计算机科学
COMPUTER SCIENCE

软件缺陷预测模型可解释性对比

李汇来, 杨斌, 于秀丽, 唐晓梅

引用本文

李汇来, 杨斌, 于秀丽, 唐晓梅.软件缺陷预测模型可解释性对比[J]. 计算机科学, 2023, 50(5): 21-30.

LI Huilai, YANG Bin, YU Xiuli, TANG Xiaomei. Explainable Comparison of Software Defect Prediction Models [J]. Computer Science, 2023, 50(5): 21-30.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

基于卷积神经网络多源融合的网络安全态势感知模型

Multi-source Fusion Network Security Situation Awareness Model Based on Convolutional Neural Network

计算机科学, 2023, 50(5): 382-389. https://doi.org/10.11896/jsjkx.220400134

结合门控机制的卷积网络实体缺失检测方法

Convolutional Network Entity Missing Detection Method Combined with Gated Mechanism

计算机科学, 2023, 50(5): 262-269. https://doi.org/10.11896/jsjkx.220400126

基于多级多尺度特征提取的CNN-BiLSTM模型的中文情感分析

Chinese Sentiment Analysis Based on CNN-BiLSTM Model of Multi-level and Multi-scale Feature Extraction

计算机科学, 2023, 50(5): 248-254. https://doi.org/10.11896/jsjkx.220400069

基于多事件语义增强的情感分析

Sentiment Analysis Based on Multi-event Semantic Enhancement

计算机科学, 2023, 50(5): 238-247. https://doi.org/10.11896/jsjkx.220400256

基于深度学习的视觉问答研究综述

Survey of Visual Question Answering Based on Deep Learning

计算机科学, 2023, 50(5): 177-188. https://doi.org/10.11896/jsjkx.220500124

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.221000028

软件缺陷预测模型可解释性对比

李汇来[1]　杨　斌[2]　于秀丽[3]　唐晓梅[4]
1 北京邮电大学人工智能学院　北京 100876
2 中国联合网络通信有限公司研究院　北京 100048
3 北京邮电大学现代邮政学院　北京 100876
4 北京联合大学信息网络中心　北京 100101
(lihuilai@bupt.edu.cn)

摘　要　软件缺陷预测已经成为软件测试中的重要研究方向,缺陷预测的全面与否直接影响着测试效率和程序运行。但现有的缺陷预测是根据历史数据进行推断,大多不能对预测过程给出合理的解释,这种黑盒的预测过程仅仅展现输出结果,使得人们难以得知测试模型内部结构对输出的影响。为解决这一问题,需挑选软件度量方法和部分典型深度学习模型,对其输入、输出及结构进行简要对比,从数据差异程度和模型对代码的处理过程两个角度进行分析,对它们的异同给出解释。实验表明,采用深度学习的方法进行缺陷预测比传统软件度量方法更加有效,这主要是由它们对原始数据处理过程不同造成的;采用卷积神经网络和长短期记忆神经网络做缺陷预测时,数据差异主要由对代码信息理解的完整程度不同造成的。综上可知,要提高对软件缺陷的预测能力,模型的计算应该对代码的语义、逻辑和上下文联系进行全面的介入,避免有用信息被遗漏。

关键词:软件缺陷预测;可解释性;软件度量;神经网络;抽象语法树
中图法分类号　TP311

Explainable Comparison of Software Defect Prediction Models

LI Huilai[1],YANG Bin[2],YU Xiuli[3] and TANG Xiaomei[4]
1 School of Artificial Intelligence,Beijing University of Posts and Telecommunications,Beijing 100876,China
2 China Unicom Research Institute,Beijing 100048,China
3 School of Modern Post,Beijing University of Posts and Telecommunications,Beijing 100876,China
4 Information Technology Center,Beijing Union University,Beijing 100101,China

Abstract　Software defect prediction has become an important research direction in software testing. The comprehensiveness of defect prediction directly affects the efficiency of testing and program operation. However,the existing defect prediction is based on historical data,and most of them cannot give a reasonable explanation for the prediction process. This black box prediction process only shows the output results,making it difficult for people to know the impact of the internal structure of the test model on the output. In order to solve this problem,it is necessary to select software measurement methods and some typical deep learning models,make a brief comparison of their input,output and structure,analyze them from the two perspectives of the degree of data differences and the processing process of the model on the code,and explain their similarities and differences. Experiments show that the method of deep learning is more effective than traditional software measurement methods in defect prediction,which is mainly caused by their different processing processes of raw data. When using convolution neural network and long-term and short-term memory neural network to predict defects,the data difference is mainly caused by the integrity of the understanding of code information. To sum up,in order to improve the prediction ability of software defects,the calculation of the model should comprehensively involve the semantics,logic and context of the code to avoid the omission of useful information.

Keywords　Software defect prediction,Explicability,Software metrics,Neural network,Abstract syntax tree

1　引言

人工智能在生产生活等各个方面得到了广泛普及,其中,深度学习是解决图像、语言识别等问题的重要手段。近年来,尤其是在自然语言处理(Natural Language Processing,NLP)[1]方面,深度学习克服了传统方法的弊端,大大提高了计算机对语言理解的准确程度。但是由于深度学习是通过标注大量数据驱动的误差后向传播来不断优化模型参数[2],

到稿日期:2022-10-07　返修日期:2023-02-22
通信作者:唐晓梅(tangxiaomei@buu.edu.cn)

也就是说，大多数学习和预测过程都相当于黑盒实验，对于使用者而言，他们难以理解其内部的数据处理过程，无法知晓各个隐藏层之间的数据对应关系，并且模型输出的结果未必完全准确，这在一定程度上影响了人们对机器决策情况的信任度。因此在使用人工智能处理自然语言问题时，在给出结果的同时需要模型能够对求解过程输出合理的解释，这就是NLP中的可解释性问题。可解释的人工智能（Explainable AI，XAI）[3]作为当前人工智能领域研究的新课题，已经在政治、金融、人机交互[4]、恶意软件检测[5]等领域得到了很好的应用，例如政府在了解、掌握群体情绪时需要及时洞察其原因，才能够给出更加合理的政策措施[6]。完整XAI包括解释的对象、解释者和解释受众，并据此构建解释结构[7]，其模型通常分为两大类：一种是透明的解释模型，相当于白盒实验，系统在做决策或预测的过程中就能够输出有用的关键信息，完成自解释；另一种是事后解释，它需要在系统输出结果后对数据进行额外的操作，常用的方法为替代模型，该方法类似于构建一个状态观测器，但该替代模型的结构方法等未必与原模型相似。通常情况下，对单个模型的决策解释需要输出额外的相关信息作为解释输出的原因，而对多种不同模型的性能进行对比分析时，可以针对具体的实验结果以及模型结构做出解释。

随着深度学习对自然语言处理问题的深化，其在软件工程中的应用也逐渐拓展开来。但在软件开发过程中，生产自动化程度不断提高反而可能会产生诸多问题，尤其在大规模的软件系统中，操作人员难以通过人工的方式解决这些故障，这在很大程度上影响了漏洞检测效率[8]。但是，目前的深度学习缺乏数学理论的支撑，从而限制了模型的实际部署和应用[9]。例如，智能模型能够根据历史数据和现有信息完成缺陷预测、测试用例生成、测试用例约简等任务，但并不能保证所得结果为最优解，更无法保证不会产生错误输出。这些偏差可能会对生产生活造成不良影响，甚至威胁企业的收益情况和操作人员的人身安全[10]，并且检测出内在缺陷的时间越晚，修复该缺陷的代价就越高[11]。因此，人们需要鉴别软件判别结果的真假，考量决策的隐含逻辑。而对于输出的差异性，研究人员也需要及时比对，以发现影响输出的局部因素。

