第47卷 第12期 计 算 机 工 程 2021年12月
Vol.47 No.12 Computer Engineering December 2021

·体系结构与软件技术· 文章编号：1000-3428(2021)12-0230-06 文献标志码：A 中图分类号：TP311

基于重子节点抽象语法树的软件缺陷预测

黄晓伟[1,2]，范贵生[1]，虞慧群[1]，杨星光[1]

（1.华东理工大学 计算机科学与工程系，上海 200237；2.上海市计算机软件测评重点实验室，上海 201112）

摘 要：在实际软件项目开发过程中，软件缺陷预测能辅助测试人员找到项目中可能存在缺陷的位置，并通过抽象语法树(AST)获取项目模块中隐藏的结构和语义信息，此类信息有助于提高缺陷预测精度。提出基于重子节点抽象语法树的缺陷预测方法，在提取节点信息时保留节点的类型信息和对应代码语义的值信息，并使用特殊字符串代替没有值信息的节点。通过树链剖分思想将AST分割为重子节点和轻子节点，优先选择重子节点作为序列化向量中的节点，同时利用深度学习网络学习节点序列中的源代码结构和语言实现软件缺陷预测。实验结果表明，与DFS方法相比，该方法在基于注意力机制的循环神经网络深度学习模型上的F1值和AUC值平均提升约3%和4%，具有更好的缺陷预测效果。

关键词：软件质量保障；软件缺陷预测；代码表征；抽象语法树；深度学习

开放科学(资源服务)标志码(OSID)：

中文引用格式：黄晓伟，范贵生，虞慧群，等.基于重子节点抽象语法树的软件缺陷预测[J].计算机工程，2021，47(12)：230-235，248.

英文引用格式：HUANG X W，FAN G S，YU H Q，et al.Software defect prediction via heavy son node-based abstract syntax tree[J].Computer Engineering，2021，47(12)：230-235，248.

Software Defect Prediction via Heavy Son Node-based Abstract Syntax Tree

HUANG Xiaowei[1,2]，FAN Guisheng[1]，YU Huiqun[1]，YANG Xingguang[1]

（1.Department of Computer Science and Engineering，East China University of Science and Technology，Shanghai 200237，China；
2.Key Laboratory of Computer Software Testing and Evaluating，Shanghai 201112，China）

【Abstract】In the actual software project development，software defect prediction can assist testers to find possible defects in the project.Through the Abstract Syntax Tree（AST），the hidden structure and semantic information in the project module can be obtained，which helps to improve the accuracy of defect prediction.This paper proposes a defect prediction method using Heavy Son（HS）node-based abstract syntax tree.In node information extraction，the type information of the node and the value information of the corresponding code semantics are retained，and the nodes without value information are replaced with a special string.Then by using the idea of tree chain division，the AST is divided into HS nodes and Light Son（LS）nodes.The HS nodes are selected in preference as the nodes in serialized vectors.At the same time，the deep learning network is used to learn the source code structure and language in the node sequence to realize software defect prediction.Experimental results show that compared with the DFS method，the proposed method improves the F1-measure by 3% and the AUC value by 4%，has a better defect prediction effect.

【Key words】software quality assurance；software defect prediction；code representation；Abstract Syntax Tree（AST）；deep learning

DOI：10.19678/j.issn.1000-3428.0060389

0 概述

在软件项目开发过程中，软件项目的最终可靠性是所有开发人员以及项目负责人都十分关注的。软件缺陷是影响软件项目可靠性的重要因素，一般通过减少软件缺陷的引入以及修正已经存在的缺陷

基金项目：国家自然科学基金(61702334，61772200)；上海市浦江人才计划(17PJ1401900)；上海市自然科学基金(17ZR1406900，17ZR1429700)；华东理工大学教育科研基金(ZH1726108)。

