北京信息科技大学

毕业设计（论文）

题　　目： 基于语句复杂度分析的软件缺陷定位方法研究与实现

学　　院： 计算机学院

专　　业： 软件工程

学生姓名： 燕子悦　**班级/学号** 软工 1601 班/2016011147

指导老师/督导老师： 崔展齐

起止时间： 2020 年 1 月 6 日至 2020 年 6 月 14 日

摘　要

软件开发过程中，用于软件调试的缺陷定位工作繁琐又不可或缺。本课题针对现有缺陷定位方法定位结果中存在大量相同可疑度语句的问题，对基于程序语句复杂度分析的软件缺陷定位技术进行了研究。采用机器学习的相关技术原理，首先，结合现有缺陷定位方法的定位结果，采集具有相同可疑度的源程序语句，并提取用于度量程序语句复杂度的特征以及类别标签，构建训练数据集。然后，采用决策树算法构建用于程序语句缺陷倾向性预测的分类器。最后，将该分类模型应用于实际，对现有定位结果中具有相同可疑度的程序语句进行区分，开发了一款基于语句复杂度分析结果进行软件缺陷定位工作的系统，以减少人工审查的代码量，从而提升软件缺陷定位工作的效率。本文在真实故障数据集 Defects4J 的项目上进行了实验。实验结果表明，本文构建的模型经十折交叉验证后的平均 F1 值约为 0.79，具有良好的分类性能。与现有基于频谱的缺陷定位方法中较为典型的 DStar 相比，本文所提出基于语句复杂度分析的方法能够有效降低需要人工审查的代码量，在一定程度上提升了软件缺陷定位的效率。

关键词：软件缺陷定位；软件缺陷预测；机器学习；语句复杂度分析

Abstract

In the process of software development, although defect locating for software debugging is very tedious, it is an essential task. Existing defect localization methods often have a large number of sentences with the same suspicious degree in the localization results. In order to solve this problem, the topic studies the software defect localization technology based on the sentence complexity analysis of program. The research on this subject was completed using the relevant technical principles of machine learning. First of all, combined with the positioning results of the existing defect positioning methods, source program sentences with the same suspicious degree were collected. At the same time, the features and category labels used to measure the complexity of the program statements are extracted to construct the training data set. Then, a decision tree algorithm is used to construct a classifier for predicting the defect tendency of program statements. Finally, the defect prediction model was applied to reality, and a software defect location system based on sentence complexity analysis was developed to distinguish program statements with the same suspicious degree in the existing location results. To reduce the amount of code that needs manual review, thereby improving positioning efficiency. In this paper, an experimental evaluation of the constructed model is carried out on the project of the real failure data set Defects4J. Experimental results show that the average F1 value of the model is about 0.79, which is based on 10-fold cross-validation. Compared with DStar, a typical defect location method in the existing spectrum-based defect location method, the method based on sentence complexity analysis proposed in this paper can effectively reduce the amount of code that requires manual review. To a certain extent, the efficiency of software defect location is improved.

Keywords: defect localization; defect prediction; machine learning; sentence complexity analysis

目　录

第一章　概述

1.1　研究背景和意义

近年来，随着互联网与信息技术的飞速发展，各种软件产品的使用者数量与日俱增。各行各业对软件产品的需求越来越迫切，也越来越复杂，各式各样的软件产品应运而生，软件数量飞速增长的同时软件规模也日趋庞大。

从软件工程的角度来看，寻找时间、质量、成本三要素之间的平衡点一直是科研人员的研究方向之一，其中影响软件质量的关键性因素就包括软件缺陷。软件中包含的缺陷数目与软件产品的质量息息相关，不难理解，软件中包含的缺陷数量越多，软件可靠性面临的挑战越大。当软件产品的数量和规模日趋庞大，发现和修复软件中缺陷的难度也就逐步增加。

对于互联网公司而言，将不稳定的、难以使用的软件产品随意发布上线不仅会带来重大的经济损失，还会降低品牌本身的信誉度导致失去用户。所以，为了保证软件的可靠性，在软件项目正式发布上线之前，通常要经过科学且系统的软件测试。软件开发及测试人员需要精准找到缺陷所在位置并进行修复，直至达到上线标准才可正式发布，这个过程往往会消耗掉大量的人力、物力与时间。有研究数据表明，在软件开发的过程中，用于修复软件缺陷的成本约占软件项目总成本的 50%~75%[1]。因此，对软件工程而言，发现并且修复软件产品缺陷的活动过程需要更加高效地方法和技术。