应用XAI的软件工程中，软件缺陷预测（Software Defect Prediction，SDP）[12]是一个重要分支。为尽早发现系统中的潜在危害，企业需要雇佣专业的测试人员进行软件测试，这将花费大量资金。为了尽可能提升软件的可靠性，研究人员通过输入语句逻辑信息，使用深度学习模型对新开发的软件系统可能存在的问题进行推断。但由于深度学习的黑盒特性，人们无法理解模型如何判断代码有无漏洞[13]。为了提高人们对机器检测的依赖度，必须清楚影响缺陷预测准确率的因素。本文仅将缺陷预测作为一个简单的下游任务，分析实验结果有差异的原因，包括对实验数据显著性差异的分析和模型具体结构的拆解，主要探讨以下两个问题的可解释性。

（1）与传统的软件度量方法相比，应用深度学习模型对软件的缺陷进行预测是否更有优势？如果有，原因是什么？

（2）不同深度学习模型对软件缺陷的预测效果差异如何？造成该差异的原因是什么？

针对问题1，本文首先针对不同的项目，分别采用传统的软件度量方式、加入抽象语法树和连续词袋模型（Continuous Bag of Words，CBOW）[14]的长短期记忆神经网络（Long-Short Term Memory，LSTM）[15]对软件的缺陷进行预测，每个模型的末端都使用相同的分类器。结果显示，使用深度学习所得的预测效果明显优于软件度量。为了进一步比较实验数据，本文引入了统计学指标来研究其差异度。最终通过对模型的分析得知，模型对数据的处理很大程度上影响了其对代码的分析，上下文的语义中所包含的诸多信息是不可忽略的。针对问题2，本文研究了卷积神经网络（Convolutional Neural Networks，CNN）[16]、单向及双向的长短期记忆神经网络的预测性能。结果表明，卷积神经网络性能最差，这是卷积神经网络对上下文联系不敏感造成的。而长短期记忆神经网络的单双向结构对系统影响不明显，这是因为两者都提取了足够多的代码逻辑信息，即使改变遍历方向也难以获得新的代码特征。

本文的主要贡献包括3个方面：

（1）使用抽象语法树和连续词袋模型对代码信息进行了数据预处理，该方法对程序这类自然语言更加适用。

（2）对比了传统方法和深度学习方法，针对数据差异的部分原因进行了可解释工作。

（3）对比了不同深度学习方法的性能，并对数据差异的部分原因做出了解释。

2 相关工作

2.1 可解释的人工智能

人工智能的可解释性在信息推送、医疗研究、金融、信息安全等领域具有重要的理论研究意义和实际应用价值[17]，目前正逐步成为一个热门研究方向。He[18]提出了人工智能的多粒度的处理，保持神经元的柔性，避免在一味地增加神经网络隐藏层的同时淹没了小粒度知识转化为大粒度知识的可能，介绍了从刚性逻辑到柔性逻辑的扩张。Zhang[19]从数据、模型和经济意义3个方面对推荐系统的可解释性进行研究，对短语级的情感模型进行分析，并根据时间先后做出了一系列动态建模，详细分析了个性化预测推荐。Wu等[20]针对深度学习在医疗、自动驾驶、金融等领域的决策风险，通过分析神经网络的特性，引入可解释性模块，从而构建出了更加合理、有效的可解释性人工智能的理论方法。Park等[21]开发了一个可靠的基于合并诊断算法的智能诊断助手，利用可解释性的人工智能，为核电站的操作决策提供了一套智能诊断辅助，从而减少了人为操作所造成的错误。Lee等[22]对两种不相关的模型的可解释性方法breakDown和shapley的能力进行了比较，用于解释多主元合金相分类，并针对一组特性给出了多种解释。Aslam等[23]使用可解释的人工智能检测、辨别互联网的恶意域名，通过对决策树、朴素贝叶斯、随机森林等模型进行建模比较，最终获得了用于基准研究的可解释性系统。Wu[24]将甲状腺的人体组织影像用深度学习的方式完成辅助诊断，采用梯度加权类激活映射方法进行解释，分析了对

分类结果有影响的像素。Sano 等[25]提出对人与机器人的交互建立一种回归模型，用于对幼儿进行人格评估，采用一种有可解释性的人工智能来理解训练后的估计倾向，该模型能够解释幼儿的气质趋势，并能够在特定情况下减小数据偏差。Zhang 等[26]将可解释系统应用到铁路预测上，对列车的时空特征进行提取，解决了高铁延时所引发的调度问题。

2.2 可解释的软件工程

可解释的软件工程是 XAI 中的重要应用方向，它使得人们对程序的处理进一步精细化。为了解决软件缺陷预测的可解释性问题，Chen 等[27]提出利用 SHAP（Shapley Additive Explanation）模型分析数据的特征，再使用集成学习的方法对大规模开源代码进行分析，在有效提高分类性能的同时给出了可解释性的分析。Wang 等[28]应用神经网络和全局平均池化，设计了可解释模型，用于对软件源码的漏洞进行检测。Zhang 等[28]针对 Android 的恶意软件检测问题，提出了一种新型的检测方法 InterDroid，该方法利用面向漂移的可解释性对软件的 4 种特征进行分析，通过集成学习完成软件检测，其输出的特征更具稳定性和可解释性。为解决 Android 的恶意软件检测问题，Wang 等[29]提出了一种图嵌入的可解释性模型，该模型克服了特征之间的耦合关系，完成了对恶意软件的自动检测。Hu 等[30]针对移动端的 App 即时缺陷预测模型的解释性问题展开研究，引入了 ExplainApp 方法，该方法能够对测试结果生成高质量的解释，其性能相比原始的 LIME 方法有大幅度的提升。Qiu[31]对现有的可解释性恶意软件检测研究方法进行改进，提出了一种稳定的局部黑盒解释方法，应用深度神经网络，增强了对存在时序相关结构的数据的处理能力。Jirayus 等[32]对自动特征提取技术进行研究，观测其对可解释性的缺陷预测模型的影响，并对自动特征提取做了新的优化，提升了模型的可解释性和性能。

3 模型构建及性能预估

软件缺陷预测是软件工程中的重要部分，该操作能够一定程度上缩减测试成本。然而，由于目前处理 SDP 问题时模型的准确性和效率不能完全得到保证，因此需要对该处理过程做出合理的解释，用以辅助测试人员判断有无缺陷。

在做可解释性的缺陷预测之前，先要获取数据集并构建处理 SDP 问题的模型。本文设置了两组对比实验，根据输出结果和模型信息分析实验结果的异同。首先，软件度量方法针对 20 种度量信息的历史数据，通过对比的方法进行预测。而长短期记忆神经网络预测系统又可以分为单双层结构，其模型构建大体可以分为两部分：数据的预处理和神经网络模型搭建。本节对各种典型模型做初步的介绍，并依次定性地估计它们的效果差异。