作者简介：黄晓伟(1995—)，男，硕士研究生，主研方向为软件缺陷预测；范贵生(通信作者)，副研究员、博士；虞慧群，教授、博士生导师；杨星光，博士研究生。

收稿日期：2020-12-24 **修回日期：**2021-01-29 **E-mail：**393046424@.qq.com

两种方法来减少软件缺陷。软件项目的开发人员和测试人员通过构建软件缺陷模式[1],能够利用积累的软件缺陷数据提高软件项目的可靠性。

软件缺陷预测(Software Defect Prediction,SDP)[2-3]利用软件开发过程存储的历史数据和测试人员对发现缺陷模块的标注,通过构建机器学习分类器对新开发的软件模块进行分类预测,判断该模块是否存在缺陷。根据源数据项目与目标数据项目的差异,软件缺陷预测可以进一步划分为项目内缺陷预测(Within-Project Defect Prediction,WPDP)[4-5]、跨项目缺陷预测(Cross-Project Defect Prediction,CPDP)[4,6]、异构缺陷预测(Heterogeneous Defect Prediction,HDP)[7-8]。根据预测程序规模,程序包括函数级、文本级、变更级等预测粒度。

传统缺陷预测方法根据项目源代码的行数、复杂度、耦合度等提取描述源代码的指标,利用构建的机器学习分类模型进行分类学习。但是,这些静态代码指标[9-10]只是代码的宏观描述,有缺陷代码和无缺陷代码很可能具有相同的代码指标,使分类器无法区分。这些宏观上无法区分的代码,具有不同语义和结构信息,因此利用项目代码的语义和结构信息能够提高缺陷预测性能。抽象语法树(Abstract Syntax Tree,AST)是以树状的形式表现编程语言的语法结构,树上的每个节点都表示源代码中的一种结构。目前,研究人员提出的基于AST的DP-CNN[11]和DP-ARNN[12]方法仅使用了部分源代码的语义和结构信息。

为利用AST中的更多信息,本文提出一种基于重子(Heavy Son,HS)节点抽象语法树的软件缺陷预测方法HSAST。将程序源代码的AST进行序列化提取节点类型和值信息,根据节点轻重分类确定舍去节点时的优先级,并使用F1值[13]和AUC值[14]作为评测指标,同时将Apache中的5个项目作为实验数据集。

1 相关工作

1.1 软件缺陷预测数据集获取

当软件项目处于初级阶段且存在历史库丢失或损坏等问题时,软件项目需要的数据无法从自身项目的历史库中获取,此时就需要借助其他的项目数据,属于跨项目缺陷预测和异构缺陷预测。在跨项目缺陷预测和异构缺陷预测方面,研究人员一般通过迁移学习等方法解决了不同项目度量元分布不同的问题。倪超等[6]采用两阶段跨项目缺陷预测方法FeCTrA,先通过聚类筛选出源项目与目标项目具有高分布相似度的特征,再基于TrAdaBoost方法选出源项目中与目标项目标注实例分布相似的实例进行预测。RYU等[15]在迁移学习过程中,考虑到少量的目标项目标记会加重缺陷预测数据集的类不平衡问题,提出一种对迁移成本敏感的学习方法,将权重分配给训练实例。BENNIN等[16]通过减弱缺陷数据集的类不平衡程度来提升缺陷模型的预测精度并提出MAHKIL方法。MAHKIL方法是一种基于染色体理论的缺陷数据集采样方法,利用两个不同的缺陷样本生成一个新实例,新实例继承父亲样本的不同特征。

1.2 深度学习模型

传统缺陷预测方法使用表示代码宏观信息和固有特征的度量元来描述代码模块。这些度量元由有经验的开发人员、测试人员和研究人员设计提出,可能更切合总结特征时选择的软件项目。当这些度量元用来描述其他软件项目时,部分指标与标签没有高度相关性,变成冗余信息,还可能影响预测模型的性能。因此,充分利用程序代码的结构和语义信息,能够使代码模块的表征更切合目标软件。

