴性(false negative,简称 FN).为可观评价预测模型性能,选取了 5 种使用最为广泛的评价指标[36,57]:假阳性率(false positive rate,简称 FPR)、假阴性率(false negative rate,简称 FNR),查准率(precision),查全率(recall)和 F1 度量(F1-measure).

- FPR 又称作误报率: $FPR = FP / (FP + TN)$.
- FNR 又称作漏报率: $FNR = FN / (TP + FN)$.
- 查准率度量分类器正确输出的真阳性(占总输出的)比率: $Precision = TP / (TP + FP)$.
- 查全率度量分类器实际输出的真阳性(占总真阳性的)比率: $Recall = TP / (TP + FN)$.
- F1 度量以调和平均数的方式平衡查准率和查全率: $F1 = 2 \times Precision \times Recall / (Precision + Recall)$.

上述 5 个评价指标的取值范围均落于[0,1]之间,其中 Precision、Recall 和 F1 属于效益型度量指标,即数值越大表示分类器的预测性能越好;相反 FPR 和 FNR 属成本型效益指标,即数值越小越好.对于一个理想的缺陷倾向性分类器而言,其既不会漏报缺陷(即 $FNR \approx 0$, $Recall \approx 1$)也不会误报缺陷(即 $FPR \approx 0$, $Precision \approx 1$),此时综合指标 $F1 \approx 1$.但在实际中,分类器很难做到这样的理想情况,经常“顾此失彼”,因此需要考察模型的多个方面.

4.4 实验环境与参数设置

实验环境为 Intel i7-6700K CPU、24G RAM、1T ROM 及 1 块 GeForce GTX 1080ti 11G 显卡.使用的深度学习建模工具为 TensorFlow 1.3.0,缺陷预测分类器的建模工具为 Scikit-learn 0.19.1,编程语言为 Python 3.6.6.使用的 C/C++代码词化工具为 Python 第三方库 Clang 7.0.0,其依赖工具 LLVM 7.0.0 和 Microsoft Visual Studio 14.0 的支持.

本文经验性参数的设置采用经典的格点搜索方法,通过五折交叉验证实验求均值后对比得到,其中下一节将以重要参数 ρ 为例,详细展示模型参数的选取过程.具体地,CNDePor 方法中语言模型的超参数设置如下:词频阈值 τ =5(BE 数据集的词汇表大小 V_{BE} =4496,RME 数据集的词汇表大小 V_{RME} =3059);词向量维度 dim_{w} =800;LSTM 层数 N_{layer} =1;LSTM 层的隐含节点 dim_{h} =800;Dropout 方法的节点保留概率 P_{keep} 均为 0.5;按时间展开步长 N_{step} =50;批处理的大小 N_{batch} =30;训练轮数 N_{epoch} =10,且每轮均对样本序列进行随机排序;允许的最大梯度范数 $grad_{\max}$ =5;LSTM 单元遗忘门偏置的初始值为 **0**,网络其他参数的初始值按[-0.08,0.08]的均匀分布随机设置;模型优化使用 SGD 算法,自适应学习率 ℓ_{rate} 参数 λ =7、η =0.5;加权语言模型的权重比例系数选取范围为 $\rho \in [1,20]$.CNDePor 方法的缺陷预测阶段,我们选取了广泛应用的 SVM 和 RF 分类器[3,34,52-53]进行对比实验,其中 SVM 的参数设置为:惩罚系数 C_{penalty} =5,少数类的权重为 2,核函数为径向基核,径向基尺度参数 $\gamma = 1/(2\sigma^2) = 10^5$,其余参数为默认配置;RF 的参数设置为:决策树数量 N_{DT} =15,特征的划分准则基于信息增益,其余参数为默认配置.在利用分类器进行缺陷预测时,我们对度量元数据进行了标准化处理,采用的方法是 Min-Max 标准化,即将各维度属性线性缩放至[0,1]之间.

5 实验过程与结果分析

实验针对表 3 所示的切片粒度数据集,利用五折交叉验证方法验证本文 CNDePor 在切片粒度缺陷预测方面的应用性能.该方法的阶段 I(语料库学习阶段)是利用 W-NLM 学习代码样本.表 4 列出了在两种缺陷数据集的五折交叉验证实验中语言模型获得的性能指标,其中包括平均困惑度(perplexity,简称 PP)值和耗时情况.