3.1 软件度量方法

在对软件进行缺陷预测的早期研究中，相关人员通常针对项目的代码和相关文件进行分析，依据源项目和目标项目设计相应的度量元[33]。将这些指标按照一定标准进行对比，找出可能存在缺陷的位置。

在特征提取阶段，已有的大部分研究工作都是仅关注代码的规则及其内部复杂度[11]，通常认为代码的复杂度以及项目的规模大小与代码缺陷的数量及严重程度直接相关。对代码的特征获取，目前都是通过手工的方式，主要分为基于软件代码和基于开发过程的两种度量。前者的度量指标主要取自代码本身，如代码行数、类的继承度深度、对象间的耦合等多种影响因素。后者则更多地考虑了开发人员的存档、历史版本缺陷记录、功能区别、代码变更等，这些数据更具有针对性，缺陷定位相较于前者更加准确。但由于基于过程的程序特征提取能获取的信息有限，不能保证检测的全面性，因此传统的软件度量方法以基于代码的方式居多[34]。

直接采用手工提取的度量信息能够很好地表征系统的宏观信息，且数据统计与获取比较容易。但由于计算机语言的特殊性，该方法可能会忽视大量语句之间的联系，而这些涉及上下文联系的信息极有可能包含错误逻辑，因此深度学习技术被逐步应用到 SDP 领域。

目前基于软件度量的软件缺陷预测方法已经有了一定的积累，例如 Solidity 智能合约预测[35]、代码气味度量的回归模型[36]等，但基于代码的多元相似度的度量[37]是该方案的主流。该方法首先将被测项目分组，再根据不同的缺陷情况和度量情况划分为不同的类别，用其中心点的欧氏距离作为相似度度量，从而判断出度量指标对代码缺陷的影响。另外，本文所采用的被测项目与文献[37]类似，共采用了 18 种相关的度量信息，分别是类的继承树深度、文件中的代码行数、类的响应集大小、对象间耦合、方法内聚性的缺失、方法内聚性的缺失、当前类中 public 方法的个数、当前类中 private 属性所占比例、当前类中方法的内聚性、当前类中方法的耦合度、当前类中方法的平均大小、依赖于当前类的其他类个数、当前类所依赖的其他类个数、当前类的子类个数、字段类型为用户定义类的字段个数、当前类中被继承方法所占比例、最大麦凯布圈复杂度和平均麦凯布圈复杂度。

3.2 针对编程语言的数据预处理

与传统的软件度量方法不同，使用深度学习模型预测软件缺陷是直接针对源代码进行处理，但由于自然语言不能直接输入神经网络参与计算，因此必须提前将语言转化为数值向量。本节介绍了将代码从分词到编码再到向量化处理的全过程。

3.2.1 代码解析及序列生成

自然语言方面的信息并非数值信息，无法直接输入神经网络进行训练，必须生成等长的数值向量才能够做数值运算。又由于编程语言含有大量的分支结构，因此使用对日常交流用语的分词处理方法不能够满足序列的生成需求。

本文以 Java 语言为例，对相关程序进行解析。对于原始的编程语言，本文使用了名为 Javalang 的 Python 包，它能够分别完成分词和解析任务。首先，运用 Javalang 将源代码进行分词[38]。在获取到单个独立单词后，使用解析器对每一个单词进行解析，得到语法树节点。但 Javalang 包含了近百种节点，本文主要选用了以下 4 种节点，分别是：方法调用节点、声明类型节点、控制流节点和定义节点，最终构建出抽象语法树[39]。

为获得数值向量，对抽象语法树的下一步操作生成词序列，我们用统一编排的元素替换节点。对于方法调用节点，我们使用节点调用的方法的名称来表示；对于声明类型的节点，使用声明节点的名称作为替代；对于控制流节点和定义节点，则直接用节点的类型作为标识符。最终，将抽象语法树按照深度优先的原则进行遍历，每一次搜索作为一个序列，最终源代码就用一系列带有先后关系的词序列表示。

3.2.2 向量化处理

为了能将数据输入神经网络做下一步的特征提取，仍然需要对生成的序列做进一步处理。与自然语言处理相比，我们可以把代码中的每一个节点的名称看作一个单词，把所得到的节点序列作为一个句子，如此，就可以用 NLP 中的词嵌套模型做向量化处理。为了充分考虑程序中语句间的逻辑关系，本文使用连续词袋模型，根据上下文推断中心词，构建词向量。

如图 1 所示，CBOW 模型主要由 3 层结构构成，分别是输入层、隐藏层和输出层，其输出充分考虑了上下文的环境。输入的单目标单词的序列应通过 one-hot 进行编码，由多个目标单词向量拼接即可构成 CBOW 的输入，避免了向量过于稀疏和不能反映上下文联系的问题。经过矩阵的乘法运算及加权平均，分别可以得到隐藏层和输出层的内容。实现了向量化处理，接下来就要对模型做训练，最终训练完成后所得的 $\boldsymbol{W}_1$ 就是词嵌入矩阵，所求目标单词的词向量为词嵌入矩阵对应的行。

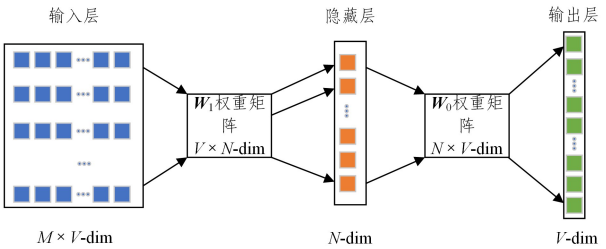


图 1 CBOW 模型结构

Fig. 1 CBOW model structure

另外，在完成数值编码后，由于语法树的深度不同，所包含的词语种类不同，因此最终生成的数值向量长度上存在差异。为了使神经网络能够更好地处理数据，要对这些向量进行规整化的处理，以最长非零向量为基准，去除后面的零元素，长度不足的向量需要补零。

3.3 深度学习模型

为比较不同深度学习模型的性能，本文挑选了部分具有代表性的神经网络模型作为对比模型，分别是卷积神经网络和长短期记忆神经网络。其中，卷积神经网络更多地专注明显特征的辨识，循环神经网络（Recurrent Neural Network，RNN）[40] 及其变体更加专注时序特征的提取，而长短期记忆神经网络又分为单层和双层结构，多用于自然语言处理。

3.3.1 卷积神经网络模型

卷积神经网络是一种具有前馈结构的神经网络，擅长从原始数据中提取抽象的相关信息，泛化能力较强。目前，卷积神经网络也被应用到了自然语言处理方面，该模型能够较好地完成对语义特征的提取，其中的权值共享、局部连接和池化操作等能够较好地降低原数据模型的复杂度。本文所使用的卷积神经网络内部结构包括输入层、输出层、卷积层、非线性激励和全连接层，其隐含层内的卷积核的参数共享。在处理每一轮卷积时均需要非线性计算，并最终通过全连接层组合输出。在对代码进行处理时，卷积神经网络能够获取抽象语法树中复杂的上下文结构信息。另外，为了提高模型对上下文逻辑的处理能力，还可以将提取的词义、语义以及句间特征和软件度量的相关指标进行融合，形成综合性的评价向量，对代码做更加细致的分析对比。然而，由于卷积神经网络与循环神经网络不同，上一时刻的输入输出无法对下一时刻的输出造成影响。此处所忽略的时序逻辑实质上是代码语句间的耦合联系，因此，在代码远程回溯或者远程调用时会忽略部分有效特征。

