学校代码：10285

学　　号：20204227007

SOOCHOW UNIVERSITY

硕士学位论文

（学术学位）

即时软件缺陷预测中

基于样本权重的数据预处理方法

Data preprocessing method based on sample weight

in just-in-time software defect prediction

研究生姓名	庄伟渊
指导教师姓名	章晓芳
专业名称	计算机科学与技术
研究方向	软件缺陷预测
所在院部	计算机科学与技术学院
论文提交日期	2023 年 7 月

苏州大学学位论文独创性声明

本人郑重声明：所提交的学位论文是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含其他个人或集体已经发表或撰写过的研究成果，也不含为获得苏州大学或其它教育机构的学位证书而使用过的材料。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人承担本声明的法律责任。

论文作者签名：_____日　期：_____

苏州大学学位论文使用授权声明

本人完全了解苏州大学关于收集、保存和使用学位论文的规定，即：学位论文著作权归属苏州大学。本学位论文电子文档的内容和纸质论文的内容相一致。苏州大学有权向国家图书馆、中国社科院文献信息情报中心、中国科学技术信息研究所（含万方数据电子出版社）、中国学术期刊（光盘版）电子杂志社送交本学位论文的复印件和电子文档，允许论文被查阅和借阅，可以采用影印、缩印或其他复制手段保存和汇编学位论文，可以将学位论文的全部或部分内容编入有关数据库进行检索。

涉密论文□

本学位论文属　　在_____年___月解密后适用本规定。

非涉密论文□

论文作者签名：_____日　期：_____

导 师 签 名：_____日　期：_____

即时软件缺陷预测中基于样本权重的数据预处理方法

摘　　要

即时软件缺陷预测技术关注项目的实际开发过程，旨在提前预测开发者每次提交的代码变更是否存在缺陷，帮助开发人员优化资源分配，提高代码质量。由于不同提交的时间新旧和质量高低的不同，因此对它们的关注程度也应该不同，即应该赋予不同的提交样本不同的权重，然而目前即时软件缺陷预测领域对样本权重的相关研究并不充分。本文面向项目的持续开发过程，提出可以随着预测模型迭代而更新的样本权重，对数据进行预处理，让预测模型关注更重要的样本，并利用样本权重来缓解数据类不平衡的问题。

(1)针对现有缺陷预测方法对新旧提交缺乏区分的问题，本文提出了一种基于时间维度的样本重要性权重。该方法从项目的版本控制系统中提取时间维度信息，然后利用时间衰减函数赋予新提交的样本较高权重，旧提交的样本较低权重。这些权重会影响缺陷预测模型对不同提交的重视程度，从而针对性地提高缺陷预测的性能。本文针对10个开源项目，每个项目选取5个数据区间，9次迭代过程，共450个任务上进行了实验，验证了基于时间维度的样本权重的有效性。

(2)针对相近时间下的提交难以区分的问题，本文提出了一种基于特征贡献度的样本重要性权重。该方法针对上一轮模型迭代过程中被误分类的提交样本，通过机器学习解释模型计算它们的特征贡献度和分类倾向性，对分类倾向性与实际类别偏差较大的提交赋予较高的权重，随后进行去噪处理。本文在450个任务上进行了实验，验证了基于特征贡献度的样本权重的有效性。

(3)针对缺陷预测中数据采样方法挑选用于合成的样本不具有代表性的问题，本文提出了一种基于样本重要性权重的数据采样方法。该方法一方面通过样本重要性权重对提交样本进行排序，挑选重要的样本用于合成新样本；另一方面计算样本距离时也会考虑样本重要性权重，合成的样本更具有多样性。本文在450个任务上进行了实验，验证了融合了时间维度和特征贡献度的数据采样方法的有效性。

本文针对现有研究中尚存的三个问题，提出了相应的解决方案，进一步提升了即时软件缺陷预测的性能表现，对及时发现缺陷、提高代码质量、调度测试资源以及降低维护成本具有重要意义。

关键词： 软件缺陷预测，数据预处理，样本权重，类不平衡

作　　者： 庄伟渊

指导教师： 章晓芳

Data preprocessing method based on sample weight in just-in-time software defect prediction

Abstract

Just-in-time software defect prediction technology focuses on the actual development process of a project, aiming to predict in advance whether there are defects in every code change submitted by developers, so as to help developers optimize resource allocation, and improve code quality. Due to the differences in time order of commits and in quality, the level of attention given to them should also be different, that is, different submitted samples should be given different weights. However, there currently needs to be more relevant research on sample weights in the field of just-in-time software defect prediction. This thesis focuses on the continuous development process of the project. It proposes sample weights can be updated as the prediction model iterates to preprocess the data, allowing the prediction model to focus on more important samples and using sample weights to alleviate the problem of data with imbalanced class.

(1) This thesis proposes a sample importance weight based on time dimension to address the need for more differentiation of time order of commits in existing defect prediction methods. This method extracts time dimension information from the version control system of the project and then uses a time decay function to assign higher weights to the newly submitted samples, while the old committed samples will have lower weights. These weights will affect the importance of defect prediction models on different commits, thereby improving the performance of defect prediction targetedly. This article conducted experiments on 10 open source projects, each with 5 data intervals and 9 iteration processes, on a total of 450 tasks in a model iteration scenario to verify the effectiveness of the sample weights.

(2) This thesis proposes a sample importance weight based on feature contribution to address the difficulty in distinguishing commits at similar times. This method focuses on submitted samples that were misclassified during the previous round of model iteration, calculates their feature contribution and classification propensity through a machine learning interpretation model, assigns higher weights to commits with significant deviations between classification propensity and actual categories, and finally performs denoising processing. Experiments on 450 tasks are performed to verify the effectiveness of sample weights based on feature contribution.

(3) This thesis proposes a data sampling method based on sample importance weights to solve the problem that the selected samples for synthesis using data sampling methods in

defect prediction that are not representative. On the one hand, this method sorts submitted samples through sample importance weights, selecting important samples for synthesizing new ones; On the other hand, when calculating the sample distance, the importance weight of the samples will also be considered, resulting in a more diverse synthesized sample. Experiments on 450 tasks in this paper verify the effectiveness of the data sampling method that integrates time dimension and feature contribution.

This thesis proposes corresponding solutions to the three remaining problems in existing research, further improving the performance of just-in-time software defect prediction. It is of great significance for timely defect detection, improving code quality, scheduling testing resources, and reducing maintenance costs.

Keywords: software defect prediction, data preprocessing, sample weights, class imbalance

Written by Weiyuan Zhuang

Supervised by Xiaofang Zhang

目　录

第一章 绪论

1.1 研究背景及意义

随着软件规模的不断扩大和软件组件的日益复杂，软件质量保障变得至关重要，而如何识别软件中的缺陷是软件质量保证的重要一环。软件缺陷是指计算机程序或系统中的错误、缺陷、bug或故障，可能会产生不准确或意外的结果，或使软件无法按预期运行。软件缺陷对软件质量的影响极大，每年都有很多因为软件缺陷而造成的事故，往往给公司及政府带来严重的经济损失。由于测试资源（人员、时间、金钱等）有限，对软件进行详细完备的测试通常很难做到。因此，软件缺陷预测技术应运而生，它能在项目开发的早期阶段，帮助开发人员和测试人员识别哪些模块更具有缺陷倾向性，从而让他们能更加合理地分配测试资源，更加快速地定位缺陷，并识别常规测试难以发现的缺陷。

软件缺陷预测技术能够帮助合理分配测试资源。由于缺陷在软件各个部分中的分布是不同的，因此对软件系统的所有部分应用相同的测试工作并不是最佳方法。因此，通过软件缺陷预测技术，对缺陷倾向性更高的模块投入更多的测试资源，在实际开发中可以降低开发和测试成本。相关研究 [1]对软件缺陷预测技术对项目的实际贡献进行评估，发现公司需要大约6%的测试工作量来收集指标、清理数据和建模，却可以减少25%的测试工作量。除此之外，那些使用传统测试方法所能发现的缺陷也可以被缺陷预测模型所识别。

软件缺陷预测技术能够在项目开发的早期阶段识别缺陷。在项目开发生命周期中，越早发现缺陷，修复缺陷所需的成本越低 [2]。因此，软件缺陷预测技术被用在项目开发过程中来迅速地识别软件模块的缺陷倾向性，从而改善软件设计，有效避免在软件构建完成后再花费大代价去改正缺陷的风险。相关调查 [3]综合了36项研究的定量和定性结果，认为软件缺陷预测技术可以有效减少最终软件系统的缺陷数量，从而提高软件质量。一些研究人员不满足于仅在开发阶段构建缺陷预测模型，为了减少修复缺陷的成本，他们选择在需求分析和概要设计阶段建立缺陷预测模型，同样发现缺陷预测技术有助于在项目早期阶段识别缺陷 [4,5]。

软件缺陷预测技术能够识别常规测试难以发现的缺陷。不同于传统的白盒测试

和黑盒测试，软件缺陷预测技术利用缺陷之间的相似性来识别模块存在缺陷的风险。缺陷预测技术可以指导测试人员从其他角度寻找软件系统的潜在缺陷，对于软件质量保证有着积极的促进作用。

即时软件缺陷预测是软件缺陷预测中的一个子领域，它面向实际的项目开发工作环境，关注开发人员的某次提交所引起的变更会不会引入缺陷。由于现代软件开发是协作和敏捷的，具有持续交付和持续部署等流行趋势，旨在以更高的速度和频率构建、修复和发布软件，所以即时软件缺陷预测面向的软件变更集通常较小，开发人员可能也只需要较少的精力就能对变更集进行审查。因此，近年来即时软件缺陷预测技术越来越受到研究人员的重视 [6, 7]。

1.2　研究现状

软件缺陷预测模型识别软件系统中容易出现缺陷的实体(例如，文件、包、类和函数)。准确的预测模型有助于开发人员专注于检查可能有缺陷的模块，并有效地利用他们的时间和精力，因此引起了很多研究者的关注。本节将先介绍软件缺陷预测各个环节的国内外研究现状，然后介绍一下软件缺陷预测领域中数据预处理方法相关研究。

1.2.1　软件缺陷预测

软件缺陷预测技术一般分为以下四步：(1)构建数据集；(2)设计软件特征；(3)构建分类模型；(4)评估模型性能。因此，相关研究也往往集中于以上四个方面。

(1)构建数据集

软件缺陷技术通常需要软件故障信息库所提供的信息，这些信息被模型学习并预测当前开发的软件项目中的缺陷倾向性。软件故障数据集主要有三组信息：软件度量集、缺陷信息(模块是否有缺陷，缺陷模块含有缺陷的数量等)，以及关于软件项目的元信息(如提交信息)。目前，缺陷预测领域有一些公共数据集，比如PROMISE，NASA等，帮助相关研究者构建缺陷预测模型。

然而，这些数据集往往面临类不平衡问题 [8]，具体来说，数据集中有缺陷的模块少，而无缺陷的模块多，这容易误导预测模型，降低模型的实际预测性能。因此，软件缺陷领域研究者也应用各种类再平衡技术 [8–11]，希望可以缓解这个问题。其

中，Song等人 [11]为了解决以前研究在不同数据集上结果的矛盾问题，进行了大规模的实验。他们选择了17种流行的类再平衡技术，在27个数据集，7种特征和7种分类器上进行综合性的实验。结果表明不平衡问题越严重的数据集越能受到类再平衡技术的积极影响，但是如果不做任何考虑，随便地采用类再平衡技术，那么可能只会产生非常小的积极影响。

(2)设计软件特征

如何寻找软件缺陷，研究人员认为软件模块具有各式各样的特征，利用软件模块的这些特征可以区分有缺陷的模块和无缺陷的模块，因此软件特征至关重要。从特征生成的不同角度出发，我们可以将特征分为手工特征和语义特征。

手工特征指研究人员根据缺陷模块与无缺陷模块之间的差别，人为制定某种特征。例如，为了描述代码的复杂度而提出的McCabe度量 [12]，基于面向对象程序的CK度量 [13]等。除了这些静态代码度量，还有一些面向代码更改的过程度量。例如Nagappan与Ball [14]用相对代码变化作为度量，得到了比绝对代码变化更好的效果。在即时软件缺陷预测领域，Mockus和Weiss [15]提出了14种更改级别的度量，可以有效描述每次提交代码的变化。除此之外，也有研究人员将目光投入到开发者身上，如Nucci等人 [16]认为专注的开发人员比非专注的开发人员更不容易引入缺陷，基于此引入分散更改作为特征，也能有效识别有缺陷的模块。

语义特征是近些年来兴起的提取特征方式，基于代码本身，更加直观地描述软件模块。Wang等人 [17]提出基于代码令牌的缺陷预测模型，他们通过在文件级别和更改级别，以及项目内和跨项目场景下的大规模实验，验证了语义特征的有效性。在此基础上，Wang等人 [18]用代码的AST序列来表示代码的结构信息，并利用门控机制将语义特征和传统手工特征相结合，取得了更优越的效果。随着图神经网络的提出，Xu等人 [19]保留了代码的AST树形结构，用图神经网络捕捉缺陷子树的潜在缺陷信息并进行剪枝，取得了优于传统方法的结果。在过程度量方面，Wen等人 [20]通过细粒度的变更分析来提取六种类型的变更序列，利用软件演化过程中由变更产生的顺序信息识别缺陷。在即时软件缺陷预测领域，Pornprasit等人 [21]基于代码的令牌向量提出了一种快速模型，并能将缺陷定位到代码行。

(3)构建分类模型

为了学习软件模块的特征和预测模块是否有缺陷，研究人员需要挑选合适的分类模型。在软件缺陷预测领域，主要采用机器学习方法来构建预测模型。

Menzies等人 [22]进行了一项实验研究，以评估朴素贝叶斯与对数滤波器在软件故障预测中的性能，结果表明，所使用的技术在软件故障预测中表现出显著的准确性。随着深度学习的流行，相关研究者开始探究如何将深度学习与缺陷预测相结合 [23–25]。Yang等人 [26]用深度置信网络对特征进行再训练，然后用线性回归进行预测，把深度学习用于软件缺陷预测领域中。Fan等人 [27]采取注意力机制，学习特征中对模型贡献较大的部分，并用循环神经网络进行训练和预测。Zhou等人 [28]利用门控图神经网络，结合代码的AST树，控制流图，数据流图等信息，提高模型识别漏洞的能力。Zhu等人 [29]将在项目内和跨项目的场景下，利用去噪自编码器对特征进行去噪，接着用卷积神经网络收集特征的局部信息，成功提高了模型的性能。Tong等人 [30]在异构缺陷预测领域上，结合核谱嵌入、转移学习和集成学习，找到源和目标数据集的潜在公共特征空间，提高模型的性能。

(4)评估模型性能

在模型对软件系统中可能存在的缺陷做出预测后，研究人员需要对模型的性能做出评估，因此设计了一系列的评估指标。在软件缺陷预测领域，评估指标主要有两类 [31]，一类是非工作量感知评估指标，一类是工作量感知指标。

非工作量感知指标主要是传统的评估机器学习模型的相关指标，比如准确度、精确度、召回率、F1 [32]和MCC [33]等，这些指标被广泛应用于软件缺陷预测的相关研究中 [34–37]。

工作量感知指标则注重测试人员检查缺陷所需要的工作量 [38]。这是因为非工作量感知对于所有软件模块都一视同仁，但是不同的软件模块复杂程度不同，检查所需的工作量也不相同。因此，相关研究人员 [39]提出用代码行和模型预测的概率值来决定测试人员应该先检测哪个模块。由此，研究者设计了一系列相关的工作量感知指标。Jiang等人 [40]提出了PofB20，这个指标用于衡量当检查总代码行的前20%时，所能发现的缺陷比例；Huang等人 [41]提出了IFA，这个指标代表测试人员找到第一个缺陷时所审查的模块数，并报告当IFA过高时，测试人员将丧失对预测模型的信心。Mende等人 [42]提出了P_{opt}，这个指标当被用于按照缺陷概率密度排序

时，衡量预测模型与理想模型之间的相似度。

除了以上四个主要内容，相关研究者还从其他角度对缺陷预测技术进行研究。比如McIntosh等人 [43]对构建模型所用数据的时间进行研究，认为项目早期的数据可能会误导预测模型，建议每三到六个月重新构建预测模型。Li等人 [44]对跨项目上缺陷预测模型的参数优化进行研究，发现自动参数优化可以显著提高了缺陷预测技术的性能。Pornprasit等人 [45]基于局部规则的模型不可知技术，对缺陷预测模型的解释做出探索。

虽然在软件缺陷预测领域中的研究已经较为充分了，但是相关研究人员很少在意项目中所产生的样本的时间顺序关系，这些时间顺序关系可能也会对模型造成一部分的影响。例如，目前很多研究工作 [46–49]采用K-折交叉验证的方式来衡量模型性能，但是在实际环境下，测试人员只能依靠项目早期的样本来预测项目晚期的样本，这样会导致报告的性能与模型的实际性能有所偏差。虽然有部分研究工作 [43]指出样本的时间标签的重要性，但是如何利用时间维度来增强预测模型性能的相关方向尚未得到充分研究。此外，由于不同样本的质量不一样，会存在一些让预测模型难以正确分类的样本，如何加强模型对难分类样本的关注也需要进一步的研究。因此，本文通过对样本施加重要性权重，希望通过利用样本的时间维度和误分类样本的特征贡献度，来提高模型的预测性能。

1.2.2 数据预处理技术

在软件缺陷预测领域，数据预处理主要关注原始特征被提取之后到输入模型之前的这个阶段，对于特征来言，由于不同特征存在某些相关性或者是量纲的偏差，导致模型不能很好地进行学习；对于样本实例来言，可能存在噪声或者类不平衡，导致模型学习的精度降低，需要对样本进行采样。围绕上述两个问题，相关研究可能分为基于列的数据预处理技术和基于行的数据预处理技术。

(1)基于列的数据预处理技术

基于列的方法主要关注特征倾斜，特征共线和特征选择的问题。在特征倾斜方向，相关技术主要通过一些变换函数，对特征数据进行处理，防止某个维度的数据与其他维度数据在量纲上差距过大，使得模型学习出现偏差。比如Duan等人 [50]用标准对数变换ln(x+1)用于每个度量，为其在八个Apache开源项目的105828个变更进

行的实证研究提供了坚实基础。

特征共线方向，相关技术主要通过分析不同特征之间的关联程度，防止模型对某个单独特征产生误解。比如Fan等人 [51]对10个Apache开源项目的126526个变更进行大规模实证研究，认为对于两个相关变量应该保留有助于简化模型解释的变量。对于特征倾斜和特征共线问题，目前的研究工作主要集中在实证研究上。

特征选择技术用来选取最重要的特征，减少特征维度，防止维度灾难问题，从而提高预测模型的性能。Chen等人 [52]提出了一种基于特征选择和迁移学习的具有度量补偿的软件缺陷预测方法，有效缓解了跨项目缺陷预测上的数据分布差异过大的问题。Awotunde等人 [53]为了解决检测时间长、软件项目的脆弱性和特征参数的高维性等问题，提出了一种混合模型，使用数据归一化方法处理数据集不平衡和特征冗余的情况，采用支持特征选择的极限梯度增强分类器，提高了预测模型的性能。Bala等人 [54]提出了一种变换和特征选择方法，使用源项目的平均分布特征和模式分布特征，从标准偏差的三倍计算变换源的每个特征的距离，以减少跨项目缺陷预测中的分布差异和高维特征问题。

(2) 基于行的数据预处理技术

在基于行的数据预处理技术，采样方法是一种比较流行的策略方法，在该策略方法中，在模型构建之前重新平衡数据分布，以便学习的分类器可以以类似于传统分类的方式执行，采样方法独立于预测模型，易于观察，并忠实代表了研究对象的性质，主要分为欠采样和过采样两种途径。

欠采样通过消除大多数类样本来提取原始数据的子集，但主要缺点是这可能会丢弃潜在的有用数据。传统的随机欠采样方法(Random Under-sampling, RUS) 就是随机从多数类中挑选一部分样本，将它们从数据集中删去，保持数据集不同类的平衡，这种做法很容易丢失有效信息，导致模型欠拟合。一些研究者关注多数类的噪声问题，比如Wu等人 [55]提出了一种基于朴素贝叶斯的重采样方法，该方法评估多数类中每个样本的信息程度后，去除信息较少的多数类样本，以重新平衡数据分布。但是这种做法仍然会可能存在过度去除噪声，导致有效信息被丢失的问题。

过采样通过生成一些少数类样本来创建原始数据的超集，然而，这可能会增加过拟合的可能性。随机过采样(Random Over-sampling, ROS) 方法随机复制属于少数

类的样本，是过采样技术的最简单形式。然而，ROS只复制现有的少数类样本，这意味着它没有为分类器学习提供新的信息，因此可能导致预测模型的过拟合。合成少数类过采样(Synthetic Minority Over-sampling Technique，SMOTE)方法 [56]通过将某个少数类样本与先前定义的少数类最近邻样本相结合来生成新合成样本。因为SMOTE通过组合而不是复制样本来生成合成样本，所以它生成的样本比ROS更多样，但是仍然未能解决模型过拟合的问题。为了解决过拟合问题，相关研究者也做了很多工作，比如Bennin等人 [57]为了增强数据集的多样性，结合染色体配对思想，提出匹配最远缺陷特征点来生成新数据的MAHAKIL方法。Agrawal等人 [58]在更大规模的数据集上，使用基于SMOTE的改良方法，利用差分进化算法自动确定参数，也获得了很好的效果。在此基础上，Feng等人 [59]通过差分进化算法确定特征的权重，并将样本根据复杂度排序来生成新的样本，在保留数据的多样性的前提下，获得了更好的效果。

上述采样方法虽然已经从数据集的不同角度出发，获得了一定的效果，但是仍然存在一些可改进的问题，比如SMOTE方法生成样本太过近似，容易让模型过拟合；Bennin等人的方法生成的样本太过多样，容易混淆多数类样本和少数类样本；Feng等人的方法将样本的复杂程度等同于样本的重要程度，这并不十分直接。因此本文基于两种样本重要性权重，拉近重要样本的距离来合成新的样本，可以有效缓解以前研究的问题。

