计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(1):29−37 [doi: 10.15888/j.cnki.csa.007726]

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

基于深度学习的软件缺陷预测模型①

陈　凯, 邵培南

(中国电子科技集团第三十二研究所, 上海 201808)
通讯作者: 陈　凯, E-mail: d-chenkai@qq.com

摘　要: 为了提高软件的可靠性, 软件缺陷预测已经成为软件工程领域中一个重要的研究方向. 传统的软件缺陷预测方法主要是设计静态代码度量, 并用机器学习分类器来预测代码的缺陷概率. 但是, 静态代码度量未能充分考虑到潜藏在代码中的语义特征. 根据这种状况, 本文提出了一种基于深度卷积神经网络的软件缺陷预测模型. 首先, 从源代码的抽象语法树中选择合适的结点提取表征向量, 并构建字典将其映射为整数向量以方便输入到卷积神经网络. 然后, 基于 GoogLeNet 设计卷积神经网络, 利用卷积神经网络的深度挖掘数据的能力, 充分挖掘出特征中的语法语义特征. 另外, 模型使用了随机过采样的方法来处理数据分类不均衡问题, 并在网络中使用丢弃法来防止模型过拟合. 最后, 用 Promise 上的历史工程数据来测试模型, 并以 AUC 和 F1-measure 为指标与其他 3 种方法进行了比较, 实验结果显示本文提出的模型在软件缺陷预测性能上得到了一定的提升.

关键词: 软件缺陷预测; 抽象语法树; 卷积神经网络; 随机过采样; 丢弃法

引用格式: 陈凯,邵培南.基于深度学习的软件缺陷预测模型.计算机系统应用,2021,30(1):29–37. http://www.c-s-a.org.cn/1003-3254/7726.html

Software Defect Prediction Model Based on Deep Learning

CHEN Kai, SHAO Pei-Nan

(The 32nd Research Institute of China Electronics Technology Group Corporation, Shanghai 201808, China)

Abstract: In order to improve the reliability of software, software defect prediction has become an important research direction in the field of software engineering. Traditional software defect prediction methods mainly design static code metrics and use machine learning classifiers to predict the defect probability of the code. However, the static code metrics do not fully consider the semantic features hidden in the code. According to this situation, this study proposes a software defect prediction model based on convolutional neural network. First, extract the characterization vectors from the appropriate nodes in the abstract syntax tree of the source code, and construct a dictionary to map them to integer vectors to facilitate input to the convolutional neural network. Then, a convolutional neural network is designed based on GoogLeNet, and the ability of the convolutional neural network to deeply mine data is used to fully mine the grammatical and semantic features of the features. In addition, this model uses the method of random oversampling to deal with the imbalance of data, and uses the method dropout in the network to prevent the model from overfitting. Finally, the historical engineering database on Promise is used to test the model, and AUC and F1-measure are used as indicators to compare with the other three methods. The results show that the proposed model has a certain improvement in software defect prediction performance.

Key words: software defect prediction; abstract syntax tree; convolutional neural network; random oversampling; dropout

① 收稿时间: 2020-05-19; 修改时间: 2020-06-16; 采用时间: 2020-06-23; csa 在线出版时间: 2020-12-31

1 引言

随着现代软件的不断发展, 软件可靠性已经成为评价软件的关键因素. 软件规模的不断扩展和功能的日益增强, 软件复杂度不断上升, 软件缺陷出现的概率也不断上升, 从而导致软件的失败. 为了帮助开发人员和测试人员及时找到软件缺陷, 软件缺陷预测已经成为软件工程领域、数据挖掘领域的研究方向之一. 软件缺陷预测技术可以在一定程度上预测软件中是否存在着缺陷, 以此帮助相关团队快速了解软件的整体情况和质量, 制定相关策略测试和改善软件, 提高软件的可靠性和稳定性.

基于此, 许多研究人员前赴后继潜心研究软件缺陷预测技术并尝试通过机器学习的方法来检测软件中是否存在着缺陷. 传统软件缺陷预测[1] 是以手工获取软件度量特征的基础进行分类学习, 而特征选择的方法直接影响软件缺陷预测的准确性和稳定性. 而在以往的软件缺陷预测研究中, 通常使用静态软件度量作为代码的特征, 静态软件度量[1] 主要包括以下几种方法:

(1) 代码度量

代码度量是最直接、应用最普遍的度量方式. 通过对源代码相关指标的简单计数得到度量值. 度量包括总行数、代码行数目、注释行数目、空白行数目和代码及注释行总数目等, 通过对总行数、代码行数、注释行数等不同的处理方式, 度量结果就会不同.

(2) McCabe 度量

McCabe 度量是一种基于程序流程图的复杂性度量方法, 度量的是程序的复杂性, 主要包括圈复杂度、基本复杂度、设计复杂度等.