3.3.2 长短期记忆神经网络模型

卷积神经网络更多地被应用在图像处理过程中，而对自然语言的处理更多的是使用循环神经网络。相比卷积神经网络，循环神经网络及其变体形式能够处理不同时间的输入序列。本文选择将 RNN 加入反向传播后形成的长短期记忆神经网络进行实验，这对于处理编程语言形成的序列能够起到重要作用，得到出色的结果。

LSTM 加入了细胞单元输入，使得当前时刻的输入不仅与上一时刻的输出有关，还与上一时刻的细胞层有关，解决了 RNN 的梯度消失问题。图 2 给出了 LSTM 大致结构图，橙色、蓝色和绿色框分别代表遗忘门、输入门和输出门。另外，LSTM 的候选细胞信息可以通过输入一个 tanh 层获得。通过 Sigmoid 函数加一层乘法操作的形式，去除多余信息，更新有用信息。最终，当前时刻的输出由上一时刻的输出、上一时刻的细胞单元和当前时刻的输入共同决定，达到了把输入转换为 0 到 1 之间的信息，进而实现数据筛选的目的。

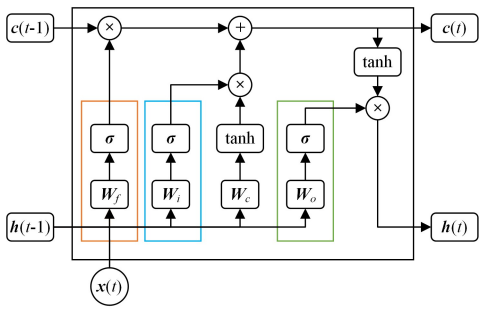


图 2 LSTM 结构（电子版为彩图）

Fig. 2 Structure of LSTM

3.3.3 双向长短期记忆神经网络模型

双向长短期记忆神经网络（Bidirectional Long Short-Term Memory，BLSTM）[41] 与常规的 LSTM 类似，但其实质是 LSTM 的双层结构。对于所生成的向量，BLSTM 会从正反两个方向分别输入模型，更加完整地保留上下文之间的关联内容。

和单层的 LSTM 不同，BLSTM 每次输入的数据量更多，模型中的参数也更多，这种情况下可能会由于阶数过高而

产生过拟合现象，从而降低模型的泛化能力。为了增加系统的鲁棒性，需对其进行正则化操作，通过加入 dropout 层，使得输入的数据受到一定的扰动，输入值按照一定概率偏离原来的位置，增加了随机性。本文以一定概率的形式使系统将一些节点设置为 0，也就是丢弃某些点。其中对每个输入的扰动为：

$$x_i' = \begin{cases} 0, & p = a \\ \dfrac{x_i}{1-a}, & p = 1-a \end{cases} \tag{1}$$

其中，p 代表数据丢失的概率。当 p 为 0.5 时，正则化的效果最好。

具体的 BLSTM 的数据处理过程如图 3 所示。对于源数值向量，要将其复制为两份，一份进行正向的特征提取，另一份进行反向的特征提取，利用 LSTM 的敏感性进行双向的序列分析，最后将提取的结果输入 softmax 层。通过综合考虑，对正向特征向量与反向特征向量分配不同的权重，加权平均后获取最终的特征向量。

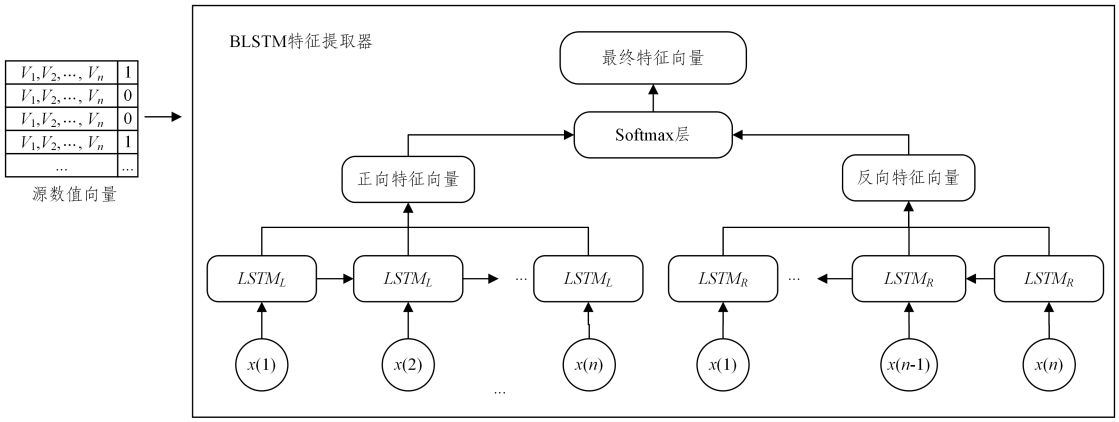


图 3 BLSTM 特征提取器框架

Fig. 3 Framework of BLSTM feature extractor

3.4 模型整体架构

对于神经网络模型，首先需要将源项目的数据随机划分，将其中 80%的数据作为训练集，剩余的 20%作为验证集。对这些信息输入模型进行训练并验证。已知源项目和目标项目数据分布相似，因此可以将训练好的模型直接用于目标项目的软件缺陷预测。完成了所有预处理和对比工作后，还需要在所有的模型之后加入一个 LR 分类器，用于对数据做缺陷倾向性分类。

4 实验基础

为测试不同模型的有效程度，本节设计了两组实验进行比对。在做具体实验之前还要完成获取数据、设置评价标准、设置对比实验等操作。本文的实验环境为 Python3.8，深度学习的框架为 tensorflow2.4.1，在装有 NVIDIA RTX 2080 的 Linux 服务器上运行。常用的数据集包括 NASA MDP，PROMISE，Relink，AEEEM[42-47]。由于 PROMISE 开源软件项目较多且专为软件缺陷预测服务，因此本文选择该数据集进行实验分析。

4.1 实验数据集

选择 PROMISE 开源数据集更加符合本实验的需求，该数据集包含了丰富的代码信息，其中有 Eclipse，Apache 等开源代码数据。本文只用 Java 程序作为预测，选择了 Lucene，Camel，synapse，velocity 这 4 个软件，软件项目的选择主要基于代码功能的多样性和适度的缺陷率，详细信息如表 1 所列。每个项目的不同版本两两组合，由历史版本预测未来版本，总共构成 10 组实验。

表 1 实验数据集细节

Table 1 Details of experimental data set

项目名称	版本号	总文件数	缺陷率/%
Lucene	2.0	186	48.92
	2.2	234	61.11
	2.4	329	61.70
Camel	1.2	595	36.30
	1.4	848	17.10
	1.6	935	20.11
synapse	1.0	157	10.19
	1.1	222	27.02
	1.2	256	33.59
velocity	1.5	214	66.35
	1.6	229	34.06

4.2 实验设计

本文将包含抽象语法树和连续词袋模型的 LSTM 称为抽象连续长短期记忆神经网络（Abstract Continuous Long Short-Term Memory，AC-LSTM）。本文对采用 AC-LSTM 的双层模型与传统的软件度量做软件缺陷预测的性能进行了对比实验。为了保证实验严谨，在传统软件度量模型的最后同样设置了 LR 分类器。

