文章编号:1671-4598(2024)09-0094-07 DOI:10.16526/j.cnki.11-4762/tp.2024.09.014 中图分类号:TP311.55 文献标识码:A

基于注意力和代价敏感的软件缺陷预测方法

毛敬恩[1,2]，周世健[1]，章树卿[1,2]，樊 鑫[1,2]
(1. 南昌航空大学 软件学院，南昌 330038；
2. 南昌航空大学 软件测评中心，南昌 330038)

摘要：软件缺陷预测的目的是预先识别容易出现缺陷的代码模块以帮助软件质量保障团队适当的分配资源和人力；当前基于稳定学习的软件缺陷预测方法在特征提取过程中缺乏代码图像的全局信息，并忽视了不平衡数据对模型性能的影响；为了解决上述问题，文章提出了一种基于注意力和代价敏感的软件缺陷预测方法；该方法在SDP-SL的神经网络中增加了全局注意力模块，重点关注图像中和缺陷代码相关的特征，并将分类器的损失函数改进为代价敏感的损失函数，降低类不平衡对模型性能的影响；为了评估SDP-SLAC的性能，在PROMISE数据库中的10个开源Java项目上进行了多组比较实验；实验结果表明，SDP-SLAC方法可以有效提升缺陷预测模型的性能。

关键词：软件缺陷预测；全局注意力；代价敏感；类不平衡；损失函数

Software Defect Prediction Method Based on Attention and Cost Sensitivity

MAO Jingen[1,2], ZHOU Shijian[1], ZHANG Shuqing[1,2], FAN Xin[1,2]
(1. School of Software, Nanchang Hangkong University, Nanchang 330038, China;
2. Software Testing and Evaluation Center, Nanchang Hangkong University, Nanchang 330038, China)

Abstract: Software defect prediction aims to identify code modules that are prone to defects in advance, assisting the software quality assurance team to appropriately allocate resources and manpower. Currently, the software defect prediction method based on stable learning lacks the global information of code images during the process of feature extraction, and disregards the influence of imbalanced data on model performance. To address these issues, a software defect prediction method based on attention and cost sensitivity is proposed. This method enhances the global attention module in the software defect prediction and stable learning (SDP-SL) neural network, which focuses on the features of defective codes in the images. Moreover, it improves the classifier's loss function to the cost sensitive loss function, reducing the influence of the class imbalance on the model performance. To evaluate the performance of software defect prediction method based on attention and cost sensitivity (SDP-SLAC), multiple comparative experiments are conducted on ten open-source Java projects in the PROMISE database. The results show that the SDP-SLAC method effectively enhances the performance of defect prediction models.

Keywords: software defect prediction; global attention; cost sensitive; class imbalance; loss function

0 引言

随着科技的飞速发展和广泛应用，软件行业已经从早期作为计算机硬件的附属品逐渐演变成一个庞大而多样的领域，包括封闭或开放式系统软件以及各种应用软件。在这样的背景下，软件缺陷预测技术变得尤为重要。它能够快速准确地检测潜在的软件缺陷，从而有效地降低软件开发的成本，并确保软件质量得到提升。

在软件缺陷预测领域，研究人员致力于利用机器学习技术从软件源代码中提取多种特征，以提高缺陷预测模型的性能和准确性[1-4]。一些学者试图弥补程序语义与缺陷预测特征之间的差距。他们利用深度置信网络（DBN，deep belief network)[5]从源代码的抽象语法树（AST，abstract syntax tree）中学习语义特征，或者通过卷积神经网络(CNN，convolutional neural network)[6]自动学习程序的语义和结构特征。迁移成分分析（TCA，transfer component analysis)[7]被用来在保留数据属性的前提下进行训练数据转移，以使源数据的分布接近目标数据。此外，构建长短期记忆网络（LSTM，long short term memory)[8]用于分析程序的结构信息和代码之间的关系。然而，这些研究中都需要一个特定工具来生成代码的中间表示，在这一过程中就可能会丢失与代码缺陷相关的信息。

因此，近年来一些研究人员都试图利用更直接的方式来进行特征提取。我们在之前的研究中提出了SDP-SL方法[9]（SDP-SL，software defect prediction method based on stable learning)，通过代码可视化技术结合稳定学习去分析软件代码，并有效的定位出现缺陷的代码模块，该方法降低了第三方工具转换代码所造成的损失，并降低了特征之间的依赖对模型预测性能的影响。但SDP-SL方法仍然存在局限性，一方面，SDP-SL的网络结构简单，对数据中的全

收稿日期:2023-03-04； 修回日期:2023-03-28。
作者简介:毛敬恩(2001-),男,硕士。
引用格式:毛敬恩,周世健,章树卿,等.基于注意力和代价敏感的软件缺陷预测方法[J].计算机测量与控制,2024,32(9):94-100.