近年来，越来越多的研究人员关注到了软件缺陷的问题，并且已经研究出了基于频谱的、基于变异的等一系列软件缺陷定位技术。但无论传统的还是新兴的软件缺陷定位方法，所面临的挑战和悬而未决的问题仍然存在，其中最突出的是缺陷定位效率不高的问题，定位结果往往存在大量语句可疑度相同，难以区分的问题。这时候的定位结果仍然需要耗费大量人力进行人工确认，而这无疑加重了软件开发过程中成本方面的负担。因此，需要研究出有效提升软件缺陷定位精确度的方法，用于提升软件缺陷定位工作的效率。

1.2　国内外研究现状

早期的软件缺陷定位是以人工实现为主的，需要手动在程序的源文件中插入输出语句，或者设置断点进行程序调试，以此来找到程序缺陷的所在位置。这样的软件缺陷定位方法要求调试人员具有较高的专业能力，要先理解程序实体的结构框架、实现方法、程序语义、具体功能以及具体执行失败的特点。不仅难度大、耗时长，而且对人力的依赖性过强，早已不再适合当今软件市场中日趋复杂的软件产品。因此，半自动或者自动化的软件缺陷定位技术成为了越来越受科研人员青睐的热门研究方向，并且已经产生了一系列研究成果。

1.2.1 基于程序频谱的软件缺陷定位

程序频谱[2]指的是在第 i 条测试用例经过第 j 个程序实体时，用 k 表示用例经过该条语句的次数。该条语句被覆盖 k 次时有 $M_{ij}=k$，而该条语句未被覆盖则有 $M_{ij}=0$。

基于频谱的软件缺陷定位方法，在定位过程中主要运用软件测试时程序实体的覆盖信息，以及相关测试用例的执行结果，从而定位软件中缺陷语句的所在位置。该技术具有计算方法易操作以及消耗资源少的特点，并且该技术的定位效果较好，是目前软件缺陷定位问题的研究热点。研

究人员已经提出了很多基于频谱的缺陷定位方法[3]，其中较为经典的有：由 Jone 等人最先提出的 Tarantula[4]方法，以及 Abreu 等人随后提出的 Jaccard[5]方法和 Ochiai[6]方法。这三种基于频谱的缺方法在缺陷定位工作中都取得了不错的定位效果，并且 Ochiai 方法相比于 Tarantula 方法在定位效果上有了一定程度的提升[7]。此外，Naish 等人提出了 Kulczynski1 方法和 Kulczynski2 方法[8]，他们在计算程序实体的可疑度时，采用了自组装映射图相似度的计算方法[9]。Gonzalez 进行了排除正确程序语句的优化，从而提出了 Zoltar 方法[10]。随着研究的不断深入，Wong 等人[11]通过进一步分析测试用例在程序实体上，成功执行的数量对缺陷定位结果的影响，得出了被成功测试用例覆盖次数越多的程序实体，其覆盖次数对可疑值的贡献度越小这一结论，从而提出了 Wong1(s)、Wong2(s)和 Wong3(s)三个公式[3]。随后为了突出在成功用例数量较多时，失败用例对定位结果的影响，又提出了 DStar[12]方法，并且通过实验对比发现 DStar 方法的缺陷定位效果优于其他同类缺陷定位方法。

1.2.2 基于程序切片的软件缺陷定位

在缺陷定位方法上，基于程序切片是不同于基于程序谱的，程序频谱方法没有考虑程序间的依赖关系，而程序切片方法则将这一点列入研究。基于程序切片的缺陷定位方法[2]，在定位工作过程中，分析程序的输出变量以及执行路径，将输出语句或者与变量有所关联的语句做程序切片，从而达到缩小程序代码规模的效果。

早在 1984 年，程序切片用于软件缺陷定位首次引起了研究人员的注意[13]。Weiser 针对连续块假设测试了切片假设，后续 Weiser 和 Lyle 又做了进一步的扩展研究，通过自动工具进行程序切片，前后的实验结果都表明了程序切片技术用在软件缺陷定位工作上的有效性。随后，研究者们不断改进，产生了一系列与程序切片相关的软件缺陷定位技术。Lin 等[14]将动态切片与正向跟踪技术相结合，提出了一种新的缺陷定位方法，该方法在前向跟踪过程中利用了来自设计规范的信息，增强上下文以减小所构造动态切片的大小[15]。谢晓媛等[16]使用变质切片的概念开发了一种可扩展 SBFL 的框架，该框架在不需要测试预言机的情况下，可提供与现有 SBFL 技术相近的性能。许高阳等[17]提出了一种基于程序层次切片的缺陷定位方法，通过在包、类及方法层进行程序切片，逐渐减小程序规模，从而定位软件缺陷，并且取得了不错的定位效果。此外，董俊华[18]将程序频谱与程序切片相结合，提出了一种程序切片缺陷定位方法，并与 Tarantula、Jaccard、Ochiai 三种频谱方法进行对比，实验结果再次证明了程序切片缺陷定位方法能够有效提高软件缺陷的定位效率，从而达到减小缺陷定位的成本的目标。