针对深度学习不同特征提取器的性能比较问题，本节同样设置了 CNN，LSTM，BLSTM 特征提取的对比实验。为了保证实验严谨，3 种模型的数据处理均使用抽象语法树和 CBOW 模型，最后的分类过程也相同。

4.3 参数设定与训练

深度学习模型受参数设定影响，在本文中模型规模较小，可以进行试错实验以确定参数。

首先，设定神经网络模型的词向量维度，常用的维度有 64，100，128 等。为保证模型有足够的容量容纳被测项目的

词表，需根据数据集中的实例数估算词表大小，并留出一定的裕度，最终词向量维度的估算值为 100。设定好词向量维度之后，根据输入层、隐藏层和输出层的关系，计算每个节点的输出，从而获取词嵌入矩阵和相应的词向量。

其次，调整生成器和鉴别器，分别先后冻结鉴别器和生成器训练另一方。CNN 沿用文献[8]中的默认参数，设置两轮卷积操作，两层卷积核大小分别为 3 * 3 和 5 * 5，学习率为 0.001。而 BLSTM 和 LSTM 的生成器 RMSprop 参数为 0.01，SGD 鉴别器学习率为 0.0005，momentum 为 0.9，nesterov 为 True。

4.4 评价指标

在统计完实验输出后，需要对结果进行初步的划分。在二分类任务处理中，对混淆矩阵的定义如表 2 所列，共分为 TP，FP，TN，FN 4 类，最后为了更准确地对实验进行评价，需再对这些统计值做进一步处理。

表 2 混淆矩阵

Table 2 Confusion matrix

	实际为正	实际为负
分类为正	TP	FP
分类为负	TN	FN

(1)精确率(Precision)，表示真实样本在正分类样本中所占的比例，其表达式如下：

$$Precision=\frac{TP}{TP+FP} \tag{2}$$

(2)召回率，也被称为真实阳性率(True Positive Rate，TPR)，表示正分类样本在真实样本中所占的比例，其表达式如下：

$$Recall=TPR=\frac{TP}{TP+FN} \tag{3}$$

(3)虚假阳性率(False Positive Rate，FPR)，表示正分类样本在假样本中所占的比例，其表达式如下：

$$FPR=\frac{FP}{FP+TN} \tag{4}$$

(4)准确率(Accuracy，ACC)，表示正确分类样本数量在所有样本集中所占的比例，其表达式如下：

$$Accuracy=\frac{TP+TN}{TP+FP+TN+FN} \tag{5}$$

(5)F1-度量是精确率和召回率的加权调和平均，其值在 0～1 之间，模型的分类性能与该值成正相关[48]。

$$F1\text{-}measure=\frac{2\times Precision\times Recall}{Precision+Recall} \tag{6}$$

5 实验结果

5.1 实验 1 结果分析

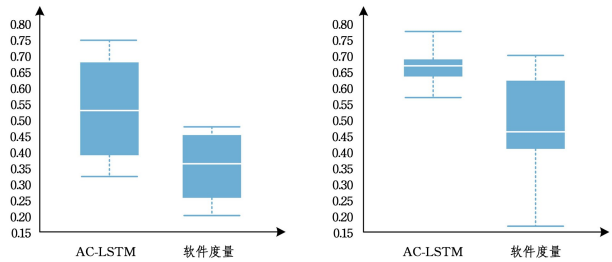
针对问题 1，实验对比了双层 AC-LSTM 方法和传统软件缺陷预测方法，研究其性能差异，表 3 列出了处理后的实验数据。

表 3 问题 1 的实验结果

Table 3 Experimental results of question 1

源项目	目标项目	*F1-measure*		*ACC*	
		AC-LSTM	软件度量	AC-LSTM	软件度量
Lucene2.0	Lucene2.2	0.757	0.434	0.624	0.432
Lucene2.0	Lucene2.4	0.725	0.489	0.669	0.447
Lucene2.2	Lucene2.4	0.736	0.477	0.584	0.405
Camel1.2	Camel1.4	0.372	0.226	0.573	0.131
Camel1.2	Camel1.6	0.426	0.194	0.692	0.120
Camel1.4	Camel1.6	0.330	0.204	0.791	0.648
Synapse1.0	Synapse1.1	0.326	0.351	0.739	0.712
Synapse1.0	Synapse1.2	0.524	0.370	0.695	0.703
Synapse1.1	Synapse1.2	0.552	0.337	0.664	0.548
Velocity-1.5	Velocity-1.6	0.604	0.456	0.651	0.371
平均值		0.535	0.354	0.668	0.452

对实验数据进行处理后，仅比较 F1 度量值和 ACC 两种指标。由表 3 可知，在 10 组实验中，双层 AC-LSTM 所得到的 F1 度量值和 ACC 值全都优于软件度量方法所得数据。从平均值看，双层 AC-LSTM 比软件度量方法的 F1 度量值高出 51.1%，ACC 值高出 47.8%。总体而言，双层 AC-LSTM 模型的输出效果远超过传统的软件度量方法，预测效果更佳。



(a)F1 measure (b)ACC

图 4 实验 1 的箱线图

Fig. 4 Box diagram of experiment 1

从图 4 所示的箱型图能够更加直观地观察出两者输出的效果差异，不论是数据的中位数还是整体趋势，AC-LSTM 模型都有着明显的优势。

5.2 实验 2 结果分析

本实验对比了 BLSTM，CNN，LSTM 这 3 种方法在进行软件缺陷预测时的性能，研究不同深度学习模型的有效程度，实验指标和实验 1 相同。结果如表 4 所列。

表 4 问题 2 的实验结果

Table 4 Experimental results of question 2

源项目	目标项目	*F1-measure*			*ACC*		
		BLSTM	CNN	LSTM	BLSTM	CNN	LSTM
Lucene2.0	Lucene2.2	0.757	0.681	0.730	0.624	0.624	0.645
Lucene2.0	Lucene2.4	0.725	0.661	0.728	0.669	0.550	0.611
Lucene2.2	Lucene2.4	0.736	0.583	0.693	0.584	0.495	0.572
Camel1.2	Camel1.4	0.372	0.317	0.345	0.573	0.661	0.689
Camel1.2	Camel1.6	0.426	0.275	0.351	0.692	0.577	0.628
Camel1.4	Camel1.6	0.330	0.331	0.369	0.791	0.750	0.755
Synapse1.0	Synapse1.1	0.326	0.319	0.278	0.739	0.730	0.733
Synapse1.0	Synapse1.2	0.524	0.230	0.417	0.695	0.660	0.684
Synapse1.1	Synapse1.2	0.552	0.331	0.503	0.664	0.652	0.659
Velocity-1.5	Velocity-1.6	0.604	0.551	0.523	0.651	0.559	0.690
平均值		0.535	0.428	0.493	0.668	0.626	0.667

由表 4 可知，从平均值来看，BLSTM 的 F1 度量比 CNN 方法高出 25%，比单层 LSTM 高出 8.5%。BLSTM 的 ACC 值比 CNN 方法高出 6.7%，但与单层的 LSTM 相比，两者几乎持平。