(3) Halstead 度量

Halstead 度量考虑了程序中出现的操作数和运算符, 具体有程序长度、操作符出现的总数量、操作数出现的总数量、程序容量、程序难度、程序级别、程序工作量等.

(4) CK 度量

CK 度量是面向对象程序的度量, 具体包括类方法复杂度带权和 (WMC)、类在继承树中的最大深度 (DIT)、继承树中类的直接子类个数 (NOC) 等.

根据代码的实际情况, 选择合适的度量方法, 或在各种度量方法中选择合适的指标组成新的特征集合, 然后根据从历史软件源码中提取出来的特征构建如逻辑回归、随机森林、支持向量机等分类器, 对新版本的软件源码进行软件缺陷预测, 以此来帮助编程人员找到可能包含缺陷的部分.

然而, 传统软件缺陷预测方法使用静态代码度量作为特征, 未能充分考虑潜藏在代码中的语义特征, 这无疑会对缺陷预测造成影响. 而抽象语法树能够表达出源代码的语义, 已经有相关的论文[2] 证实了其可以用于源码的完整性和缺陷的检测. 抽象语法树是基于源代码采用树状结构来描述代码上下文之间的关系, 其中包含了程序模块的语法结构和语义信息. 从抽象语法树中提取表征用于软件缺陷预测, 可以充分考虑到代码的语法语义特征.

近年来, 深度学习作为数据挖掘的技术之一得到了充足的研究. 在软件缺陷预测领域, 深度学习同样可以用于挖掘代码中隐含的特征. Iqbal 等[3] 介绍了用静态度量的方法获得特征, 然后用 4 种方法对特征进行投票, 选取出最合适的特征, 然后构建一个多层感知机 (MLP) 网络来对样本进行学习分类. Wang 等[2] 介绍了用多层受限玻尔兹曼机叠加而成的深度置信网络 (DBN), 自动提取源代码中的语法语义特征, 并用提取出来的特征构建软件缺陷预测模型. Li 等[4] 利用卷积神经网络 (CNN) 提取源码特征后, 将其与传统静态特征进行连结, 构建逻辑回归分类器来对软件缺陷进行预测. 然而, 这些方法依然存在着数据挖掘不足的问题, 所使用的网络大多是简单模型, CNN 也仅使用单层卷积层. 基于这种情况, 本文以抽象语法树为特征来源, 提出了一种卷积神经网路模型来进行软件缺陷预测, 并利用 Promise 官网上的历史工程数据来对模型进行实验, 取得了较好的结果.

2 基于抽象语法树的代码表征

能否从源代码中提取到合适的特征, 是影响软件缺陷预测性能的一个关键因素. 在过去的研究中, 常常用静态软件度量的方法来处理源代码, 忽视了潜藏在代码中的语义特征. 而本文使用了一种基于抽象语法树的方法来获取代码表征.

2.1 抽象语法树

抽象语法树 (Abstract Syntax Tree, AST) 是源代码关于抽象语法结构的树状表示, 源代码中的每一种结构都表征为树上的结点. 之所以说是抽象的, 是因为 AST 并不会将源代码的细节表示出来, 例如, 一串 For 语句, 在结点中就记录为“ForStatement”以及一些关键

要素, 而具体循环的内容并不会被记录下来. 另外, 抽象语法树并不依赖源代码语言的语法, 也就是说, 语法分析时所采用的是上下文无关文法, 因为在写文法时, 通常会对文法进行等价的转换 (消除左递归, 二义性, 回溯等), 这样会在文法分析时引入一些冗余成分, 对后续阶段造成不好的影响.

抽象语法树能有效保存代码的语法结构和语义信息. 如图 1 所示, 代码 a 和代码 b 十分相似, 且有着几乎一样的静态代码度量, 也就是说, 他们有着几乎一样的特征, 在特征空间中, 它们的距离会非常小, 用分类器分类的话, 很有可能将两份代码归为一类, 但显然代码 a 是没有缺陷的, 而代码 b 是有缺陷的, 这就造成了错误的检测结果. 图 2 展示两份代码的抽象语法树, 代码 a 的抽象语法树比代码 b 的抽象算法树多了两个结点, 在提取表征向量时, 两份代码在序列上就会有一定的区别, 而这种区别就可以方便模型区分两种代码, 得到更好的软件缺陷预测性能.

```
Package com;
Public class A{
    public static void main(String[]args){
        int i=0;
        while(i<10){
            i++;
        }
        System.out.println(i);
    }
}
```

(a) 代码 a

```
Package com;
Public class B{
    public static void main(String[]args){
        int i=0;
        while(i<10){

        }
        System.out.println(i);
    }
}
```