1.3 研究内容

虽然相关研究 [43]已经指出，对整个项目开发周期来言，用项目早期数据所构建的预测模型不能很好地预测项目晚期的模块是否具有缺陷，并且建议三到六个月就重新构建预测模型。但是目前仍然缺乏对于如何重新构建预测模型的研究，即如何平衡项目早期数据和新产生的数据，如何处理相近时间下的难分类样本，以及如何处理类不平衡问题。

本文将在更大的数据集上对相关研究 [43]的观点进行验证，并基于以上三个问题进行进一步的研究。具体地，本文基于模型迭代的场景下，设计了两种样本重要性权重和一种基于样本权重的数据采样方法，来进一步提高预测模型的性能。具体

研究内容结构如图 1-1所示。

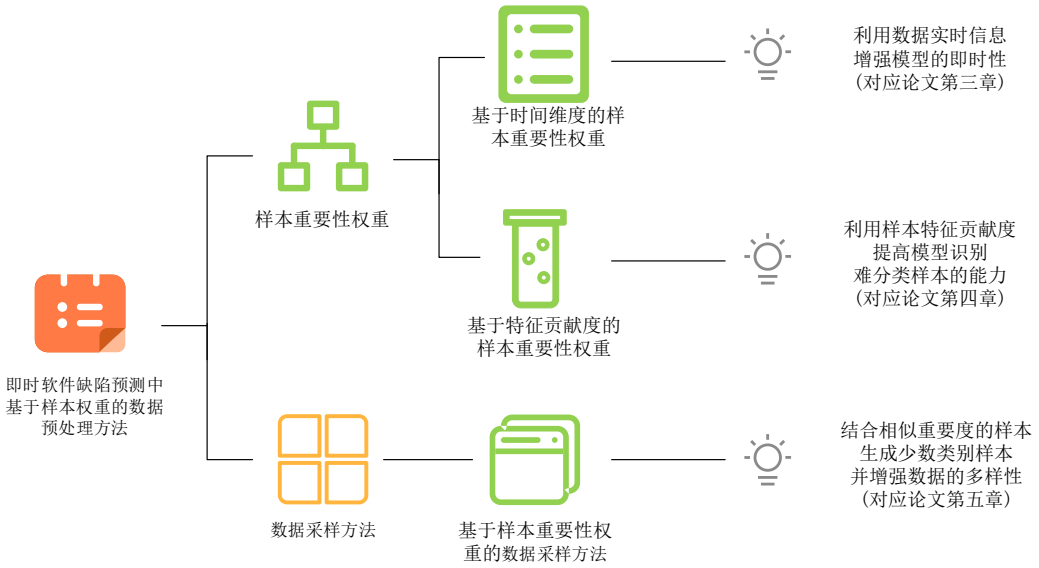


图 1-1 本文的研究内容结构

1.3.1 基于时间维度的样本重要性权重

在项目实际开发环境下，项目的故障数据库中存在一部分已经打好标签的历史数据，测试人员可以根据这些历史数据来构建缺陷预测模型，对新产生的样本进行预测。但是项目的开发周期很长，使用早期历史数据构建的预测模型往往不能准确判断项目晚期新产生的提交是否具有缺陷 [60]。此外，不同于其他领域，软件项目开发过程中所产生的样本具有时间顺序，在实际环境下，只能拿项目早期的样本来训练模型，以预测项目晚期的样本。并且，样本的这种时间标签对模型的性能也具有影响。相关实证研究 [43]指出，项目早期的数据可能会对模型产生误导，而最近新打上标签的数据对于模型性能有着积极影响。

为了充分利用提交序列的时间维度，让模型更多关注时间较新的提交，本文基于项目开发过程中缺陷预测模型不断迭代的场景下，提出一种基于时间维度的样本重要性权重(Time-based Weights, TBW)。该方法在数据预处理阶段，通过时间衰减函数，给新旧提交施加不同的权重，来使模型更关注新打上标签的样本。

1.3.2 基于特征贡献度的样本重要性权重

在软件工程领域中，由于实际开发环境的复杂性，所产生的不同样本质量参差

不齐，重要程度也各不相同。比如，代码行变动大的样本一般都含有缺陷，如果模型过多关注这些样本，很容易陷入仅用代码行的变动大小对样本做出预测，所以需要对多维特征进行综合考虑。此外，一些提交样本很容易被误分类，这类样本被称为难分类样本，需要模型给予更多的关注。因此，本文通过对样本施加另外的权重，让模型关注价值更高的样本。

为了区别时间相近的提交，以及让模型准确识别容易被误分类的样本，本文提出了一种基于特征贡献度的样本重要性权重(Contribution-based Weights, CBW)。该方法分为两个阶段，一阶段根据上一轮模型迭代所产生的预测标签和真实标签之间的差异，对被误分类的样本设置更高的权重；二阶段针对被误分类的样本，通过机器学习解释模型计算样本的特征贡献度和分类倾向性，根据分类倾向性来对与真实标签差异更大的样本施加更高的权重，最后还进行了去噪处理，从而有效提高预测模型的性能。

1.3.3 基于样本重要性权重的过采样方法

目前，在软件缺陷预测领域，存在较为严重的类不平衡问题。根据Pareto法则，80%的缺陷往往集中于20%的软件模块中，因此如何缓解类不平衡问题也是缺陷预测研究的一项重点内容。考虑到多样的数据能使得模型更能准确地识别不同阶段的样本，而传统的数据合成采样方法(如SMOTE)有以下几个方面的问题：第一，挑选中心样本过于随机，可能会挑选到噪声样本；第二，总是与最近邻样本合成新的样本数据，使得生成的数据太过近似，可能会导致模型过拟合，影响预测模型的性能；第三，对所有特征维度同等看待，不能反映不同特征的重要程度。

为了缓解上述三个方面的问题，本文提出了一种基于样本重要性权重的数据采样方法(Weight-base Oversampling, WBO)。该方法通过样本重要性权重来挑选中心样本，用扩增的特征矩阵来计算近邻样本，来合成新样本。该方法有以下三个方面的好处：首先，通过计算每个样本的两个权重来对样本的重要程度进行排序，只保留部分最重要的样本作为中心样本，这些样本更具有代表性；其次，通过连接样本的特征向量和权重的方式，拉近重要样本之间的距离，使得合成的数据更加多样化，从而有效提高预测模型的性能；最后，样本重要性权重有一部分是通过计算特征贡献度得到的，能够反映不同特征的重要程度。

1.4 论文组织结构

本文全文共六章，结构组织如下：

第一章简要介绍了软件缺陷预测技术研究背景以及意义，相关研究现状以及本文的主要研究内容。

第二章系统介绍了目前即时软件缺陷预测领域相关的背景知识。

第三章介绍了一种基于时间维度的样本重要性权重。该方法从项目的版本控制系统中提取时间维度信息，然后利用时间衰减函数赋予新提交的样本较高权重，旧提交的样本较低权重，针对性地提高缺陷预测的性能，在预测模型迭代的场景下，针对10个开源项目，每个项目挑选5个数据区间，9次迭代过程，共450个任务上进行了实验，验证了该样本权重的有效性。

第四章介绍了一种基于特征贡献度的样本重要性权重。该方法针对上一轮模型迭代过程中被误分类的提交样本，通过机器模型解释框架计算它们的特征贡献度和分类倾向性，对分类倾向性与实际类别偏差较大的提交赋予较高的权重，最后还进行去噪处理。本章进行了详尽的实验来验证该方法的性能表现。

第五章介绍了一种基于样本重要性权重的数据采样方法。该方法一方面通过样本重要性权重对提交样本进行排序，挑选重要的样本用于合成新样本；另一方面计算样本距离时也会考虑样本重要性权重，合成的样本更具有多样性，从而有效提高预测模型的性能。

第六章主要总结了本文所做的即时软件缺陷预测中基于样本权重的数据预处理方法研究的具体贡献，并就相关研究的进一步实施进行了展望。

第二章 背景知识

本章将对本论文涉及到的背景知识分三个部分进行简要介绍。首先介绍即时软件缺陷预测的基本流程，其次将介绍即时软件缺陷预测中的所必需的软件特征，最后将介绍类不平衡技术的相关概念。

2.1 即时软件缺陷预测基本流程

现代软件开发是协作和敏捷的，具有持续交付和持续部署等流行趋势，旨在以更高的速度和频率构建、修复和发布软件。由于这一增长趋势，近年来在缺陷预测的一个新的子领域，即时软件缺陷预测的相关研究越来越多 [61]，该领域旨在预测每次软件更改的缺陷可能性。

图 2-1展示了整个即时软件缺陷预测工作流程的概述。开发人员通常使用源代码管理系统或版本控制系统，来记录每次提交产生的软件更改，并通过缺陷追踪系统记录每次提交是否具有缺陷。接下来，数据采集步骤是将提交信息转换为原始变更数据(例如，从变更集计算软件变更度量)以及记录提交是否具有缺陷。从原始变更数据中，我们准备好用于构建模型的特征矩阵，通常将此步骤称为数据预处理。最后就是构建预测模型来识别新产生的提交是否有缺陷。

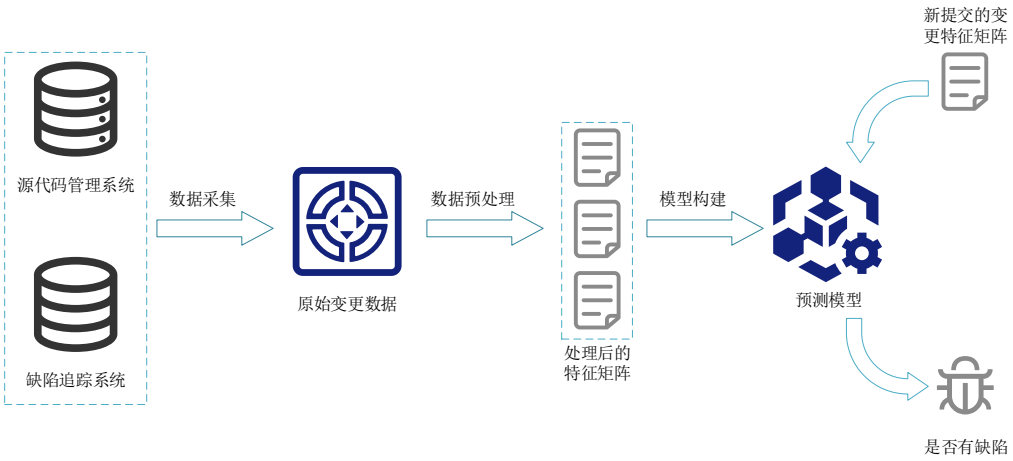


图 2-1 即时软件缺陷预测的基本流程框架

2.1.1 数据采集

数据采集包括特征提取和数据标注，特征提取指将代码变更信息提取为原始特征数据，数据标注指将软件变更标注为缺陷引入变更和非缺陷引入变更。特征提取将在第2.2节具体阐述，这一节主要阐述数据标注相关背景知识。

在即时软件缺陷预测领域，数据标注是非常具有挑战的一个研究领域，因为首先软件变更本来就是不直观，难以确定的 [62, 63]。软件变更是版本控制系统中软件的两个修订版之间的差异，在完成对软件项目的修改后，开发人员使用提交命令将更改记录到版本控制系统中，从而在版本控制系统中创建新的修订号。我们将软件更改计算为版本控制系统中两个连续修订的差异。讨论提取影响数据质量的软件更改数据集非常复杂，有各种不同的diff算法来计算两个文件之间的差异。表 2-1中列出了目前缺陷预测领域比较流行的四种diff算法。

表 2-1 四种软件变更的diff算法

算法名称	描述
Myers	从同一文件的两个版本的第一行开始依次扫描代码行，以找到彼此匹配的行对
Minimal	通过比较两个Myers对象，使得补丁大小尽可能小
Patience	注意标记行的最长公共子序列，关注出现次数最少但至关重要的行
Histogram	构建直方图，对代码行有序匹配，统计元素存在次数

其次，测试并没有向我们揭示哪种软件更改会在模块中引入导致测试失败的缺陷，那我们如何为软件更改分配缺陷标签或干净标签？为了解决这个问题，Sliwerski等人 [64]提出了一种基于缺陷修复变化自动识别缺陷诱导变化的算法，该算法现在被称为SZZ，以三位发明人的首字母缩写命名。SZZ 算法的一般框架包含了以下几个步骤：首先，检查版本控制系统中的提交日志消息或发布报告，通过在文本中搜索自然语言模式或者寻找发布报告的相关信息等方式来识别缺陷修复更改；其次，对于这些缺陷修复更改代码，通过diff算法，定位到这些缺陷的代码行；然后在更改历史中向后搜索，以找到引入缺陷的更改集；最后，进行去噪处理，丢弃提交时间晚于缺陷报告时间的代码变更。

2.1.2 数据预处理

用于构建即时软件缺陷预测模型的输入数据通常是一个向量，由特定变更集的特征值组成。这些向量是数据采集步骤的结果，多个变更集的向量集合形成一个矩

阵。在大多数情况下，矩阵数据（例如变更集的软件度量）在构建即时软件缺陷预测模型之前需要额外的转换。基于这样的矩阵，我们将转换技术分为两类：基于行和基于列。

基于行的方法主要是数据采样方法，分为过采样技术和欠采样技术。过采样技术通过增加多数类样本来平衡数据集，比如随机过采样方法。欠采样技术通过减少多数类样本来平衡数据集，比如随机欠采样方法。在缺陷预测领域中，欠采样不如过采样常见，因为从多数类中删除的样本可能包含可供分类器学习的重要信息。

基于列的方法主要关注特征倾斜，特征共线的问题 [50,51]。特征倾斜指软件变更数据的大多数特征都是高度偏斜的(例如，一些变量在0和1之间变化，而其他变量则高出几个数量级)，许多特征不是正态分布的。大多数统计和机器学习算法都期望自变量的值是标准化的，例如，流行的机器学习库，处理这一标准化问题的最流行方法是对软件度量应用对数变换。特征共线是指两个特征或者一个特征与其他多个特征相互关联。这会带来两方面的问题，一方面由于特征的共线，逻辑回归模型可能出现数值非唯一性，其中数值非唯一是指无法确定一组唯一的模型参数；另一方面相关特征也可能导致对单个特征在预测缺陷引起的软件变化方面的影响的误解。主要解决方法是使用相关性分析、独立性分析和冗余分析的组合消除相关或多重相关的变量。

2.1.3 模型构建

在即时软件缺陷预测领域，大多数模型是机器学习模型，关于缺陷模型的研究非常丰富，在下面做些陈述。

按数据是否有标签，可分为有监督模型和无监督模型。传统的有监督即时软件缺陷预测模型通常需要训练数据带有正确的标签，利用机器学习分类器比如逻辑回归、朴素贝叶斯、支持向量机、决策树和神经网络等对数据进行学习和分类，不过会存在由于数据标注较为困难，使得传到预测模型的数据往往存在噪声的问题。无监督即时软件缺陷预测是指利用未知标签变更数据构建预测模型的技术。最简单的无监督即时预测模型就是按代码行来分类，可以将代码变更较多的提交预测为有缺陷的，或者将代码变更较少的提交预测为无缺陷的。目前即时软件缺陷预测领域的无监督模型大多不复杂，但是在一些指标上许多简单的无监督模型比最先进的有监

督模型表现得更好。不过，一般来言，无监督模型并不比有监督模型更有竞争力。

按关注的任务不同，分为非工作量感知模型和工作量感知模型。非工作量感知模型将所有的提交同等看待，对于预测结果和模型性能的衡量，通常采取机器学习领域常用的性能指标,包括机器学习领域常用的精确度(precision)、召回率(recall)、$F1$和正确率(accuracy)等。工作量感知模型关注项目提交的实际复杂程度，观察到较小的模块在比例上更容易出现缺陷，应该首先进行检查，评估模型性能时通常采取即时软件缺陷预测领域研究者提出的工作量感知指标，比如关注检查前20%的代码行所能找到的缺陷比例的$PofB20$、软件测试人员在首次成功检查到缺陷之前所需要检查模块数的IFA等。

按实验的场景不同，分为项目内缺陷预测和跨项目缺陷预测。项目内缺陷预测是根据一个项目的历史软件更改和缺陷数据来训练预测模型，并使用该模型对同一项目进行缺陷预测或工作量感知预测。跨项目缺陷预测 [65]是根据一个或多个源项目的变化和缺陷数据建立预测模型，并使用该模型来预测目标项目的提交是否具有缺陷。由于不同项目数据往往差异很大，存在特征漂移等问题，因此跨项目缺陷预测的难度往往大于项目内缺陷预测。

2.2　软件特征

软件特征用来描述软件模块，帮助预测模型区分软件模块是否具有缺陷。在即时软件缺陷预测领域，研究者们提出了多种特征 [66]，包括基于变更元数据的特征、基于变更代码内容的特征、基于软件演进过程的特征和基于多源软件制品的特征。本节对这4种特征分别进行概述。

2.2.1　基于变更元数据的特征

变更元数据是指描述变更属性(例如开发者、提交时间、变更日志、修改文件、每个文件增加和减少的代码行数等)的数据，通常来说，缺陷预测研究者可以从项目的历史提交记录中直接获取这些变更元数据。表 2-2中列出一些常用的基于变更元数据的特征。在变更元数据基础之上，Mockus等人 [15]提出了变更增加减少行数、修改文件数量、开发者经验、变更类型等特征。由于这些特征显而易见，容易被理解和计算，具有普遍性，能够较好地反映项目的变更情况，因此被应用到大量的即时

软件缺陷预测研究中。

表 2-2 基于变更元数据的特征

特征名	描述
NS	修改子系统的数量 [15]
Entropy	在每个文件中分发修改过的代码 [67]
LA/LD	添加/删除代码行数 [14]
LT	更改前文件中的代码行 [68]
FIX	变更是否是缺陷修复 [69]
NDEV	更改修改文件的开发人员数量 [70]
NUC	对修改文件的唯一更改数 [67]
EXP	开发者的经验 [15]

2.2.2 基于变更代码内容的特征

上述基于变更元数据的特征是有一定的局限性的，主要在于没有考虑到变更的具体内容，因此，基于变更代码内容的特征被研究者提出。比如代码复杂度特征，对变更的具体内容进行详尽分析，并使用词频率信息来量化项目提交的变更内容。在语义特征兴起之后，与变更代码相关的特征也随之提出，比如基于变更的抽象语法树，分析相同节点类型数量的差值，用差值来表示变更的具体情况。这些特征都依赖于具体的变更代码，并可以与基于变更元数据的特征一起表征项目变更，构建即时软件缺陷预测模型。

表 2-3中列出一些常用的基于变更代码内容的特征。基于变更代码内容的优点在于所表征的信息更加直接和具体，有效反映了代码的变更情况，但其同样存在一些缺点，主要有以下三个方面：

(1)特征维度太多。在蕴含更多可能反映项目提交变更内容信息的同时，也会带来维度灾难的问题。有实证研究表明，过多的特征会对即时软件缺陷预测模型产生负面影响。对于这个缺点，即时软件缺陷预测研究者一方面引入深度学习，借助深度学习强大的学习能力来获取变更信息；另一方面，提出了特征选择技术，减少无效的特征数量。只保留重要的、必要的特征，可以显著提升预测模型的性能。

(2)计算复杂。计算基于变更代码内容的特征时，我们需要提取项目变更前后的代码文件的内容差异，有时需要分别对变更前后的代码文件构建抽象语法树，这样比较费时。然后还需要计算复杂度特征和词频率特征，与基于变更元数据的特征相

比，计算难度较大。

(3)泛用性不强。由于不同语言的抽象语法树有所区别，计算复杂度特征可能也会存在差异，因此基于变更代码内容的特征的实用性会受到影响。

表 2-3 基于变更代码内容的特征

特征名	描述
Complexity	变更相关文件在变更提交前后复杂度指标的差值 [71]
LogTF	变更日志的词频率特征 [71]
FileTF	变更文件的词频率特征 [71]
Structure	代码变更前后抽象语法树的差异 [40]

2.2.3　基于软件演进过程的特征

在传统的基于文件的缺陷预测中，相关研究者提出了基于软件版本的动态特征，受此启发，即时软件缺陷预测研究者提出了基于软件演进过程的变更特征，这些特征可以量化项目代码的修改历史量。

表 2-4中列出一些常用的基于软件演进过程的特征。在传统基于文件的缺陷预测研究中，基于软件演进过程的特征构建的模型很多时候比基于静态度量构建的模型能获得更好的预测效果。在即时软件缺陷预测研究中，也有研究者提出了一些基于软件演进过程的特征。Kamei等人 [72]提出从文件修改历史角度提取变更特征，例如变更相关文件被修改的次数、修改变更相关文件的开发者人数等。这些基于软件演进过程的特征，和基于变更代码内容的特征相比，一方面减少了特征的维度，避免了维度灾难的问题；另一方面，由于这些特征是从项目提交的历史信息中提取的，所以获取直接，计算简单，而且具有更强的普遍性，因此被后来的缺陷预测研究广泛运用。

表 2-4 基于软件演进过程的特征

特征名	描述
NDEV	对该相关变更文件进行过修改的开发者数量 [72]
NUC	相关变更文件被修改的次数 [72]
NFIX	相关变更文件被修复的缺陷数量 [72]
AGE	相关变更文件最近修改的时间与该修改时间差的平均值 [73]

2.2.4　基于多源软件制品的特征

上述3种变更特征的提取都是基于软件项目的代码版本控制系统。在软件项目的

开发过程中，除了版本控制系统之外，还会有其他的项目管理系统，比如，代码审查系统，代码测试系统等等。因此，相关研究者提出从多个项目管理系统中提取多维特征来表示项目的变更情况。

表 2-5中列出一些常用的基于多源软件制品的特征。McIntosh等人 [43]首次从项目的代码审查系统提取特征来表征代码变更。随着代码审查的兴起和代码审查技术的发展，这些基于多源软件制品的特征具有一定的泛用性，可以与代码审查领域的技术相结合，在未来的相关工作中得到更广泛的应用。

表 2-5 基于多源软件制品的特征

特征名	描述
Iterations	变更被合并到项目代码仓库之前被修正的次数 [43]
Reviewers	对该变更进行审查的人数 [43]
Comments	该变更审查意见数量 [43]
Review Window	该变更代码审查时间 [43]

2.3 类不平衡

在软件缺陷预测领域，经常遇到的一个问题是，真实项目的软件缺陷数据仅由少数有缺陷的组件和大量无缺陷的组件组成。因此，软件缺陷数据的分布是高度不均衡的，在机器学习领域被称为类不平衡数据。关于类不平衡问题，研究者们已经展开了大量的研究工作，现有的类不平衡算法主要有采样方法，代价敏感学习方法和集成学习方法等。本节对这3种方法分别进行概述。