局信息关注不足，使得网络性能有限；另一方面，SDP-SL 方法没有考虑类不平衡对模型性能的影响。如果不能有效地解决这两个问题，将会对模型的预测性能产生影响。

针对以上缺点，我们提出了基于注意力机制和代价敏感的软件缺陷预测方法（SDP-SLAC，software defect prediction method based on attention and cost sensitivity），来进一步提升软件缺陷预测模型的性能。具体来说 SDP-SLAC 在 SDP-SL 将代码文件转换为代码图像的基础上，将全局注意力层引入神经网络中，融合网络中的通道注意和空间注意信息，从而获得更有效的全局信息；此外，数据集的类不平衡问题也通过代价敏感学习技术的加入从而得到改善。最后，利用 SDP-SLAC 构建缺陷预测模型并进行实验，实验结果也验证了 SDP-SLAC 方法的有效性。综上所述，本研究的主要贡献如下：

1）本文提出了 SDP-SLAC 方法，该方法引入全局注意力模块融合通道注意和空间注意信息，增加了模型提取特征的有效性。

2）引入代价敏感学习技术到损失函数上，设计了一个代价敏感的损失函数，有效的缓解了类不平衡问题。

3）利用本文提出的方法在 10 个开放的 Java 项目上进行了丰富的对比实验，实验结果表明我们的方法提高了软件缺陷预测的性能。

1　相关工作

1.1　软件缺陷预测技术

早期软件缺陷预测的研究重点再如何获取优秀的度量元，被研究人员所肯定的度量元大致可以分为静态代码特征和开发过程特征两种。静态代码特征包括基于操作符和操作数的数量来评估代码复杂程度的 Halstead 特征[10]、基于代码依赖关系来衡量代码中控制流复杂性的 McCabe 特征[11]、基于面向对象概念评估面向对象软件复杂性的 CK 特征[12]。上述特征都离不开代码复杂性一词，这是因为研究人员应用上述特征构建软件缺陷预测模型的基本假设是：源程序代码结构越复杂，存在缺陷的可能性就越大。而开发过程特征也被很多研究证明在预测软件质量上是有效的，包括代码修改次数[13]、历史提交信息[14]和代码变更记录[15]等。但是上述研究使用的静态特征和开发过程特征都受人主观思想的影响，并不包含代码中隐含的结构或语义信息。

因此，一些学者提出，利用代码中所包含的结构语义特征构建缺陷预测模型。文献［5］利用 DBN 从 AST 中提取的标记向量中学习语义特征。文献［6］则利用 CNN 从源代码的 AST 表示中生成特征，并尝试跟部分静态特征相结合，进一步提升预测模型性能。

1.2　注意力机制

注意力机制随着深度学习的流行而走入大家的视野，这种机制可以帮助网络进一步聚焦特定特征，忽视其他不重要的信息，提升特定特征的重要性。文献［16］第一次创新性的将注意力机制和深度学习网络相结合，提出了一种经典的注意策略。文献［17］在这基础上提出了空间变换网络（STN，spatial transformer network），通过引入一个子网络的方式来区分输入中的重要信息和无效信息。自注意力机制由文献［18］提出，是注意力机制的一种变形，随后被广泛应用在自然语言处理和目标检测等领域并获得了巨大的成功。

1.3　代价敏感学习技术

代价敏感学习最早由文献［19］提出，但在之后的很长一段时间都没有学者继续深入研究。直到机器学习渐渐成为主流，在使用机器学习研究分类问题时往往会面对误分类的问题，因此就有学者提出使用代价敏感学习技术来缓解误分类问题。文献［20］将代价敏感引入权重更新公式中，提出了 AdaCost 方法，提高误分类样本的权重。文献［21］则选择将 softmax 分类器的损失函数和代价敏感思想相结合，提高模型分类的准确性。现在主流的研究思路就是将分类算法和代价敏感学习相互结合，提升分类的准确性。

2　SDP-SLAC 总体架构

本文设计的 SDP-SLAC 模型的整体框架如图 1 所示，在之前 SDP-SL 的研究内容基础上，在网络的残差块中增加全局注意力模块，并针对类不平衡问题，在最后的分类器中设计了一个基于代价敏感的损失函数。网络的整体构造分为代码文件可视化、深度特征提取、样本重加权、模型的构建 4 个部分，具体描述如下。

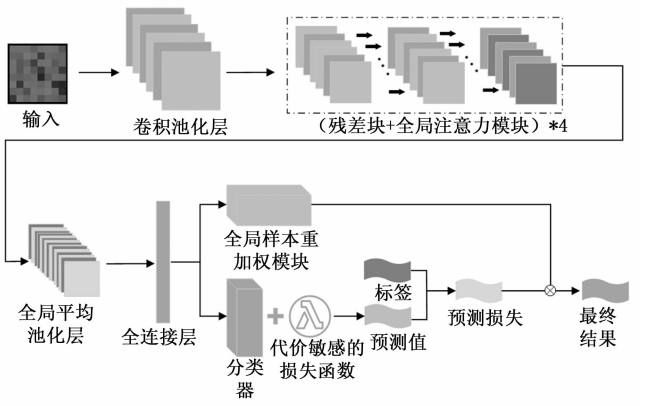

图 1　SDP-SLAC 模型的整体框架

1）代码文件可视化：我们按照和 SDP-SL 方法相同的策略将源项目的所有代码文件转换为对应的 RGB 图像。首先，每个代码文件中的源代码字符都将被转换为相对应的 ASCII 十进制值，例如，符号“i”被转换为“105”，字母“n”被转换为“110”。转换完成后一个代码文件中的所有字符将被组合成一个整数向量，将该向量中每 3 个值进行分组，这 3 个值将分别代表 R、G、B 通道的值，以此方式将整数向量转换为具备颜色信息的像素点。最后将代码图像输入到训练网络中进行训练。