从单体的实验看，BLSTM 模型在 F1 度量值上比 CNN 和单层的 LSTM 更优的实验占比分别为 90% 和 80%，在 ACC 值上比 CNN 和单层的 LSTM 更优的实验占比分别为 80% 和 70%。从该角度来看，BLSTM 明显优于 CNN 模型，但和 LSTM 相比，BLSTM 虽有优势，但两者数值差异较小，部分实验结果会出现 LSTM 更优的情况。

将以上数据绘制成柱状图，如图 5 和图 6 所示。其中 x 轴为实验序号，y 轴为对应的 F1 度量值和 ACC 值。从图中可知，BLSTM 和单层的 LSTM 性能都明显优于 CNN 方法，BLSTM 和单层的 LSTM 数据相差较小，但前者的性能略优于后者。

综上所述，在 3 种模型中，BLSTM 的性能最优，其次是单层 LSTM，而 CNN 在本次实验中表现最差。其中 BLSTM 和单层 LSTM 相比性能差异较小，实验数据相近，但前者略优于后者。

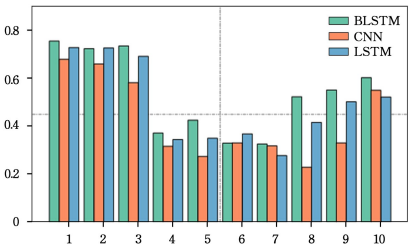


图 5　实验 2 在 F1 度量上的柱状图

Fig. 5　Histogram of experiment 2 on F1 measurement

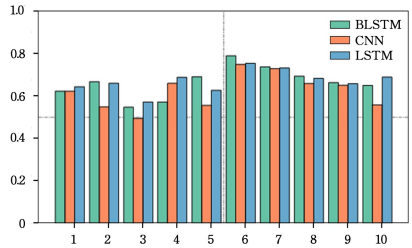


图 6　实验 2 在 ACC 上的柱状图

Fig. 6　Histogram of experiment 2 on ACC

6　模型对比的可解释性分析

本节针对上述两组实验结果进行分析，应用统计学指标深入分析它们的差异程度，从而对模型性能差异做出合理的解释。

6.1　差异性解释

为了对比模型之间的差异性，需要对数据做进一步的处理，此处我们选取两种检验方法，分别用来表征数据的显著与否以及差异性的大小。

(1)Wilcoxon 符号秩检验[49]。对于两个相互独立的样本，该方法可以检验它们之间的差异程度。在置信度达到 95%时，p 值小于 0.05 表示两组数据的差异明显，反之，表示两组数据差异不显著。

(2)δ 值检验。该方法能够划分区间，定性地分析两组数据的差异程度。首先要计算两组数据的增量函数，然后按照定义计算 δ 值，具体如下：

$$\delta(i,j)=\begin{cases}+1, & x_i>y_i\\ 0, & x_i=y_i\\ -1, & x_i<y_i\end{cases} \tag{7}$$

$$\delta=\frac{1}{mn\sum_{i=1}^{n}\sum_{j=1}^{m}\delta(i,j)} \tag{8}$$

根据不同的 δ 值对两组实验数据的差异显著性水平进行划分，具体的划分标准如表 5 所列。

表 5　不同 δ 值对应的差异水平

Table 5　Difference level corresponding to different δ values

δ 值大小	差异水平
$0.474\leqslant\lvert\delta\rvert$	高
$0.33\leqslant\lvert\delta\rvert<0.474$	中
$0.147\leqslant\lvert\delta\rvert<0.33$	较低
$\lvert\delta\rvert<0.147$	可忽略不计

6.2　针对实验 1 的可解释性分析

实验 1 针对双层 AC-LSTM 模型和传统的软件度量方法，分别对其软件缺陷预测水平做了对比实验。实验数据表明，采用神经网络提取特征向量的方式解决 SDP 问题所得到的效果远远超过传统手工提取软件度量的预测方式。

Wilcoxon 符号秩检验和 δ 值检验的结果如表 6 所列。其中两种度量指标所对应的 p 值均为 0.000391，该值远远小于 0.05，因此可以认为两者模型的实验结果具有显著性差异。F1 度量和 ACC 对应的 δ 值分别为 0.56 和 0.58，通过查询可知两组数据的差异水平很高。

表 6　Wilcoxon 符号秩检验和 δ 值检验结果

Table 6　Wilcoxon signed rank test and δ value inspection results

	F1-measure	*ACC*
p 值	3.91×10^{-3}	3.91×10^{-3}
δ 值	0.56	0.58

计算机程序的语法结构与我们常规的书面用语不同。在人们的日常交流中，前后的语句往往仅存在内容上的关系，而逻辑关系通常用逻辑连词进行连接。但在代码中，前后文关系紧密，如果程序设计有微小错误，可能会导致整个代码无法运行或错误执行。也就是说，计算机语言的上下文语义更加明确，耦合程度更深。因此，在双层 AC-LSTM 中采用抽象语法树和 CBOW 模块能够更好地保留各类信息，尤其是逻辑类信息，实现更加准确的缺陷预测。

使用抽象语法树的优势明显，如图 7 所示，代码示例 1 与代码示例 2 的结构、语句极为相似，但所表达的内容完全不同。其中，代码 1 能够正常执行，不产生错误信息，但代码 2 的累加语句放在了循环结构的外面，微小的结构改变，上下文的语义信息就会发生巨大变化，导致该段代码陷入无限循环中。而采用右边语法树的结构表示，不仅得到了所有关键语句，并且整个执行顺序清晰易懂，更加完整地保留了程序的相关数据。

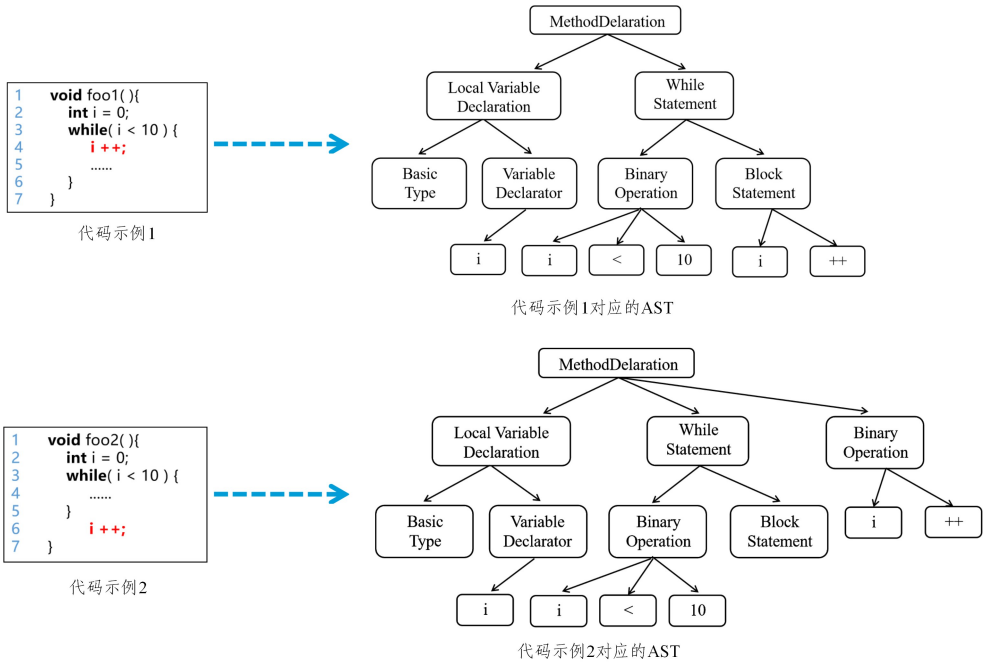


图 7 抽象语法树对比实例

Fig. 7 Example of abstract syntax tree comparison

而双层 AC-LSTM 对数据双向反复训练，将事件的发展按照时间顺序展开，关注上一时刻对未来事件发展的影响，并非只提取重要指标，因此这种方法有着更好的预测性能。

6.3 针对实验 2 的可解释性分析

实验 2 针对 BLSTM 模型、CNN 方法和单层的 LSTM 模型，分别对其软件缺陷预测水平做了对比实验。实验数据表明，采用 BLSTM 进行软件缺陷预测在这三者中的有效程度最高，CNN 方法的效果最差。而 BLSTM 中采用 BLSTM 特征提取器的效果和单层的 LSTM 相比差异较小。