在表 4 中,ρ 表示权重比例系数,其代表了语言模型对有/无缺陷代码学习的抑制/增强程度.当 ρ =1 时,不同质量类型的样本权重相同,本质上 W-NLM 已退化为非加权的语言模型.从整体上看,模型在测试集上 PP 值与训练集的结果接近,说明模型的泛化性能良好;模型在 RME-ALL 数据集上的指标值比 BE-ALL 更低,说明前者数据较后者更易学习;权重比例系数 ρ 的增大会影响测试集 PP 值,这是由于模型降低了对有缺陷代码的学习权重;逆向语言模型的 PP 值与正向语言模型得分接近,说明了软件代码亦可进行逆序学习.

Table 4 5-fold cross validation results of language model

表 4 语言模型的五折交叉验证结果

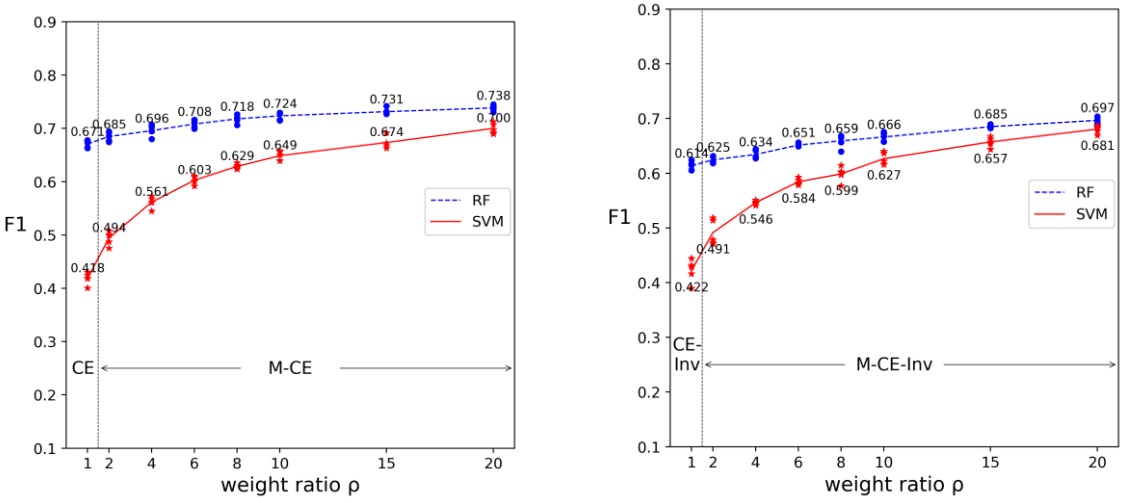
语言模型		BE-ALL 数据集			RME-ALL 数据集		
		平均困惑度		平均耗时/s	平均困惑度		平均耗时/s
		训练集	测试集		训练集	测试集	
W-NLM(ρ=1)	正向	1.556	1.523	800.5	1.398	1.409	466.8
	逆向	1.554	1.523	785.3	1.401	1.412	459.5
W-NLM(ρ=10)	正向	1.550	1.692	792.9	1.420	1.570	460.6
	逆向	1.547	1.691	783.7	1.422	1.571	459.5
W-NLM(ρ=20)	正向	1.550	1.727	792.8	1.421	1.630	460.8
	逆向	1.547	1.727	786.0	1.421	1.637	460.9

本文 CNDePor 方法的阶段 II(交叉熵度量阶段)是基于训练过的语言模型进行软件模块的熵值度量.最后在第 III 阶段的缺陷预测应用中,将 CE 类度量元与传统度量元相融合,一同训练分类器并实现对软件模块有/无缺陷倾向性的判别.下面依据上一节提出的研究问题,先后从三个方面进行实验与结果分析.

5.1 针对RQ1的结果分析

为提高软件缺陷预测性能,本文 CNDePor 方法使用了新颖的双向语言模型 W-NLM 对代码进行度量.其中,W-NLM 可以有效利用软件模块的质量类型信息(即有/无缺陷的标签信息)对样本进行加权,进而提高学习的针对性.经加权学习后,语言模型将更倾向于赋予有/无缺陷模块更高/低的 CE 值,从而得到的代码 CE 类度量元可更有效地完成缺陷预测.在 W-NLM 中,权重比例系数 ρ 是一个十分关键的参数,其代表了语言模型对有/无缺陷代码学习的抑制/增强程度.当 ρ=1 时,本质上 W-NLM 已退化为 NLM 模型,同时模型生成的 M-CE 和 M-CE-Inv 度量元退化为 CE 和 CE-Inv 度量元.

经五折交叉验证实验,图 5 和图 6 分别绘制了 CE 类度量元在 BE-ALL 数据集和 RME-ALL 数据集上的缺陷预测性能(F1 指标)随权重比例系数 ρ 的变化情况,其中纵坐标表示 F1 指标,横坐标表示 ρ 的取值;缺陷预测分类器包括 RF(蓝虚线)和 SVM(红实线)两种;缺陷预测过程仅使用单一度量元.