2）深度特征提取：在初始模型中采用 ResNet18 神经网络提取代码图像通用特征，在该部分中，在 ResNet18 网络的残差结构中添加全局注意力模块，图像特征输入到全

局关注层中会分别为代码的结构特征和语义特征分配权重，以提高结构特征和语义特征在预测模型中的重要性，并抑制其他信息，得到更能代表缺陷代码模块的特征。

3）样本重加权：和代码可视化部分一致，我们按照和 SDP-SL 方法相同的策略为样本进行重加权，以此来消除部分空间中样本特征之间的依赖性。

4）模型的构建：该部分对分类器进行了重新设计，为了降低软件缺陷数据集类不平衡问题对模型性能产生的影响，将原先分类器中的损失函数更换为代价敏感的损失函数，在算法层面缓解类不平衡问题，避免采样方法导致的数据缺失或过拟合问题。而当一个代码模块经过上述 3 个步骤之后，训练网络可以从中提取到该代码模块的代表性特征，最后将该特征输入到重新设计的分类器中进行预测，并和真实标签进行对比来评估模型的性能。

2.1 全局注意力模块

全局注意力模块包括了通道注意力模块、空间注意力模块。通道注意力模块可以将跨通道空间依赖放大，减少信息丢失，扩大全局交互特征。空间注意力模块可以将空间信息进行融合。在每一个残差块之前添加一个全局注意力模块，也就是说在特征图像在进入每个残差块之前，要先进入全局注意力模块。增加了全局注意力模块后的残差结构具体描述如下。

第一个残差结构：包含一个全局注意力模块和 4 个基本卷积块。全局注意力模块包括了通道注意力模块和空间注意力模块，由于注意力模块的作用是强调和缺陷相关的特征并抑制和缺陷不相关的特征，不会对特征图产生维度上的改变，因此输出的大小和输入一致，不会对模型后续的结构产生影响。每个基本卷积块由两个 3×3 的卷积层和一个跳跃连接组成。输入通道数为 64，输出通道数为 64。卷积核大小为 3×3，步长为 1，填充为 1。每个卷积层后面跟着批标准化和 ReLU 激活函数。在第一个基本卷积块中，输入特征图通过跳跃连接添加到最后的 ReLU 激活函数前，保留特征信息。其他基本卷积块中，输入和输出通道数保持一致。

第二个残差结构：包含一个全局注意力模块和 4 个基本卷积块。每个基本卷积块的输入通道数为 64，输出通道数为 128。其他配置与第一个残差结构相同。

第三个残差结构：包含一个全局注意力模块和 4 个基本卷积块。每个基本卷积块的输入通道数为 128，输出通道数为 256。其他配置与第一个残差结构相同。

第四个残差结构：包含一个全局注意力模块和 4 个基本卷积块。每个基本卷积块的输入通道数为 256，输出通道数为 512。其他配置与第一个残差结构相同。

全局注意力模块将通道注意力和空间注意力串行连接到模型的整体网络中，如图 2 所示。根据输入模型的特征图 M，全局注意力模块先通过一个一维的通道注意力图来学习通道以及跨通道的注意力信息 M_c，再通过一个二维的空间注意力图来学习空间注意力信息 M_s，最后将 M_c 和 M_s 融合到网络后续的结构中，用于模型网络继续训练。该融合过程可以由下式表示：

$$M_1 = M_c(M) \otimes M$$
$$M_2 = M_s(M_1) \otimes M_1 \tag{1}$$

式中，M_1 代表通道注意力模块的输出 M_c 和特征图 M 的融合，M_2 代表 M_1 继续和空间注意力模块的输出 M_s 融合得到的最终输出，$\otimes$代表的是元素逐个相乘。接下来将分别对通道注意力模块和空间注意力模块的细节进行详细介绍。

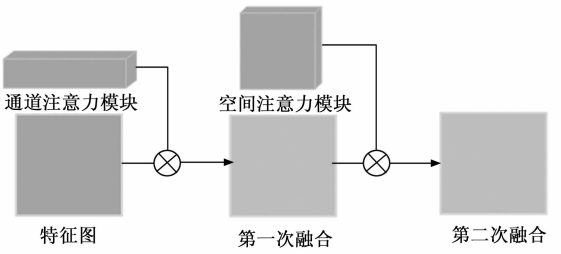

图 2 全局注意力模块

1）通道注意力模块：

通道注意力图是利用特征通道之间的关系生成的，特征所映射的每一个通道都可以被看作是一个特征检测装置[22]，通道注意力集中于研究输入图像是“什么”的信息。为了得到更有效的通道注意力，我们将特征图像的空间维度进行了压缩，同时使用平均池化方法和最大池化方法将全局的空间信息进行了聚合，详细的流程如下。