AST和控制流图(Control Flow Graph,CFG)是当前常用的利用数据结构对程序代码进行表示的方法。AST是以树状形式表示软件源代码的语法结构,树上的每个节点都表示源代码中的一种结构。CFG是以图的形式表示软件代码中一个模块在执行过程中的可能流向。LI等[11]采用AST表示程序代码,遍历AST时选择具有特定类型的节点,通过字典映射并利用节点转换向量构建向量组来描述整个程序代码。通过卷积神经网络(Convolutional Neural Networks,CNN)产生一个向量来表征整个程序代码,这个向量具有类似度量元的功能。PAN等[17]在卷积层后增加了dense层和dropouts层来减少模型过拟合问题。FAN等[12]提出一种基于注意力的循环神经网络缺陷预测(DP-ARNN)模型来构建代码表征向量,该模型通过注意力机制对前后节点的关联信息进行有效利用。VIET等[18]通过编译程序代码获得的汇编指令构造CFG,使用基于有向图的卷积神经网络(DGCNNs)构建代码表征向量。上述方法根据深度优先遍历获得AST节点,并且仅选取特定类型的节点。这可能使不相关的节点被视为有关联的节点,导致不正确的缺陷预测。因此,在本文方法中将使用轻重链来区分节点在树中的位置,并仅以轻重程度作为条件来选择需要的节点。

2 基于重子节点的抽象语法树缺陷预测

本文提出的HSAST方法首先将AST中的每个节点进行标记,在节点数超过预设的代码描述向量组时,根据标记的优先级选择被舍弃的节点,然后使用深度学习方法将向量组转换为代码模块的表征向量,以此进行精确的软件缺陷预测。图1给出了HSAST的总体框架。首先将程序模块的源代码解析为AST;然后根据子树的大小对节点进行标记,根据节点标记和节点包含的信息从AST中获取节点序列;之后通过字典映射和词嵌入方法将节点序列编

码为多维向量,这些向量会作为深度神经网络的输入,通过深度神经网络完成代码模块的结构和语义学习;最后将学习获得的代码模块表征向量输入到逻辑回归分类器中进行分类完成缺陷预测。

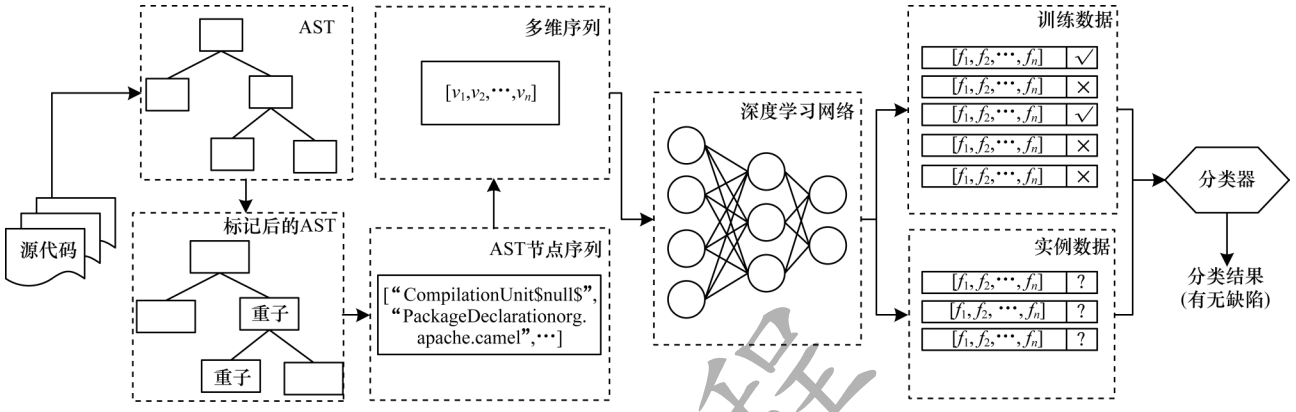

图1 基于重子节点抽象语法树的缺陷预测框架

Fig.1 Defect prediction framework via heavy son node-based abstract syntax tree

2.1 源代码解析

神经网络模型无法将完整的文本信息直接转换成描述文本信息的向量,需要合适的方法将计算机代码先转换成神经网络模型能够运用的实数向量数据,再通过深度学习网络进行学习。根据已有研究,研究人员使用AST来表示一段程序,这种数据结构能够很好地反映代码模块的结构和语义信息。在本文实验中,将使用一个名为javalang的开源Python包来完成代码模块的解析任务,通过https://github.com/c2nes/javalang获取,能够将Java源代码解析成对应的AST。