(b) 代码

图 1 抽象语法树示例代码

2.2 提取表征向量

本文使用了一款开源的 Python 依赖包 Javalang 来对源代码进行解析, 它提供了一个基于 Java 语言规范的词法分析器和解析器, 可以构造 Java 源代码的抽象语法树. 在得到源代码的抽象语法树后, 按照深度优先的顺序来遍历 AST 的所有节点, 然后主要选择以下 3 类结点作为抽象语法树的表征向量元素:

(1) 表示为方法调用的结点

(2) 表示为声明的结点, 包括方法声明、类声明、接口声明等

(3) 控制流结点, 譬如说条件分支、循环控制等

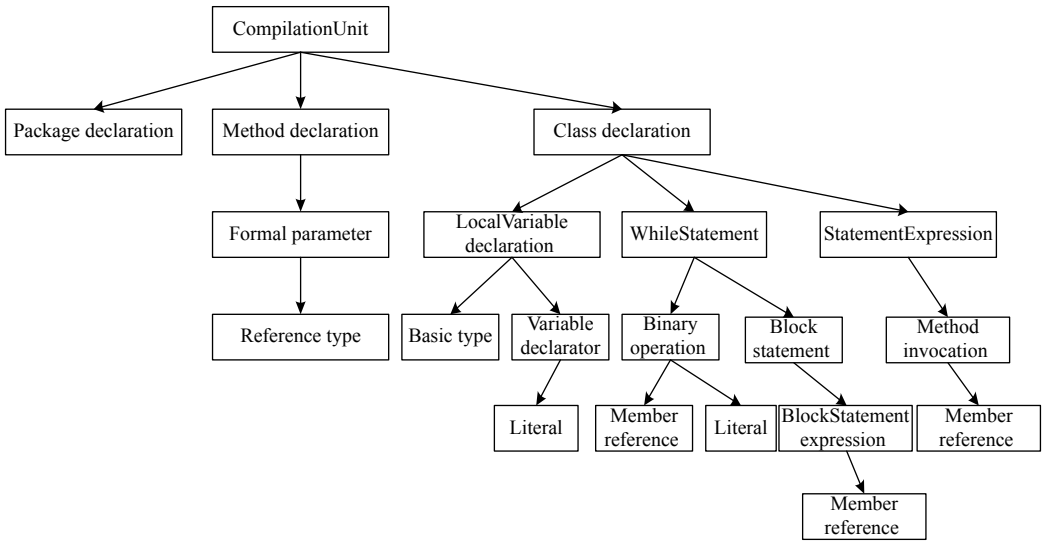
根据结点的类型, 我们选取不同的要素来作为结点的特征. 对于第 1 类结点, 我们使用结点对应的方法名来作为结点在特征向量中的标识; 第 2 类结点选择结点的名称来作为标识; 第 3 类控制流结点选择节点的类型来作为标识, 比如表征条件分支的结点, 记录结点的类型名“IfStatement”作为该结点的特征. 表 1 列出了本文所使用的所有的结点类型.

由此, 我们可以得到一棵树即一份代码的表征向量.

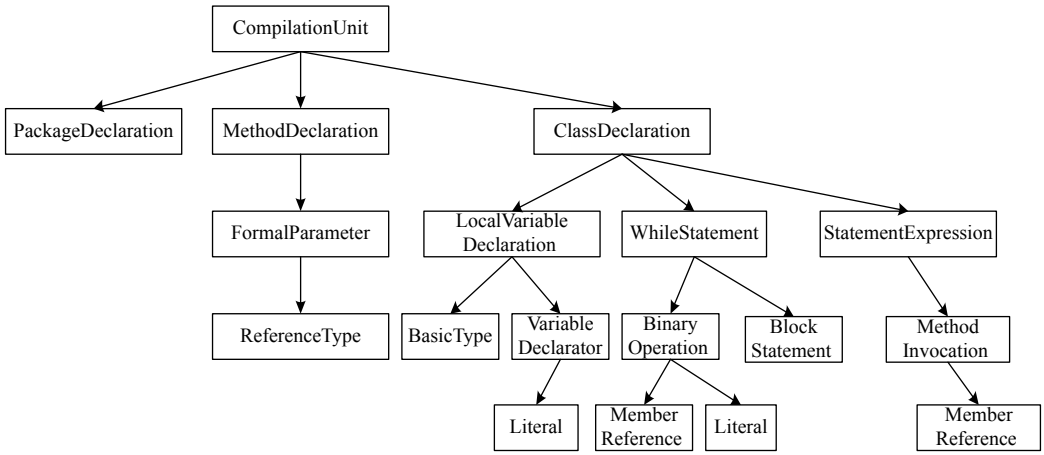
2.3 整数向量映射

从抽象语法树中获得的表征向量不能直接用于神经网络, 训练神经网络需要输入整数向量, 所以我们需要在获得的特征和整数之间建立一个映射, 将表征向量转换为整数向量.