通道注意力模块的第一步是将代码图像生成的特征图依次进行平均池化操作和最大池化操作，可以分别得到两个不一样的空间信息描述值 M_a 以及 M_m 该步骤的目的是将聚合特征图的空间信息，同时进行平均池化操作和最大池化操作相比于任何一种单一的池化操作效果都要好。在网络的后续阶段，M_a 和 M_m 会被送入到一个共享的多层感知机网络中，来生成我们所需要的通道注意力图 M_c。该共享网络由多层感知机和一个隐藏层组成。通道注意力的计算公式可以由下式表示：

$$M_c(M) = \sigma\{\mathrm{MLP}[\mathrm{AvgPool}(M)] + \mathrm{MLP}[\mathrm{MaxPool}(M)]\} = \sigma\{W_1[W_0(M_a)] + W_1[W_0(M_m)]\} \tag{2}$$

式中，σ 表示 sigmoid 函数，W_0 和 W_1 分别代表 M_a 和 M_m 在共享网络中的权重，该权重是网络中是互相共享的。

2）空间注意力模块：

空间注意力模块和通道注意力模块不同的是空间注意力模块关注的是“在哪里”的信息，和通道注意力相互补充。为了得到输入的特征图的空间注意力信息，首先要先针对特征图的每个通道的特征进行平均池化操作和最大池化操作，通过沿着通道轴进行池化操作可以在保留通道信息的同时，突出图像中重要的信息区域。池化操作结束后可以得到每个通道的平均值和最大值，再将两个池化操作的结果连接起来，形成一个全新的特征描述子。我们在利用一个卷积层从全新的特征描述子上生成一个空间注意力图 M_s。具体的过程描述如下。

和通道注意力类似的，也通过使用两个池化操作来聚合特征图的信息，沿着通道轴进行平均池化操作和最大池化操作可以分别得到空间注意力的平均池化特征图 M_a^1 以及最大池化特征图 M_m。再将 M_a^1 和 M_m^1 通过一个卷积层进行连接和卷积，最终得到我们所需要的空间注意力图 M_s。空间注意力的计算公式可以由下式表示：

$$M_s(M)=\sigma\{f[(\mathrm{AvgPool}(M);\mathrm{MaxPool}(M))]\}=\sigma[f(M_a^1;M_m^1)] \tag{3}$$

式中，σ 表示 sigmoid 函数，f 表示卷积运算。

2.2　构建代价敏感的交叉熵损失函数

随着技术的发展和经验的积累，开发人员更加熟悉编码规范、最佳实践和常见的错误模式。这使得他们能够更好地编写高质量、健壮性强的代码。因此一个完善的软件中，存在缺陷的代码模块数量肯定远远少于不存在缺陷的代码模块。因此在软件缺陷预测中一个无法避免的问题就是数据集的类不平衡问题，这往往就会导致网络在训练过程中会更倾向于识别出多数类的样本，使得预测结果不够理想。为了更好地学习存在缺陷的代码模块的特征，本节将网络中初始的分类器改进为代价敏感的分类器，在算法层面对类不平衡问题进行处理。

初始分类器的损失函数采用的是交叉熵损失函数，该函数如下所示。在二分类任务中，交叉熵损失函数可以衡量模型输出的概率分布和真实标签之间的差异，促使模型学习正确的预测策略，能使得模型的预测结果更接近实际。

$$l=-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{k}\{(y^i)_c*\log[s(x^i)]_c\} \tag{4}$$

其中：N 代表的是训练数据中样本也就是代码模块的数量，k 代表的是类别的数量，由于我们执行的是二分类任务，故此处 k 的值为 2，c 代表标签的类别，y^i 代表样本的真实标签，$s(x^i)$ 代表样本的预测结果，其中 $s(\ )$ 代表激活函数，此处选择的是 softmax 激活函数。分类器分类错误有两种情况，第一种分类错误是将有缺陷代码模块预测为无缺陷代码模块，而另一种分类错误是将无缺陷代码预测为有缺陷代码模块，通常情况下这两种误分类所产生的代价是相同的。而对分类器进行改进的核心思路就是通过给损失加上权重，提高模型误分类时的代价。该权重可由代价函数得出，而代价函数的构造离不开代价矩阵，二分类的代价矩阵如表 1 所示。

表 1　代价矩阵

	真实有缺陷	真实无缺陷
预测有缺陷	cost(f, f)	cost(f, t)
预测无缺陷	cost(t, f)	cost(t, t)

其中：cost（f，t）代表将类别 f 误分类为 t 的代价值。代价函数的构造可以分为统计、构造代价函数以及设置权重因子三步来进行，首先计算出数据仓库中缺陷代码模块的数量 P_f，以及无缺陷代码模块的数量 P_t，再据此构造具体的代价函数，最后为不同的误分类设置不同的权重因子。初步的代价函数可以设置如下：

$$\mathrm{cost}(f,t)=\begin{cases}\left(\dfrac{P_f}{P_t}\right)^m, & P_f>P_t\\ 1, & P_f=P_t\\ \left(\dfrac{P_t}{P_f}\right)^n, & P_f<P_t\end{cases} \tag{5}$$