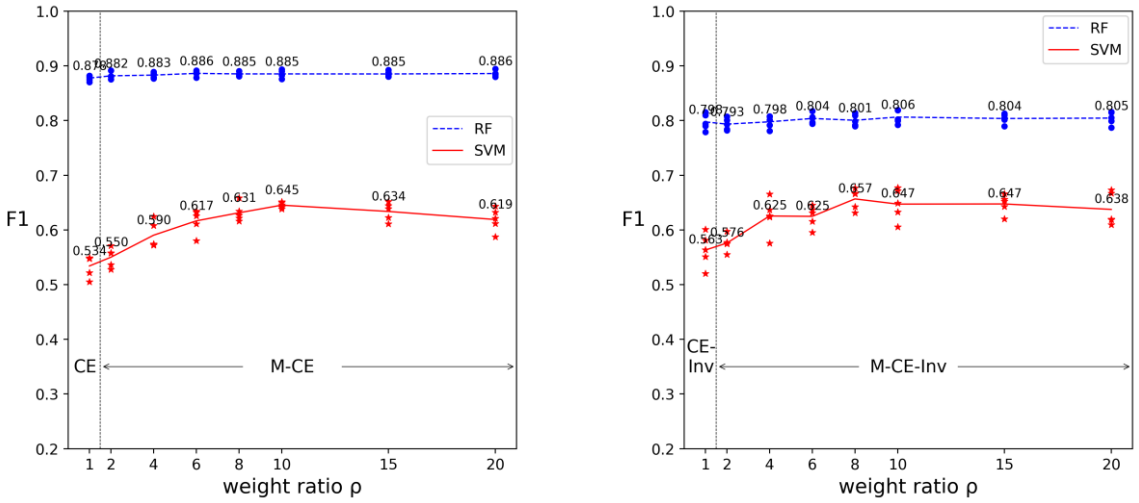
(a) M-CE 度量元 (b) M-CE-Inv 度量元

Fig.5 Defect prediction performance (F1-measure) changes of CE-type metrics with weight ratio factor ρ in BE-ALL dataset

图 5 CE 类度量元的缺陷预测性能(F1 指标)随权重比例系数 ρ 的变化情况(BE-ALL 数据集)

图 5 的(a)图和(b)图分别绘制了 M-CE 和 M-CE-Inv 度量元的缺陷预测性能(BE-ALL 缺陷数据集),其中当

ρ=1 时,它们退化为 CE 和 CE-Inv 度量元.从图中可以看出,基于 CE 类度量元的 F1 性能随 ρ 的增大而上升,其中 $1\le\rho\le10$ 时上升趋势明显,当 $\rho>10$ 后,性能改善趋于平缓.这一情况无论对于 RF 分类器还是 SVM 分类器来说均是如此,但整体上 RF 的预测性能更优.最终当 ρ=20 时(以 RF 分类器为例),基于 M-CE 和 M-CE-Inv 度量元的缺陷预测 F1 值可分别达到 73.8%和 69.7%,较原始 CE 和 CE-Inv 度量元的 F1 值 67.1%和 61.4%绝对增长 6.7%和 8.3%,说明了通过调节权重比例系数 ρ,可以改变语言模型对有/无缺陷软件模块学习的抑制/增强程度,进而可以有效提升 CE 类度量元的缺陷预测性能.



(a) M-CE 度量元 (b) M-CE-Inv 度量元

Fig.6 Defect prediction performance (F1-measure) changes of CE-type metrics with weight ratio factor ρ in RME-ALL dataset

图 6 CE 类度量元的缺陷预测性能(F1 指标)随权重比例系数 ρ 的变化情况(RME-ALL 数据集)

图 6 的(a)图和(b)图分别绘制了 M-CE 和 M-CE-Inv 度量元的缺陷预测性能(RME-ALL 缺陷数据集),其中当 ρ=1 时,它们退化为 CE 和 CE-Inv 度量元.从图中可以看出,整体上基于 CE 类度量元的 F1 性能随 ρ 的增大而上升($1\le\rho\le10$),当 $\rho>10$ 后,性能改善趋于平缓甚至停滞或下降.这一情况对于 SVM 分类器来说现象明显,但对于 RF 分类器,其预测性能变化不大.上述表现与图 5 描绘的 BE-ALL 数据集情况有所差别,反映出 RME-ALL 数据本身更易预测和学习;RF 分类器在 RME-ALL 数据集上已可以获得较好的性能;权重比例系数 ρ 过大会影响语言模型的学习效果.因此,最终取 ρ=10(以 SVM 为例),基于 M-CE 和 M-CE-Inv 度量元的缺陷预测 F1 值可分别达到 64.5%和 64.7%,较原始 CE 和 CE-Inv 度量元的 F1 值 53.4%和 56.3%绝对增长 11.1%和 8.4%.