1.2.3 基于机器学习的软件缺陷定位

近年来，人工智能可谓备受关注。如今，对于各个领域中重复性或规律性较强的工作任务，用机器取代人力是优先选择，所带来的效益不容小觑，而取得解放双手这场革命性进步的背后推手正是机器学习。机器学习的过程模拟人类的学习过程，一般指机器通过分析现有的数据或者说是解决问题的经验，给出解决当下问题的方案。

随着机器学习技术的不断发展，越来越大量的研究者通过机器学习技术，解决自身研究领域中的难题。其中，较为常见的有计算机视觉、密码学以及生物信息学等。当然，软件缺陷定位的研究者们也结合机器学习的相关技术，将程序实体的执行信息等作为输入，利用机器学习算法，学习并推导出软件缺陷的位置，改进了已有方法或提出了新的缺陷定位方法。陈理国等[19]提出了

一种高斯过程缺陷定位的方法（简称 GPBL），并对高斯计算过程进行优化。通过与基于隐含狄利克雷分布（LDA）模型的缺陷定位方法进行比较实验，证明了高斯缺陷定位的效果明显优于 LDA 模型。随后，何海江[20]把 Logistic、SGD、SMO 和 LibLinear 一系列分类学习算法应用于缺陷定位模型构建，并且通过实验证明了这类模型的缺陷定位效果都优于 SBFL 方法。紧接着，解铮和黎铭[21]基于损失函数对模型预测性能的影响，提出了一种代价敏感的间隔分布优化损失函数（简称 CSMDO)，并通过将该方法与深度卷积神经网络相结合，为进一步提高软件缺陷定位方法的精准度做出了贡献。就在 2019 年，邹大明等在方法级别的缺陷定位研究中，借助了自然语言处理的 learning-to-rank 进行排序学习，并且取得了良好的缺陷定位效果。

1.3 主要研究内容

在上一节的研究现状中已经大致描述了不同基础的现有缺陷定位技术，并且阐明了这些技术能够有效地进行缺陷定位工作，有助于提升软件开发效率从而降低成本。尽管如此，现有缺陷定位技术仍然存在不同的问题。其中，目前研究最广泛的频谱类缺陷定位方法也存在软件缺陷定位效率不高的问题，定位结果中往往含有大量因可疑度相同而难以区分的语句。这意味着在自动化定位缺陷之后，仍然需要耗费大量人力进行人工审查代码的工作，无疑会在一定程度上加重软件开发中的成本负担。

为了解决上述问题，提高现有缺陷定位方法的定位效率，从而实现最大化降低软件开发成本，本文在程序的语句级进行了缺陷定位研究，设计并实现了基于语句复杂度的缺陷定位方法，主要工作如下：

1） 分析程序语句复杂度。根据现有缺陷定位方法定位结果中具有相同可疑度的语句特点，设置并选择度量元，将其作为衡量程序语句复杂度的特征属性。

2） 构建缺陷预测模型。运用由 1）中的特征属性构成的训练数据集，通过机器学习算法训练合适的分类器，并根据分类效果进行模型优化，从而实现有效的缺陷预测模型。

3） 实现缺陷定位方法。将 2）中构建的预缺陷测模型与现有基于频谱的缺陷定位方法结合，实现输出定位到的软件缺陷位置信息。

经过上述系列工作后，本文主要做出了以下贡献：

1） 提出了一组用于分析程序语句复杂度的特征属性。能够有效分析现有缺陷定位方法中，具有相同可疑度的程序语句复杂度。

2） 构建出了通过程序语句复杂度进行缺陷预测的模型。能够对具有完整语句复杂度信息的程序语句进行分类，并给出有效的缺陷预测结果。

3） 设计并实现了一款用于缺陷定位的系统。将上述研究应用于实践，实现将语句复杂度分析的方法用于软件缺陷定位工作中，减少了人工审查代码过程的工作量，提升现有缺陷定位方法的精确度。

1.4 论文组织结构

第一章为概述。描述了软件缺陷定位的研究背景以及该方向的研究意义，突出了软件缺陷定位研究的重要性。同时介绍了较为主流且研究广泛的缺陷定位方法，主要包括基于程序频谱、切片以及基于机器学习的缺陷定位方法。还介绍了本论文的主要工作内容、贡献以及本论文的组织结构。

第二章为相关技术基础。介绍了软件缺陷预测和定位的具体技术，同时介绍了本文所涉及的

机器学习内容。在机器学习方面，重点介绍了模型构建过程，CART 算法实现分类决策树以及数据平衡算法 SMOTE。

第三章为基于语句复杂度分析的缺陷定位方法具体分析。介绍了特征属性的选取，以及如何设计数据提取算法，实现从现有故障库中提取相关特征数据。还介绍了如何构建缺陷预测模型并对所得模型进行评估验证。