根据DP-CNN方法[11]和javalang的源代码文件tree.py,有3种类型的AST节点含有代码模块的语义信息,具体为:1)方法调用的节点;2)声明的节点,包括方法声明、构造函数声明和类声明等;3)引用的包、参数定义等。其他结构信息主要包括控制流节点(分支、循环、异常引发、捕获等)以及类的起始与终止、声明的起始等表示代码模块结构的信息。

首先,对于AST中不含代码语义信息的节点,本文选择一段特殊的字符串$null$作为第一部分表示该节点不包含文本信息,并记录描述节点类型的字符串作为描述节点值的第二部分。然后,对于AST中包含代码语义信息的节点,提取节点的名称或节点对应的值作为第一部分,记录描述节点类型的字符串作为第二部分,结合两者对节点进行描述。最后,按照规则将每个代码模块的AST转换为向量,此时向量是AST标记字符串的组合,无法直接作为深度学习网络的输入。本文在标记字符串与整数之间建立映射字典表,先计算每个标记的出现频率,再根据标记频率进行排序,使出现频率高的标记位于字典表前端。在完成映射后,为避免向量过于稀疏,选择合适的向量长度。对于小于指定长度的向量,将通过填充0来增加长度。对于长度大于指定长度的向量,将通过基于重子节点的标记按规则舍去部分长度,直到向量长度与指定长度相等。

2.2 基于重子节点的AST编码和类不平衡处理

HSAST方法采用树链剖分思路将树分割成多个部分,具体定义如下:

定义1 重子节点为在所有子节点中以该节点为根的子树规模(即树的节点数目)最大的节点。以重子节点为根的子树为重子树。

定义2 轻子(Light Son,LS)节点为除了重子节点外的其他子节点。以轻子节点为根的子树为轻子树。

算法1描述了将AST分割成重子节点和轻子节点的过程。输入是AST的节点$node_i$和节点对应的下标$next_i$。输出是以节点为根的子树大小$size_i$和该节点的重子节点son_i。

算法1 重子节点分割算法

输入 $node_i, next_i$

输出 $size_i, son_i$

1.initialize:Set $size_i = 1$
2.for index in $next_i$ do
3.MarkAndCountSize $node_{index}$
4.Set $size_i = size_i + size_{index}$
5.if the heavy son of $node_i$ is unmark then
6.Set son_i = index
7.else if $size_{index} > size_{son_i}$ then
8.Set son_i = index
9.end if
10.end for
11.return

在将代码模块解析成AST的过程中,包含源代码中更多文本信息和控制流信息的部分在AST中形成的子树更大,同时这一部分更有可能形成缺陷。若舍去代码模块中出现频率低的标记字符串,则有可能会舍去重子树中具有特异点的标记。因此,保

留具有更多信息的重子树,从信息较少的轻子树开始舍去能提高模型预测性能。

算法2描述了重子树的选择过程。输入是AST的节点$node_i$和节点对应的下标$next_i$、节点的重子节点son_i和序列化向量的最大长度MaxSize。输出是序列化后的向量V。采用嵌入到网络中的词作为可训练的单词字典,将标记转换为高维向量。

算法2 重子数选择算法

输入 $node_i$, $next_i$, son_i, MaxSize

输出 Serialized Vector(V)

1.if len(V) ≥ MaxSize then
2.return
3.end if
4.Adding $node_i$ into V
5.ProcessingSubtree of $node_{son_i}$
6.for index in $next_i$ then
7.if index ≠ son_i then
8.ProcessingSubtree of $node_{index}$
9.end if
10.end for
11.return