结论 1:在切片粒度缺陷数据集上,基于 CE 类度量元的缺陷预测性能随权重比例系数 ρ 的增大而上升,其中 $\rho\in[1,10]$;当 $\rho>10$ 后,性能提升趋于平缓甚至停滞或下降,这种情况因数据集和分类器差异而不同.基于实验结果,对于 BE 缺陷数据集,本文在缺陷预测应用中取 ρ=20;对于 RME 缺陷数据集,取 ρ=10.

结论 2:权重比例系数 ρ 代表了语言模型对有/无缺陷代码学习的抑制/增强程度,通过 ρ 的调节可影响 CE 类度量元的缺陷预测性能,进一步说明了 CE 类度量元具有可学习性,它们蕴涵了语言模型从代码语料库中学习到的(关于代码有/无缺陷的)知识,进而可以用于判别软件缺陷模块.

5.2 针对RQ2的结果分析

判别力分析旨在衡量度量元在区分样本类别方面的能力,也即在衡量度量元与类标间的相关程度.一般情况下,度量元的判别力越强,缺陷预测模型就越有可能得到好的预测结果.为此,RQ2 考察 CE 类度量元在切片粒度缺陷数据集上的判别力表现.针对这一问题,表 5 展示了 4 种度量元(包括 M-CE、M-CE-Inv、LOC 和 TSize)

在五折交叉验证实验测试集上的平均判别力分数,其中当权重比例系数 ρ=1 时 M-CE 和 M-CE-Inv 度量元退化为 CE 和 CE-Inv 度量元;判别力指标包括 Pearson 相关性、Fisher 准则和信息增益,它们均是效益型指标.

Table 5 Metrics' discriminative performance

表 5 度量元的判别力对比

数据集	度量元	Pearson 相关性	Fisher 准则(×10^{-5})	信息增益(×10^{-1})
BE-ALL	TSize	0.048	0.185	0.347
	LOC	0.131	1.283	0.485
	CE (ρ=1)	0.017	0.022	0.209
	CE-Inv (ρ=1)	0.007	0.004	0.229
	M-CE (ρ=10)	0.284	6.142	1.369
	M-CE-Inv (ρ=10)	0.250	4.662	1.154
	M-CE (ρ=20)	**0.325**	**8.252**	**1.783**
	M-CE-Inv (ρ=20)	0.299	6.860	1.529
RME-ALL	TSize	0.037	0.183	0.326
	LOC	0.102	1.153	0.393
	CE (ρ=1)	0.110	1.305	0.189
	CE-Inv (ρ=1)	0.119	1.520	0.321
	M-CE (ρ=10)	0.173	3.273	0.910
	M-CE-Inv (ρ=10)	0.156	2.666	1.001
	M-CE (ρ=20)	**0.212**	**4.978**	0.769
	M-CE-Inv (ρ=20)	0.207	4.734	**1.077**

由表 5 可以看出,规模度量元 LOC 的判别力要优于 TSize,代码自然性特征 CE 和 CE-Inv 在 BE-ALL 数据集上表现弱于规模度量元,但在 RME-ALL 数据集上判别力与 LOC 接近.因此整体上,原始自然性特征在切片粒度缺陷数据集上的判别力表现一般.但可以看到,当引入改进机制,利用双向 W-NLM 度量软件模块后,得到的改进度量元 M-CE 和 M-CE-Inv 具有优越的判别力表现,明显优于原始自然性度量元和传统规模度量元.以 BE-ALL 数据集上的 Pearson 相关性为例,CE(ρ =1)度量元、M-CE(ρ =10)度量元和 M-CE(ρ =20)度量元的得分分别为 0.017、0.284 和 0.325,改进幅度超过 26.7%;CE-Inv(ρ =1)度量元、M-CE-Inv(ρ =10)度量元和 M-CE-Inv(ρ =20)度量元的得分分别为 0.007、0.250 和 0.299,改进幅度超过 24.3%.可以看出,随着权重比例系数 ρ 的增大,度量元判别力提升明显;同时整体上看 CE/M-CE 度量元的判别力强于 CE-Inv/M-CE-Inv 度量元.

结论 3: 在切片粒度缺陷数据集上,CE 类度量元具有一定判别力,其中 CE 和 M-CE 度量元的判别力在整体上强于 CE-Inv 和 M-CE-Inv 度量元;改进的度量元 M-CE 和 M-CE-Inv 的判别力明显优于原始度量元 CE 和 CE-Inv 以及传统规模度量元 TSize 和 LOC.