Wilcoxon 符号秩检验和 δ 值检验的结果如表 7 所列。表中数据代表 BLSTM 模型的实验数据分别与 CNN 和单层的 LSTM 相比的统计学指标。从表中可知，与 CNN 对比时，BLSTM 的 F1 度量和 ACC 对应的 p 值分别为 0.00391 和 0.00440，均小于 0.05，可以认为两组实验数据之间有显著性差异。与单层的 LSTM 相比，BLSTM 的 p 值分别为 0.02480 和 0.62500，两者的 F1 度量值有明显差别，但两组 ACC 值没有显著性差异。进一步观察 δ 值，与 CNN 对比时，BLSTM 的 F1 度量和 ACC 对应的 δ 值分别为 0.42 和 0.35，这表明它们的差异性水平为中等。与单层的 LSTM 相比，BLSTM 的 δ 值分别为 0.22 和 0.06，说明两者的 F1 度量数据的差异水平不可忽略，但两者的 ACC 数据的差异水平可以忽略不计。

表 7 Wilcoxon 符号秩检验和 δ 值检验结果

Table 7 Wilcoxon signed rank test and δ value inspection results

	F1-measure		*ACC*	
	CNN	LSTM	CNN	LSTM
p 值	0.00391	0.02480	0.04400	0.62500
δ 值	0.42	0.22	0.35	0.06

综上所述，首先，卷积神经网络的训练模型和另外两者相比，其前后时刻的输出缺少关联性，在处理编程语言问题时性能不及循环神经网络及其变体形式。虽然其可以将上下文特征与软件度量信息融合在一起，通过获取更有代表性的特征来优化性能，但从数据处理的本质上看，CNN 更加关注局部的上下文逻辑，对于间隔距离较长的代码逻辑关系无法进行合理训练。

从模型的整体结构看，两者的唯一差别在于特征提取器有无逆向操作以及 softmax 层，这也是两者输出指标差异性不大的原因。但由于代码信息中包含着丰富的上下文逻辑，因此在进行特征提取时，如果模型能够对数据做反复的运算处理，那么其性能也会有一定程度的提升。

从实验结果看，两种方法的 ACC 值差别较小。这是因为双向结构并不会对准确性有太大提升，且也没有加入额外的代码数据，仅仅是将代码向量反向遍历，相当于特征提取器细化对代码的研读。但是就 F1 度量值而言，双层 AC-LSTM 的多数单体实验要优于单层的 AC-LSTM，这是因为 F1 度量综合了精确率和召回率的影响，能够反映系统整体的性能。

实际上，在进行特征提取时，正向和反向的顺序体现的是对不同分支的遍历顺序。假设代码回溯问题较多时，尤其是回溯的距离较远时，那么正向和反向输入极有可能提升缺陷预测效果，但是这种反向输入应当从构建抽象语法树时就尝试逆向构建。而本文中的正反向向量输入仅仅指的是输入顺序，但 CBOW 模型的输入窗口大小是固定的，因此并不会因为代码回溯问题而对预测性能产生影响。本文中 BLSTM 的实质是颠倒上下文顺序，即增加一组相似的输入，类似于一种对抗学习，只是此处没有真实对抗，仅仅取最优部分。

另外，使用深度学习解决 SDP 问题时，其本身的优化结果与理论的真实最优值较为接近，因此所预留的改进空间被进一步压缩，这也是实验 2 中输出数据差异性不大的原因。

结束语 软件的缺陷预测问题一直是软件测试中倍受关注的部分，它对提升系统的鲁棒性有着重要意义。本文为了初步理解不同模型对软件缺陷预测效果的差异问题，分别

设置了两组实验。通过实验比较可知，AC-LSTM 相比传统的软件度量方法有明显的性能提升；而深度学习模型中 BLSTM 和单层 LSTM 相比 CNN 模型有一定的提升，但是 LSTM 模型的层数对预测效果的影响较小。为了理解实验结果，本文对两种实验输出的原因进行了可解释性分析。BLSTM 与软件度量方法的输出差异，主要是由输入的数据内容、形式以及处理数据的方法不同造成的，采用深度学习对软件缺陷做预测时其效果远优于基于代码的软件度量方法。而 BLSTM 与 CNN、单层 LSTM 对比时所产生的输出差异，主要是由长距离调用关注度、数据输入形式、输入数量以及改进裕度决定的。加入反向传播后形成的长短期记忆神经网络，在处理编程语言形成的序列时能够得到出色的结果。综上所述，模型的数据处理方式、模型的结构以及训练方法都会对软件缺陷预测的结果产生不同程度的影响。

在以后的研究中，我们将对本文模型及其可解释性做出相应的改进。首先，对于抽象语法树的生成，未来也可以考虑更多的生成方式，从而充分获取数据，避免遗漏长距离代码回溯造成的缺陷。其次，为进一步实现模型的可解释性，可以在预测过程输出中间变量，用以辅助解释，构建出透明的深度学习系统。最后，可以找寻简易的可解释的相关模型，通过后续的比较，完成对系统的事后解释。

参 考 文 献

[1] ZHAO J S, SONG M X, GAO X. A review of the development and application of natural language processing[J]. Information Technology and Informatization, 2019(7): 142-145.

[2] ZHANG J Y, WANG H L, GUO Y, et al. Review of deep learning[J]. Application Research of Computers, 2018, 35(7): 1921-1928, 1936.

[3] KONG X W, TANG X Z, WANG Z M. A survey of explainable artificial intelligence decision[J]. Systems Engineering-Theory & Practice, 2021, 41(2): 524-536.

[4] ZHANG K Y, ZHANG W N, LIU T. A survey of multi-party dialogue research based on deep learning[J]. Scientia Sinica(Informationis), 2021, 51(8): 1217-1232.

[5] WANG X G. Research on malware detection technology of mobile intelligent terminal based on behavior sequence[D]. Guangzhou: Guangzhou University, 2022.

[6] ZHAO M. Group Emotion analysis based on multiple dimension [D]. Beijing: Beijing University of Technology, 2018.

[7] DING X. Constructing network learning object model with interpretive structure model[J]. Journal of Fujian Computer, 2010, 26(2): 173.

[8] ZHANG Q. Research on software defect prediction based on convolutional neural network[J]. Computer Programming Skills & Maintenance, 2020, (9): 14-16.

[9] WANG Y X. Research on interpretability method of deep recognition network[D]. Harbin: Harbin Institute of Technology, 2021.