2.3.1 采样方法

采样方法是一种数据预处理阶段的策略方法，在该策略方法中，在模型构建之前重新平衡数据分布，以便学习的分类器可以以类似于传统分类的方式执行，采样方法独立于预测模型，易于观察，并忠实代表了研究对象的性质，主要分为欠采样和过采样两种途径。

欠采样通过消除大多数类样本来提取原始数据的子集，优点在于可以去除多数类中的噪声，并且减少数据规模，使得机器学习模型学习更快，但主要缺点是这可能会丢弃潜在的有用数据 [55]。传统的随机欠采样方法(Random Undersampling, RUS)

就是随机从多数类中挑选一部分样本，将它们从数据集中删去，保持数据集不同类的平衡，这种做法很容易丢失有效信息，导致模型欠拟合。

过采样通过生成一些少数类样本来创建原始数据的超集，然而，这可能会增加过拟合的可能性。随机过采样(Random Oversampling, ROS) 方法随机复制属于少数类的样本，是过采样技术的最简单形式。然而，ROS只复制现有的少数类样本，这意味着它没有为分类器学习提供新的信息，因此可能导致预测模型的过拟合。合成少数类过采样(Synthetic Minority Over-sampling Technique，SMOTE)方法通过将某个少数类样本与先前定义的少数类最近邻样本相结合来生成新的合成样本。因为SMOTE通过组合而不是复制样本来生成合成样本，所以它生成的样本比ROS更多样，但是由于在基于SMOTE的过采样技术中使用了KNN算法，用于生成合成样本的样本距离太近，这仍然可能导致预测模型的过拟合。

2.3.2　代价敏感学习方法

代价敏感学习可以自然地应用于解决不平衡的学习问题 [74–76]。代价敏感学习不是通过子抽样来平衡数据分布，而是使用代价矩阵来优化训练数据，该矩阵定义了每个类别的不同错误分类代价。表 2-6中列出了一种简单的二分类代价矩阵。相关研究者使用代价矩阵，已经开发了许多代价敏感的学习方法，从代价矩阵应用的阶段出发，可以分为以下三个方面：

(1) 数据预处理阶段：将代价用于权重的调整，给多数类一个低权重，少数类一个高权重，这样模型会对少数类更加关注，从而缓解类不平衡问题。

(2) 模型学习阶段：研究者们将代价矩阵与机器学习模型如K最邻近，决策树，神经网络等相结合，分别提出了其代价敏感的版本，用于适配不平衡的数据集。以决策树为例，在选择决策阈值，选择分类标准和剪枝等方面都可以使用代价矩阵来提高对不平衡数据集的学习能力。

(3)模型分类阶段：在这一方面，研究者主要从贝叶斯风险理论出发，在获得模型的分类结果之后，对其进行进一步的处理。比如，先按照传统学习方法产生一个模型，设置损失最小目标函数来对结果进行调整，其优点在于它可以不依赖所用具体的分类器。

目前代价敏感学习方法存在的问题主要是所使用的代价矩阵需要自己定义，因此，为不同类别的样本分配适当的成本是一个需要解决的问题。

表 2-6 二分类代价矩阵

	真实正例	真实负例
预测正例	$C(0,0)=C_{00}$	$C(0,1)=C_{01}$
预测负例	$C(1,0)=C_{10}$	$C(1,1)=C_{11}$

2.3.3 集成学习方法

集合学习是增强泛化能力的基础，每个分类器都会出错，但不同的分类器已经在不同的数据上进行了训练，相应的错误分类样本不一定相同，所以综合多个分类器的结果可以提高模型的分类准确率。也正因为如此，集成学习被用于大量的研究工作中 [77–79]。最广泛使用的方法是Bagging和Boosting [80]，它们在各种分类问题中的应用带来了显著的改进。Bagging算法是一种随机抽样来构建训练集的方法，在类不平衡问题上，通常对少数类和多数据都随机挑选样本m次，这种行为重复n次，就得到了n个平衡的数据集，在这n个数据集上构建弱分类器，通过投票获得最终的预测结果。Boosting算法的步骤如下：首先从初试训练集训练出一个基学习器；接着再根据基学习器的表现对训练样本进行调整，使得先前基学习器做错的训练样本在后续训练中受到更多关注；然后基于调整后的训练样本继续训练下一个基学习器；如此重复进行，直至基学习器数目达到事先设定的值T，最终将这T个基学习器进行加权结合。

由于集成学习是在模型构建阶段，因此可以将集成学习与上述类不平衡技术相结合，以进一步解决不平衡数据分类的问题。其思想是将数据预处理技术嵌入到集成学习方法中 [81, 82]，以创建一个不平衡的集成学习器。例如，先用SMOTE方法对数据集过采样，生成一个平衡的数据集后，再采用Bagging算法或者Boosting算法得到更好的性能表现。

2.4 本章小结

本章主要介绍了即时软件缺陷预测领域相关的一些背景知识，首先介绍即时软件缺陷预测的基本流程；接着介绍了即时软件缺陷预测领域常见的几种特征；最后介绍了缺陷预测领域的类不平衡问题，以及各种类不平衡技术。

第三章　基于时间维度的样本重要性权重

这一章将在项目开发过程中预测模型迭代的场景下，基于更改级别的缺陷预测构建模型，并在模型构建的数据预处理阶段对样本施加基于时间维度的样本重要性权重。该方法可以有效利用项目样本的时间维度，通过让模型更关注较新提交的样本的方式提高模型的预测性能。

3.1　研究动机

即时软件缺陷预测旨在预测每次软件更改的缺陷可能性，通过项目的历史更改数据集构建训练模型，对新产生的提交进行预测。在实际的项目开发环境下，由于项目提交的先后顺序不同，所以提交样本具有明显的时间相关性。然而，即时软件缺陷预测领域研究者却很少关注项目提交的时间维度，最为典型的例子就是，仍然存在大量的研究工作 [46–48]采取K-折交叉验证的方式来对模型性能进行评估。在实际开发环境下，这种做法是不可能的，因为我们无法拿项目晚期的数据作为训练集来构建缺陷预测模型，并用于预测项目早期的数据。

除此之外，对于开发周期较长的软件项目，项目早期提交的风格往往与晚期提交的风格有着较大的差距，使得用项目早期数据构建的模型的性能表现往往会随时间推移而减弱。虽然相关研究 [43]已经指出，对整个项目开发周期来言，用项目早期数据所构建的预测模型不能很好地预测项目晚期的模块是否具有缺陷，并且建议三到六个月就重新构建预测模型。但是目前仍然缺乏对于如何重新构建预测模型的研究，即如何平衡项目早期数据和新产生的数据，如何利用项目开发的时间维度，以及预测模型何时应该更新等等。

为了解决以上几个问题，本文将在更大的数据集上对相关研究 [43]的观点进行验证，并提出一种基于时间维度的样本重要性权重，在模型迭代的场景下，进一步提高预测模型的性能。

3.2　研究方法

首先我们将定义如何在即时软件缺陷预测领域中预测模型迭代的过程。如图3-1所示，对于项目内缺陷预测来言，一般已经有一部分历史故障信息(比如图 3-1中

的[Commit_1, Commit_2, Commit_3, Commit_4 ... Commit_n])，可以用它们来构建缺陷预测模型，在对新产生的未知样本进行预测后，交由测试人员进行审查，并打上标签，这样就会出现一批刚打上标签的样本(比如图 3-1中的[Commit_n+1, Commit_n+2, Commit_n+3, Commit_n+4 ... Commit_m])。这些样本在被测试人员检查之后，将被输入模型中来更新训练集和预测模型，这样，更新后模型又能对新产生的一批无标签样本(比如图 3-1中的[Commit_m+1, Commit_m+2, Commit_m+3, Commit_m+4 ... Commit_o])进行预测。

为了合理利用历史样本以及刚被打上标签的样本，构建更加准确的模型，每有一批新的刚被打上标签的样本，我们将用历史数据和新的有标签样本对模型进行重新训练，并对待测试样本进行预测。此时，被纳入训练的有标签样本都将成为故障信息库中的历史样本。随着项目继续开发一段时间，测试人员又会对新一批的样本打上标签，同时也会产生新一批待测试数据，这样不断重复之前的步骤，模型也会不断更新。

除此之外，本文还利用了样本的时间维度来使模型能更多关注重要的样本，具体来说，我们首先进行项目划分，从项目中挑选出活跃的数据区间；其次，从这些数据区间的软件变更中提取出所有样本的特征；接着，我们对特征进行初步的数据规范化；然后，我们用时间衰减函数对样本施加基于时间维度的样本重要性权重；最后，将这些样本及其权重输入到机器学习模型中进行训练。

3.2.1 项目划分

本文所采用的数据集是Github上的开源数据集。由于在项目的实际开发中，项目的提交的活跃程度随着项目进度的变化而变化。在有新功能需求的时，项目开发较为活跃，提交较为频繁；而无新功能需求时，项目开发进入沉寂期，提交频率降低。如图 3-2所示，以ant项目为例，从直方图中明显看出ant项目的提交数量是随着天数而波动的，具有明显的峰顶和谷底，峰顶表明项目在这一段时间进入活跃期，谷底则表明项目在这一段时间进入沉寂期。为了排除项目的活跃程度带来的干扰，本文从每个项目中寻找提交最频繁的5个阶段，这样保证提交的数量是足够的同时，也能保证数据的一致性和稳定性。

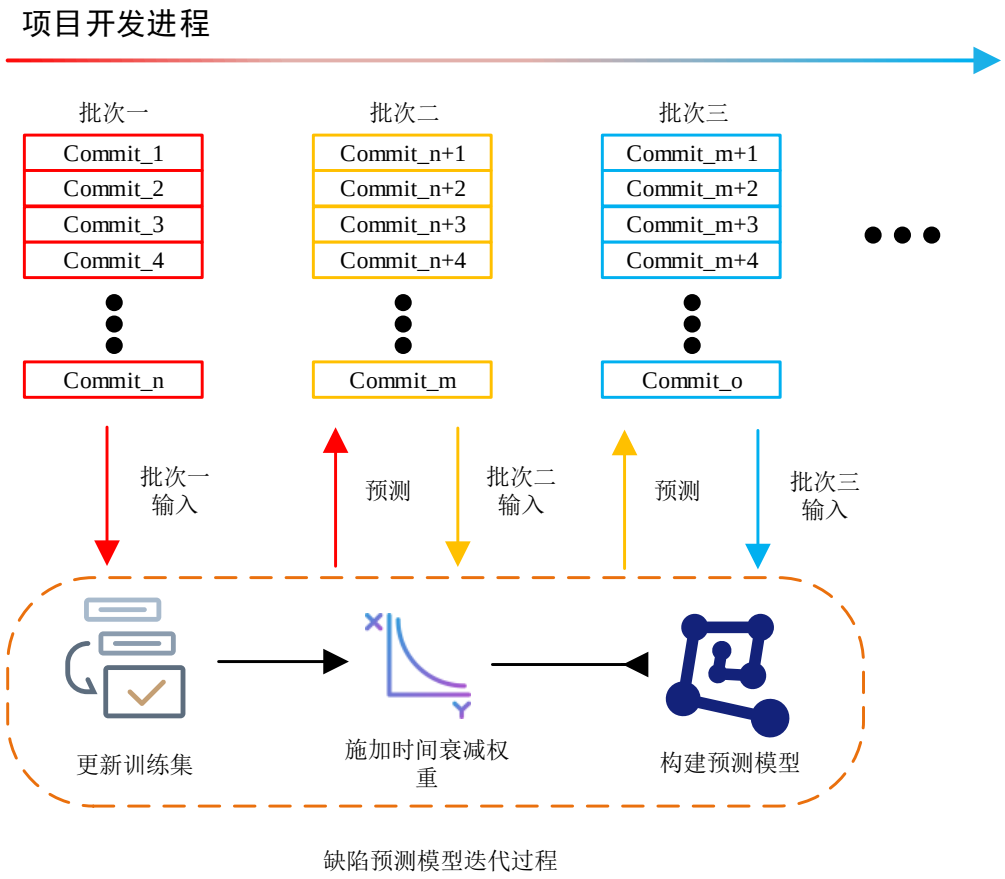


图 3-1 即时软件缺陷预测中模型迭代过程

3.2.2　特征提取

本文所采用的是Kamei等人 [73]提出的14个特征。如表 3-1所示，这些特性基于软件演化过程，从由于项目提交所产生的变动的影响、大小、目的、历史信息和开发者的经验等五个维度去描述提交所具有的缺陷倾向性。这些数据的大小是与提交的缺陷倾向性紧密相关的，因此它们被广泛应用于即时软件缺陷预测中 [26,29,83,84]。

3.2.3　数据规范化

我们对数据集进行数据规范化，以消除不同度量的不同权重对预测模型和复杂度计算的负面影响。在应用数据规范化后，所有样本的度量都会落入一个类似的值范围，从而消除对任何特定度量的偏见。在本研究中，我们采用最小-最大归一化方法，将所有值调整到0到1的范围内。

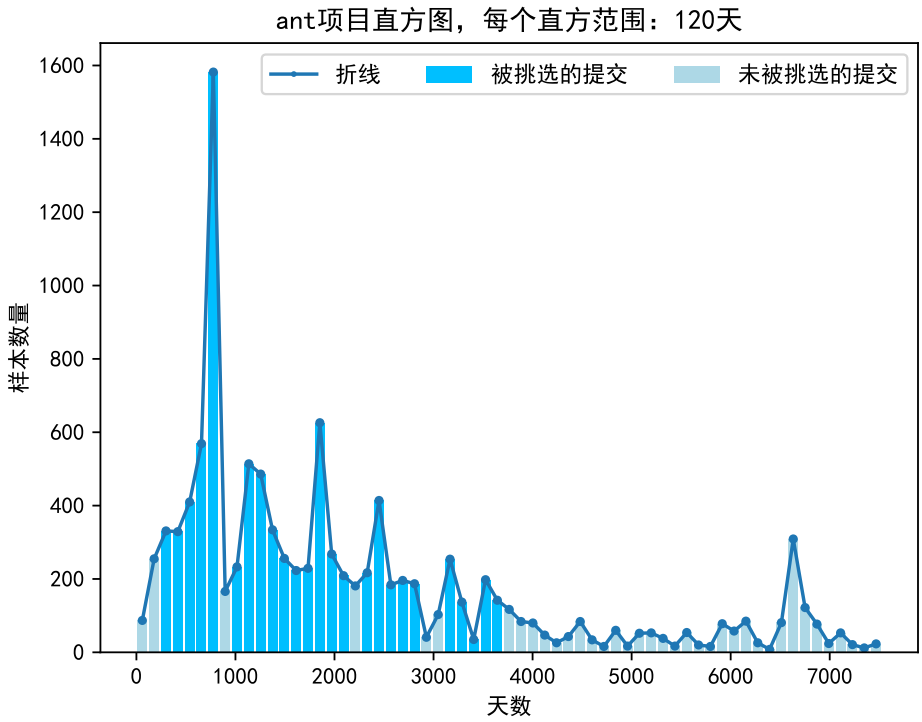


图 3-2 从项目中寻找提交活跃期

3.2.4 计算样本的时间权重

除了特征复杂度之外，我们还利用样本的时间维度。具体来说，我们希望模型更重视接近现在时间的样本，为此，我们对样本加上一个时间权重，这个权重由时间衰减因子所控制，这样越靠近现在时间的样本，它的时间权重越高，会获得模型更多的关注；越是项目早期的数据，它的时间权重越低，对模型的影响越小。我们设计了三种不同的时间衰减函数，分别为线性衰减函数，指数衰减函数和高斯衰减函数，它们的表达式如下：

线性衰减函数：

$$f(T) = W_{min} + \frac{W_{max} - W_{min}}{1 + \sigma|T - t|} \tag{3.1}$$

指数衰减函数：

$$f(T) = W_{min} + \frac{W_{max} - W_{min}}{e^{\sigma|T-t|}} \tag{3.2}$$

表 3-1 14种更改特征的描述

特征类别	特征名	描述
Diffusion	NS	修改子系统的数量 [15]
	ND	已修改目录的数量 [15]
	NF	修改文件的数量 [85]
	Entropy	在每个文件中分发修改过的代码 [67]
Size	LA	添加代码行数 [14]
	LD	删除代码行数 [14]
	LT	更改前文件中的代码行 [68]
Purpose	FIX	变更是否是缺陷修复 [69]
History	NDEV	更改修改文件的开发人员数量 [70]
	AGE	上次更改和当前更改之间的平均时间间隔 [70]
	NUC	对修改文件的唯一更改数 [67]
Experience	EXP	开发者的经验 [15]
	REXP	最近的开发人员的经验 [15]
	SEXP	开发人员在子系统上的经验 [15]

高斯衰减函数：

$$f(T) = W_{min} + \frac{W_{max} - W_{min}}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{T-t}{\sigma})^2} \tag{3.3}$$

其中$f(T)$指目标样本的时间标签为T时，时间衰减因子的值；W_{min}和W_{max}分别代表样本的最小权重和最大权重。在本方法中对样本权重做归一化处理，即W_{min}设为0，W_{max}设为1；σ为衰减常数，与项目所挑选的活跃期长度有关；T表示目标样本的时间；t表示预测模型本次迭代的时间。目标样本时间越久远，T越小，$|T-t|$越大，而$f(T)$越小。

3.2.5 模型训练

生成时间权重之后，我们构建即时软件缺陷预测模型，每次重新训练模型时，以样本的时间权重作为训练时的样本权重，让模型注重学习较重要的样本。在模型构建上，我们选取随机森林(Random Forest, RF) 和逻辑回归(Logistic Regression, LR)作为分类器，这两种分类器常用于软件工程实证研究工作的建模中 [86, 87]。

3.3 实验设置

这一节主要介绍本次实验中的实验数据集和描述、对比实验设置、性能评价指标等相关设置。

3.3.1 实验数据集

本文所挑选的项目来自Apache和Github上的开源项目，这些项目开发时间较长，足以支撑起本文的研究工作，并且也在软件工程领域被研究人员所广泛使用 [26,29,73]。它们的名称、描述、总时长、所挑选的活跃期长度、提交总数以及缺陷比例如表 3-2所示。项目的总时长在2至20年之间，因此有不同时间跨度的项目可以更加全面地反映模型对不同项目的适配性。有7个项目期限超过10年，说明这些项目都具有足够长的时间跨度并产生足够多的提交用于我们的研究和实验。项目总提交的数量在8845到49927之间，所挑选的活跃期长度在5至20个月之间。这些数据都意味着这些项目足以进行实证研究。缺陷率在9%到26%之间，这意味着在模型训练过程中会出现类不平衡的问题，导致模型的预测性能的评估可能出现偏差，为了保证实验公平，本章的所有方法都没有采用类不平衡技术，这样实验结果的有效性是有保障的。

为了获取这些项目在开发周期中的每次提交产生的变更，以及变更所对应的特征，我们采取比较成熟的一种对提交的分析和风险预测技术 [88]来提取项目开发过程中产生的提交，并生成如表 3-1所示的各类度量。这个技术已经被广泛应用于各类对项目代码提交的研究当中 [89–91]，受到广泛认可，生成的数据集具有较高的可靠性。

另外需要注意的是，由于我们对每个项目都挑选了5个活跃期来作为缺陷预测的任务数据，因此总共可以划分出50(5*10)个任务数据集，在每个数据集上我们依据时间进行划分成10个时长相等的阶段，在这10个阶段上进行模型迭代，第一个阶段作为历史数据集，剩下九个阶段在每次模型迭代中依次输入训练集，即在每个数据区间上，预测模型迭代9次，这样共生成了450个缺陷预测任务。

3.3.2 对比方法

为了能更充分地展现本文提出的方法的性能表现，这一节将选取以下方法作为对比：

(a)TRD (Tradition) [49]: 每次模型迭代数据集不会更新，仅使用项目早期数据去预测项目晚期数据的模型。

表 3-2 数据集描述

项目	描述	总时长(月/年)	活跃期长度(月)	提交总数	缺陷比例(%)
ambari	管理Hadoop集群的工具	09/2011-06/2020	10	24588	22
ant	基于java的构建工具	01/2000-08/2020	20	14667	24
aptoide	Android App Store客户端	04/2016-11/2020	5	12981	22
camel	一个强大的开源集成框架	03/2007-02/2021	15	49927	23
cassandra	高度可扩展的分区行存储	03/2009-01/2021	10	25936	14
egeria	开放的元数据和管理类型系统	06/2018-02/2021	5	12707	9
felix	一个实现OSGi R4规范的项目	08/2005-03/2020	15	15556	19
jackrabbit	JCR的实现	09/2004-08/2020	15	8845	20
jenkins	开源自动化服务器	11/2006-08/2020	15	29447	17
lucene	开源搜索软件	09/2001-08/2020	20	34317	26

(b)Base [37]: 每次模型迭代使用项目的所有历史数据作为训练集，去预测新的阶段的样本是否有缺陷。

(c)SP (Short Period) [43]: 每次模型迭代仅使用最新阶段的数据作为训练集，其他历史数据均抛弃掉。

(d)TBW (Time-based Weights)+liner: 在Base模型的基础上使用线性衰减函数作为样本权重系数。

(e)TBW+exp: 在Base模型的基础上使用指数衰减函数作为样本权重系数。

(f)TBW+gauss: 在Base模型的基础上使用高斯衰减函数作为样本权重系数。

3.3.3 评价指标

本文主要采取非工作量感知场景下的性能评价指标，非工作量感知场景指开发或测试人员对缺陷预测模型给出的结果进行人工核查时，相关资源能够满足对所有被报告有缺陷的测试样本均进行核查的场景。在这样的场景下，用于缺陷预测任务的性能评价指标应该与一般的分类任务的评价指标相同。因此，本研究选择了在分类任务中广泛使用的*Acc*，*F*1和*Mcc* [33]作为本文的性能评价指标。