为了在获得的特征和整数之间建立映射, 我们建立一个字典[2] 将表征和正整数一一对应起来. 将训练样本和测试样本中的代码全部提取为表征向量后, 统计各个特征的频率, 将其转换为对应的整数. 假设不同的特征的数量是 m, 每个特征都有着对应的整数, 那么正整数的范围也是 1～m. 具体的, 在从训练样本和测试样本中提取表征向量后, 首先, 计算各个特征在所有样本中出现的频数, 并且根据频数将它们排列; 然后, 为排列好的特征建立一个序列字典, 频数高的特征排在前面, 这意味着出现频率越高的特征对应的正整数越小; 构建完字典后, 就可以将之前的表征向量转化为整数向量. 但因为神经网络的输入要求有固定的长度, 为了避免向量过于稀疏, 选择适当的向量长度对向量进行处理. 如果一个向量的长度小于设定的长度, 那么我们就在向量末尾添 0, 而 0 在神经网络的计算过程中没有意义; 如果一个向量的长度大于设定的长度, 那就在向量中寻找最大的正整数将它删去, 因为最大的整数对应的是频数最小的特征, 循环往复, 直到向量的长度符合设定的长度. 由此, 我们得到了每份源代码对应的整数向量. 图 3 给出了从源代码到整数向量的全部流程.



(a) 代码 a 的抽象算法树



(b) 代码 b 的抽象算法树

图 2 示例代码的抽象语法树

表 1 使用到的所有结点类型

类别	结点类型
方法调用结点	MethodInvocation, SuperMethodInvocation
声明结点	PackageDeclaration, InterfaceDeclaration, ClassDeclaration, MethodDeclaration, ConstructorDeclaration, VariableDeclaration
控制流结点	IfStatement,WhileStatement, DoStatement, ForStatement, SwitchStatement, AssertStatement, BreakStatement, ContinueStatement, ReturnStatement, ThrowStatement, TryStatement, SynchronizedStatement, BlockStatement, TryResource, CatchClause, EnhancedForControl
其他结点	FormalParameter, BasicType, CatchClauseParameter, MemberReference, SuperMemberReference, ReferenceType

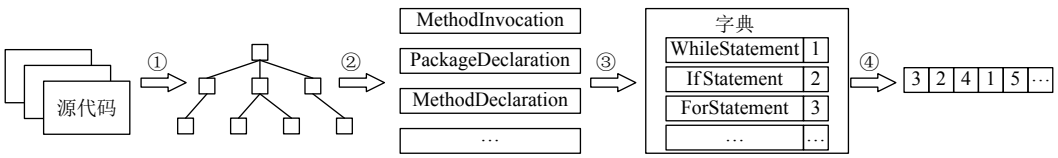


图 3 从源代码到得到整数向量的流程图 (①从源代码中解析出抽象语法树; ②从抽象语法树中提取表征向量; ③根据提取的特征构建字典; ④将表征向量映射为整数向量)

3 卷积神经网络设计与结构

卷积神经网络是一种含卷积计算且具有深度结构的前馈式神经网络, 具有表征学习、深度挖掘数据的能力. 卷积神经网络已经被证实在图像处理、语音识别、自然语言处理等领域有着不错的性能表现. 而且, 卷积神经网络的隐含层内的卷积核参数共享、层与层之间的稀疏连接使得卷积神经网络能够以较小的计算量对格点化特征.

3.1 数据预处理

在软件缺陷预测领域, 数据分类不均衡问题是普遍存在的, 如何处理这种不均衡问题也是具有挑战性的. 在获得的数据集中, 往往有缺陷的样本数是要少于没有缺陷的样本数, 如果直接用这样的样本对模型进行训练, 训练出来的模型往往会对样本数量较少的类别的识别能力较弱, 在软件缺陷预测时, 模型便会偏向没有缺陷的结果. 因此, 我们需要对数据进行预处理以弥补数据集分类不均衡带来的偏差.

一般而言, 针对数据集中存在的样本分类不均衡问题, 通常采用过采样或欠采样的方法来使得数据集中各类的样本数量达到均衡. 欠采样是从多数样本中随机选择和少数样本一样数量的样本, 以此构建分类平衡的数据样本, 但这样会大大减少数据的训练样本, 造成信息的损失, 会造成模型的欠拟合, 因此我们选择过采样的方法来处理分类不均衡问题. 由于 AST 数值向量并非是特征向量, 所以类似像 SMOTE[5] 等基于样本间的欧式距离来生成新样本的分类不均衡处理方法并不一定适用. 因此, 在这里, 我们采取简单的随机过采样的方法来对数据进行预处理, 在少数样本即有缺陷的样本中随机选择进行复制, 使得有缺陷的样本数量和没有缺陷的样本数量保持一致, 以此保证数据类别间的均衡. 具体的算法如算法 1 所示.