5.3 针对RQ3的结果分析

为进一步说明本文方法的有效性,RQ3 关注与其他方法的比较.表 6 展示了在相同缺陷数据集上,CNDePor 方法与其他方法的性能对比情况,其中本文方法使用了表 2 设计的 4 种度量元;性能得分为五折交叉验证实验结果的均值;依据 RQ1 的结论,对于 BE 缺陷类型,权重比例系数 ρ 取 20,对于 RME 缺陷类型 ρ 取 10;依据 RQ1 实验表现,使用的分类器为 RF.

为尽可能确保评价的客观性,选取了 5 种性能度量指标,包括 FPR、FNR、Precision、Recall 和 F1 指标;选取了 3 类基准方法,包括基于传统度量元的缺陷预测方法、基于深度学习的缺陷检测方法 VulDeePecker[30]、基于代码自然性的缺陷预测方法 DefectLearner[48].其中,第一类方法使用的度量元为 LOC 和 TSize,使用的分类器为 SVM 和 RF,模型参数同本文方法;DefectLearner 使用的度量元为 CE、LOC 和 TSize,使用的分类器为 RF,其他参数同本文方法.

Table 6 Performance comparisons between our CNDePor and other methods

表 6 CNDePor 方法与其他方法的性能对比

数据集	方法	评价指标(%)				
		FPR	FNR	Precision	Recall	F1
BE-ALL	SVM	34.5	27.1	43.4	72.9	54.4
	RF	10.8	69.3	50.8	30.7	38.3
	VulDeePecker[30]	**2.9**	18.0	**91.7**	82.0	**86.6**
	DefectLearner[48]	10.2	36.5	69.3	63.5	66.3
	CNDePor	7.4	**11.2**	81.3	**88.8**	84.9
RME-ALL	SVM	17.3	14.6	71.9	85.4	78.1
	RF	9.6	20.6	81.1	79.4	80.2
	VulDeePecker[30]	**2.8**	**4.7**	**94.6**	**95.3**	**95.0**
	DefectLearner[48]	4.6	15.1	90.5	84.9	87.6
	CNDePor	6.2	8.7	88.5	91.3	89.9

由表 6 实验结果可以看出,在两类缺陷数据集上数据集上,本文 CNDePor 得到的查准率分别为 81.3%和 88.5%,查全率分别为 88.8%和 91.3%,F1 指标分别为 84.9%和 89.9%,说明本文方法可以有效识别两类缺陷.

在对比基于传统度量元的缺陷预测方法方面,本文选取了 2 种广泛使用的分类器:SVM 和 RF.与 CNDePor 不同的是,它们仅基于传统度量元进行预测,没有使用代码自然性特征.相较而言,本文 CNDePor 方法性能优势明显,其中 F1 指标在 BE-ALL 和 RME-ALL 数据集上分别提升 30.5%~46.6%和 9.7%~11.8%.

在对比基于深度学习的方法方面,本文选取了实验数据集贡献者的方法 VulDeePecker.该方法基于 Bi-LSTM 网络分析切片后的代码 token 序列,然后利用神经网络自动生成的语义特征进行分类器学习和对新模块的缺陷检测,因此该方法具有专家经验依赖少的优点,但同时也造成了模型和语义特征的可解释差等问题.与 VulDeePecker 相比,本文 CNDePor 在两类缺陷数据集上得到了相近结果,其中在 BE-ALL 上本文方法的漏报率、查全率较对比方法均改善 6.8%.同时,CNDePor 以语言模型和 CE 类度量元为基础,因此相比基于端到端深度学习的“黑箱”检测方法来说,具有更好的可解释性.

在对比基于代码自然性的方法方面,本文选取了我们已发表的工作 DefectLearner.该方法将 NLM 度量的代码 CE 度量元引入到缺陷预测中,以增强分类器的预测性能.但是 NLM 仅单向学习语料库,致使模型对软件模块自然性的度量没有本文方法全面.其次就缺陷预测任务而言,NLM 的训练是无监督的,它们未利用模块的缺陷信息,致使生成的 CE 度量元的缺陷判别力有限.从数据对比来看,本文 CNDePor 方法性能整体上优势明显,其中在 BE-ALL 数据集上查准率、查全率和 F1 指标分别提升 12.0%、25.3%和 18.6%.

结论 4: 在相同数据集上,本文 CNDePor 方法较对比的传统缺陷预测方法和基于代码自然性的方法 DefectLearner 有显著优势;较先进的基于深度学习的方法 VulDeePecker 具有可比性性能,但本文方法可解释性更强.