式中，m 代表缺陷代码模块大于无缺陷代码模块的数量时，将缺陷代码模块误分类为无缺陷代码模块的权重因子，n 代表缺陷代码模块小于无缺陷代码模块的数量时，将缺陷代码模块误分类为无缺陷代码模块的权重因子。权重因子使得模型在将缺陷代码模块预测为无缺陷代码模块时，会有更大的损失值。综上所述，本节可以将误分类代价值设置为：

$$f(c)=\begin{cases}1, & P_f=P_t\\ \left(\dfrac{P_f}{P_t}\right)^m, c=0, & P_f>P_t\\ \left(\dfrac{P_t}{P_f}\right)^n, c=0, & P_f<P_t\\ \left(\dfrac{P_t}{P_f}\right)^m, c=1, & P_f>P_t\\ \left(\dfrac{P_f}{P_t}\right)^n, c=1, & P_f<P_t)\end{cases} \tag{6}$$

其中：c 代表具体的类别，$c=0$ 和代价矩阵中的 cost（f，t）对应，$c=1$ 和代价矩阵中的 cost（t，f）对应。将误分类代价值和交叉熵损失函数相结合，得到代价敏感的损失函数为：

$$l=-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{k}\{(y^i)_c*\log[s(x^i)]_c*f(c)\} \tag{7}$$

3　实验与分析

3.1　实验环境和数据集

本文的实验在处理器为 Intel（R）Core（TM）i5-12400F 2.50 GHz，显卡为 NVIDIA GeForce RTX 3060，内存为 16G 的计算机上运行。

为了便于将我们的工作和其他学者的研究进行比较，我们使用数据集为 PROMISE 数据库中的公开数据，PROMISE 数据库包括了多个软件项目，由多名研究人员共同整理并免费供大家使用，是专门研究软件缺陷的数据库。数据库中包含的信息非常详细，例如多种缺陷度量元属性、软件源代码以及对应的缺陷标签等。我们从 GitHub 上从 PROMISE 数据库中分别获取了 10 个 Java 项目的多个不同版本，并且收集了这些项目的项目名称、版本号以及每个代码文件的缺陷标签，将每个项目可用的文件总数、平均文件大小、平均缺陷率等信息进行了统计。这 10 个项目的详细信息可见表 2 所示。

为了全面的对 SDP-SLAC 方法进行验证，针对该方法设计了多组对比实验。在实验中，使用表 1 中列出的每个项目的连续版本进行缺陷预测，即选取较旧的版本作为源项目用来训练模型，较新的版本作为目标项目用于测试（例如选取 ant-1.5 项目作为训练集，ant-1.6 项目作为测试集）。

表 2 数据集详细信息

项目	版本	文件数量	平均文件数量	平均大小/kb	缺陷率/%
ant	1.5,1.6,1.7	1 465	488	6.2	13.4
camel	1.2,1.4,1.6	3 140	1 046	2.9	18.7
jEdit	3.2,4.0,4.1	1 935	645	8.7	19.2
log4j	1.0,1.1	300	150	3.4	49.7
lucene	2.0,2.2,2.4	607	402	3.8	35.8
xalan	2.4,2.5	1 984	992	4.6	29.6
xerces	1.2,1.3	1 647	549	2.9	15.7
ivy	1.4,2.0	622	311	4.1	20.0
synapse	1.0,1.1,1.2	661	220	3.8	22.7
poi	1.5,2.5,3.0	1 248	416	3.6	40.7

3.2 数据预处理

数据库中的数据并不能直接输入模型，因此在代码文件可视化的步骤中，将源项目数据和目标项目数据按照文件级别分别生成代码图像，并将图像按照代码文件的缺陷标签分别保存到 clean 和 buggy 两个文件夹中。

3.3 实验设计

为了全面评估 SDP-SLAC 模型在软件缺陷预测中的预测性能，我们分别将其与多种基准方法进行了比较，并且为了验证 SDP-SLAC 新增全局注意力模块和代价敏感学习是否有效提高了缺陷预测性能，还将 SDP-SL 模型也加入了对比方法中，方法的具体描述如下。

1）DP-CNN[6]：是一种基于 CNN 的软件缺陷预测方法，专注于从软件源代码的 AST 结构中学习特征表示，例如代码语法和结构，DP-CNN 能有效捕捉代码中的模式和结构信息，从而提高缺陷预测的准确性。

2）DP-LSTM[8]：是一种基于 LSTM 的软件缺陷预测方法，LSTM 是一种递归神经网络，在软件缺陷预测中，DP-LSTM 利用 LSTM 网络构建代码的演化历史或变化模式，从而预测潜在的缺陷。

3）SDP-SL[9]：是一种最新的软件缺陷预测方法，利用稳定学习中的样本重加权技术将样本特征进行关系的解耦，寻找和缺陷更相关的特征，从而提升缺陷预测的性能。

4）PROMISE-DP[5]：是一种基于数据集 PROMISE 的软件缺陷预测方法，基于该数据集原始的 20 个特征构建分类器，用于缺陷预测。

5）SDP-SLAC：该方法是在 SDP-SL 模型的基础上进一步优化实现的，在神经网络中增加了全局注意力模块，在图像输入神经网络中会多次经过该模块，从而提取到更有效的特征，并将分类器的初始损失函数改进为代价敏感的损失函数，通过缓解类不平衡问题进一步提升模型的预测性能。