算法 1. 随机过采样算法

输入:分类不均衡的数据集 D

输出: 分类均衡的数据集 D'

(1) 初始化已复制样本集合 C 为空集.

(2) 将数据集 D 复制, 组成数据集 D'.

(3) 在数据集 D 中筛选出有缺陷的样本集 d.

(4) 在数据集 d 中随机选择样本 a, 如果 a 不在已复制样本集合 C 中, 将样本 a 加入数据集 D' 和已复制样本集合 C; 如果已复制样本集 C 的大小和 d 一样, 则清空 C.

(5) 重复步骤 (4) 直至数据集 D' 中无缺陷样本和有缺陷样本数量保持一致.

3.2 网络结构

首先介绍所要构建的网络中用到的基础卷积块 Inception 块[6], 这个卷积模块的结构如图 4 所示.

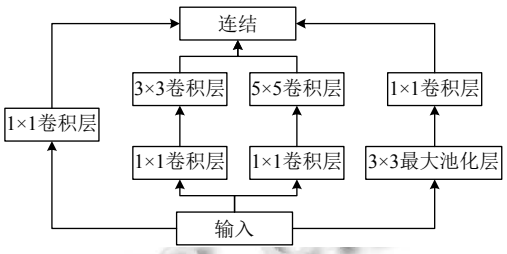


图 4 Inception 块

如图 4 所示, Inception 块含有 4 条并行的网络线路. 前三条线路所使用的卷积核大小分别是 1×1、3×3、5×5, 以用来抽取不同空间尺寸下的信息, 第 2, 3 层会先对输入做 1×1 卷积操作以减少输入通道数, 以降低模型复杂度. 第 4 条线路会先使用 3×3 最大池化层, 然后接一个 1×1 卷积层来改变通道数. 并且, 根据 4 条线路的具体情况, 设置合适的填充来使得输入和输出的高和宽一致. 最后将每条线路的输出在通道维上连结, 以此完成卷积的功能, 将结果输入到下一层. 相较于普通的卷积层, Inception 块因为使用了 3 个卷积层和 1 个池化层, 能够更深层次的挖掘数据中的特征, 以此帮助模型进行更好的学习分类.

为了能够更深层次地挖掘出潜藏在向量中的语法语义特征, 本文基于 GoogLeNet[7] 设计了一个卷积神经网络, GoogLeNet 最初设计出来是用来进行图像处理的, 在 ImageNet 图像识别挑战赛中大放异彩, GoogLeNet 串联了多个 Inception 块来对图像进行深度挖掘, 以此进行更好的分类. 本文基于 GoogLeNet 设计一个卷积神经网络, 具体的网络结构如图 5 所示, 除了最后的输出层使用 Sigmoid 函数作为激活函数外, 其他层的激活函数均使用 ReLU 函数, 并且根据实际情况调整网络中各个层的参数, 网络整体分为主要分为以下 3 个部分:

(1) 输入层: 输入层主要是一个嵌入层[8], 嵌入层的主要作用是将输入的整数向量中的整数元素转换成整数向量, 使得向量可以进行卷积操作. 嵌入层有两个重要参数: 嵌入层字典的大小 (num_embeddings) 和每个产出向量的大小 (embedding_dim). 这里, 本文将 num_embeddings 设置为 2.3 节构建的字典中所含有的

特征的数量, 将 embedding_dim 设置为 2.3 节中通过映射得到的整数向量的长度. 将长度为 n 的整数向量输入到嵌入层, 嵌入层将给出一个 $n \times n$ 的矩阵向量. 并且, 为了提高内存利用率和训练模型的速度, 本文选择分批量进行训练, 设置每次训练样本个数 (批尺寸, Batch_Size) 为 16, 即一次输入 16 个样本进行训练.

(2) 卷积部分: 卷积部分是网络的主体部分, 共由 5 个模块组成. 模块与模块之间使用步幅为 2 的 3×3 最大池化层来减小输出高度. 第 1 个模块包含 3 层的 3×3 卷积层; 第 2 个模块使用 2 个卷积层, 首先接一个 64 通道的 1×1 卷积层, 然后接了一个将通道数扩大 3 倍的 3×3 卷积层; 第 3 个模块串联了 2 个完整的 Inception 块; 第 4 模块串联了 5 个 Inception 块; 第 5 模块串联了 2 个 Inception 块. 通过多层的不同空间尺寸的卷积操作, 来深度挖掘数据中的特征, 从而进行性能更好稳定性更高的学习分类.

(3) 输出层: 输出层主要是根据之前卷积层输出的结果来输出分类结果. 首先使用一个全局平均池化层来将每个通道的高和宽都变成 1, 然后接上一个全连接层, 输出通道数为标签类别数, 最后, 连结一个 Sigmoid 函数构建逻辑回归分类器来计算程序代码的缺陷概率, 从而得到分类结果.