*Acc*是评价机器学习模型性能的最常见指标，*F*1则是*Precision*和*Recall*的调和平均值。在缺陷预测模型的性能表现中，较高的*Precision*值表明模型预测为有缺陷的样例中，真正含有缺陷的比例越高，这将减少浪费在检查假阳性样本的相关资源；较高的*Recall*值则表明模型能够识别出更多确实含有缺陷的样例，通常这对于安全攸关的软件来说极为重要。然而，*Precision*和*Recall*都有各自的局限性，并不能全面地反映模型的性能表现。例如，一个输出全为*True*地模型的*Recall*值总是1，但这样的

预测结果不具有参考意义。因此，本文选取了$F1$来均衡$Precision$和$Recall$，并作为本文工作的评价指标。

虽然$F1$广泛用于软件缺陷预测领域，但一些研究 [11]指出由于$F1$忽略了$True\ Negative$，它仍然无法很好地评估不平衡域下的性能。因此在本文中，我们选取了马修斯相关系数(Matthews Correlation Coefficient, MCC) [33], 一个用于衡量不平衡数据下的模型性能的度量 [92]，来更全面地评估本文方法的性能表现。MCC指标考虑了真阳性(True Positive, TP)、真阴性(True Negative, TN)和假阳性(False Positive, FP)和假阴性(False Negative, FN)，具体定义如下:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.4}$$

MCC展现标准互信息度量算法结果与标准结果之间的相似度，取值范围为[-1，1]，其中1表示模型预测的分类和实际分类完全正相关，0表示无关联，-1表示完全负相关，MCC值越大说明模型效果越好。

最后，为了排除随机因素的干扰，我们在每个任务上的实验都重复三十次，来获得更加可靠的实验结果。

3.4 实验结果和分析

在这一节，本文将本文方法与其他方法进行对比，并对实验结果进行分析。

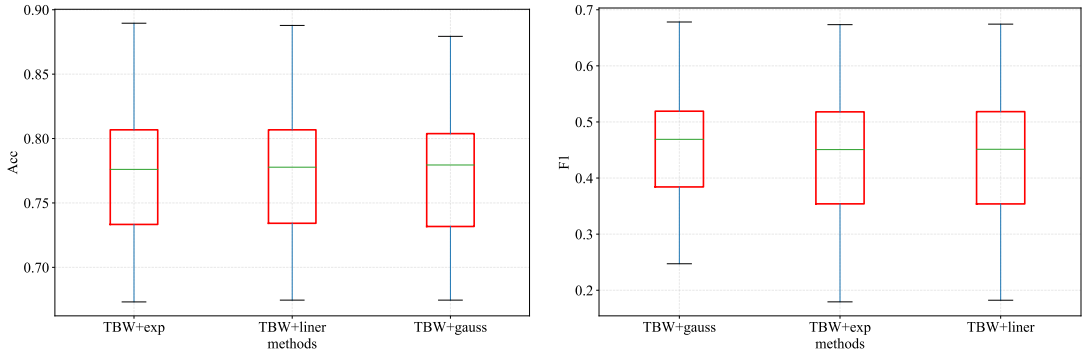
3.4.1 三种时间衰减因子的性能比较

在本小节中，我们探讨三种时间衰减因子的性能比较。三种时间衰减因子各有不同，线性衰减因子的衰减速率总是一致的，指数衰减因子的衰减速率是越来越大，高斯衰减因子的衰减速率是先增大后减少，具体哪一种衰减因子更适合即时软件缺陷预测，我们在50个数据区间上进行实验来验证。对于每个时间衰减因子参数$sigma$，都调节到使得预测模型性能最优。最后我们会选取最优的时间衰减因子作为本文工作的方法与其他方法进行比较。

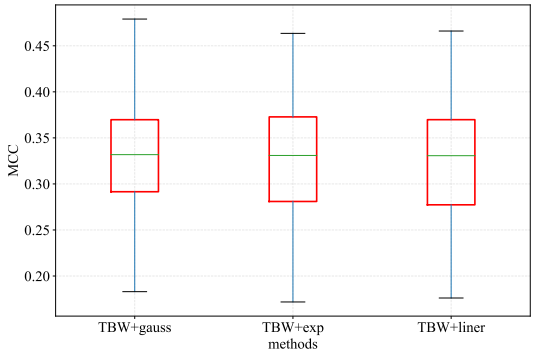
如图 3-3所示，我们采取SK(Scott & Knott)图 [93]来表现不同方法之间的性能差异。在图中，模型根据预测性能从左到右排序，性能相当的模型的框用相同的

颜色表示。从图中可以看出，在指标*Acc*上，TBW+exp性能最高，平均值为0.775，TBW+gauss的性能最低，为0.772。而在*F*1和*MCC*上，TBW+gauss的性能最高，分别为0.459和0.331，TBW+liner的性能最低，分别为0.441和0.325。不过三种函数的图像颜色相同，说明当参数最优时，这三种衰减因子的性能差异并不大。

从综合考虑，TBW+gauss在两个指标上的性能都是最优，而在*Acc*仅与最优相差0.003，因此本文在与其他方法进行对比时，都采用高斯衰减函数作为本章所提方法的时间衰减因子。



(a) Acc上的性能差值　　(b) F1上的性能差值



(c) MCC上的性能差值

图 3-3 三种时间衰减系数的性能差异

3.4.2　与对比方法的性能比较

在本小节中，我们将本文提出的基于时间维度的样本重要性权重与其他方法进行比较，它们的性能如图 3-4，图 3-5和图 3-6所示。对于指标*Acc*，模型按照性能从优到劣进行排序，分别为TBW+gauss、Base、SP以及TRD，平均值分别为0.772、0.769、0.755和0.733；对于指标*F*1，模型按照性能从优到劣进行排序，分

别为TBW+gauss、TRD、SP以及Base，平均值分别为0.459、0.430、0.429和0.425；对于指标*MCC*，模型按照性能从优到劣进行排序，分别为TBW+gauss、Base、SP以及TRD，平均值分别为0.331、0.306、0.287和0.272。

从总体上来看，本文提出的TBW+gauss总是在图的最左边，并且平均值也是最高的，表明在所有指标上，TBW+gauss总是最优的，这说明了我们提出的基于时间维度的样本权重是有意义的。此外，传统的即时预测模型TRD的性能表现不佳，说明了模型迭代本身对即时软件缺陷预测也是具有重大意义的。对于SP，我们将其纳入对比方法是因为有研究人员 [43]只用最新的数据作为训练集，结果要比把所有数据都作为训练集效果要好；不过从我们的实验结果来看，似乎并不总是这样，有历史数据信息的模型往往表现更好。而Hoang等人 [46]的研究和我们的实验结果相似。这说明不能简单地抛弃项目早期的数据，因此我们采取时间权重的方式，只让模型重点关注比较接近当前时间的数据。

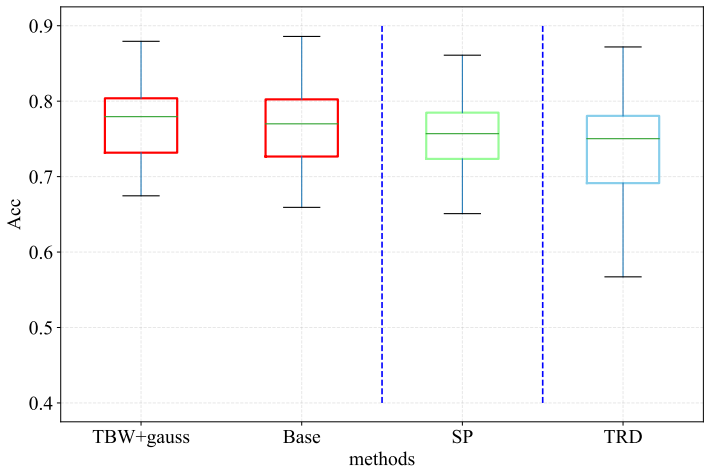


图 3-4 基于时间维度的样本重要性权重与其他方法在Acc值上的性能差异

3.4.3　参数实验和讨论

这一节主要讨论在即时软件缺陷预测领域，窗口期的大小对模型性能的影响。在这里我们以两个元素分别作为窗口期：固定时长和固定样本数量。固定时长是由于在即时软件缺陷预测中，项目的提交具有明显的时间顺序，包括本文的研究是以时间权重作为研究内容，因此时长是值得关注的一个元素。固定样本数量是因为，在即时软件缺陷预测研究中，仍然存在大量的研究工作是以样本数量作为训练集和

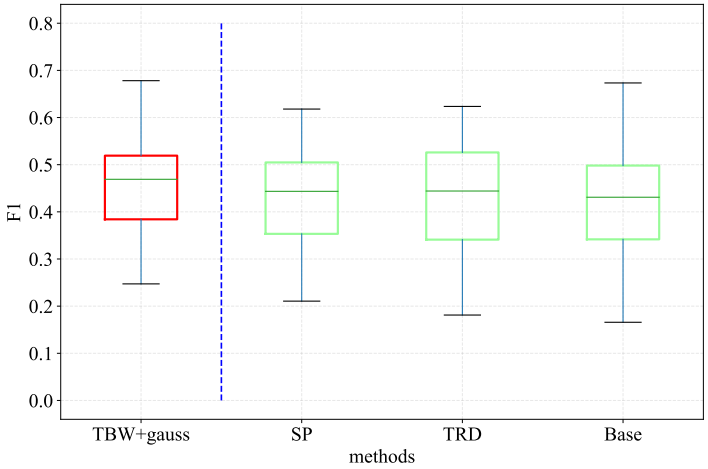


图 3-5 基于时间维度的样本重要性权重与其他方法在F1值上的性能差异

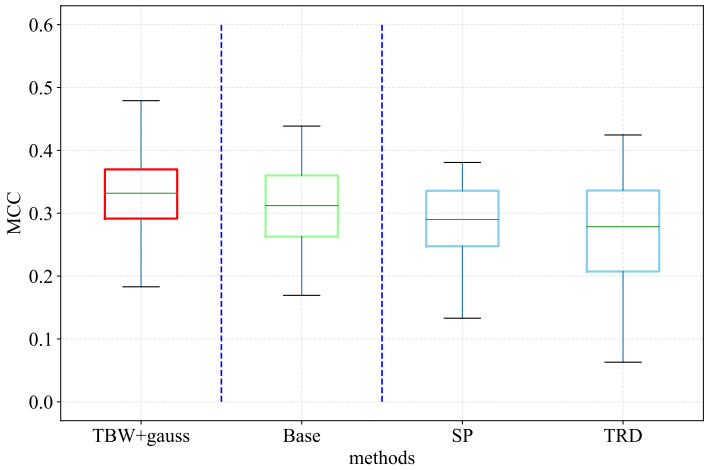


图 3-6 基于时间维度的样本重要性权重与其他方法在MCC值上的性能差异

测试集的划分依据，即以固定样本数量作为训练集，剩下的作为测试集，因此本文也将固定的样本数量作为关注的一个元素。

在本参数实验中，由于任务数据集的大小是固定的，因此窗口期越大，每次模型迭代模型时输入的数据越多，迭代的次数越少。反之，窗口期越小，输入的数据越少，迭代次数越多。

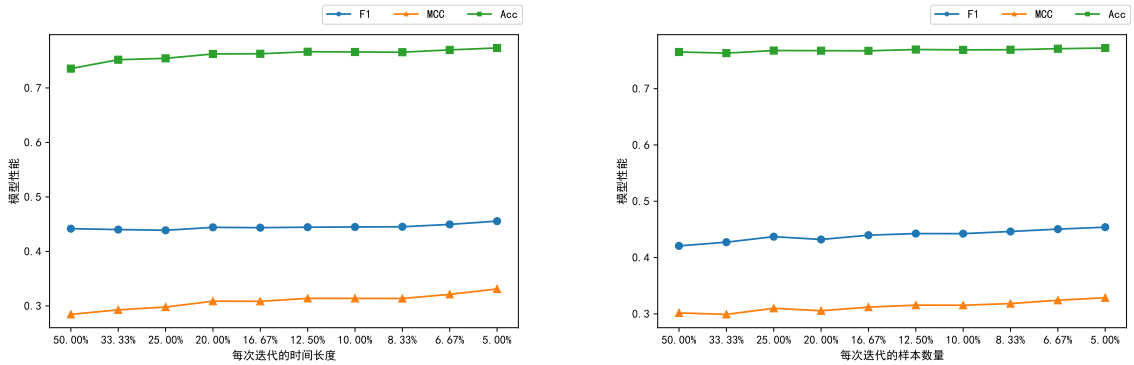
(1)按照时间划分窗口期

对于时间窗口期，我们按比例进行划分。如图 3-7 (a)所示，我们将缺陷预测模型每次迭代的时长分别划分为总时长的50.00%、33.33%、25.00%、20.00%、16.67%、12.50%、10.00%、8.33%、6.67% 和5.00%。值得注意的是，当时间窗口期为50%时，

相当于拿项目早期50%时长的数据，去预测项目晚期50%时长的数据，这样我们实际上并没有对预测模型进行迭代，训练集和测试集的划分将与传统的即时软件缺陷预测方法类似。但是与之不同的是，为了确保参数实验的前后一致性，我们仍然对训练集施加时间权重。

从图 3-7 (a)可以看出，对于*F*1,*MCC*和*Acc*，这三个指标的性能都会随着窗口期的减小而增大，这意味着在同样的数据集上，模型迭代的次数越多，模型的性能越好。从图中可以看出，模型性能在横坐标为8.33%至5.00%的范围内变化幅度比12.5%至8.33%大一些，这是因为横坐标为5.00%时，划分为20个阶段；横坐标为8.33%时，划分为12个阶段；横坐标为12.5%时，划分为8个阶段。因此8.33%至5.00%多了8个阶段，12.5%至8.33%只多了4个阶段；这也使得横坐标在8.33%至5.00%上升比较明显。

这个结果对于实际项目中进行即时软件缺陷预测是具有实际意义的，结果表明，在测试资源允许的情况下，我们应该尽可能频繁地去更新预测模型，以获得更高的预测性能。在本文中，考虑到我们在50个数据区间上的实验工作量较为庞大，受到服务器性能的制约，我们采用10%作为预测模型迭代的窗口期，而不是最高的5%，但即便如此，我们的方法也比其他方法更加优秀。



(a) 按照时间划分窗口期　　(b) 按照样本数量划分窗口期

图 3-7 参数实验结果

(2)按照样本数量划分窗口期

在相关即时软件缺陷预测研究中 [46–48]，一些研究人员以样本数量作为划分依

据，比如用固定比例的样本数量作为训练集，去预测剩下的样本；或者采取X折交叉验证的方式。另一方面，考虑到在实际开发过程中，测试人员可以在提交的数量足够多时来更新缺陷预测模型，这样的做法也是具有实际意义的。即项目活跃的时候预测模型更新会变得较为频繁，项目不活跃的时候预测模型更新会变得较为缓慢。因此我们也把样本数量作为窗口期进行实验研究。

与时间窗口期类似，我们将缺陷预测模型每次迭代的样本数量分别划分为总样本数量的50.00%、33.33%、25.00%、20.00%、16.67%、12.50%、10.00%、8.33%、6.67% 和5.00%。在每次模型迭代学习的过程中，我们也同样按照本文的方法对不同的任务给予相应的时间权重。结果如 3-7 (b)所示，实验结果随着窗口期的减小而增大，这意味着以样本数量来划分窗口期时，缺陷预测模型的性能也同样随着迭代次数的增大而增加。

以上的窗口期参数实验表明，在数据集时长和总样本数量一定的情况下，预测模型迭代的越频繁，模型性能越高。这意味着在实际项目开发过程中，测试人员应该频繁地更新预测模型，而不是像传统即时软件缺陷预测那样，只用项目早期的数据作为训练集，去预测项目晚期的数据。

结合以上两个图可知，当数据集大小相同时，窗口期越小，模型迭代的越频繁，模型性能越强。这证明了模型迭代在即时软件缺陷预测领域的有效性和必要性。对于项目开发过程中，具体应该多长时间来更新缺陷预测模型，根据我们的实验结果以及相关的研究 [43]，我们推荐每隔3或6个月就应该更新一次缺陷预测模型，当然，如果测试资源允许，去更加频繁地更新缺陷预测模型，也能获得更好的预测性能。

(3)本章方法的时间开销

由于本章方法在基础方法上添加了时间权重，考虑到测试资源是有限的，因此本章所提出的权重的时间开销是值得考虑的事情。如表 3-3所示，表里列举了三种时间衰减因子在每个项目上生成时间权重所需时间开销之和。从表中可以看出，在时间开销最大的项目lucene上，三种时间衰减函数的时间开销都小于2毫秒，因此本章权重的时间开销是完全可以接受的；由于时间权重的生成时间开销只与样本数量相关，因此样本数量越多的项目所需要的时间开销越大，考虑到模型训练的时间同样也随样本数量成正比，而本章方法的时间开销在样本数量相同的情况下远小于模型

训练的时间，因此可以认为本章方法所提出的权重并不会造成很多额外的测试资源花费。

表 3-3 三种时间衰减函数的时间开销

项目	TBW+liner(毫秒)	TBW+exp(毫秒)	TBW+gauss(毫秒)
ambari	0.97	0.97	0.94
ant	0.56	0.55	0.52
aptoide	0.44	0.43	0.44
camel	0.96	0.94	0.93
cassandra	0.81	0.79	0.82
egeria	0.66	0.68	0.67
felix	0.59	0.60	0.60
jackrabbit	0.34	0.36	0.34
jenkins	1.10	1.05	1.10
lucene	1.29	1.20	1.22

3.5 本章小结

本章在即时软件缺陷预测领域，基于少有人关注的时间维度，探究提交的时间对预测模型的影响，以及如何去利用时间维度来增强即时预测模型。本文设计了三种不同的时间衰减因子，在10个项目，每个项目5个数据区间上，9次模型迭代上进行实证研究，通过三种评价指标评估模型性能，并将本文的方法与其他方法进行比较，最终的实验结果表明了模型迭代的必要性，并且验证了本文所提出的基于时间维度的样本重要性权重的有效性。接着，我们还对时间窗口期和样本数量窗口期对缺陷预测模型迭代的影响进行了实证研究，结果表明，加快预测模型的迭代频率可以在总体上获得性能的提升。最后，我们还对方法的时间开销进行分析，表明我们方法在时间开销上并没有太大花费。

第四章　基于特征贡献度的样本重要性权重

这一章将在项目开发过程中预测模型迭代的场景下，基于更改级别的缺陷预测构建模型，并在模型构建的数据预处理阶段对样本施加基于特征贡献度的样本重要性权重。该方法关注易被误分类的样本，通过计算被误分类样本的特征贡献度和分类倾向性，让模型对分类倾向性与实际类别偏差较大的提交赋予较高的权重，来提高模型的预测性能。

4.1　研究动机

在上一节，我们已经讨论了项目的样本之间在时间维度上是有区别的，因此应该施加不同的权重进行训练。但是如图 3-2所示，项目的样本数量-天数关系图并非是平滑的直线，而是呈现波峰状的，这是由于在实际的开发过程中，项目样本的提交往往呈现活跃期和沉寂期交替进行的状况。这样会使得在某一活跃期所提交的样本它们的时间权重较为相似，此时的时间权重将无法很好地区分这些样本。

为了更好地区分相近时间权重下的样本，本文将利用机器学习解释框架对同一训练批次的样本进行解释，计算它们的特征贡献度和分类倾向性，生成基于特征贡献度的样本重要性权重来对这些样本进行区分，让预测模型关注这些时间相近的样本中相对重要的部分。

此外，由于项目的时间维度贯穿整个开发周期，因此时间维度权重可以视为一种纵向权重；而特征贡献度权重是赋予相同批次的样本，因此可以视作是横向权重。显而易见的是，这两种权重是从不同方面对样本的重要性进行描述，因此，本章还讨论了这两种权重的融合对模型性能的影响。

4.2　研究方法

这一小节阐述如何利用特征贡献度来对样本的重要性进行一个评估。如图 4-1所示，我们将赋予项目样本权重的过程分成两个阶段，第一是分配基础权重阶段，将样本的两种标签(模型的预测标签和人工打上的正确标签)进行比较，筛选出模型预测错误的样本，我们可以让模型重点关注这些错误的样本；第二是计算特征贡献度权重阶段，目的是为了进一步对预测错误的样本进行区分，具体来说，利用机器学习

解释模型，探寻分类错误的样本为什么会出错，计算特征贡献度来判别样本的分类倾向性，根据分类倾向性的大小施加对应的权重，然后进行去噪。经过两阶段的权重赋予之后，我们用带权重的项目样本去更新预测模型。

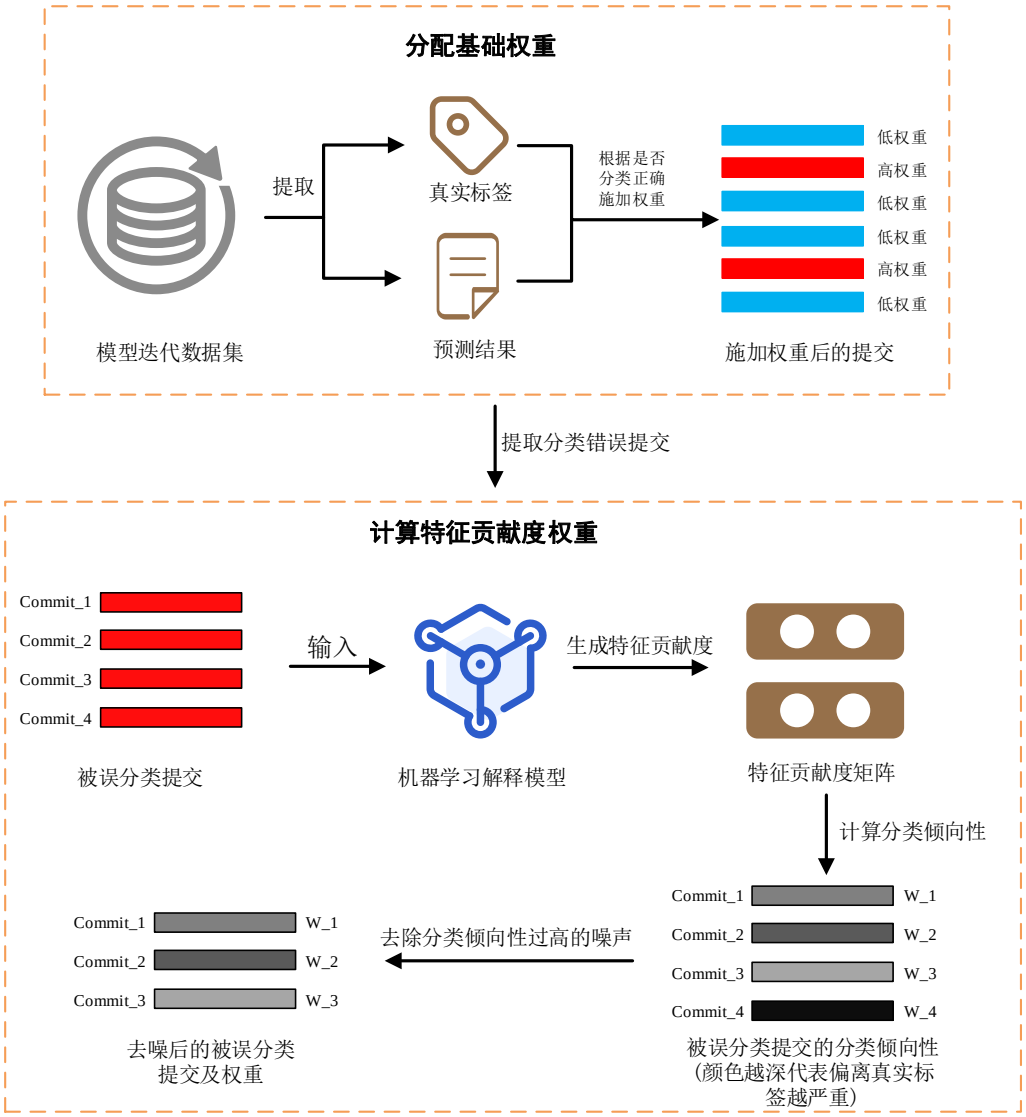


图 4-1 基于特征贡献度的样本重要性权重方法框架

4.2.1　分配基础权重

我们的研究是基于预测模型迭代的场景下，即重点关注项目样本在有了模型的预测标签和人工打上的正确标签之后，即将去更新预测模型的阶段。这样在分配基础权重阶段，我们首先可以获取样本的两种标签，即模型的预测标签和人工打上的