在一个软件项目的正常生命周期中,无缺陷的模块总是多于有缺陷的模块,因此在预测模型数据集时,无缺陷样本往往多于有缺陷样本。如果直接使用未处理过的源数据集进行训练,则结果将偏向多数类,即更容易将目标软件模块视为无缺陷的样本。在类不平衡问题上,研究人员进行了大量研究[19-20]并取得了一定的研究成果。为进行更有效的对比,本文使用过采样方法[12]处理类不平衡问题,从有缺陷样本中随机复制训练样本,生成类平衡的训练数据集。

2.3 基于深度学习模型的软件缺陷预测

本文采用3种深度学习模型提取代码模块中的结构和语义信息:1)循环神经网络(Recurrent Neural Network,RNN)模型,利用双向长短期记忆(Bi-directional Long Short-Term Memory,Bi-LSTM)网络对代码模块的信息进行学习,能够获取源代码的上下文信息;2)基于注意力机制的循环神经网络(Attention-based RNN,ARNN)模型,在Bi-LSTM的基础上加入注意力层,能够突出对预测结果更具影响的关键信息;3)CNN模型,对程序代码模块信息先进行局部感知,再在更高层次对局部进行综合操作,从而得到全局信息。

软件缺陷的预测主要分为4个部分:1)词嵌入层,将处理后的AST数字标记向量转换成固定长度的高维实数向量;2)信息抓取,通过深度学习模型获取输入序列中的关键信息;3)代码表征,将提取到的特征进行综合形成表示代码模块特征的向量;4)输出层,根据表征向量输出缺陷预测结果。

3 实验验证

通过实验对HSAST方法进行有效性验证,从多个角度分析HSAST方法的性能优劣。介绍实验过程中采用的数据集、性能评价指标以及实验流程和参数设定,选择Keras、Tensorflow以及Sklearn构建深度学习模型,运行于配置为Windows 10操作系统、i7-6700HQ 2.60 GHz处理器、16 GB内存的计算机上。

3.1 研究问题

HSAST方法需在具有一定历史标记数据的目标项目中完成预测任务,从源代码中提取代码模块的结构和语义信息,利用AST中的节点类型和值信息来构建代码表征向量。为验证HSAST方法的有效性,提出以下2个研究问题:

1)RQ1。将HSAST方法与采用DFS[12]搜索并根据字典频率舍去节点的处理方法(简称为DFS方法)进行性能分析与对比。

2)RQ2。将不同深度学习模型下的HSAST方法性能进行对比与分析。

3.2 实验数据

采用Apache开源项目中的5个开源Java项目,每个项目都包含两个版本。Java项目数据集是专门用于软件工程研究的公开数据集,被大量研究人员认可和使用,提供了每个文件的类名、静态度量元以及缺陷标签。使用公开数据集可确保实验所用项目的历史数据标记是被广泛认可的。表1给出了Java项目数据集信息。使用每个项目中版本号靠前的版本作为训练数据集,版本号靠后的版本作为测试数据集。在使用训练数据集进行模型训练时,随机选择其中的70%作为训练集,剩下的30%作为验证集。

表1 Java项目数据集信息

Table 1 Dataset information of Java project

项目	描述	版本(前,后)	平均文件数	平均缺陷率/%
Camel	Enterprise integration framework	1.4,1.6	789	52.4
Lucene	Text search engine library	2.0,2.2	217	56.7
Xalan	A library for transforming XML files	2.5,2.6	829	50.0
Synapse	Data transport adapters	1.1,1.2	290	41.8
Poi	Java library to access Microsoft format files	2.5,3.0	455	57.9

3.3 评测指标

在进行样本预测后，根据样本有无缺陷的实际情况和预测结果可以产生4种预测结果：实际有缺陷的样本被预测为有缺陷，即T_{TP}；实际无缺陷的样本被预测为无缺陷，即T_{TN}；实际无缺陷的样本被预测为有缺陷，即F_{FP}；实际有缺陷的样本被预测为无缺陷，即F_{FN}。基于这4种预测结果，得到查准率(P)和查全率(R)这两个基础的评测指标。