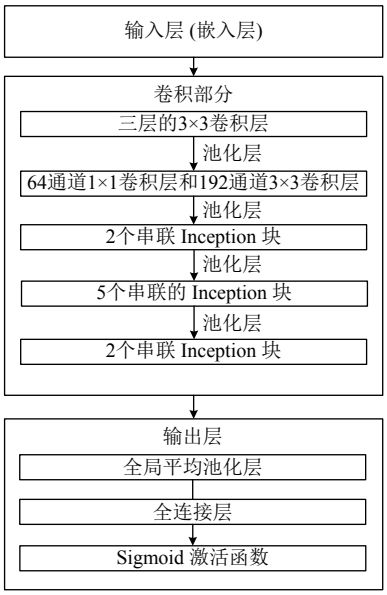


图 5 网络结构图

3.3 模型优化

之前, 我们在数据预处理时采用随机过采样的方法来解决数据分类不均衡问题, 提升了模型的泛化能力, 但是这样也有一定的过拟合的风险, 因此我们选择使用丢弃法 (Dropout)[9], 通过随机丢弃一部分神经元来避免过拟合. 在训练过程中, 随机丢弃一部分隐藏层神经单元, 即所有神经单元都有可能被清零, 这样就减少了神经元之间的依赖性, 输出层的计算也无法过度依赖任何一个隐含层神经元, 从而在训练模型时起到正则化的作用, 用来应对过拟合. 在测试模型时, 为了拿到更加准确的结果, 我们不使用丢弃法.

另外, 在训练模型的过程中, 为了得到最优的模型参数, 我们需要根据损失函数的梯度不断地对参数进行迭代, 这里我们选择使用 Adam[10] 优化器来更新参数. Adam 算法结合了 AdaGrad 和 RMSProp 两种优化算法的优点[10], 能够自动调整学习率, 并且参数的更新不受梯度的伸缩变换影响. Adam 算法能够从梯度均值及梯度平方两个角度进行自适应地调节, 综合考虑梯度的一阶矩估计和二阶矩估计计算出更新步长, 而不是直接由当前梯度决定.

4 实验与结果分析

4.1 数据集

为了评估训练出来的模型的性能, 本文从 Promise[11] 上下载了 5 个工程, 总共 11 个项目, 组成 6 组软件缺陷预测任务, 用于模型的测试. 在同一软件工程中, 将旧版本的工程项目作为训练集, 将新版本的工程项目作为测试集, 根据测试结果来评估模型的预测能力. 例如, 对于 Camel 工程, 我们将 Camel 1.4 版本的工程代码用来训练模型, 然后用 Camel 1.6 版本的代码用来测试模型. 表 2 列出了测试时所使用的软件项目的基本信息.

表 2 测试使用的工程信息

工程	版本	平均代码数	平均缺陷率(%)
Camel	1.4, 1.6	918	18.1
Lucene	2.0, 2.2	209	55.7
Synapse	1.0, 1.1, 1.2	212	25.5
Poi	2.5, 3.0	413	64.0
Xalan	2.6, 2.7	844	47.3

另外, 在数据集中, 每个项目不仅含有工程的源代码, 还统计了源代码的静态代码度量和缺陷注释, 度量方法主要是针对面向对象编程的静态代码度量, 具体的指标内容如表 3 所示. 这些指标可以用于其他的软件缺陷预测方法, 来和本文模型进行比较.

表 3 数据集中所使用的 20 个静态代码度量

简写	全称
LOC	Lines of code
DIT	Depth of inheritance tree
NOC	Number of children
RFC	Response for a class
CBO	Coupling between object classes
LCOM	Lack of cohesion in methods
LCOM3	Lack of cohesion in methods
NPM	Number of public methods
DAM	Data access metric
MOA	Measure of aggregation
MFA	Measure of function abstraction
IC	Inheritance coupling
CAM	Cohesion among methods of class
CBM	Coupling between methods
AMC	Average method complexity
Ca	Afferent couplings
Ce	Efferent couplings
Avg(CC)	Average McCabe
Max(CC)	Maximum McCabe
WMC	Weighted methods per class

4.2 评估指标

本文采用 AUC[12] 和 F1-measure[13] 这两个指标来评估模型性能, AUC 主要用来评估模型的识别缺陷的能力, 而 F1-measure 主要用来评估模型的稳定性.