正确标签，然后将这两种标签进行比较，就能获取模型预测正确的样本和模型预测错误的样本。由于被误分类的样本可能是模型之前未学习过的，所以应该是更有价值，模型更应该关注的。因此，我们对模型预测正确的样本赋予一个较低权重W_l，对模型误分类的样本赋予一个较高权重W_h。这样可以让模型更多地关注之前误分类的项目样本，提高去预测后面项目样本的准确性。

4.2.2　计算特征贡献度权重

在分配基础权重阶段，我们给所有被预测正确的样本一个同样大小的较低权重，以及对被误分类的样本一个同样大小的较高权重，但考虑到不同样本之间仍然具有较大的差异性，赋予相同的权重并不能很好让模型针对性的关注和学习这些样本，因此，考虑到样本之间的差异，我们针对相对更重要的被预测错误的样本，再次给它们加上一个新的权重，这个权重计算步骤如下：

(1) 模型解释

首先我们通过机器学习解释模型对我们的预测模型和样本进行分析，解释样本为什么会被认为是有缺陷或者无缺陷的。如图 4-2所示，这是一个样本在被模型解释之后生成的特征贡献度柱状图。图中上方的“Local explanation for class 1”指的是这个柱状图表示的是样本被预测为有缺陷的情况；Y轴表示这些样本的特征值归一化后所在范围，如la ≤ 0.08，指的是这个样本增加的代码行数在归一化后小于等于0.08，是一个很小的数值；X轴表示的特征贡献度，图中的样本特征la所赋予的贡献度为-0.13左右，即对被分类为有缺陷的贡献度是负的。这其实也符合直觉，代码增加的少，此次提交就不容易有缺陷。机器学习解释相关领域已经提出了大量的解释模型，本文采用的是被广泛运用于各领域的Lime框架 [94]。

(2) 确定特征区间和贡献度

Lime框架将特征值划分成几个区间，对每一个特征区间给一个对应的特征贡献度，因此我们需要找到这些特征空间来计算特征贡献度。由于Lime框架未给出直接显示特征区间的接口，并且对样本进行一次次解析较为耗时，因此本文采取贪婪的数据构造方法。具体来说，首先我们会对训练集和测试集都做归一化处理；然后构建一个特征值全为0的样本，即该样本的所有特征值都为0；接着用Lime框架解析该样本，获取特征值范围和贡献度；之后再构造一个样本，这个样本所有特征值刚好

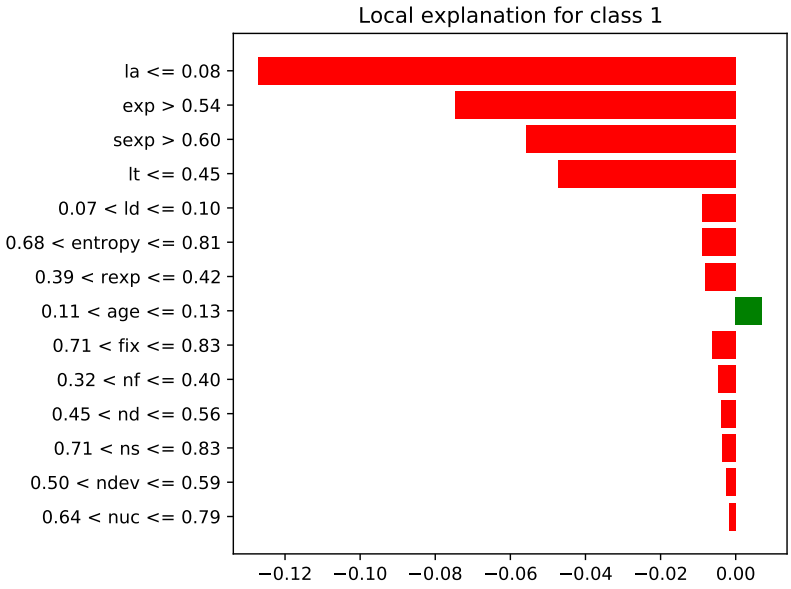


图 4-2 一个样本的模型解释

在上一个样本的特征值范围之外；接着再用Lime框架进行解析，获取新的特征值范围和贡献度，如此重复，直到新的构造样本所有特征值为1为止。这样我们就能得到所有特征值区间的特征贡献度。

(3) 计算分类倾向性

在确定特征区间及其对应的贡献度之后，我们可以构造出特征值和特征贡献度之间的映射函数。假设一个样本的特征值分别为$f_1, f_2, f_3, \ldots, f_n$；其中$n$为特征数量。接下来通过映射函数，我们就能获得对应的特征贡献度$c_1, c_2, c_3, \ldots, c_n$；由此我们可以得到分类倾向性$W_c$=$\sum_{i=1}^{n} c_i$。当$W_c > 0$时，说明预测模型认为该样本是具有缺陷的；当$W_c < 0$时，说明预测模型认为该样本是无缺陷的。

通过上述步骤，我们可以获取被缺陷预测模型误分类的样本的分类倾向性，并通过分类倾向性W_c绝对值的大小来判断预测模型将其分成某一类的意愿强度，我们引入难样本分类的思想，认为预测模型的分类意愿强度与实际类别偏差越大，说明该样本越可能是难分类样本，因此预测模型应该给予更多的关注。

(4) 去噪

一些研究指出 [81]，在项目的实际开发过程中，由于各式各样的原因，可能会出现测试人员打错标签的情况，这样会导致某些项目样本其实已经被预测模型正确

分类了，但由于测试人员打错标签而被认为是误分类，从而影响预测模型的准确性，因此有必要对项目样本进行去噪处理。在缺陷预测领域，已经出现了较多对项目提交进行去噪的研究，本文则通过分类倾向性进行去噪。具体来说，我们将$|W_c| > m$的误分类样本标记为噪声，其中m为去噪阈值。这是因为误分类样本$|W_c|$过大时，说明其严重偏离了预测模型，因此有可能是被打错标签的样本。我们去除掉被标记为有噪声的样本之后，将分类倾向性权重$|W_c|$加到原有的高权重W_h上。由于分类倾向性权重是一个数据区间，原有的高权重是一个自定义的数值，为了防止量纲不同影响预测模型的性能，因此我们将分类倾向性权重$|W_c|$归约到$[W_h, 2 \times W_h]$之间。这样最后我们得到最终的基于特征贡献度的样本重要性权重：预测模型分类正确的样本权重为W_l，预测模型分类错误的样本权重为$W_h + |W_c|$。之后，将去噪后的样本和权重输入至第三章的分类模型中进行学习和预测。

4.3　实验设置

这一节主要介绍本次实验中的实验数据集和描述、对比实验设置、性能评价指标以及参数设置等相关设置。

4.3.1　实验数据集

本章所选取的项目和第三章相同，即来自Apache和Github上的10个开源项目，项目的总时长在2至20年之间，有7个项目期限超过10年。项目总提交的数量在8845到49927之间。每个项目的都会挑选出5个活跃期，所挑选的活跃期长度在5至20个月之间。对于活跃期，本文依据时间进行划分成10个时长相等的阶段，在这10个阶段上进行缺陷预测模型的迭代，第一个阶段作为历史数据集，剩下九个阶段在每次模型迭代中依次输入训练集，即在每个任务上，预测模型迭代9次，总共有450项任务。

4.3.2　对比方法

为了能更充分地展现本文提出的方法的性能表现，这一节将选取以下方法作为对比：

(a)TRD [49]: 每次模型迭代数据集不会更新，仅使用项目早期数据去预测项目晚期数据的模型。

(b)Base [37]: 每次模型迭代使用项目的所有历史数据作为训练集，去预测新的阶段的样本是否有缺陷。

(c)TBW+gauss: 本文在第三章提出的研究方法，即在Base模型的基础上使用高斯衰减函数作为样本权重系数。

(d)CBW (Contribution-based Weights): 本章的研究方法。

(e)CBW_without_D (Contribution-based Weights without Denoising): 在计算特征贡献度权重阶段中没有进行去噪，其他步骤和本节研究方法保持一致。

(f)CBW_without_C (Contribution-based Weights without Contribution): 给分类错误的样本和分类正确的样本一个固定的权重，然后直接进行缺陷预测(即只有分配基础权重阶段的权重)。

4.3.3 评价指标

本章所选取的评价指标和第三章相同，同样使用Acc，$F1$和Mcc作为基本的评价指标。为了排除随机因素的干扰，我们在每个任务上的实验都重复三十次，来获得更加可靠的实验结果。

4.3.4 参数设置

在本实验中，一共有四个参数，分别为低权重W_l，高权重W_h，特征贡献度权重W_c和去噪阈值m。其中W_l作为统一的低权重，我们将其设置为1，W_h作为统一的高权重，根据后文的参数实验结果，我们将其设置为200，W_c是通过机器模型解释框架计算得来的，无须人工确定。为了去除噪声，但又要防止过度去除了有效信息，我们参考了缺陷预测领域先前的去噪工作的研究 [55]，m在实验中设置为90%$|W_c|$。

4.4 实验结果和分析

在这一章节，我们将本文方法与其他方法进行对比，并对实验结果进行分析。

4.4.1 与对比方法的性能比较

在本小节中，本文将基于特征贡献度的样本重要性权重与其他方法进行比较，它们的性能如图 4-3，图 4-4和图 4-5所示。我们采取SK(Scott & Knott)图来表现不同方法之间的性能差异。在图中，模型根据预测性能从左到右排序，性能相当的

模型的框用相同的颜色表示。对于指标*Acc*，模型按照性能从优到劣进行排序，分别为CBW、TBW+gauss、Base以及TRD，平均值分别为0.778、0.772、0.769和0.733；对于指标*F*1，模型按照性能从优到劣进行排序，分别为CBW、TBW+gauss、TRD以及Base，平均值分别为0.465、0.459、0.429和0.425；对于指标*MCC*，模型按照性能从优到劣进行排序，分别为CBW、TBW+gauss、Base以及TRD，平均值分别为0.345、0.331、0.306和0.272。

从总体上来看，本文提出的CBW总是在图的最左边，并且平均值也是最高的，表明在所有指标上，CBW总是最优的，这说明了我们提出的基于特征贡献度的样本权重是有意义的。不过，从实验结果来看，基于时间维度的样本重要性权重和基于特征贡献度的样本重要性权重相差并不大，这表明了两种权重虽然从两种角度出发，但都具有相当的性能。另一个值得注意的是，从图中可以看出，CBW方法的波动程度要比TBW+gauss要大，为了探究这个现象的原因，我们分析了实验数据，发现CBW由于设置了一个高权重W_h，使得CBW方法生成的样本权重不如TBW+gauss生成的样本权重平滑，也就是说，CBW方法可能会过于关注被误分类的样本，这在大多数时候都能帮助模型更好地关注重要的样本，从而提高模型的性能；但某些时候，新产生的提交与之前被误分类的样本差异较大时，就可能会影响预测模型的性能。

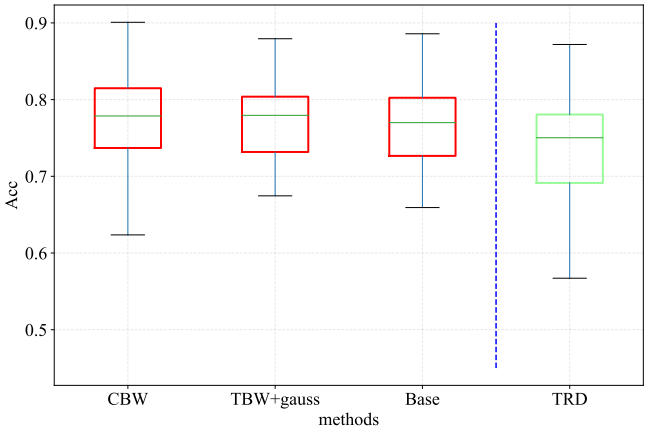


图 4-3 基于特征贡献度的样本重要性权重与其他方法在Acc值上的性能差异

4.4.2　消融实验

在本小节中，我们探讨方法的两个阶段是否具有意义，即对我们的方法进行消

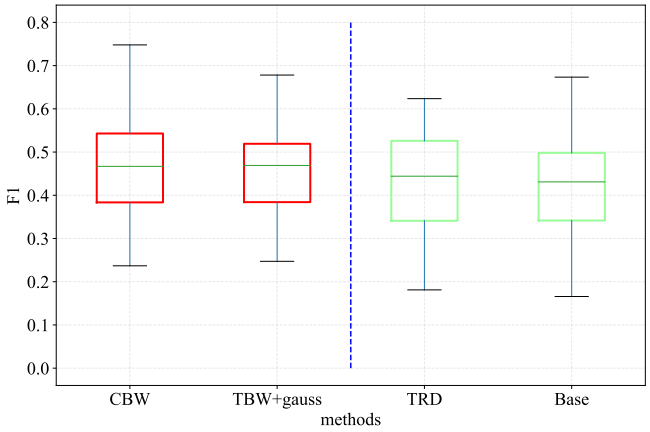


图 4-4 基于特征贡献度的样本重要性权重与其他方法在F1值上的性能差异

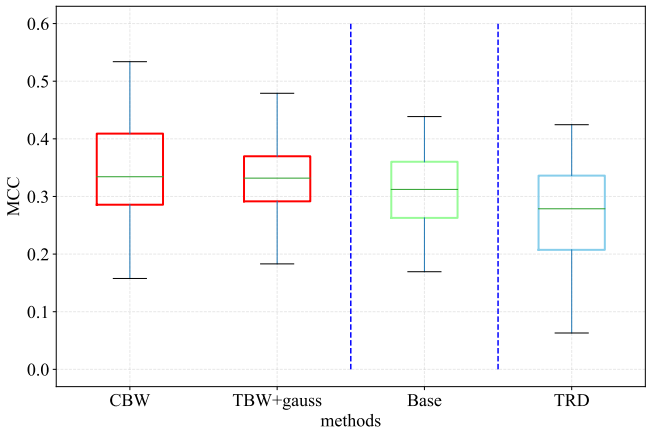


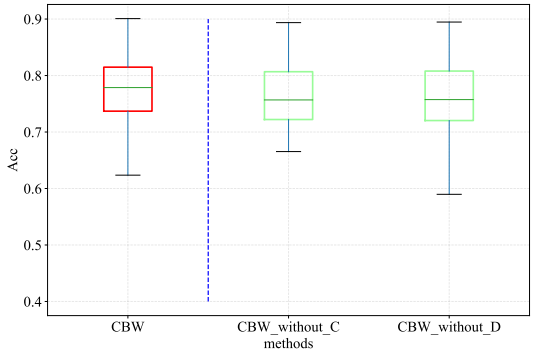
图 4-5 基于特征贡献度的样本重要性权重与其他方法在MCC值上的性能差异

融实验。首先，本章提出的方法CBW作为基准对比方法，我们在CBW方法步骤中省略去噪步骤，这样产生第一个对比方法CBW_without_D，在CBW_without_D方法的基础上，我们省略了计算特征贡献度和分类倾向性来确定W_c的步骤，这样出现了第二个对比方法CBW_without_C。

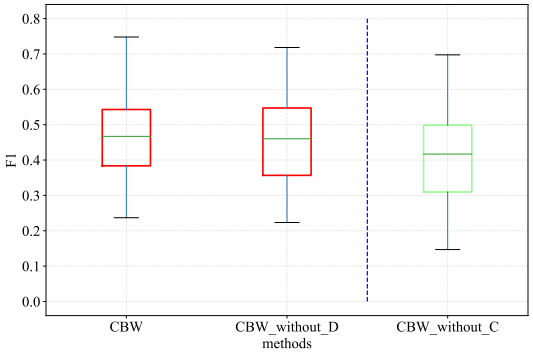
如图 4-6所示，从图中可以看出，对于指标*Acc*，模型按照性能从优到劣进行排序，分别为CBW、CBW_without_C以及CBW_without_D，平均值分别为0.778、0.775和0.753；对于指标*F*1，模型按照性能从优到劣进行排序，分别为CBW、CBW_without_D以及CBW_without_C，平均值分别为0.465、0.458和0.415；对于指标*MCC*，模型按照性能从优到劣进行排序，分别为CBW、CBW_without_D以及CBW_without_C，平均值分别为0.345、0.323和0.303。

从总体上来看，本文提出的CBW总是在图的最左边，并且平均值也是最高的，

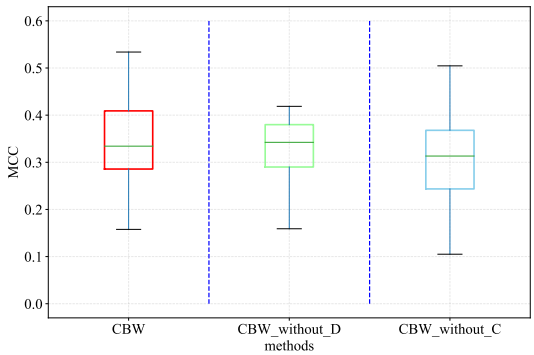
说明了本章所提出的方法步骤是有效的。不过值得注意的是，CBW_without_C只保留了分配基础权重阶段的权重，然而在指标*Acc*上，CBW_without_C的性能却优于CBW_without_D。我们仔细分析了实验数据，发现CBW_without_D关注被误分类的样本，对它们计算特征贡献度和分类倾向性，对与真实标签差距更大的样本施加更高的权重，但由于项目提交中存在噪声，在没有去噪的情况下，可能会存在过多关注噪声的问题。此外，由于数据不平衡的原因，被误分类的样本可能大部分是有缺陷的样本，这种情况下对它们施加高权重，一方面与数据采样的方法类似，可以让模型更关注有缺陷样本，提高模型性能；另一方面也可能造成模型的过拟合，反而在一些指标上降低性能。不过总的来说，在更被广泛认可的指标*F*1和*MCC*上，CBW_without_D的性能都优于CBW_without_C，这说明了我们方法计算特征贡献度权重阶段的重要性。



(a) Acc上的性能表现



(b) F1上的性能表现



(c) MCC上的性能表现

图 4-6 消融实验的结果

4.4.3　参数实验和讨论

在这一小节，我们进行参数实验和讨论，首先探讨基于时间维度的样本重要性权重与基于特征贡献度是否可以融合，以及融合之后的表现情况；然后我们对W_h这个参数做一些参数实验，寻找最为合适的参数值。

(1) 时间维度权重和特征贡献度权重是否可以融合？基于时间维度的权重是从纵向的角度出发，贯穿项目开发的整个流程，对比较新的提交施加一个较高权重，对较旧的提交施加一个较低权重；而基于特征贡献度的权重从横向的角度出发，面对同一批次时间相近的样本，找出在上一轮模型迭代过程中被误分类的样本，并对与真实标签差距较大的样本施加更高的权重。这两种权重是从不同的角度出发，应该存在可以融合的可能，这里采取两种不同的权重融合方法，一种是*TBW_ADD_CBW*，对两种权重进行归一化之后得到两个权重向量，再将两个权重向量进行相加；另一种是*TBW_MUL_CBW*，对两种权重进行归一化之后得到两个权重向量，再将两个权重向量进行一一对应相乘。

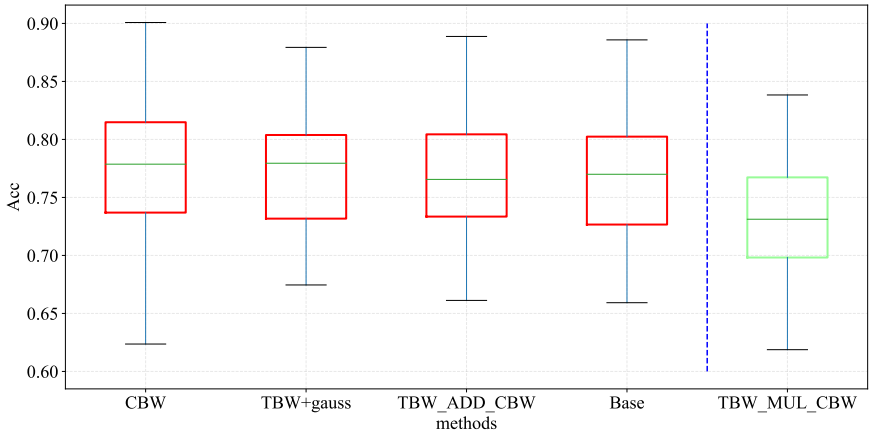


图 4-7 融合权重与非融合权重在Acc值上的性能差异

如图 4-7，图 4-8和图 4-9所示，为了更全面的展示两种融合方法的优劣程度，我们将这两种权重融合方法与CBW、TBW+gauss还有Base方法进行对比。从图中可以看出，对于指标*Acc*，模型按照性能从优到劣进行排序，*TBW_ADD_CBW*在五个方法中间排第三，*TBW_MUL_CBW*在五个方法中间排第五；对于指标*F*1，模型按照性能从优到劣进行排序，*TBW_ADD_CBW*在五个方法中间排第二，*TBW_MUL_CBW*在五个方法中间排第四；对于指标*MCC*，模型按照性能从优到