查准率也称为正确率，指预测结果为有缺陷样本中，实际有缺陷样本的比例，计算公式如下：

$$P=T_{TP}/(T_{TP}+F_{FP}) \tag{1}$$

查全率也称为召回率，指所有实际有缺陷的样本中，被预测为有缺陷样本的比例，计算公式如下：

$$R=T_{TP}/(T_{TP}+F_{FN}) \tag{2}$$

单独通过一个指标对模型进行测评是不够准确的，比如模型将所有测试样例都预测为有缺陷样本，那么该模型能够得到100%的查全率，但是其查准率较低。为更综合地评价HSAST方法，将F1值(F)和AUC值也作为评测指标。F1值是查准率与查全率的调和平均值，计算公式如下：

$$F=\frac{2\times P\times R}{P+R} \tag{3}$$

AUC被定义为ROC曲线下与坐标轴围成的面积，该面积的数值不会大于1。AUC值越接近1，检测方法的真实性越高。

3.4 实验结果

在以下3种深度学习模型的基础上利用HSAST和DFS方法进行缺陷预测：

1)RNN。使用基于LSTM单元的双向循环神经网络来捕获和提取代码模块中的结构和语义信息后，进行软件缺陷预测。

2)ARNN。在Bi-LSTM的基础上加入注意力层提取对预测结果更具影响的关键信息后，进行软件缺陷预测。

3)CNN。通过文本序列卷积的方式提取代码模块中局部和全局信息后，进行软件缺陷预测。

采用HSAST和DFS方法生成模型输入，使用相同参数进行模型训练。具体而言，CNN设置10个长度为5的过滤器，全连接层的隐藏节点数为100。RNN与ARNN设置LSTM节点数为16、第一层隐藏节点数为16、第二层隐藏节点数为8。每个样本形成的向量以固定长度(2 000)输入到模型中，所有模型的词嵌入层将数字向量转换成维度为30的高维向量。表2和表3给出了模型在5个项目下的运行结果，显示了同一项目中两种方法在相同深度学习网络下的比较统计数据，其中，粗体显示了每个项目的最佳预测结果，最后一行显示了两种方法对5个项目预测结果的平均值。表4给出了HSAST与DFS方法在5个项目下的时间成本对比结果。

表2 基于HSAST和DFS方法的深度学习模型F1值比较

Table 2 Comparison of F1 values of deep learning models based on HSAST and DFS methods

项目	HSAST-ARNN	DSF-ARNN	HSAST-RNN	DFS-RNN	HSAST-CNN	DFS-CNN
Camel	0.531	0.506	0.517	0.494	**0.554**	0.519
Lucene	0.700	0.635	0.720	0.700	**0.748**	0.676
Xalan	0.609	**0.632**	0.616	0.624	0.604	0.619
Synapse	0.503	0.436	0.519	0.477	**0.541**	0.484
Poi	0.741	0.736	0.726	0.746	0.728	**0.758**
Average	0.617	0.589	0.619	0.608	**0.635**	0.611

表3 基于HSAST和DFS方法的深度学习模型AUC值比较

Table 3 Comparison of AUC values of deep learning models based on HSAST and DFS methods

项目	HSAST-ARNN	DSF-ARNN	HSAST-RNN	DFS-RNN	HSAST-CNN	DFS-CNN
Camel	**0.677**	0.614	0.629	0.612	0.669	0.632
Lucene	0.675	0.664	0.661	0.673	**0.679**	0.677
Xalan	0.711	0.688	0.686	0.712	0.731	**0.767**
Synapse	**0.729**	0.689	0.706	0.632	0.722	0.693
Poi	**0.773**	0.719	0.712	0.760	0.672	0.704
Average	**0.713**	0.675	0.679	0.678	0.695	0.695

表4 HSAST与DFS方法的时间成本比较

Table 4 Comparison of time cost between HSAST and DFS methods s

项目	HSAST方法	DSF方法
Camel	40.0	821.0
Lucene	15.0	1 404.0
Xalan	32.0	2 358.0
Synapse	14.0	767.0
Poi	72.0	4 242.0
Average	34.6	1 918.4