在二分类的学习预测过程中, 根据分类结果可以将其分为 4 类: (1) 若一个实例为正类且被预测为正类, 即为真正类 (True Positive, *TP*); (2) 若一个实例为正类但被预测负类, 即为假负类 (False Negative, *FN*); (3) 若一个实例为负类但被预测为正类, 即为假正类 (False Positive, *FP*); (4) 若一个实例为负类且被预测为负类, 即为真负类 (True Negative, *TN*). 基于这 4 个数据, 可以得到击中概率 (*TPR*) 和虚报概率 (*FPR*), 其计算公式如式 (1) 和式 (2) 所示:

$$TPR = \frac{TP}{TP+FN} \tag{1}$$

$$FPR = \frac{FP}{TN+FP} \tag{2}$$

然后以 *FPR* 为横轴, *TPR* 为纵轴, 就可以绘制出 ROC 曲线, 而 ROC 曲线下的面积就是 AUC. 根据 AUC 的定义, 识别能力更好的模型对应着更高的 *TPR* 和更低的 *FPR*, 所以有 AUC 值越大的预测方法越好.

F1-measure 是精确率 (*P*) 和召回率 (*R*) 的调和平均, 其中精确率和召回率的计算公式如式 (3) 和式 (4) 所示:

$$P = \frac{TP}{TP+FP} \tag{3}$$

$$R = \frac{TP}{TP+FN} \tag{4}$$

通常情况下, 我们希望精确率越高越好, 召回率也越高越好, 但事实上这两个指标在某些情况下是矛盾的, 而 F1-measure 则综合考虑了这两个指标. F1-measure 的计算公式如式 (5) 所示.

$$F_1 = \frac{2 \cdot P \cdot R}{P+R} \tag{5}$$

另外, 用于评估软件缺陷预测模型的指标还有很多, 例如 MCC[14] 和 G-mean[15], MCC 考虑所有的有缺陷数据和无缺陷数据来综合评估预测结果和真实结果之间的关系, G-mean 是在数据不平衡时十分有参考价值的一个指标, 但因为 AUC 和 F1-measure 综合评估了模型的准确性和稳定性, 具有广泛的代表意义.

4.3 实验方法

为了能够正确估计模型对于软件缺陷预测的性能, 将本文提出的模型与以下 3 种方法进行比较:

(1) 静态代码度量+逻辑回归 (LR): 以数据集中提供的 20 个静态代码度量作为代码特征, 并用逻辑回归的方法进行分类

(2) 深度置信网络+逻辑回归 (DBN)[2]: 使用深度置信网络从源代码中提取特征, 然后使用逻辑回归的方法进行分类

(3) 卷积神经网络+逻辑回归 (CNN)[4]: 利用单层卷积神经网络对源代码进行特征提取, 然后使用逻辑回归分类器得到分类结果

对于传统软件缺陷预测算法, 因为使用的是 20 个静态代码度量所构成的特征向量, 所以在数据预处理时, 可以使用 SMOTE 方法进行过采样来处理数据集分类不均衡问题; 而对于 DBN、CNN 和本文模型, 只能简单地采用随机过采样的方法来对数据进行预处理.

本文使用 Python 环境以及深度学习框架 PyTorch 来实现本文提出的模型, 所有的实验均在一台带有 NVIDIA GTX 1080 的 Linux 服务器上运行. 此外, 因为随机过采样和丢弃法都具有一定的随机性, 因此实验中每个方法都执行 10 次, 取平均值来进行模型性能的比较.

4.4 实验结果及分析

本文采用 AUC 和 F1-measure 来比较 4 种方法在

6 组预测任务上的性能. 表 4 和表 5 分别记录了这 4 种方法关于 AUC 和 F1-measure 的实验结果, 每次测试任务的表现最好的已在表格中加粗.

表 4 4 种方法关于 6 项测试任务的 AUC

训练集	测试集	LR	DBN	CNN	本文模型
Camel 1.4	Camel 1.6	0.599	0.641	0.687	**0.709**
Lucene2.0	Lucene2.2	0.628	0.626	0.635	**0.641**
Synapse1.0	Synapse1.1	0.600	0.639	0.594	**0.646**
Synapse1.1	Synapse1.2	0.637	**0.697**	0.622	0.674
Poi2.5	Poi3.0	0.665	0.651	0.710	**0.718**
Xalan2.6	Xalan2.7	0.651	**0.683**	0.675	0.674
平均值		0.630	0.656	0.654	**0.677**

表 5 4 种方法关于 6 项测试任务的 F1-measure

训练集	测试集	LR	DBN	CNN	本文模型
Camel 1.4	Camel 1.6	0.355	0.342	0.489	**0.513**
Lucene2.0	Lucene2.2	0.611	0.666	0.709	**0.724**
Synapse1.0	Synapse1.1	0.444	0.457	0.456	**0.461**
Synapse1.1	Synapse1.2	0.471	**0.493**	0.487	0.473
Poi2.5	Poi3.0	0.707	0.716	0.756	**0.776**
Xalan2.6	Xalan2.7	0.633	0.642	0.654	**0.681**
平均值		0.537	0.553	0.592	**0.605**