为了实现公平的比较，我们在同一台电脑上进行了实验并报告了平均结果，并且在实现基线方法时，我们使用与介绍它们的论文中描述的相同的网络架构和参数设置。

3.4 统计检验

为了在实验中能更好的比较 SDP-SLAC 以及其他基线模型的性能，我们采用 Scott-Knott ESD 测试方法来对比，这是一种均值比较方法，将一组测量值（在本文中是 F-measure 测量值）划分为具有统计显著性差异的不同组。该测试方法包括以下两个主要步骤：

1）寻找最佳分组：首先，通过层次聚类的方法寻找一个能最大化组件测量平均值的划分。这个步骤有利于将测量值相似的样本分在一组，从而更好的比较他们的性能表现。

2）分组或合并：在得到最佳分组后，根据样本之间的显著性差异，将某些组合并成一组或者将某一组拆分成两组。这个步骤的目的是寻找不同组之间的显著性差异，从而确定哪些模型在性能上具有显著优势。

3.5 评价指标

为了全面的评估模型性能，确保模型在正负样本中取得较好的平衡，我们采用 F-measure 指标作为模型的评价指标。F-measure 结合了两个指标：精确率和召回率，评估分类模型的性能离不开混淆矩阵，其中记录了模型对样本分类 4 种可能的结果，如表 3。

表 3 混淆矩阵

	预测无缺陷	预测有缺陷
真实无缺陷	*TP*	*FN*
真实有缺陷	*FP*	*TN*

具体的说，如果模型正确的将无缺陷的文件预测为正例则被称为真阳性（TP，true positive），否则为假阳性（FN，false negative）。类似的，如果模型错误的将存在缺陷的文件预测为正例则为假阴性（FP，false positive），否则为真阴性（TN，true negative）。精确度（P，precision）、召回率（R，recall）和 F-measure 的具体定义为：

$$P = \frac{TP}{TP + FP} \tag{8}$$

$$R = \frac{TP}{TP + FN} \tag{9}$$

$$F - \text{measure} = \frac{2 * PR}{P + R} \tag{10}$$

4 实验结果与分析

1）SDP-SLAC 和其他基线方法相比效果：

我们将 SDP-SLAC 与 4 种基线方法进行比较，4 种基线方法中既包括基于传统特征进行缺陷预测的 PROMISE-DP 方法，也包括基于 AST 提取语义结构特征的 DP-CNN 等方法，还包括了最新的结合了稳定学习的 SDP-SL 方法。我们分别使用上述方法进行了 16 组 WPDP 实验，如表 4。每组都选取旧版本作为源项目用来训练模型，新版本作为目标项目用于测试。

主要在 3 个方面进行了对比实验，SDP-SLAC 模型和基于传统特征的 PROMISE-DP 模型进行对比，SDP-SLAC 和结合传统特征的深度学习模型（DP-CNN、DP-LSTM）进行对比，SDP-SLAC 和最新的缺陷预测模型 SDP-SL 模型进行对比。最终目的是验证本文提出的 SDP-SLAC 方法的

有效性。

为了验证从神经网络中提取代码特征对软件缺陷预测模型的优化作用，对表 4 中的第 5 列和第 7 列数据进行了对比分析。结果显示，相比于使用静态特征的 PROMISE-DP 方法，SDP-SLAC 方法在 16 组实验中的效果均优于 PROMISE-DP 方法，平均 F-measure 提高了 27.9%。综合上述比较结果，可以判断 SDP-SLAC 模型的预测性能高于 PROMISE-DP 模型。

为了验证 SDP-SLAC 模型的预测性能高于常规深度学习模型，对表 4 中的第 3 列、第 4 列分别和第 7 列进行对比分析。结果显示，和 DP-CNN、DP-LSTM 相比，SDP-SLAC 在 F-measure 的平均值上分别提高了 15.8%、27.4%。并且在 16 组实验中，和 DP-CNN、DP-LSTM 相比分别有 14 组和 13 组实验占优。综合上述内容，可以判断 SDP-SLAC 的预测性能高于常规深度学习模型。

为了验证全局注意力模块和代价敏感对 SDP-SLAC 模型性能的提升作用，对表 4 中的第 6 列和第 7 列进行对比分析。结果显示，相较于没加全局注意力模块和代价敏感的 SDP-SL 方法，SDP-SLAC 方法在 16 组实验中，有 11 组实验占优，并且 F-measure 的平均值提高了 1.4%。综合上述比较结果，可以判定全局注意力模块和代价敏感的加入对模型性能有一定的提升作用。