3.5 实验结果分析

3.5.1 针对RQ1的结果分析

利用3种深度学习模型(CNN、RNN和ARNN)进行分析与比较，证明了HSAST方法优于DFS方法不是使用特定深度学习模型出现的偶然情况。使用表1中列出的这些项目进行实验，每个项目采用两个版本，版本靠前的作为训练数据集用于训练预测模型，版本靠后的作为测试数据集用于评估模型性能。

表2列出了分别采用HSAST和DFS方法的3种深度学习模型在每个项目上的F1值。以Camel为例,采用HSAST方法处理AST,ARNN、RNN和CNN分别取得的F1值为0.531、0.517和0.554;采用DFS处理AST时,ARNN、RNN和CNN分别了取得的F1值为0.506、0.494和0.519。显然,在3个深度学习模型上,HSAST方法均比DFS方法表现更好。由表2的最后一行可以看出,HSAST方法在F1平均值方面也比DFS方法表现更好,在ARNN上平均提升约3%。这些结果表明了基于HSAST方法的深度学习缺陷预测模型具有更好的预测性能。

表3列出了每个项目的AUC值。HSAST方法同样在大部分项目上比DFS方法表现更好。总体而言,HSAST方法在使用ARNN深度学习模型时具有最佳性能,而基于其他深度学习模型时,与DFS方法具有相近的性能。由表3的最后一行可以看出,HSAST方法在AUC平均值方面也比DFS方法表现更好,在ARNN上平均提升约4%。

从表4可以看出,HSAST方法比DFS方法平均多花费了55倍的时间成本,主要在Xalan和Poi项目上花费了更多的时间成本。HSAST方法相比DFS方法在语法树结构重构以及节点所在树的位置识别上花费了更多的时间成本,该成本受树的大小即项目源代码的文件大小影响。

实验模型的主要差别在于通过深度学习方法提取代码模块信息前的数据处理阶段。HSAST方法对AST的所有节点进行处理,并优先保留可能具有更多特征信息的重子树,但HSAST方法也保留了更多的冗余信息,这些信息会干扰分类器的分类结果,在实验结果上表现为:在个别项目上DFS方法具有比HSAST方法更优的性能。基于上文分析得出以下结论:采用深度学习模型捕获代码模块的隐藏信息,再进行项目内软件缺陷预测时,HSAST方法优于DFS方法。

3.5.2 针对RQ2的结果分析

为分析HSAST对不同深度学习模型的影响程度,采用ARNN、RNN和CNN深度学习模型来捕获和提取代码模块中的信息。深度学习网络采用同一个数据处理结果作为输入,并转换成相同维度的高维向量进行特征学习。

从表2和表3可以看出,CNN深度学习模型在F1指标下性能最佳,ARNN深度学习模型在AUC指标下性能最佳。这些结果说明了在需要综合考虑查准率和查全率的场景下,提取节点序列局部特征的CNN深度学习模型具有更好的性能,而相对实际应用场景,捕获上下文信息的ARNN深度学习模型具有更好的性能,因为ARNN深度学习模型具有略低于CNN深度学习模型的F1指标结果和略高于CNN深度学习模型的AUC指标结果。

4 结束语

本文提出基于重子节点的抽象语法树软件缺陷预测方法,利用AST中的语义和结构信息,通过字典映射和词嵌入方法将程序源代码的AST转换为高维数字向量,并将结果输入ARNN和CNN深度学习模型中提取代码表征向量。实验结果表明,该方法保留了更多代码模块中的结构和语义信息,并通过深度学习模型提取关键特征,提高了缺陷预测性能。后续将设计基于AST的树形结构,拼接代码模块静态度量元,进一步提高软件缺陷预测性能。

参考文献

[1] 曾福萍,靳慧亮,陆民燕. 软件缺陷模式的研究[J]. 计算机科学,2011,38(2):127-130.
ZENG F P,JIN H L,LU M Y. Study on software defect patterns[J]. Computer Science,2011,38(2):127-130.(in Chinese)

[2] CATAL C,DIRI B N. A systematic review of software fault prediction studies[J]. Expert Systems with Applications,2009,36(4):7346-7354.