[10] CHEN D J. Bug interpretation-based input generation for software testing[D]. Nanjing: Nanjing University, 2021.

[11] CHEN X, GU Q, LIU W S, et al. Survey of static software defect prediction[J]. Journal of Software, 2016, 27(1): 1-25.

[12] GAO T, GUO X. Integrated software defect prediction model for code evolution[J]. Journal of Wuhan University(Natural Science Edition), 2022, 68(3): 279-288.

[13] ZHU Y W. Research on automatic generation of vulnerability features for program code[D]. Wuhan: Huazhong University of Science and Technology, 2020.

[14] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space[J]. arXiv: 1301.3781, 2013.

[15] LI J, HE P, ZHU J, et al. Software defect prediction via convolutional neural network[C]// 2017 IEEE International Conference on Software Quality, Reliability and Security(QRS). IEEE, 2017: 318-328.

[16] LI Z W, LIU F, YANG W J, et al. A survey of convolutional neural networks: analysis, applications, and prospects[J]. arXiv: 2004.02806, 2020.

[17] CHENG K Y, WANG N, SHI W X, et al. Research advances in the interpretability of deep learning[J]. Journal of Computer Research and Development, 2020, 57(6): 1208-1217.

[18] HE H C. Retrieving the interpretability of AI[J]. CAAI Transactions on Intelligent Systems, 2019, 14(3): 393-412.

[19] ZHANG Y F. Research on explainable personalized recommender systems[D]. Beijing: Tsinghua University, 2016.

[20] WU F, LIAO B B, HAN Y H. Interpretability for deep learning [J]. Aero Weaponry, 2019, 26(1): 39-46.

[21] PARK J H, JO H S, LEE S H, et al. A reliable intelligent diagnostic assistant for nuclear power plants using explainable artificial intelligence of GRU-AE, LightGBM and SHAP[J]. Nuclear Engineering and Technology, 2022, 54(4): 1271-1287.

[22] LEE K, AYYASAMY MUKIL V, JI Y F, et al. A comparison of explainable artificial intelligence methods in the phase classification of multi-principal element alloys[J]. Scientific Reports, 2022, 12(1): 11591.

[23] ASLAM N, KHAN I U, MIRZA S, et al. Interpretable machine learning models for malicious domains detection using explainable artificial intelligence(XAI) [J]. Sustainability, 2022, 14(12): 1-22.

[24] WU S P. Research on assisted diagnosis of thyroid nodules on SPECT image based on deep learning[D]. Harbin: Harbin Institute of Technology, 2021.

[25] SANO T, HORII T, ABE K, et al. Temperament estimation of toddlers from child-robot interaction with explainable artificial intelligence[J]. Advanced Robotics, 2021, 35(17): 1068-1077.

[26] ZHANG D, XU Y, PENG Y, et al. An interpretable station delay prediction model based on graph community neural network and time-series fuzzy decision tree[J]. IEEE Transactions on Fuzzy Systems, 2022, 31(2): 421-433.

[27] CHEN L Q, WANG C, SONG S L. Just-in-time software defect prediction model and its interpretability research[J]. Journal of Chinese Computer Systems, 2022, 43(4): 865-871.

[28] ZHANG B, WEN Z, WEI X Y, et al. InterDroid: an interpretable Android malware detection method for conceptual drift[J]. Journal of Computer Research and Development, 2021, 58(11):

2456-2474.

[29] WANG Y L, LU M M. Interpretable automatic detection of Android malware based on graph embedding[J]. Computer Engineering and Applications, 2021, 57(23): 122-128.

[30] HU X Y, CHEN X, XIA H L, et al. Interpretable method of just-in-time defect prediction model for mobile App[J]. Application Research of Computers, 2022, 39(7): 1-5.

[31] QIU M M. Research on detection method of interpreted malware based on deep learning[D]. Nanjing: Nanjing University of Technology, 2021.

[32] JIRAYUS J, CHAKKRIT T, CHRISTOPH T. The impact of automated feature selection techniques on the interpretation of defect models[J]. Empirical Software Engineering: An International Journal, 2020, 25(5): 3590-3638.

[33] CHENG M. Research on some key technologies of software defect prediction[D]. Wuhan: Wuhan University, 2016.

[34] GONG L N, JIANG S J, JIANG L. Research progress of software defect prediction[J]. Journal of Software, 2019, 30(10): 3090-3114.

[35] YANG H W, CUI Z Q, CHEN X, et al. Defect prediction for solidity smart contracts based on software measurement[J]. Journal of Software, 2022, 33(5): 1587-1611.

[36] HALL T, ZHANG M, BOWES D, et al. Some code smells have a significant but small effect on faults[J]. ACM Transactions on Software Engineering and Methodology, 2014, 23(4): 1-39.

[37] WANG C, YU Q, HAN H. Similarity-based training set recommendation for software defect prediction[J]. Computer Engineering and Applications, 2022, 1(10): 10-31.

[38] CHIBA S, TATSUBORI M. Yet another java. lang. class[J]. Lecture Notes in Computer Science, 1998, 1543: 372-373.

[39] CHEN C T, PAN L M, GONG J, et al. The vulnerability detection method based on compression coding of abstract syntax tree [J]. Journal of Information Security Research, 2022, 8(1): 35-42.

[40] GOODFELLOW I, BANGIO Y, COURVILLE A. Deep learning [M]. Cambridge: MIT Press, 2016: 367-415.

[41] RAY A, RAJESWAR S, CHAUDHURY S. Text recognition using deep BLSTM networks [C]// 2015 Eighth International Conference on Advances in Pattern Recognition(ICAPR). IEEE, 2015: 1-6.

[42] GRAY D, BOWES D, DAVEY N, et al. Reflections on the NASA MDP data sets[J]. Software Let, 2012, 6(6): 549-558.

[43] ZHANG Q, WU B. Software Defect Prediction via Transformer [C]// 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference(ITNEC). IEEE, 2020.

[44] WANG S, LIU T Y, NAM J, et al. Deep Semantic Feature Learning for Software Defect Prediction[J]. IEEE Transactions on Software Engineering, 2020, 46(12): 1267-1293. .

[45] WU R X, ZHANG H Y, KIM S H, et al. Relink: Recovering links between bugs and changes[C]// Proceeding of the ACM SIGSOFT Symposium and the European Conference on Foundations of Software Engineering. 2011: 15-25.

[46] D'AMBROS M, LANZA M, ROBBES R. An extensive comparison of bug prediction approaches[C]// Proceeding of the Mining Software Repositories. 2010: 31-41.

[47] SUN Y, JING X Y, WU F, et al. Adversarial Learning for Cross-project Semi-supervised Defect Prediction [J]. IEEE Access, 2020, 8: 32674-32687.

[48] XING Y, QIAN X M, GUAN Y, et al. Cross-project defect prediction method using adversarial learning[J]. Journal of Software, 2022, 33(6): 2097-2112.

[49] ZENG Y, LI G H, ZHUANG L. A comparison of the power of the Wilcoxon rank-sum test and K-S test on completely randomized design of independent samples[J]. Chinese Journal of Health Statistics, 2011, 28(4): 372-374.

LI Huilai, born in 1998, postgraduate, is a member of China Computer Federation. His main research interests include software test and deep learning.

TANG Xiaomei, born in 1978, senior engineer. Her main research interest is network and information security.

（责任编辑：何杨）