表 4　实验中 6 种方法的 F-measure 对比

项目	版本信息	DP-CNN	DP-LSTM	PROMISE-DP	SDP-SL	SDP-SLAC
ant	1.5－>1.6	0.485	0.391	0.49	0.606	**0.631**
	1.6－>1.7	0.571	0.279	0.513	0.593	**0.617**
camel	1.2－>1.4	0.429	0.343	0.395	0.414	**0.520**
	1.4－>1.6	0.292	0.422	0.318	0.419	**0.513**
jEdit	3.2－>4.0	0.589	0.466	0.502	**0.696**	0.605
	4.0－>4.1	0.685	0.492	0.551	**0.704**	0.632
log4j	1.0－>1.1	0.776	0.519	0.602	**0.789**	0.790
lucene	2.0－>2.2	0.691	0.710	0.498	0.759	**0.764**
	2.2－>2.4	0.736	0.742	0.612	**0.773**	0.752
xalan	2.4－>2.5	0.456	**0.701**	0.54	0.631	0.680
xerces	1.2－>1.3	0.279	0.196	0.233	0.397	**0.402**
ivy	1.4－>2.0	0.435	0.129	0.33	0.485	**0.508**
synapse	1.0－>1.1	0.308	0.434	0.495	0.518	**0.521**
	1.1－>1.2	0.428	0.563	0.546	0.647	**0.682**
poi	1.5－>2.5	**0.877**	0.838	0.612	0.785	0.793
	2.5－>3.0	0.775	0.781	0.727	**0.832**	0.776
胜/平/负		14/0/2	13/0/3	16/0/0	11/0/5	
平均值		0.550	0.500	0.498	0.628	**0.637**

为了进一步全面验证 SDP-SLAC 方法的性能，我们针对 5 种方法的 F-measure 值进行了 Scott-Knott ESD 测试，测试结果如图 3。从图中可以看出 SDP-SLAC 方法不仅平均 F-measure 最高，且该方法的中值除了略微低于 SDP-SL 之外，均高幅领先其他方法。测试结果表明 SDP-SLAC 方法在软件缺陷预测任务中的优越性，它在平均 F-measure、性能稳定性等方面都表现更为出色，相比其他方法具有更好的预测性能和可靠性。

此外，图中 Rank 3 有异常数据可能归因于两个主要因素。一方面，模型可能在某些数据集上过度拟合，从而影响其性能。另一方面，这可能是由于超参数调整问题。模型的超参数需要根据不同的数据集进行调整，以实现最佳性能。其次，其他方法没有表现出异常值的原因是它们有效地利用了代码的语义和结构信息进行预测。与 PROMISE-DP 方法相比，这些方法具有更强的泛化能力，使它们能够适应更广泛的数据集，从而降低出现异常值的可能性。

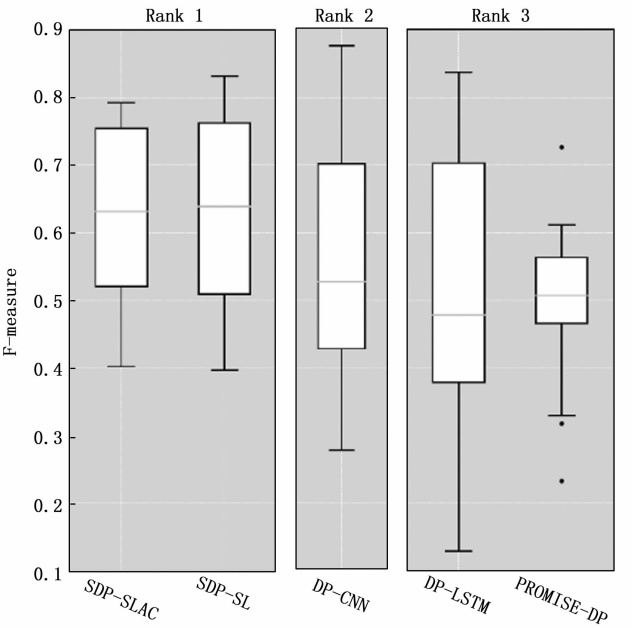

图 3　Scott-Knott ESD 测试结果

2）为什么我们的 SDP-SLAC 方法有效？

上述的实验结果表明，在软件缺陷预测任务中，我们的 SDP-SLAC 方法和多种基线方法相比具有更好的预测性能。可能有以下两个原因：

SDP-SLAC 方法和基于传统特征以及抽象语法树提取特征的方法相比，从更细粒度（字母粒度）的图像中提取代码的深度特征，有效的避免了手工统计特征和第三方工具提取特征带来的信息丢失。

SDP-SLAC 方法和 SDP-SL 方法相比，我们进一步考虑了全局信息和类不平衡对模型的影响。我们在神经网络中加入了全局注意力层，全面的利用了空间注意力和通道注意力，并将其结合起来，获得更完整的全局信息。此外，通过对损失函数进行改进，利用代价敏感的损失函数有效解决了类不平衡问题，使得模型的性能得到了进一步的提升。

3）案例研究：

如表 5 所示为本文进行的一项案例研究，展示了本文的 SDP-SLAC 方法和基线 SDP-SL 方法在 5 个真实代码文件上的预测结果，其中 0 代表预测无缺陷，1 代表预测有缺陷。在该案例中，文件 1 和文件 3 为真实的缺陷代码文件。

表 5 案例研究

文件编号	真实标签	SDP-SL	SDP-SLAC
1	1	1	1
2	0	0	0
3	1	0	1
4	0	0	0
5	0	0	1

从表中可以看出，由于结合了代价敏感的 SDP-SLAC 方法加大了将缺陷代码文件误分类为无缺陷代码文件的惩罚，因此在一定程度上缓解了模型误分类的情况，提升了模型的预测性能。但是还存在将无缺陷代码文件误分类为缺陷代码文件的情况，未来设计一个更优秀的误分类代价值是可行措施。