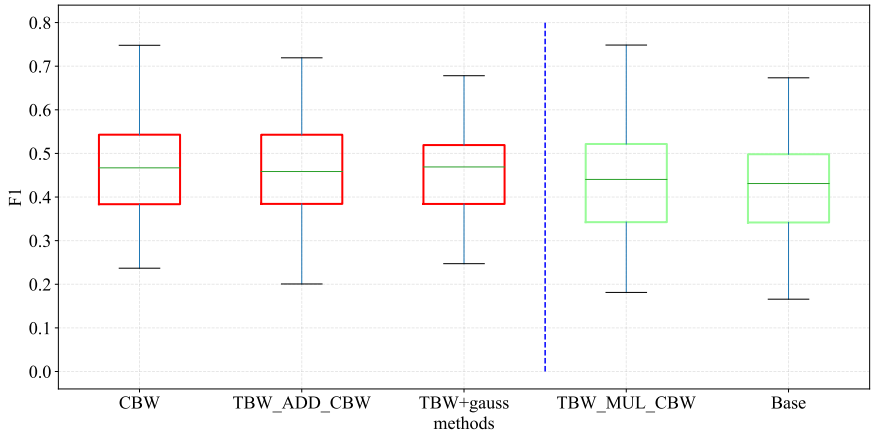


图 4-8 融合权重与非融合权重在F1值上的性能差异

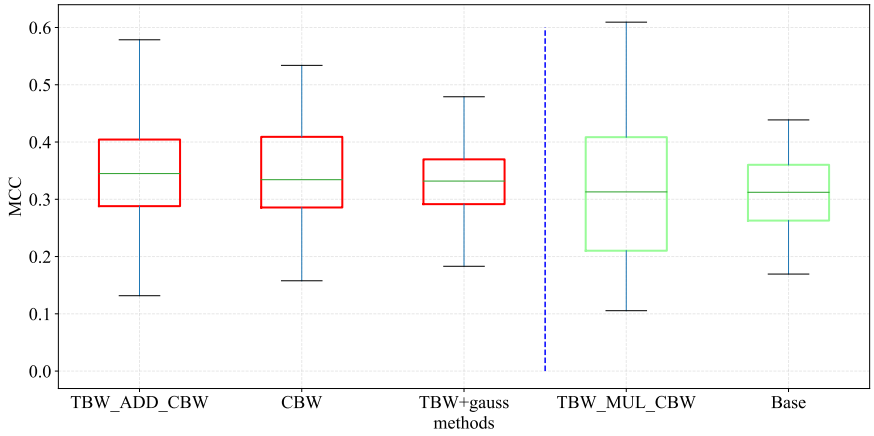


图 4-9 融合权重与非融合权重在MCC值上的性能差异

劣进行排序，*TBW_ADD_CBW*在五个方法中间排第一，*TBW_MUL_CBW*在五个方法中间排第四。

从以上结果发现，两种权重融合之后的效果并不一定比单独的权重要好，尤其是对于*TBW_MUL_CBW*来言，甚至出现了比单独的权重都要差的情况，这是因为对于*TBW_MUL_CBW*来言，由于要进行权重相乘，只要样本有一个权重较小，那么总权重就会被拉低到不被关注的程度，特别是基于特征贡献度的权重给很多没被误分类的样本的权重很低，这就导致大部分样本的总权重都很低，只有非常少的两个权重值都很高的样本才会被模型关注，导致模型泛化能力降低。与*TBW_MUL_CBW*不同的是，*TBW_ADD_CBW*将两种权重相加，哪怕样本有一个权重较小，总权重也不会被拉低到不被关注的程度，因此表现较为良好，在指标*MCC*上取得了比单独权重更优秀的效果。这些实验结果表明对于权重简单粗暴地融合是行不通的，应该考虑

权重的性质，寻找适合的方法来融合权重。

(2) 被误分类样本和分类正确样本之间的权重比对模型性能的影响

这一部分我们探讨W_h与的W_l之比对模型性能的影响，在本章的方法中W_h和W_l之比显然会影响模型的性能，因为当W_h和W_l之比越大时，模型就会越多关注被误分类的样本。我们把W_l的大小固定为1，所以只需要调节W_h的大小即可。

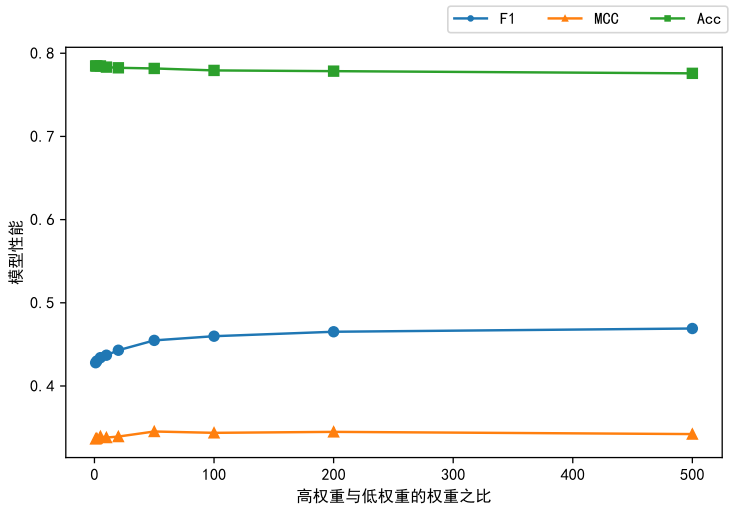


图 4-10 不同权重比下模型性能的大小

如图 4-10所示，我们设置W_h大小分别为1、2、5、10、20、50、100、200和500。当W_h为1时，代表将被误分类样本和被正确分类样本同等看待。结果表明，随着W_h的值逐渐增大，模型在指标$F1$和MCC的性能也逐渐提高，但在指标Acc上的性能反而逐渐下降。我们对实验数据进行了分析，发现当W_h的值增大时，模型对误分类样本提供了更多的关注，由于误分类样本其中很多是有缺陷的样本，这样模型会加大对缺陷样本的关注，从而有效提高在指标$F1$和MCC上的表现；同时，由于被误分类样本占样本总数不多，当模型对这些样本施加更多关注时，也可能会有些忽视大多数被正确分类的样本，导致在指标Acc上的表现有所下降。不过总体来言，随着W_h值的逐渐增大，缺陷预测模型的性能表现会越来越好。我们综合三个指标，最后确定W_h的值为200。

(3) 本章方法的时间开销

表4-1展示了本章方法的时间开销，并与第三章的方法进行对比。从表中可以看出，本章方法的时间开销是毫秒级的，这意味着本章方法的时间开销是可以接受的。不过与时间权重不同的是，时间权重的时间开销与项目的时长以及样本数量紧密相关；而特征贡献度权重的时间开销却更加稳定。这是因为基于特征贡献度的权重计算花费主要来源于机器学习解释模型对样本特征进行解释的时间，由于本章采取贪婪的方法，每次任务只需要构造几个特殊样本即可获取特征区间至特征贡献度的映射函数，从而有效降低了时间开销。虽然与时间权重相比，基于特征贡献度的权重时间开销较大，但是仍然远低于很多机器学习模型学习的时间花费，因此本章方法所提出的权重同样不会造成很多额外的测试资源花费。

表 4-1 两种权重时间开销对比

项目	TWB+gauss(毫秒)	CWB(毫秒)
ambari	0.94	5.84
ant	0.53	5.31
aptoide	0.44	5.43
camel	0.93	5.62
cassandra	0.82	5.49
egeria	0.67	5.80
felix	0.60	5.60
jackrabbit	0.34	5.62
jenkins	1.10	5.67
lucene	1.22	5.99

4.5　本章小结

本章在即时软件缺陷预测领域，针对时间相近的样本以及在模型上一轮迭代中被误分类的样本，提出了一种基于特征贡献度的样本重要性权重。本章在10个项目，每个项目5个数据区间上，9次模型迭代上进行实证研究，通过三种评价指标评估模型性能，将本文的方法与其他方法进行比较，并进行了消融实验，最终的实验结果表明了本文所提出的基于特征贡献度的样本重要性权重的有效性。我们还探究了基于时间维度的样本重要性权重和基于特征贡献度的样本重要性权重的融合，结果表明需要寻找合适的方法进行融合才能保证预测模型的性能。最后，我们进行参数实验，讨论被误分类样本权重与被正确分类样本权重之比对模型性能的影响。结果表明，在一定范围内，对被误分类样本关注越多，模型性能越好。

第五章　基于样本重要性权重的数据采样方法

这一章将在项目开发过程中预测模型迭代的场景下，基于更改级别的缺陷预测构建模型，并针对类不平衡问题，在模型构建的数据预处理阶段对数据进行采样处理。该方法通过两种样本重要性权重，对样本重要程度进行排序并挑选部分最重要的样本作为中心样本；并将权重矩阵和特征矩阵进行拼接之后来寻找近邻样本，这样使得用于合成新样本的样本都是重要样本，从而提高即时软件缺陷预测的性能。

5.1　研究动机

在即时软件缺陷预测领域，软件更改数据通常是类不平衡的，在大部分项目上，会引入缺陷的变更数量往往远少于不含有缺陷的变更数量 [11]。如果不对不平衡的数据进行处理，会导致大多数机器学习算法只关注多数类(无缺陷样本)中的特征，而忽视了学习少数类(有缺陷样本)中的特征。

为了解决即时软件缺陷预测中的类不平衡问题，数据采样技术被广泛采用，用于在数据预处理阶段平衡训练集数据的分布，以提高预测模型的性能。在缺陷预测领域，一般认为过采样比欠采样更可取，因为在欠采样过程中丢弃的样本可能包含预测模型的有用和重要信息。在过采样中，基于少数样本合成新样本的方法最为流行，比如SMOTE方法。SMOTE方法通过挑选某个样本，寻找它的近邻样本来合成新的样本。然而它的问题有三个：一是随机挑选中心样本，导致挑选的中心样本不具有代表性，不能很好地反映数据集的特征；二是中心样本只与空间距离最近样本结合，生成的数据不具有多样性，容易让模型陷入过拟合；三是对所有特征维度同等看待，不能反映不同特征的重要程度。

针对上述问题，本章结合第三章和第四章提出的两种样本重要性权重，来挑选部分重要样本作为中心样本，以改进随机挑选的问题；合并特征矩阵和权重矩阵，来计算近邻样本，以改进生成数据不多样的问题；权重有一部分是通过计算特征贡献度得到的，以改进对所有特征同等看待的问题。

5.2　研究方法

这一节阐述如何利用两种样本重要性权重对数据不平衡问题进行处理。如图

5-1所示，首先我们从不平衡数据集中提取少数类，并获取这些少数类样本的权重系数；接着将样本特征矩阵和权重系数矩阵合并，并根据权重进行排序，挑选最重要的部分样本作为中心样本；然后我们对中心样本进行遍历，并根据合并后的加权特征矩阵来计算近邻样本，最后合成新的少数类样本，使数据集平衡。值得注意的是，由于我们在计算基于特征贡献度的样本重要性权重的时候，会对样本进行去噪处理，因此我们的方法并不是完全的过采样方法，可以视为一种混合采样方法。

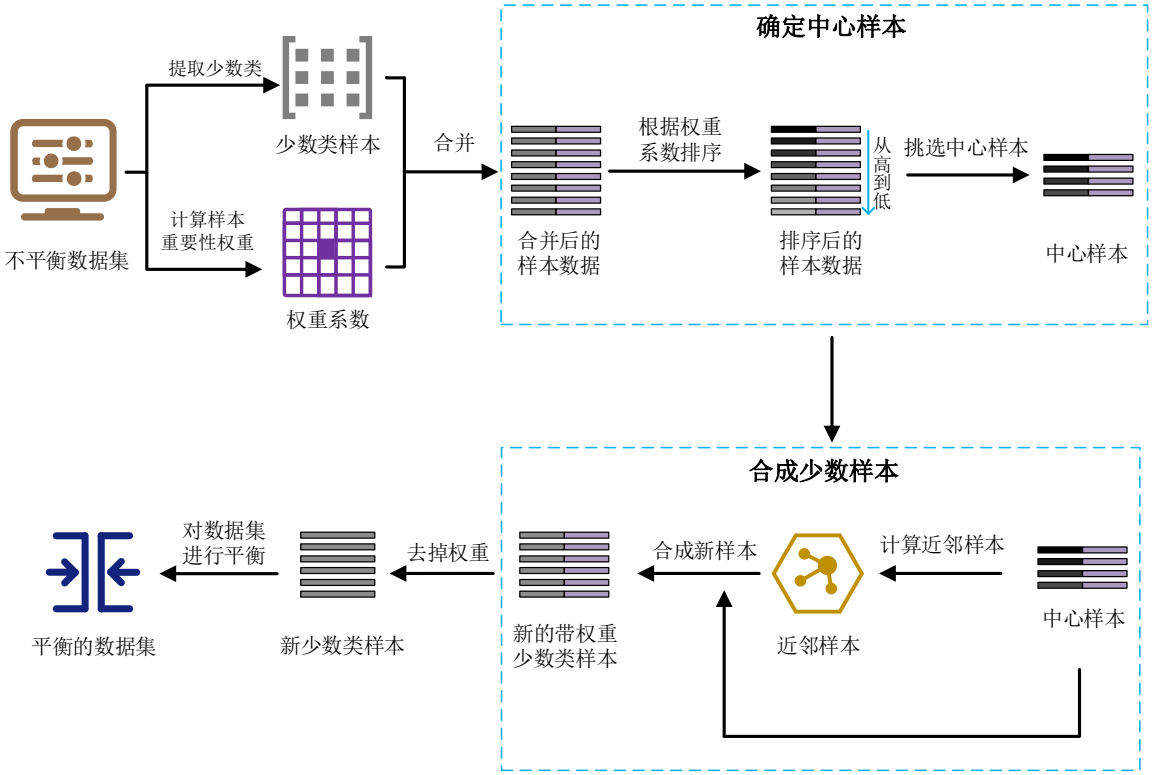


图 5-1 基于样本重要性权重的数据采样方法框架

5.2.1　计算权重系数

根据第三章和第四章，我们可以为已经打上真实标签的样本计算两种样本重要性权重。这样，我们在从不平衡的数据集中提取少数类样本之后，也能得到少数类样本的两种权重。对于两种权重，首先为了保证量纲一致，我们对所有的特征向量和权重都进行归一化处理，接着，根据第四章关于两种特征融合的讨论结果，我们取两种权重的平均值作为这些少数类样本的权重系数。这样，对于每一个少数类样本，我们都可以得到它的特征向量$[f_1, f_2, f_3, \ldots, f_n]$以及一个权重系数$w$。

5.2.2　挑选中心样本

在得到少数类的特征向量和权重系数之后，我们将用它们来挑选中心样本。首先，根据特征向量的维度，我们对权重系数进行重复扩增，得到一个维度和特征向量相同的权重向量，并将特征向量和权重向量进行拼接，生成一个扩增特征向量。这样，对于每一个少数类样本，我们都可以得到它的扩增特征向量[$f_1, f_2, f_3, \dots, f_n, w_1, w_2, w_3, \dots, w_n$]。

得到扩增特征向量之后，我们按照权重系数w的大小对样本进行排序，这样排在前面的就是相对比较重要的样本。排序之后，我们选取前α%个样本作为中心样本，α是一个可调节的参数，根据后文的参数实验，本文将α设置为50。由于这些中心样本是根据基于时间维度的样本权重和基于特征贡献度的样本权重平均排名得到的，所以这些样本要么是时间较新，更能代表当前项目提交特征的；要么如图 5-2所示，是靠近类的边缘，容易被误分类的样本。我们认为这些比较重要的样本作为中心样本时，将比随机挑选的中心样本更具有代表性，更能提高预测模型的性能。

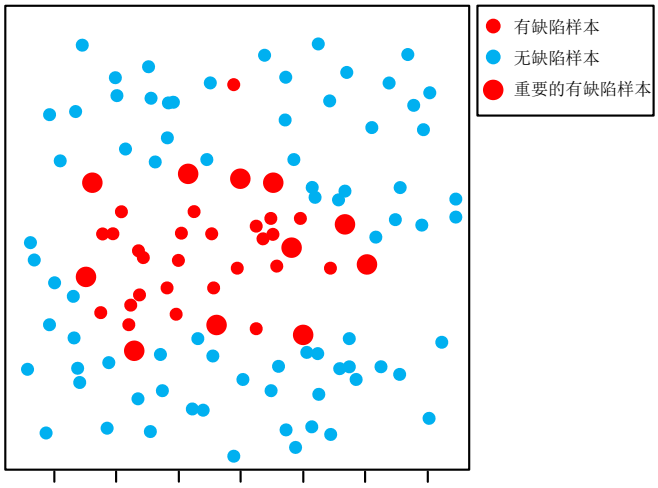


图 5-2 根据权重挑选重要有缺陷样本的示意图

5.2.3　合成少数样本

在挑选出中心样本之后，我们就可以根据中心样本来寻找近邻样本，并合成新的少数类样本。对于中心样本的集合，我们对其按权重由大到小进行遍历，对于每个中心样本，我们按照扩增的特征向量来计算近邻样本，这样做的好处在于可以拉

近重要程度相近的样本之间的距离，与比只按照特征向量来计算近邻样本的方法相比，会有两方面的好处：一方面可以让两个比较重要的样本有更大的概率结合，合成的新样本更具有代表性；另一方面重要程度相近的样本不一定空间距离相近，这样合成的新样本更多样。

计算完所有中心样本的近邻样本之后，对于一个中心样本C_i，可以得到一个近邻样本序列[N_{i1}, N_{i2}, N_{i3}, ... , N_{im}]，在序列中近邻样本按照离中心样本的距离从近到远进行排列。值得注意的是，序列长度m的大小为少数类样本的长度减一，即所有少数类样本都在中心样本选择近邻样本的范围之内，这样做的目的同样是为了增强合成新样本的多样性。这样在第一轮遍历中，对于中心样本C_i，它将与N_{i1}进行结合产生新样本；如果第一轮遍历结束，新生成的样本数量仍然不足以让少数类样本扩增到与多数类样本相同大小，我们将进行下一轮遍历，这样C_i将与N_{i2}相结合，以此类推，直到能让数据集平衡为止。在遍历过程中，往往会出现一个样本既作为中心样本，又作为某个样本的近邻样本，可能导致重复合成相同的样本，对于这种情况我们会记录合成历史，不重复合成相同样本。

当然，可能会存在一种极端情况，即在数据集严重不平衡的情况下，经过m次遍历，中心样本与其他所有样本都合成一遍之后，数据集仍然不平衡。对于这种情况，我们将把合成的新样本与原来的少数类样本进行合并，生成新的少数类样本集合，之后我们在新的少数类样本集合上重复执行本章的基于样本权重的数据采样方法，直到能让数据集平衡为止。

具体的合成方法与SMOTE中的合成方法一致，即在中心样本与近邻样本之间随机产生一个新的样本。

5.2.4　平衡数据集

在生成足够多的少数样本之后，由于现在少数类样本的特征向量都是包含权重系数的扩增特征向量，因此，我们需要对样本去除权重系数矩阵，只保留特征矩阵，然后与原来的多数类样本组成一个平衡的数据集。值得注意的是，与前两章方法不同，为了保证后续实验的公平性，所有样本将不会给它们施加任何权重。

5.3 实验设置

这一节主要介绍本次实验中的实验数据集和描述、对比实验设置以及性能评价指标等相关设置。

5.3.1 实验数据集

本章所选取的项目和第三章相同，即来自Apache和Github上的10个开源项目，项目的总时长在2至20年之间，有7个项目期限超过10年。项目总提交的数量在8845到49927之间。每个项目的都会挑选出5个活跃期，所挑选的活跃期长度在5至20个月之间。对于活跃期，本文依据时间进行划分成10个时长相等的阶段，在这10个阶段上进行缺陷预测模型的迭代，第一个阶段作为历史数据集，剩下九个阶段在每次模型迭代中依次输入训练集，即在每个任务上，预测模型迭代9次，总共有450项任务。

5.3.2 对比方法

为了能更充分地展现本文提出的方法的性能表现，这一节将选取以下方法作为对比：

(1) 未使用数据采样技术的基础模型

(1-a)Base [37]: 使用项目的所有历史数据作为训练集，去预测新的阶段的样本是否有缺陷。

(2) 过采样方法

(2-a)ROS (Random Oversampling) [95]: 使用随机过采样算法来中进行类再平衡。

(2-b)SMOTE [56]: 使用SMOTE算法来进行类再平衡。

(2-c)ORB (Oversampling Rate Boosting) [96]:一种过采样方法，使用Boosting方法动态地调节每次迭代过程中的过采样率。

(3) 欠采样方法

(3-a)RUS (Random Undersampling) [95]: 使用随机欠采样算法来进行类再平衡。

(3-b)Filtering [97]: 一种欠采样方法，计算欧式距离来保留与最新项目提交较接近的项目历史数据。

(4) 本文提出的方法

(4-a)WBO (Weight-based Oversampling): 本节的研究方法。

(4-b)WBO_without_T : 去掉时间权重，仅使用特征贡献度权重，其他步骤与本节研究方法保持一致。

(4-c)WBO_without_C: 去掉特征贡献度权重，仅使用时间权重，其他步骤与本节研究方法保持一致。

5.3.3 评价指标

本章所选取的评价指标和第三章相同，同样使用*Acc*，*F*1和*Mcc*作为基本的评价指标。为了排除随机因素的干扰，我们在每个任务上的实验都重复三十次，来获得更加可靠的实验结果。