6 总结与展望

针对现有语言模型仅对代码进行单向度量和未能利用样本缺陷信息的问题,提出了一种基于代码自然性特征的缺陷预测方法(CNDePor).该方法利用软件质量类型信息对样本加权,增强/抑制了语言模型对无/有缺陷代码的学习强度,提升了 CE 类度量元的缺陷判别能力;另一方面还实现了对输入序列的正逆双向学习,得到的两种改进度量元 M-CE 和 M-CE-Inv 更全面地刻画了代码的自然性.

针对粗粒度缺陷预测难以辅助代码审查及缺陷查找的不足,研究了一种新的细粒度缺陷预测问题——面向语句的切片级缺陷预测,并在此问题上实现了度量元设计和缺陷预测方法应用.具体地,针对切片粒度缺陷预测设计了 4 种度量元,其中包括 2 种代码自然性度量元和 2 种规模度量元,并在两类安全缺陷数据集上验证了度量元和 CNDePor 方法的有效性.实验结果表明:CE 类度量元具有可学习性,它们蕴涵了语言模型从代码语料库中学习到的相关知识;改进的度量元 M-CE 和 M-CE-Inv 的判别力明显优于原始度量元和传统规模度量元;本文 CNDePor 方法较传统缺陷预测方法和已有的基于代码自然性的方法有显著优势,较先进的基于深度学习

的方法具有可比性性能,但本文方法可解释性更强.

近年来,随着软件技术的快速发展,可以获取的代码等软件资源日渐丰富.在这些数据的支持下,研究基于语言模型的代码自然性分析及缺陷预测技术具有重要价值和广阔前景.本文围绕这一话题提出了一些解决方案,但工作还有许多值得进一步完善和探索的地方.首先,研究基于代码自然性特征的跨项目缺陷预测.考虑到实际中存在样本标记难、历史数据少的挑战,研究跨项目缺陷预测具有重要价值[57-59].其次,研究切片粒度软件模块的度量元设计.本文对面向语句的切片粒度缺陷预测问题进行了初步探讨并设计了 4 种度量元.设计更多的、不同类型的度量元并验证它们的有效性是下一步工作的重点.最后,研究代码自然性特征在不同粒度缺陷预测问题上的应用.我们已在文件粒度和切片粒度上研究了代码自然性的缺陷预测应用.关于其他预测粒度上,特别是细粒度方面(如函数粒度、变更粒度),此类度量元的有效性及应用方法值得进一步研究.

References:

[1] 马晓星, 刘譞哲, 谢冰, 等. 软件开发方法发展回顾与展望[J]. 软件学报, 2019, 30(1): 3-21.

[2] 陈翔, 顾庆, 刘望舒, 等. 静态软件缺陷预测方法研究[J]. 软件学报, 2016, 27(1): 1-25.

[3] Rathore S S, Kumar S. A study on software fault prediction techniques[J]. Artificial Intelligence Review, 2017: 1-73.

[4] Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction[J]. IEEE Transactions on Software Engineering, 2019, 45(2): 111-147.

[5] 蔡亮, 范元瑞, 鄢萌, 等. 即时软件缺陷预测研究进展[J]. 软件学报, 2019, 30(5): 1288-1307.

[6] 宫丽娜,姜淑娟,姜丽. 软件缺陷预测技术研究进展[J]. 软件学报, 2019, 30(10): 3090-3114.

[7] Radjenović D, Heričko M, Torkar R, et al. Software fault prediction metrics: A systematic literature review[J]. Information and Software Technology, 2013, 55(8): 1397-1418.

[8] Dam H K, Tran T, Ghose A. Explainable software analytics[C]. //Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results. New York: ACM Press, 2018: 53-56.

[9] Du M, Liu N, Hu X. Techniques for Interpretable Machine Learning[J]. Communications of the ACM, 2020, 63(1): 68-77.

[10] Wan Z, Xia X, Hassan A E, et al. Perceptions, expectations, and challenges in defect prediction[J]. IEEE Transactions on Software Engineering, 2018. DOI: 10.1109/TSE.2018. 2877678.

[11] Kamei Y, Shihab E, Adams B, et al. A large-scale empirical study of just-in-time quality assurance[J]. IEEE Transactions on Software Engineering, 2013, 39(6): 757-773.

[12] Miltiadis A, Barr E T, Premkumar D, et al. A survey of machine learning for big code and naturalness[J]. ACM Computing Surveys, 2018, 51(4):1-37.

[13] Manning C D. Foundations of Statistical Natural Language Processing[M]. Massachusetts: MIT Press, 1999.

[14] Ray B, Hellendoorn V, Godhane S, et al. On the naturalness of buggy code[C]. //Proceedings of the 38th International Conference on Software Engineering. New York: ACM Press, 2016: 428-439.