表 3 和表 4 分别列出了 4 种方法关于每个测试任务的 AUC 值和 F1-measure. AUC 评估了模型分类的准确性, 而 F1-measure 评估了模型的稳定性. 从表 3 和表 4 中我们可以看到, 总体而言, 本文提出的模型在软件缺陷预测性能方面和模型稳定性明显优于 LR、DBN 和 CNN 的. 而本文模型的 AUC 和 F1-measure 的均值也都高于其他方法, 这也证实了本文提出模型的合理性和可行性. 此外, 从两张表中我们可以看出, 相较于传统的软件缺陷预测方法, 应用深度学习方法在软件缺陷预测性能和模型稳定性上都得到一定的提高. 这也证实了, 在软件缺陷预测性能方面, 深度学习方法优于传统的机器学习方法.

综上所述, 针对传统软件缺陷预测方法中对源代码语义特征挖掘不足的问题, 本文测试实验结果表明, 在软件缺陷预测领域, 相比于传统的预测方法, 应用深度学习方法得到了一定的提高. 而本文也根据前人的工作, 提出了用多层卷积神经网络对基于抽象语法树得到的表征向量进行分类学习, 有效提高了缺陷预测的准确性.

5 结束语

本文针对传统软件缺陷预测方法应用静态代码度量而忽视代码语义的缺点, 从代码的抽象语法树中提取出向量, 再利用卷积神经网络深度挖掘数据的能力挖掘代码中的语法语义特征, 从而对软件缺陷进行学习分类. 并且, 通过与 LR、DBN、MLP 方法的实验比较, 由 AUC 和 F1_measure 两个指标我们可以看出本文提出的模型在软件缺陷预测性能上得到了一定的提高. 然而, 关于数据集分类不均衡、模型优化等问题, 本文的处理方法相对粗糙, 这也是未来需要继续研究的方向.

参考文献

1 刘童. 基于机器学习算法的软件缺陷预测技术研究 [硕士学位论文]. 武汉: 华中师范大学, 2018.

2 Wang S, Liu TY, Tan L. Automatically learning semantic features for defect prediction. 2016 IEEE/ACM 38th International Conference on Software Engineering. Austin, TX, USA. 2016. 297–308.

3 Iqbal A, Aftab S. A classification framework for software defect prediction using multi-filter feature selection technique and MLP. International Journal of Modern Education and Computer Science, 2020, 12(1): 18–25. [doi: 10.5815/ijmecs.2020.01.03]

4 Li J, He PJ, Zhu JM, *et al*. Software defect prediction via convolutional neural network. 2017 IEEE International Conference on Software Quality, Reliability and Security. Prague, Czech Republic. 2017. 318–328.

5 Chawla NV, Bowyer KW, Hall LO, *et al*. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 2002, 16(1): 321–357.

6 Szegedy C, Ioffe S, Vanhoucke V, *et al*. Inception-v4, Inception-ResNet and the impact of residual connections on learning. Proceedings of the 31st AAAI Conference on Artificial Intelligence. San Francisco, CA, USA. 2017. 4278–4284.

7 Tang PJ, Wang HL, Kwong S. G-MS2F: GoogLeNet based multi-stage feature fusion of deep CNN for scene recognition. Neurocomputing, 2017, 225: 188–197. [doi: 10.1016/j.neucom.2016.11.023]

8 Mikolov T, Sutskever I, Chen K, *et al*. Distributed representations of words and phrases and their compositionality. Proceedings of the 26th International Conference on Neural Information Processing Systems. Red Hook, NY, USA. 2013. 3111–3119.

9 Ba LJ, Frey B. Adaptive dropout for training deep neural

networks. Proceedings of the 26th International Conference on Neural Information Processing Systems. Red Hook, NY, USA. 2013. 3084–3092.

10 Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv: 1412.6980, 2014.

11 Cukic B. Guest editor's introduction: The promise of public software engineering data repositories. IEEE Software, 2005, 22(6): 20–22. [doi: 10.1109/MS.2005.153]

12 Fawcett T. An introduction to ROC analysis. Pattern Recognition Letters, 2006, 27(8): 861–874. [doi: 10.1016/j.patrec.2005.10.010]

13 Powers DMW. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. Journal of Machine Learning Technologies, 2011, 2: 37–63.

14 Olsson PQ, Cotton WR. Balanced and unbalanced circulations in a primitive equation simulation of a midlatitude MCC. Part II: Analysis of balance. Journal of Atmospheric Sciences, 1997, 54(4): 479–497.

15 Guo HP, Liu HB, Wu CA, *et al*. Logistic discrimination based on G-mean and F-measure for imbalanced problem. Journal of Intelligent & Fuzzy Systems, 2016, 31(3): 1155–1166.