5 结束语

本研究主要是针对基于稳定学习的 SDP-SL 方法进行了改进，构建了基于注意力机制和代价敏感的 SDP-SLAC 缺陷预测模型。该方法在 SDP-SL 的神经网络中增加全局注意力模块，并将分类器中的损失函数改进为代价敏感的损失函数，使得模型在更关注缺陷特征的同时，降低类不平衡对模型性能的影响，提升模型的缺陷预测性能。

我们的实验结果验证了 SDP-SLAC 的有效性，虽然在一定程度上实现了对模型的优化，然而 SDP-SLAC 还是存在一些局限。本文提出的方法仅仅是面对 Java 项目，但随着人工智能时代的到来，未来 Python 等语言的使用率将会大幅上升，因此在未来希望能设计出可以适用于 Python 或是通用的软件缺陷预测模型。此外，本文神经网络的多种参数是手动设置的，会在一定程度上对模型的预测性能产生影响，因此未来将针对性的采用参数自动优化技术，使得模型可以自动调整模型参数，进一步提升模型的缺陷预测性能。

参考文献：

[1] LI N, SHEPPERD M, GUO Y. A systematic review of unsupervised learning techniques for software defect prediction [J]. Information and Software Technology, 2020, 122: 106287.

[2] CROFT R, XIE Y, BABAR M A. Data preparation for software vulnerability prediction: A systematic literature review [J]. IEEE Transactions on Software Engineering, 2022, 49 (3): 1044 - 1063.

[3] ZOU Q, LU L, YANG Z, et al. Joint feature representation learning and progressive distribution matching for cross-project defect prediction [J]. Information and Software Technology, 2021, 137: 106588.

[4] LIN G, WEN S, HAN Q L, et al. Software vulnerability detection using deep neural networks: a survey [J]. Proceedings of the IEEE, 2020, 108 (10): 1825 - 1848.

[5] WANG S, LIU T, TAN L. Automatically learning semantic features for defect prediction [C] // Piscataway: IEEE Press, 2016: 297 - 308.

[6] LI J, HE P, ZHU J, et al. Software defect prediction via convolutional neural network [C] // Piscataway: IEEE Press, 2017: 318 - 328.

[7] PAN S J, TSANG I W, KWOK J T, et al. Domain adaptation via transfer component analysis [J]. IEEE Transactions on Neural Networks, 2010, 22 (2): 199 - 210.

[8] DAM H K, PHAM T, NG S W, et al. Lessons learned from using a deep tree-based model for software defect prediction in practice [C] // Piscataway: IEEE Press, 2019: 46 - 57.

[9] FAN X, MAO J, LIAN L, et al. Software defect prediction method based on stable learning [J]. Computers, Materials & Continua, 2024: 1 - 30. https://doi. org/10. 32604/cmc. 2023. 045522.

[10] MAURICE HOWARD HALSTEAD, et al. Elements of software science [M]. New York: Elsevier Science Ltd Press, 1977: 7.

[11] MCCABE T J. A complexity measure [J]. IEEE Transactions on software Engineering, 1976, 4: 308 - 320.

[12] CHIDAMBER S R, KEMERER C F. A metrics suite for object oriented design [J]. IEEE Transactions on Software Engineering, 1994, 20 (6): 476 - 493.

[13] MOSER R, PEDRYCZ W, SUCCI G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction [C] // Piscataway: IEEE Press, 2008: 181 - 190.

[14] PORNPRASIT C, TANTITHAMTHAVORN C K. Deeplinedp: Towards a deep learning approach for line-level defect prediction [J]. IEEE Transactions on Software Engineering, 2022, 49 (1): 84 - 98.

[15] KONDO M, GERMAN D M, MIZUNO O, et al. The impact of context metrics on just-in-time defect prediction [J]. Empirical Software Engineering, 2020, 25: 890 - 939.

[16] MNIH V, HEESS N, GRAVES A. Recurrent models of visual attention [C] // Berlin: Springer International Press, 2014: 2204 - 2212.

[17] JADERBERG M, SIMONYAN K, ZISSERMAN A. Spatial transformer networks [C] // Berlin: Springer International Press, 2015: 2017 - 2025.

[18] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need [C] // Berlin: Springer International Press, 2017: 6000 - 6010.

[19] BREIMAN L, FRIEDMAN J, OLSHEN R, STONE C. Classification and Regression Trees [M]. London: Chapman and Hall Press, 1984: 50 - 51.

[20] FAN W, STOLFO S J, ZHANG J, et al. AdaCost: misclassification cost-sensitive boosting [C] // New York: PMLR Press, 1999: 97 - 105.

[21] SHEN W, WANG X, WANG Y, et al. Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection [C] // Piscataway: IEEE Press, 2015: 3982 - 3991.

[22] ZEILER M D, FERGUS R. Visualizing and understanding convolutional networks [C] // Berlin: Springer International Press, 2014: 818 - 833.