[3] 刘文杰,江贺. 基于特征选择的软件缺陷报告严重性评估[J]. 计算机工程,2019,45(8):80-85.
LIU W J,JIANG H. Severity assessment of software defect reports based on feature selection[J]. Computer Engineering,2019,45(8):80-85.(in Chinese)

[4] JING X Y,WU F,DONG X W,et al. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems[J]. IEEE Transactions on Software Engineering,2017,43(4):321-339.

[5] ZHAO L C,SHANG Z W,ZHAO L,et al. Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks[J]. Neurocomputing,2019,352:64-74.

[6] 倪超,陈翔,刘望舒,等. 基于特征迁移和实例迁移的跨项目缺陷预测方法[J]. 软件学报,2019,30(5):1308-1329.
NI C,CHEN X,LIU W S,et al. Cross-project defect prediction method based on feature transfer and instance transfer[J]. Journal of Software,2019,30(5):1308-1329.(in Chinese)

[7] LI Z Q,JING X Y,ZHU X K,et al. Heterogeneous defect prediction with two-stage ensemble learning[J]. Automated Software Engineering,2019,26(3):599-651.

[8] JIANG K Y,ZHANG Y T,WU H B,et al. Heterogeneous defect prediction based on transfer learning to handle extreme imbalance[J]. Applied Sciences,2020,10(1):396.

[9] HALSTEAD M H. Elements of software science(operating and programming systems series)[M].[S. l.]:Elsevier Science Inc.,1977.

[10] MCCABE T J. A complexity measure[J]. IEEE Transactions on Software Engineering,1976,SE-2(4):308-320.

[11] LI J,HE P J,ZHU J M,et al. Software defect prediction via convolutional neural network[C]//Proceedings of 2017 IEEE International Conference on Software Quality,Reliability and Security. Washington D. C.,USA:IEEE Press,2017:318-328.

(下转第248页)

（上接第235页）

[12] FAN G S, DIAO X Y, YU H Q, et al. Software defect prediction via attention-based recurrent neural network[EB/OL]. [2020-11-14]. https://www. semanticscholar. org/paper/Software-Defect-Prediction-via-Convolutional-Neural-Li-He/95badade580eeff6a2f512268a017b9b66c6a212? p2df.

[13] POWERS D. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation[J]. Journal of Machine Learning Technologies, 2007, 2: 37-63.

[14] LOBO J M, JIMÉNEZ-VALVERDE A, REAL R. AUC: a misleading measure of the performance of predictive distribution models[J]. Global Ecology and Biogeography, 2008, 17(2): 145-151.

[15] RYU D, JANG J I, BAIK J. A transfer cost-sensitive boosting approach for cross-project defect prediction[J]. Software Quality Journal, 2017, 25(1): 235-272.

[16] BENNIN K E, KEUNG J, PHANNACHITTA P, et al. MAHAKIL: diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction[J]. IEEE Transactions on Software Engineering, 2018, 44(6): 534-550.

[17] PAN C, LU M Y, XU B, et al. An improved CNN model for within-project software defect prediction [J]. Applied Sciences, 2019, 9(10): 2138.

[18] PHAN A V, LE NGUYEN M, BUI L T. Convolutional neural networks over control flow graphs for software defect prediction [C]//Proceedings of the 29th International Conference on Tools with Artificial Intelligence. Washington D. C. , USA: IEEE Press, 2017: 45-52.

[19] GAO Y, YANG C H. Software defect prediction based on manifold learning in subspace selection[C]//Proceedings of 2016 International Conference on Intelligent Information Processing. New York, USA: ACM Press, 2016: 1-6.

[20] WEI H, HU C Z, CHEN S Y, et al. Establishing a software defect prediction model via effective dimension reduction[J]. Information Sciences, 2019, 477: 399-409.

编辑　陆燕菲