5.4 实验结果和分析

在这一章节，我们将本文方法与其他方法进行对比，并对实验结果进行分析。

5.4.1 与对比方法的性能比较

在本小节中，本文将基于样本重要性权重的数据采样方法与其他方法进行比较，它们的性能如图 5-3，图 5-4和图 5-5所示。我们采取SK(Scott & Knott)图来表现不同方法之间的性能差异。在图中，模型根据预测性能从左到右排序，性能相当的模型的框用相同的颜色表示。对于指标*Acc*，模型按照性能从优到劣进行排序，分别为Base、WBO、SMOTE、ORB、ROS、RUS以及Filtering，平均值分别为0.769、0.703、0.702、0.701、0.701、0.689和0.673；对于指标*F*1，模型按照性能从优到劣进行排序，分别为WBO、ORB、ROS、SMOTE、Filtering、RUS以及Base，平均值分别为0.534、0.523、0.522、0.520、0.512、0.499和0.425；对于指标*MCC*，模型按照性能从优到劣进行排序，分别为WBO、ORB、ROS、Filtering、SMOTE、RUS以及Base，平均值分别为0.349、0.342、0.331、0.329、0.328、0.327和0.306。

从总体上来看，本文所提的方法WBO在两个指标上排第一，一个指标上排第二，说明了WBO方法的有效性。对于指标*Acc*，没有用不平衡技术进行处理的Base方法效果最好，这是因为在对于不平衡的数据集，模型会倾向于将结果预测为多数类，这

样就能得到虚高的*Acc*性能，但是从另两个指标*F*1和*MCC*上看，Base方法的效果总是最差的，这也说明了对于不平衡数据集，应该综合多种指标，尤其是能全面反映模型性能的指标。

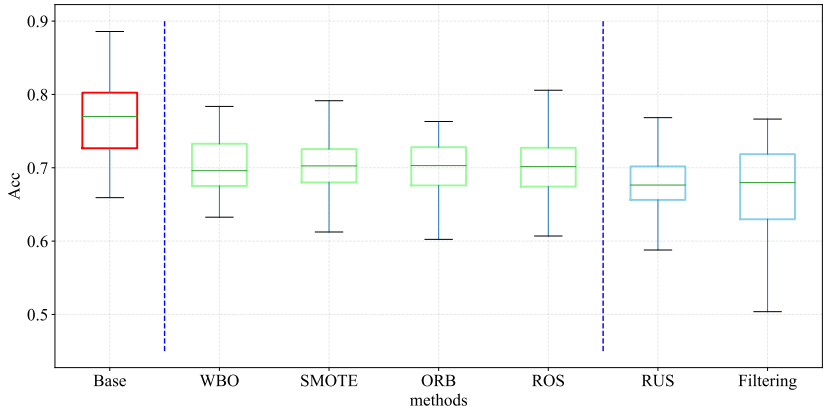


图 5-3 基于样本重要性权重的过采样方法与其他过采样方法在Acc值上的性能差异

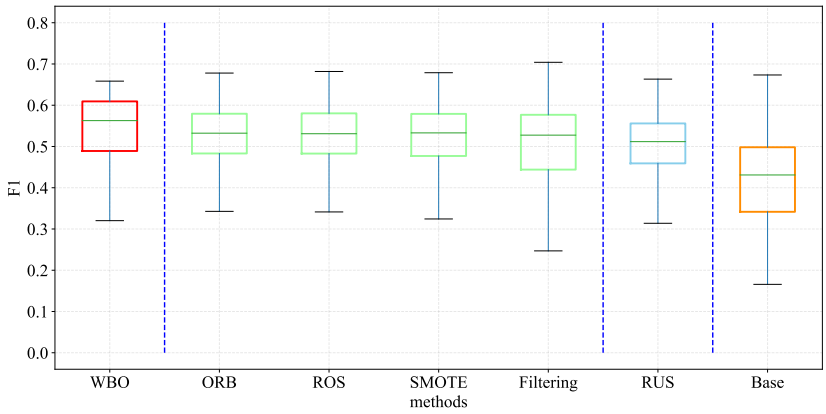


图 5-4 基于样本重要性权重的过采样方法与其他过采样方法在F1值上的性能差异

将过采样方法与欠采样方法进行对比，我们发现，两种欠采样方法RUS和Filtering，在三个性能指标上的大部分情况下都劣于SMOTE、ORB和ROS这三种过采样方法。这些结果表明在即时软件缺陷预测中，欠采样方法的性能表现不如过采样方法，这是因为欠采样方法在丢弃多数类样本的同时，也会丢失许多有重要信息的数据，因此，我们认为，在即时软件缺陷预测领域，为了获得更好的预测性能，我们应该采用过采样方法来对数据集进行平衡。

另一个值得注意的现象是，在指标*F*1和*MCC*上，SMOTE方法的性能表现不如ROS，这个结果与以前的研究报告的不太一样。我们分析了实验数据，发现正如

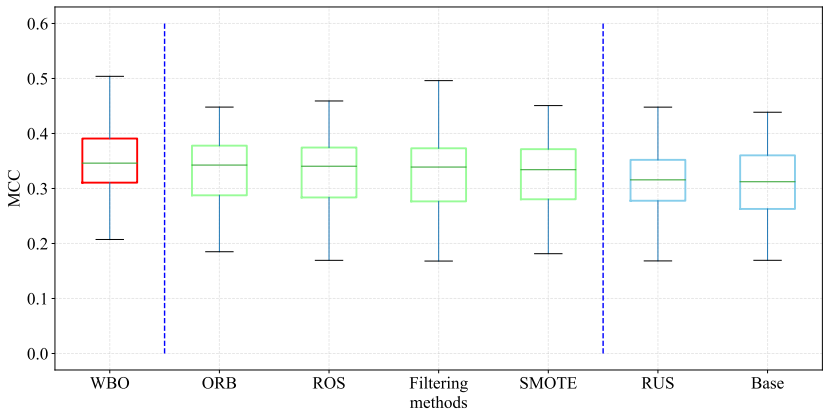


图 5-5 基于样本重要性权重的过采样方法与其他过采样方法在MCC值上的性能差异

相关研究 [98]指出的那样，SMOTE方法从处于类边界的少数类样本中合成新样本时，其近邻样本也是在边界，这样合成的新样本也会落在类的边界，使得边界变得更加模糊，从而导致预测性能受到负面影响。而本文提出的方法WBO通过扩增特征向量来计算近邻样本，拉近了重要样本彼此之间的距离，使得生成的新样本不一定落在类边界，在一定程度上改善了这个问题。

5.4.2　消融实验

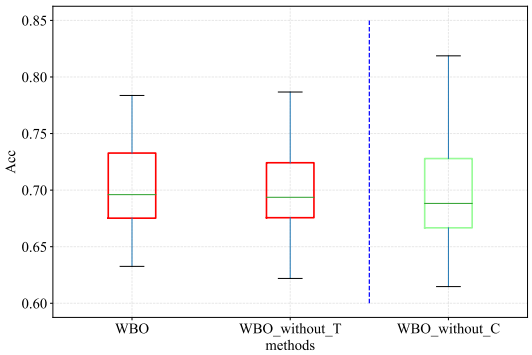
在本小节，我们对本章所提出的基于样本重要性权重的数据采样方法进行消融实验，设置了两个对比方法WBO_without_T和WBO_without_C，分别为只用基于特征贡献度的权重和只用时间维度的权重，其他地方与本文提出的方法WBO保持一致。

如图 5-6所示，从图中可以看出，对于指标*Acc*，模型按照性能从优到劣进行排序，分别为WBO、WBO_without_C和WBO_without_T，平均值分别为0.703、0.699和0.694；对于指标*F*1，模型按照性能从优到劣进行排序，分别为WBO、WBO_without_T和WBO_without_C，平均值分别为0.547、0.535和0.535；对于指标*MCC*，模型按照性能从优到劣进行排序，分别为WBO、WBO_without_C和WBO_without_T，平均值分别为0.349、0.342和0.341。

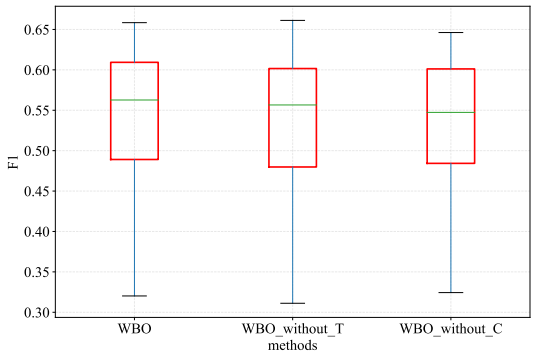
从总体上来看，本文提出的WBO总是在图的最左边，并且平均值也是最高的，说明了本章所提出的方法中两种权重融合的步骤是有效的，不过从图中也可以看出，三种模型的性能其实是相当的，这与第四章的讨论结果类似，当我们对权重进行平均来作为混合权重时，只能得到很小的改善。不过与第四章的讨论有所区别的是，

由于实验方法和实验场景的不同，实验结果也有所差异。

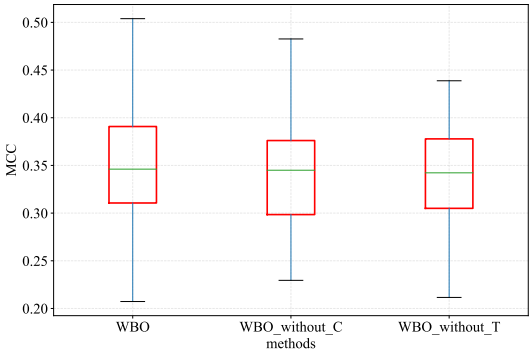
从实验方法来看，第四章的权重混合是在模型训练的时候根据两种权重来关注更加重要的样本，这样每个样本的重要程度具有绝对的顺序；而本章方法是根据权重大小将少数类样本划分成重要样本和非重要样本，并以重要样本为中心样本来合成新样本，这样本章的样本重要程度并没有绝对的顺序，不同权重的参与使得中心样本的选择更加全面，生成样本更具有代表性和多样性。



(a) Acc上的性能比较



(b) F1上的性能比较



(c) MCC上的性能比较

图 5-6 两种权重的过采样方法与单个权重方法的性能比较

从实验场景来看，第四章实验时由于只探讨权重的不同对预测模型性能的影响，为了实验的公平性，我们对数据集不做任何修改，因此某个样本被标记为噪声，只是将权重设为0，而不是直接剔除。这样在权重混合时，这些噪声样本可能会有另一个权重而仍然受到模型的关注，从而降低预测模型性能。与之不同的是，第五章的各种数据采样方法本来就会对数据集进行修改，因此此时被标记为噪声的样本就会从数据集中剔除，不会被用来合成新的样本，所以数据集的可靠性得到增强，从而提高预测模型的性能。

5.4.3　参数实验和讨论

(1) 少数样本采样范围对模型性能的影响

这一部分我们探讨对少数样本设置的采样范围α%的大小对模型性能的影响。采样范围α%的不同，会使得用来作为中心样本的少数样本发生变化，同时影响遍历次数，使得合成的新样本也有所不同，因此我们认为参数α会对预测模型的性能产生影响。

如图 5-7所示，我们设置参数α值分别为10、20、30、40、50、60、70、80、90以及100。值得注意的是，哪怕参数α值等于100，我们的方法与SMOTE方法也有极大的区别，首先是我们在计算近邻样本时，采用的是扩增特征向量进行计算的；其次，我们是采用权重从高到低依次遍历的方式来合成新样本，而不是随机挑选。

从实验结果上来看，我们提出的基于样本重要性权重的数据采样方法的性能随着α的增大先变大后减少，在α等于50的时候，综合所有指标取得最大值，因此在本章方法的参数α设置为50。这是因为当采样范围过低时，中心样本不得不与距离较远的样本相结合，而两个距离较远的样本结合时，可能会导致合成的新样本落入到多数类的范围里，从而对预测模型的性能产生负面影响 [59]；当采样范围过大时，会使得不重要的样本结合生成新的样本，这种新生成的样本同样也很难对模型起到正面影响。除此之外，观察三条曲线，我们可以发现指标*Acc*的曲线趋势往往与$F1$和MCC相反，这与之前的实验结果保持一致，这似乎说明，当我们想要在类平衡的指标上取得更优秀的成绩时，往往不得不牺牲*Acc*指标。最后，我们还能看出，参数实验几个指标的波动不是非常剧烈，说明我们的方法其实是参数不敏感的，这是因为我们没有采用随机选择的方式，而是固定地从高权重的样本开始合成新的样本，这在一定程度上保持了方法的稳定性。

(2) 任务数据的不平衡率对模型性能的影响

这一部分我们探讨任务数据的不平衡率对模型性能的影响。按照理想情况，数据集的不平衡率越高，原本不平衡的预测模型对多数类关注越高，使得实际性能下降越多，这样当采用不平衡技术时，模型所能获得的改善越高，因此我们设计了实验，探究WBO方法在不同缺陷率任务下的对预测模型的性能改善。由于MCC指标对TP，TN，FP和FN都进行了综合考量，更能全面反映预测模型在不平衡数据集上

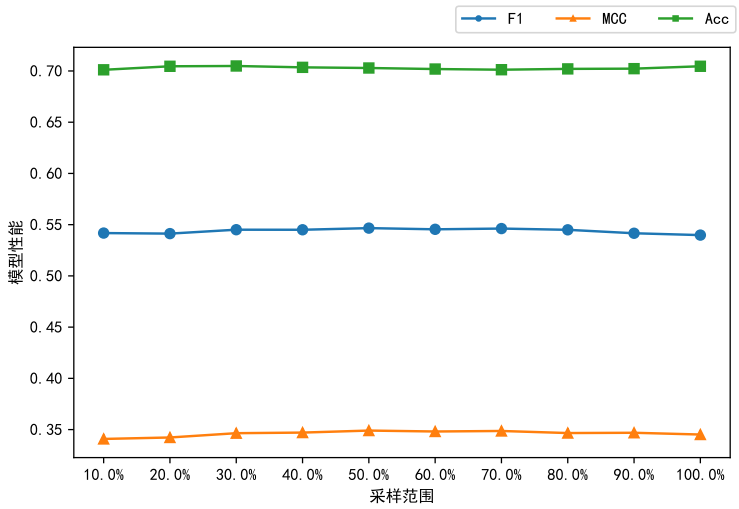


图 5-7 少数样本不同采样范围下模型性能的变化情况

的性能表现，因此这一部分我们以*MCC*作为评价模型性能的指标。

如图 5-8所示，这个散点图展示了WBO方法在450个任务上对即时软件缺陷预测模型的性能改善情况，其中横坐标为某次任务的数据集中有缺陷样本占总样本的比率，当横坐标小于0.5时，说明在不平衡数据集中有缺陷的样本为少数类；当横坐标等于0.5时，说明这次任务的数据集是平衡的；当横坐标大于0.5时，说明在不平衡数据集中有缺陷的样本为多数类。纵坐标为预测模型采用WBO方法对数据进行平衡和预测模型没有采用任何不平衡方法之间的性能差值，当纵坐标大于0时，说明WBO方法在这个任务上有效提高了预测模型的性能；当纵坐标等于0时，说明WBO方法在这个任务上对预测模型没有任何影响；当纵坐标小于0时，说明WBO方法在这个任务上反而对预测模型有负面影响。为了更清晰地展现WBO方法在450个任务上的性能表现，我们在图中绘制了$y = 0$的红色虚线。

从图中的横坐标来看，在绝大多数任务的数据集，有缺陷样本是少数类，只有极少部分任务上有缺陷样本是多数类。所有任务数据集缺陷率的中位数是0.254，即有接近一半的任务数据集中，有缺陷样本在总样本的比例不到四分之一，这说明我们实验所用的任务数据集存在比较明显的类不平衡问题。

从图中的纵坐标来看，在大部分任务上，应用WBO方法前后，预测模型的*MCC*差值都大于0，这说明了本文提出的WBO方法的有效性。结合横坐标的缺陷

率来看，我们发现，当缺陷率在小于0.4的情况下，在$y = 0$虚线上方的实验结果远远多于在$y = 0$虚线下方；但是当缺陷率在0.5附近的时候，在$y = 0$虚线上方的实验结果反而少于在$y = 0$虚线下方，这说明WBO方法更加适合不平衡程度较高的数据集，并且从总体上来说，数据集的不平衡程度越高，WBO方法所能改善的效果越好。

虽然从整体的实验结果来说，WBO方法对不平衡数据集上的预测模型有着明显的改善效果，但是仍然在部分任务上表现不佳。我们对数据集和实验过程进行了详细分析，发现问题主要出现在数据集上，数据集的一些内在数据集特征，即小析取、缺乏密度、缺乏类可分性、噪声数据和数据集偏移等问题对类不平衡技术的影响较大，对于一些上述问题较多的数据集，在进行平衡处理之后，效果不但没有提升，反而有所下降。这种现象不是我们的WBO方法独有的，相关实证研究 [11]报告了常见的类不平衡技术也会出现类似的问题。

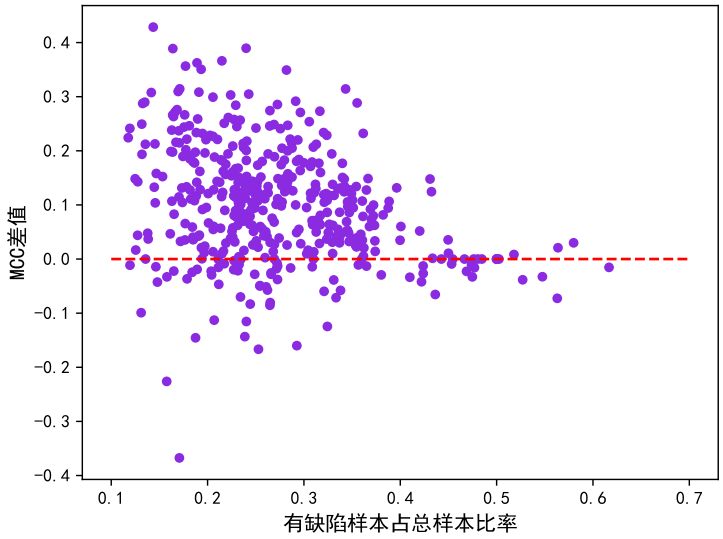


图 5-8 不同缺陷率下应用WBO数据采样方法前后模型的性能差异

5.5　本章小结

本章在即时软件缺陷预测领域，基于两种样本重要性权重，提出了一种数据采样方法，用于解决数据集类不平衡的问题。本章在10个项目，每个项目5个数据区间上，9次模型迭代上进行实证研究，通过三种评价指标评估模型性能，将本文的

方法与其他方法进行比较，并进行了消融实验，最终的实验结果表明了本文所提出的基于样本重要性权重的数据采样方法的有效性。我们还进行参数实验，探讨对少数样本设置的采样范围$\alpha\%$的大小对模型性能的影响，结果表明α值过大和过低可能会被模型的性能产生负面影响。最后我们还讨论了任务数据的不平衡率对模型性能的影响，实验结果表明，从总体上来说，数据集的不平衡程度越高，本文所提出的WBO方法所能改善的效果越好。

第六章　总结与展望

6.1　总结

即时软件缺陷预测关注敏捷迭代的项目开发过程，在软件更改提交到存储库后，立即预测潜在的引起缺陷的软件更改。由于软件变更集通常较小，每个变更集都有一条描述性的日志消息，并且每个变更都有一个提交人，因此软件测试人员可以较为轻松地完成变更审阅。然而目前即时软件缺陷预测领域仍然存在诸如不够重视提交的时间维度、对难分类样本无法很好地区分以及数据采样方法挑选用于合成的样本不具有代表性的问题。本文基于模型迭代的场景下，从区分样本的重要程度出发开展研究工作，论文主要工作包括以下三个部分：

(1)为了解决即时软件缺陷预测领域存在不够重视提交时间维度的问题，本文提出了一种基于时间维度的样本重要性权重——TBW，设计了三种时间衰减函数，在模型迭代的场景下，通过让模型更关注较新提交的样本的方式提高模型的预测性能。为了验证时间维度的重要意义，本文针对10个开源项目，每个项目挑选5个数据区间，9次迭代过程，共450个任务上进行了实证研究，在指标*Acc*、*F*1和*MCC*上的结果表明：用项目的早期提交构建的预测模型往往不能正确预测项目晚期的提交，因此模型的更新迭代是有必要的，且本文所提出的TBW能够让模型更关注较新的样本，有效提高了预测模型的性能。

(2)为了解决即时软件缺陷预测领域中对难分类样本无法很好地区分的问题，本文提出了一种基于特征贡献度的样本重要性权重——CBW，通过提取上一轮模型迭代过程中被误分类的样本，计算这些样本的特征贡献度和分类倾向性，给与真实标签差别更大的样本一个更高权重，从而能让模型关注更难被正确分类的样本。在450个任务上的实验结果表明，CBW在指标*Acc*、*F*1和*MCC*上优于对比方法；消融实验结果表明了CBW方法每个步骤的有效性；最后参数实验结果表明在一定范围内，对被误分类样本关注越多，模型性能越好。

(3)为了解决即时软件缺陷预测领域数据采样方法挑选用于合成的样本不具有代表性的问题，本文提出了一种基于样本重要性权重的数据采样方法——WBO。通过两种样本重要性权重，对样本重要程度进行排序并挑选部分最重要的样本作为中心

样本；并将权重矩阵和特征矩阵进行拼接之后来寻找近邻样本，这样使得用于合成新样本的样本更大概率都是重要样本，从而提高即时软件缺陷预测的性能。在450个任务上的实验结果表明，WBO在指标*Acc*、*F*1和*MCC*上优于对比方法；消融实验结果表明WBO方法通过两个权重融合是优于单个权重的；最后的讨论表明数据集的不平衡程度越高，本文所提出的WBO方法的效果提升越明显。

6.2 展望

本文关注不同样本的重要程度，提出了一系列适用于即时软件缺陷预测领域中的数据预处理方法，并进行了充分的实验来证明所提出方法的有效性。未来与本文研究相关的工作还可以从以下方面开展：

(1)本文中所使用的两种样本重要性权重可以进一步融合。本文所提的两种样本重要性权重分别从纵向和横向的角度出发，对不同样本的重要程度进行区分。我们也讨论了两种权重简单融合后对模型性能表现的影响，发现不具有太大优势。因此未来工作可以去寻找一种更加适合的融合方式，来有效利用两种甚至更多种权重，从而提高预测模型性能。

(2)本文中所提出的两种样本重要性权重可以扩展到其他场景。在本文的450个任务的实验中，训练集和测试集所在的项目是相同的。但是在跨项目场景下，不同项目的时间标签和特征分布都具有较大的差异性，如何将我们的工作扩展到跨项目场景下是一个有意义且充满挑战的工作。

(3)本文中所涉及到的特征可以扩展到其他种类特征。本文的研究是基于14种手工特征之上的，除了这14种手工特征之外，即时缺陷预测领域近些年语义特征的兴起给本文研究带来了新的挑战，我们的方法在其他种类特征之上的性能表现如何也是值得关注的。未来工作可以改造本文方法，让其适用于高维的语义特征，以增强本文方法的通用性。

参考文献

[1] Akito Monden, Takuma Hayashi, Shoji Shinoda, Kumiko Shirai, Junichi Yoshida, Mike Barker and Ken-ichi Matsumoto. Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing[J]. IEEE Trans. Software Eng., 2013, 39(10): 1345–1357.