[15] Hindle A, Barr E T, Su Z, et al. On the naturalness of software[C]. //Proceedings of the 34th International Conference on Software Engineering. Piscataway: IEEE Press, 2012:837-847.

[16] Hindle A, Barr E T, Gabel M, et al. On the naturalness of software[J]. Communications of the ACM, 2016, 59(5): 122-131.

[17] Tip F. A survey of program slicing techniques[J]. Journal of Programming Languages, 1995, 3(3): 1-65.

[18] Devanbu P. New initiative: The naturalness of software[C]. //Proceedings of the 37th International Conference on Software Engineering. Piscataway: IEEE Press, 2015: 543-546.

[19] Tu Z, Su Z, Devanbu P. On the localness of software[C]. //Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2014: 269-280.

[20] Franks C, Tu Z, Devanbu P, et al. Cacheca: A cache language model based code suggestion tool[C]. //Proceedings of the 37th International Conference on Software Engineering-Volume 2. Piscataway: IEEE Press, 2015: 705-708.

[21] Campbell J C, Hindle A, Amaral J N. Syntax errors just aren't natural: Improving error reporting with language models[C]. //Proceedings of the 11th Working Conference on Mining Software Repositories. New York: ACM Press, 2014: 252-261.

[22] Jimenez M. Evaluating Vulnerability Prediction Models[D]. Luxembourg: University of Luxembourg, 2018. https://www.researchgate.net/publication/328215078.

[23] Jimenez M, Maxime C, Le Traon Y, et al. On the impact of tokenizer and parameters on n-gram based code analysis[C]. //Proceedings of the 34th International Conference on Software Maintenance and Evolution. Piscataway: IEEE Press, 2018: 437-448.

[24] 李必信. 程序切片技术及其在面向对象软件度量和软件测试中的应用[D]. 南京: 南京大学, 2000.

[25] Pan Kai, Sunghun Kim, E. James Whitehead. Bug classification using program slicing metrics[C]. //Proceedings of the 6th International Workshop on Source Code Analysis and Manipulation. Piscataway: IEEE Press, 2006: 31-42.

[26] Black S, Counsell S, Hall T, et al. Using program slicing to identify faults in software[C]. Beyond Program Slicing. No. 05451 in Dagstuhl Seminar Proceedings, 2006.

[27] Black S, Counsell S, Hall T, et al. Fault analysis in OSS based on program slicing metrics[C]. //Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications. 2009: 3-10.

[28] Yibiao Yang, Yuming Zhou, Hongmin Lu, et al. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study[J]. IEEE Transactions on Software Engineering, 2015, 41(4): 331-357.

[29] 王俊. 基于程序切片的软件缺陷预测[D]. 上海: 上海交通大学, 2014.

[30] Li Zhen, Zou D, Xu S, et al. VulDeePecker: A deep learning-based system for vulnerability detection[C]. //Proceedings of the Network and Distributed System Security Symposium, 2018.

[31] Malhotra R. A systematic review of machine learning techniques for software fault prediction[J]. Applied Soft Computing, 2015, 27: 504-518.

[32] 于巧, 姜淑娟, 张艳梅, 等. 分类不平衡对软件缺陷预测模型性能的影响研究[J]. 计算机学报, 2018(04): 809-824.

[33] Li Z, Jing X Y, Zhu X, et al. On the multiple sources and privacy preservation issues for heterogeneous defect prediction[J]. IEEE Transactions on Software Engineering, 2019, 45(4):391-411.

[34] Laradji I H, Alshayeb M, Ghouti L. Software defect prediction using ensemble learning on selected features[J]. Information and Software Technology, 2015, 58: 388-402.

[35] Jing X Y, Ying S, Zhang Z W, et al. Dictionary learning based software defect prediction[C]. //Proceedings of the 36th International Conference on Software Engineering. New York: ACM Press, 2014: 414-423.

[36] Nam J, Pan S J, Kim S. Transfer defect learning[C]. //Proceedings of the 35th International Conference on Software Engineering. Piscataway: IEEE Press, 2013: 382-391.

[37] Tantithamthavorn C, McIntosh S, Hassan A E, et al. Automated parameter optimization of classification techniques for defect prediction models[C]. //Proceedings of the 38th International Conference on Software Engineering. Piscataway: IEEE Press, 2016: 321-332.

[38] Yang X, Lo D, Xia X, et al. Deep learning for just-in-time defect prediction[C]. //Proceedings of the International Conference on Software Quality, Reliability and Security. Piscataway: IEEE Press, 2015: 17-26.

[39] Wen M, Wu R, Cheung S C. How well do change sequences predict defects? Sequence learning from software changes[J]. IEEE Transactions on Software Engineering, 2018. DOI: 10.1109/TSE.2018.2876256.