[2] Carol S. Smidts, Robert Stoddard and Martin A. Stutzke. Software reliability models: an approach to early reliability prediction[C]. Seventh International Symposium on Software Reliability Engineering, 1996: 132–141.

[3] Tracy Hall, Sarah Beecham, David Bowes, David Gray and Steve Counsell. A Systematic Literature Review on Fault Prediction Performance in Software Engineering[J]. IEEE Trans. Software Eng., 2012, 38(6): 1276–1304.

[4] N. C. Shrikanth, Suvodeep Majumder and Tim Menzies. Early Life Cycle Software Defect Prediction. Why? How?[C]. IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021: 448–459.

[5] Rana Ozakinci and Ayça Tarhan. Early software defect prediction: A systematic map and review[J]. J. Syst. Softw., 2018, 144: 216–239.

[6] 刘旭同，郭肇强，刘释然，张鹏，卢红敏，周毓明. 软件缺陷预测模型间的比较实验:问题,进展与挑战[J]. 软件学报, 2023, 34(2): 582–624.

[7] Yunhua Zhao, Kostadin Damevski and Hui Chen. A Systematic Survey of Just-in-Time Software Defect Prediction[J]. ACM Comput. Surv., 2023, 55(10): 201:1–201:35.

[8] Shuo Wang and Xin Yao. Using Class Imbalance Learning for Software Defect Prediction[J]. IEEE Trans. Reliab., 2013, 62(2): 434–443.

[9] Kiran Kumar Bejjanki, Jayadev Gyani and Narsimha Gugulothu. Class Imbalance Reduction (CIR): A Novel Approach to Software Defect Prediction in the Presence of Class Imbalance[J]. Symmetry, 2020, 12(3): 407–421.

[10] Ming Tan, Lin Tan, Sashank Dara and Caleb Mayeux. Online Defect Prediction for Imbalanced Data[C]. 37th IEEE/ACM International Conference on Software Engineering, 2015: 99–108.

[11] Qinbao Song, Yuchen Guo and Martin J. Shepperd. A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction[J]. IEEE Trans. Software Eng., 2019, 45(12): 1253–1269.

[12] Thomas J. McCabe. A Complexity Measure[J]. IEEE Trans. Software Eng., 1976, 2(4): 308–320.

[13] Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design[J]. IEEE Trans. Software Eng., 1994, 20(6): 476–493.

[14] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density[C]. 27th International Conference on Software Engineering, 2005: 284–292.

[15] Audris Mockus and David M. Weiss. Predicting risk of software changes[J]. Bell Labs Tech. J., 2000, 5(2): 169–180.

[16] Dario Di Nucci, Fabio Palomba, Giuseppe De Rosa, Gabriele Bavota, Rocco Oliveto and Andrea De Lucia. A Developer Centered Bug Prediction Model[J]. IEEE Trans. Software Eng., 2018, 44(1): 5–24.

[17] Song Wang, Taiyue Liu, Jaechang Nam and Lin Tan. Deep Semantic Feature Learning for Software Defect Prediction[J]. IEEE Trans. Software Eng., 2020, 46(12): 1267–1293.

[18] Hao Wang, Weiyuan Zhuang and Xiaofang Zhang. Software Defect Prediction Based on Gated Hierarchical LSTMs[J]. IEEE Trans. Reliab., 2021, 70(2): 711–727.

[19] Jiaxi Xu, Fei Wang and Jun Ai. Defect Prediction With Semantics and Context Features of Codes Based on Graph Representation Learning[J]. IEEE Trans. Reliab., 2021, 70(2): 613–625.

[20] Ming Wen, Rongxin Wu and Shing-Chi Cheung. How Well Do Change Sequences Predict Defects? Sequence Learning from Software Changes[J]. IEEE Trans. Software Eng., 2020, 46(11): 1155–1175.

[21] Chanathip Pornprasit and Chakkrit Tantithamthavorn. JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction[C]. 18th IEEE/ACM International Conference on Mining Software Repositories, 2021: 369–379.

[22] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang and Ayse Basar Bener. Defect prediction from static code features: current results, limitations, new approaches[J]. Autom. Softw. Eng., 2010, 17(4): 375–407.

[23] Safa Omri and Carsten Sinz. Deep Learning for Software Defect Prediction: A Survey[C]. Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops, 2020: 209–214.

[24] Wenjian Liu, Baoping Wang and Wennan Wang. Deep Learning Software Defect Prediction Methods for Cloud Environments Research[J]. Sci. Program., 2021, 2323100: 1–11.

[25] Lov Kumar, Triyasha Ghosh Dastidar, Lalita Bhanu Murthy Neti, Shashank Mouli Satapathy, Sanjay Misra, Vipul Kocher and Srinivas Padmanabhuni. Deep-Learning Approach with DeepXplore for Software Defect Severity Level Prediction[C]. Computational Science and Its Applications - ICCSA - 21st International Conference, 2021: 398–410.

[26] Xinli Yang, David Lo, Xin Xia, Yun Zhang and Jianling Sun. Deep Learning for Just-in-Time Defect Prediction[C]. IEEE International Conference on Software Quality, Reliability and Security, 2015: 17–26.

[27] Guisheng Fan, Xuyang Diao, Huiqun Yu, Kang Yang and Liqiong Chen. Software Defect Prediction via Attention-Based Recurrent Neural Network[J]. Sci. Program., 2019, 6230953: 1–14.

[28] Yaqin Zhou, Shangqing Liu, Jing Kai Siow, Xiaoning Du and Yang Liu. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks[C]. Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, 2019: 10197–10207.

[29] Kun Zhu, Nana Zhang, Shi Ying and Dandan Zhu. Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network[J]. IET Softw., 2020, 14(3): 185–195.

[30] Haonan Tong, Bin Liu and Shihai Wang. Kernel Spectral Embedding Transfer Ensemble for Heterogeneous Defect Prediction[J]. IEEE Trans. Software Eng., 2021, 47(9): 1886–1906.

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

[32] Yue Jiang, Bojan Cukic and Yan Ma. Techniques for evaluating fault prediction models[J]. Empir. Softw. Eng., 2008, 13(5): 561–595.

[33] Pierre Baldi, Søren Brunak, Yves Chauvin, Claus A. F. Andersen and Henrik Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview[J]. Bioinform., 2000, 16(5): 412–424.

[34] Xiao-Yuan Jing, Shi Ying, Zhi-Wu Zhang, Shanshan Wu and Jin Liu. Dictionary learning based software defect prediction[C]. Proceedings of the 36th international conference on software engineering, 2014: 414–423.

[35] Tim Menzies, Jeremy Greenwald and Art Frank. Data Mining Static Code Attributes to Learn Defect Predictors[J]. IEEE Trans. Software Eng., 2007, 33(1): 2–13.

[36] Yuming Zhou, Yibiao Yang, Hongmin Lu, Lin Chen, Yanhui Li, Yangyang Zhao, Junyan Qian and Baowen Xu. How Far We Have Progressed in the Journey? An Examination of Cross-Project Defect Prediction[J]. ACM Trans. Softw. Eng. Methodol., 2018, 27(1): 1:1–1:51.

[37] Liqiong Chen, Shilong Song and Can Wang. A Novel Effort Measure Method for Effort-Aware Just-in-Time Software Defect Prediction[J]. Int. J. Softw. Eng. Knowl. Eng., 2021, 31(8): 1145–1169.

[38] Hongyu Zhang and S. C. Cheung. A cost-effectiveness criterion for applying software defect prediction models[C]. Proceedings of the 9th Joint Meeting on Foundations of Software Engineering, 2013: 643–646.

[39] Thilo Mende and Rainer Koschke. Effort-Aware Defect Prediction Models[C]. 14th European Conference on Software Maintenance and Reengineering, CSMR, 2010: 107–116.

[40] Tian Jiang, Lin Tan and Sunghun Kim. Personalized defect prediction[C]. 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013: 279–289.

[41] Qiao Huang, Xin Xia and David Lo. Supervised vs Unsupervised Models: A Holistic Look at Effort-Aware Just-in-Time Defect Prediction[C]. IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017: 159–170.

[42] Thilo Mende and Rainer Koschke. Revisiting the evaluation of defect prediction models[C]. Proceedings of the 5th International Conference on Predictor Models in Software Engineering, 2009: 1–10.

[43] Shane McIntosh and Yasutaka Kamei. Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction[J]. IEEE Trans. Software Eng., 2018, 44(5): 412–428.

[44] Ke Li, Zilin Xiang, Tao Chen, Shuo Wang and Kay Chen Tan. Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: an empirical study[C]. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, 2020: 566–577.

[45] Chanathip Pornprasit, Chakkrit Tantithamthavorn, Jirayus Jiarpakdee, Michael Fu and Patanamon Thongtanunam. PyExplainer: Explaining the Predictions of Just-In-Time Defect Models[C]. 36th IEEE/ACM International Conference on Automated Software Engineering, 2021: 407–418.

[46] Thong Hoang, Hoa Khanh Dam, Yasutaka Kamei, David Lo and Naoyasu Ubayashi. DeepJIT: an end-to-end deep learning framework for just-in-time defect prediction[C]. Proceedings of the 16th International Conference on Mining Software Repositories, MSR, 2019: 34–45.

[47] Qiao Huang, Xin Xia and David Lo. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction[J]. Empir. Softw. Eng., 2019, 24(5): 2823–2862.

[48] Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi and Ahmed E Hassan. Studying just-in-time defect prediction using cross-project models[J]. Empirical Software Engineering, 2016, 21(5): 2072–2106.

[49] Yuli Tian, Ning Li, Jeff Tian and Wei Zheng. How Well Just-In-Time Defect Prediction Techniques Enhance Software Reliability?[C]. 20th IEEE International Conference on Software Quality, Reliability and Security, 2020: 212–221.

[50] Ruifeng Duan, Haitao Xu, Yuanrui Fan and Meng Yan. The Impact of Duplicate Changes on Just-in-Time Defect Prediction[J]. IEEE Trans. Reliab., 2022, 71(3): 1294–1308.

[51] Yuanrui Fan, Xin Xia, Daniel Alencar da Costa, David Lo, Ahmed E. Hassan and Shanping Li. The Impact of Mislabeled Changes by SZZ on Just-in-Time Defect Prediction[J]. IEEE Trans. Software Eng., 2021, 47(8): 1559–1586.

[52] Jinfu Chen, Xiaoli Wang, Saihua Cai, Jiaping Xu, Jingyi Chen and Haibo Chen. A software defect prediction method with metric compensation based on feature selection and transfer learning[J]. Frontiers Inf. Technol. Electron. Eng., 2022, 23(5): 715–731.

[53] Joseph Bamidele Awotunde, Sanjay Misra, Emmanuel Abidemi Adeniyi, Abiodun Kazeem Moses, Manju Kaushik and Morolake Oladayo Lawrence. A Feature Selection-Based K-NN Model for Fast Software Defect Prediction[C]. Computational Science and Its Applications?ICCSA 2022 Workshops, 2022: 49–61.

[54] Yahaya Zakariyau Bala, Pathiah Abdul Samat, Khaironi Yatim Sharif and Noridayu Manshor. Improving Cross-Project Software Defect Prediction Method Through Transformation and Feature Selection Approach[J]. IEEE Access, 2023, 11: 2318–2326.

[55] Yumei Wu, Jingxiu Yao, Shuo Chang and Bin Liu. LIMCR: Less-Informative Majorities Cleaning Rule Based on Nave Bayes for Imbalance Learning in Software Defect Prediction[J]. Applied Sciences, 2020, 10(23): 8324–8348.

[56] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall and W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique[J]. J. Artif. Intell. Res., 2002, 16: 321–357.

[57] Kwabena Ebo Bennin, Jacky Keung, Passakorn Phannachitta, Akito Monden and Solomon Mensah. MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction[J]. IEEE Trans. Software Eng., 2018, 44(6): 534–550.

[58] Amritanshu Agrawal and Tim Menzies. 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, 2018: 1050–1061.

[59] Shuo Feng, Jacky Keung, Xiao Yu, Yan Xiao, Kwabena Ebo Bennin, Md. Alamgir Kabir and Miao Zhang. COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction[J]. Inf. Softw. Technol., 2021, 129: 106432.

[60] Jayalath Ekanayake, Jonas Tappolet, Harald C. Gall and Abraham Bernstein. Tracking concept drift of software projects using defect prediction quality[C]. Proceedings of the 6th International Working Conference on Mining Software Repositories, 2009: 51–60.

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

[62] Yusuf Sulistyo Nugroho, Hideaki Hata and Kenichi Matsumoto. How different are different diff algorithms in Git?[J]. Empir. Softw. Eng., 2020, 25(1): 790–823.

[63] Giovanni Rosa, Luca Pascarella, Simone Scalabrino, Rosalia Tufano, Gabriele Bavota, Michele Lanza and Rocco Oliveto. Evaluating SZZ Implementations Through a Developer-informed Oracle[C]. 43rd IEEE/ACM International Conference on Software Engineering, 2021: 436–447.

[64] Jacek Sliwerski, Thomas Zimmermann and Andreas Zeller. When do changes induce fixes?[J]. ACM SIGSOFT Softw. Eng. Notes, 2005, 30(4): 1–5.

[65] 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(2): 266–281.

[66] 梅元清，郭肇强，周慧聪，李言辉，陈林，卢红敏，周毓明. 面向对象软件度量阈值的确定方法：问题、进展与挑战[J]. 软件学报, 2023, 34(01): 50–102.

[67] Ahmed E. Hassan. Predicting faults using the complexity of code changes[C]. 31st International Conference on Software Engineering, 2009: 78–88.

[68] Akif Günes Koru, Dongsong Zhang, Khaled El Emam and Hongfang Liu. An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules[J]. IEEE Trans. Software Eng., 2009, 35(2): 293–304.

[69] Ranjith Purushothaman and Dewayne E. Perry. Toward Understanding the Rhetoric of Small Source Code Changes[J]. IEEE Trans. Software Eng., 2005, 31(6): 511–526.

[70] Todd L. Graves, Alan F. Karr, J. S. Marron and Harvey P. Siy. Predicting Fault Incidence Using Software Change History[J]. IEEE Trans. Software Eng., 2000, 26(7): 653–661.

[71] Sunghun Kim, E. James Whitehead Jr. and Yi Zhang. Classifying Software Changes: Clean or Buggy?[J]. IEEE Trans. Software Eng., 2008, 34(2): 181–196.

[72] Emad Shihab, Ahmed E. Hassan, Bram Adams and Zhen Ming Jiang. An industrial study on the risk of software changes[C]. 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), 2012: 62.

[73] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha and Naoyasu Ubayashi. A Large-Scale Empirical Study of Just-in-Time Quality Assurance[J]. IEEE Trans. Software Eng., 2013, 39(6): 757–773.

[74] Neha Gupta, Vinita Jindal and Punam Bedi. CSE-IDS: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems[J]. Comput. Secur., 2022, 112: 102499.

[75] Zhijun Ren, Yongsheng Zhu, Wei Kang, Hong Fu, Qingbo Niu, Dawei Gao, Ke Yan and Jun Hong. Adaptive cost-sensitive learning: Improving the convergence of intelligent diagnosis models under imbalanced data[J]. Knowl. Based Syst., 2022, 241: 108296.

[76] Debashree Devi, Saroj K. Biswas and Biswajit Purkayastha. Correlation-based Oversampling aided Cost Sensitive Ensemble learning technique for Treatment of Class Imbalance[J]. J. Exp. Theor. Artif. Intell., 2022, 34(1): 143–174.

[77] Faseeha Matloob, Taher M. Ghazal, Nasser Taleb, Shabib Aftab, Munir Ahmad, Muhammad Adnan Khan, Sagheer Abbas and Tariq Rahim Soomro. Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review[J]. IEEE Access, 2021, 9: 98754–98771.

[78] Akshma Chadha and Baijnath Kaushik. A Survey on Prediction of Suicidal Ideation Using Machine and Ensemble Learning[J]. Comput. J., 2021, 64(11): 1617–1632.

[79] Santosh Singh Rathore and Sandeep Kumar. An empirical study of ensemble techniques for software fault prediction[J]. Appl. Intell., 2021, 51(6): 3615–3644.

[80] Yoav Freund and Robert E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting[J]. J. Comput. Syst. Sci., 1997, 55(1): 119–139.

[81] Wenqiang Xu, Liangxiao Jiang and Chaoqun Li. Resampling-based noise correction for crowdsourcing[J]. J. Exp. Theor. Artif. Intell., 2021, 33(6): 985–999.

[82] Justin M. Johnson and Taghi M. Khoshgoftaar. Cost-Sensitive Ensemble Learning for Highly Imbalanced Classification[C]. 21st IEEE International Conference on Machine Learning and Applications, 2022: 1427–1434.

[83] Takafumi Fukushima, Yasutaka Kamei, Shane McIntosh, Kazuhiro Yamashita and Naoyasu Ubayashi. An empirical study of just-in-time defect prediction using cross-project models[C]. Proceedings of the 11th Working Conference on Mining Software Repositories, 2014: 172–181.

[84] Yibiao Yang, Yuming Zhou, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu and Hareton Leung. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models[C]. Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016: 157–168.

[85] Nachiappan Nagappan, Thomas Ball and Andreas Zeller. Mining metrics to predict component failures[C]. 28th International Conference on Software Engineering), Shanghai, China, May 20-28, 2006, 2006: 452–461.

[86] Rodrigo Morales, Shane McIntosh and Foutse Khomh. Do code review practices impact design quality? A case study of the Qt, VTK, and ITK projects[C]. 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, 2015: 171–180.

[87] Minghui Zhou and Audris Mockus. Does the initial environment impact the future of developers[C]. Proceedings of the 33rd International Conference on Software Engineering, 2011: 271–280.

[88] Christoffer Rosen, Ben Grawi and Emad Shihab. Commit guru: analytics and risk prediction of software commits[C]. Proceedings of the 10th Joint Meeting on Foundations of Software Engineering, 2015: 966–969.

[89] Khine Yin Mon, Masanari Kondo, Eunjong Choi and Osamu Mizuno. Commit-Based Class-Level Defect Prediction for Python Projects[J]. IEICE Trans. Inf. Syst., 2023, 106(2): 157–165.

[90] Farshad Kazemi, Maxime Lamothe and Shane McIntosh. Exploring the Notion of Risk in Code Reviewer Recommendation[C]. IEEE International Conference on Software Maintenance and Evolution, 2022: 139–150.

[91] Md. Nadim, Debajyoti Mondal and Chanchal K. Roy. Leveraging structural properties of source code graphs for just-in-time bug prediction[J]. Autom. Softw. Eng., 2022, 29(1): 27.

[92] Martin J. Shepperd, David Bowes and Tracy Hall. Researcher Bias: The Use of Machine Learning in Software Defect Prediction[J]. IEEE Trans. Software Eng., 2014, 40(6): 603–616.

[93] Andrew Jhon Scott and M Knott. A cluster analysis method for grouping means in the analysis of variance[J]. Biometrics, 1974, : 507–512.

[94] Marco Túlio Ribeiro, Sameer Singh and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier[C]. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016: 1135–1144.

[95] Jason Van Hulse, Taghi M. Khoshgoftaar and Amri Napolitano. Experimental perspectives on learning from imbalanced data[C]. Proceedings of the 24th international conference on Machine learning, 2007: 935–942.

[96] George G. Cabral, Leandro L. Minku, Emad Shihab and Suhaib Mujahid. Class imbalance evolution and verification latency in just-in-time software defect prediction[C]. Proceedings of the 41st International Conference on Software Engineering, 2019: 666–676.

[97] Sadia Tabassum, Leandro L. Minku and Danyi Feng. Cross-Project Online Just-In-Time Software Defect Prediction[J]. IEEE Trans. Software Eng., 2023, 49(1): 268–287.

[98] Hui Han, Wenyuan Wang and Binghuan Mao. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning[C]. Advances in Intelligent Computing, International Conference on Intelligent Computing, 2005: 878–887.

攻读硕士学位期间发表的论文

已发表论文

[1].Zhuang W, Wang H, Zhang X. Just-in-time defect prediction based on AST change embedding[J]. Knowledge-Based Systems, 2022, 248: 108852.(SCI一区期刊，已发表)

致　　谢

写到这里之时，三年的研究生生涯，乃至近20年的学生生涯，也即将画上句号，坐在电脑前，喜悦与不舍一时又涌上心头，自己的人生即将迈入新的道路，当然是值得高兴的；但是站在路口回首凝望过去的点点滴滴，自然也有很多不舍，不舍的是老师和同学们。所以借这个机会，对我身边关心，鼓励和支持我的人表示感谢。

首先是我的导师章晓芳老师，非常幸运能成为她的研究生。研究生三年以来，我在各个方面做的都不尽人意，不过章老师没有放弃我，一直都很耐心地在学习和生活上给与我很多帮助，让我得以顺利完成学业。毫不夸张地说，章老师是我人生中遇到的最好老师。

其次就是我实验室的同学们，感谢吴秉庭、杨馨悦在平日对我的帮助，还有其他师弟师妹，吴忆凡、汪创伟、毛瑞、薛琦、李立轩、杨慎之等等等等，感谢他们的支持。希望毕业以后仍有机会多与同学们相聚。

当然，还要感谢我的家人，三年以来，在物质上和精神上都无私地支持我，他们是我一直以来前进的动力。所幸以后开始工作，能对家人有所报答。

在我25岁的人生中，有7年是在苏大度过的，这里的一草一木我都很喜欢，以前司空见惯的风景以后可能很少有机会再看到，今年毕业季得多拍几张照片了。