[40] Clemente C J, Jaafar F, Malik Y. Is predicting software security bugs using deep learning better than the traditional machine learning algorithms?[C]. //Proceedings of the International Conference on Software Quality, Reliability and Security. Piscataway: IEEE Press, 2018: 95-102.

[41] Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction[C]. //Proceedings of the 38th International Conference on Software Engineering. Piscataway: IEEE Press, 2016: 297-308.

[42] Wang S, Liu T, Nam J, et al. Deep semantic feature learning for software defect prediction[J]. IEEE Transactions on Software Engineering, 2018. DOI: 10.1109/ TSE.2018.2877612.

[43] Dam H K, Pham T, Ng S W, et al. A deep tree-based model for software defect prediction[J]. arXiv preprint:1802.00921, 2018.

[44] Li J, He P, Zhu J, et al. Software defect prediction via convolutional neural network[C]. //Proceedings of the International Conference on Software Quality, Reliability and Security. Piscataway: IEEE Press, 2017: 318-328.

[45] Phan A V, Le Nguyen M. Convolutional neural networks on assembly code for predicting software defects[C]. //Proceedings of the 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems. Piscataway: IEEE Press, 2017: 37-42.

[46] Dong F, Wang J, Li Q, et al. Defect prediction in android binary executables using deep neural network[J]. Wireless Personal Communications, 2018, 102(3): 2261-2285.

[47] 董枫, 刘天铭, 徐国爱, 等. 面向 Android 二进制代码的缺陷预测方法[J]. 北京邮电大学学报, 2018, 41(01): 13-23.

[48] Zhang Xian, Ben Kerong, Zeng Jie. Cross-Entropy: A new metric for software defect prediction[C]. //Proceedings of the 18th IEEE International Conference on Software Quality, Reliability and Security. Lisbon, Portugal: Piscataway: IEEE Press, 2018: 111-122.

[49] Zhang Xian, Ben Kerong, Zeng Jie. Using cross-entropy value of code for better defect prediction[J]. International Journal of Performability Engineering, 2018, 14(9): 2105-2115.

[50] 张献. 基于语言模型的代码分析及缺陷预测方法研究[D]. 武汉: 海军工程大学, 2019. https://github.com/TOM-ZXian/Research-on-the-Language-Model-Based-Code-Analysis-and-Defect-Prediciton-Method-

[51] Zaremba W, Sutskever I, Vinyals O. Recurrent neural network regularization[J]. arXiv preprint arXiv:1409.2329, 2014.

[52] Agrawal A, Menzies T. Is 'better data' better than 'better data miners'? : On the benefits of tuning SMOTE for defect prediction[C]. //Proceedings of the 40th International Conference on Software Engineering. New York: ACM Press, 2018: 1050-1061.

[53] Nam J, Fu W, Kim S, et al. Heterogeneous defect prediction[J]. IEEE Transactions on Software Engineering, 2018, 44(9): 874-896.

[54] Chandrashekar G, Sahin F. A survey on feature selection methods[J]. Computers & Electrical Engineering, 2014, 40(1): 16-28.

[55] Xu Z, Liu J, Yang Z, et al. The impact of feature selection on defect prediction performance: An empirical comparison[C]. //Proceedings of the 27th International Symposium on Software Reliability Engineering. Piscataway: IEEE Press, 2016: 309-320.

[56] Ghotra B, McIntosh S, Hassan A E. A large-scale study of the impact of feature selection techniques on defect classification models[C]. //Proceedings of the 14th International Conference on Mining Software Repositories. Piscataway: IEEE Press, 2017: 146-157.

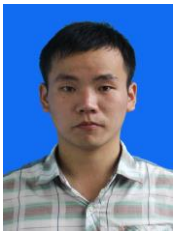
[57] 陈翔, 王莉萍, 顾庆, 等. 跨项目软件缺陷预测方法研究综述[J]. 计算机学报, 2018, 41(1): 254-274.

[58] 陈曙, 叶俊民, 刘童. 一种基于领域适配的跨项目软件缺陷预测方法[J]. 软件学报, 2020, 31(02): 266-281.

[59] Zhou Y, Yang Y, Lu H, et al. How far we have progressed in the journey? An examination of cross-project defect prediction[J]. ACM Transactions on Software Engineering and Methodology, 2018, 27(1):1-51.

张献 (1990-), 男, 河北南宫人, 博士, 讲师,CCF 会员,主要研究领域为软件质量保障、机器学习.

贲可荣(1963-),男,江苏海安人,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程、人工智能.



曾杰(1993-),男,河南固始人,在读博士生,主要研究领域为软件分析、机器学习.