第四章为缺陷定位系统的分析设计与实现。按照软件开发的不同阶段，介绍了基于语句复杂度分析的软件缺陷定位系统的开发过程。描述了定位系统各功能模块，从分析设计到实现的完整过程。并对本文重点研究的语句复杂度分析方法进行了叙述。

第五章为实验评估与分析。介绍了本文实验原始数据的来源 Defects4J 故障库，以及本文的实验过程，并且对实验结果进行了统计和分析总结。

第六章为总结与展望。总结了本文研究过程中的主要工作，包括当前工作中的不足之处，以及对未来工作的可改进之处的展望。

最后结束语向为本文工作提供支持的老师等相关人员致谢。

第二章　　相关技术基础

2.1　软件缺陷预测技术基础

软件缺陷预测的主要工作是充分利用软件产品中的信息，预测其所含缺陷[22]。纵观其技术发展，大体方向有静态技术和动态技术两种。其中，静态软件缺陷预测工作又根据其预测目标的不同，分为以下 3 种类型，并且可以分别与机器学习中所解决的三种问题类型相对应：

1） 缺陷倾向性预测：预测程序实体中是否含有缺陷，是典型的二分类问题。

2） 缺陷密度预测：预测程序实体中所含缺陷的数量和分布情况，可以视为回归分析的问题。

3） 缺陷严重程度预测：是一种介于倾向性和密度预测中间的类型，需要先预测倾向性概率或者密度，再根据预测结果的大小对程序实体按照降序排列，找出排名靠前的 *M* 个程序实体，可与排序问题相对应。

由于本文工作的重点之一是通过机器学习的方法，对具有相同可疑度的程序语句进行缺陷倾向性预测，所以在此重点解释软件倾向性预测技术的具体流程。如图 2-1 所示，基于机器学习的缺陷预测技术可分为 4 个步骤：

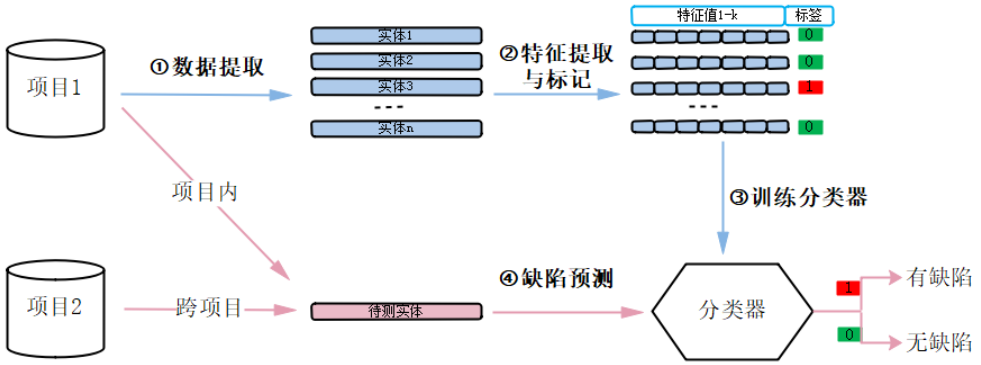

图 2-1 软件缺陷倾向性预测流程

1） 数据提取：利用人工或自动化技术，从现有项目中提取相应的程序实体，如文件、方法、语句等均可成为程序实体。此处提取出的程序实体一部分是用于构建分类器的训练数据，另一部分则是项目中待检测的程序实体。根据实际需求，待检测的实体和用于模型训练的实体可以来自同一项目或者不同项目。

2） 特征提取与标记：通过对程序实体的分析，提取其某些属性（特征），如代码行数、方法被调用次数、语句长度等。每一个实体的特征值都形成一个能够输入机器学习模型的特征向量，并且，在使用有监督学习算法构建模型时还需要对训练数据进行分类，也称为标记、标签。

3） 训练分类器：如本小节前文所述，软件缺陷倾向性预测对应机器学习中的二分类问题，此步骤即是利用机器学习中的分类算法构建一个能够用于二分类的分类器。所构建的分类器用于判断待测实体是否有缺陷。

4） 缺陷预测：利用构建好的分类器对待测实体进行缺陷预测。实际应用中，此时的待实体一般不能直接输入分类器，需要类似处理训练数据的方式，对其进行特征提取，形成特征向量，输入分类器后输出其预测结果，即是否有缺陷。

2.2 软件缺陷定位技术基础

软件缺陷定位工作的主要目的是找到源程序中隐含的错误实体，这里的实体可以是文件、类、函数或者程序语句等。软件缺陷定位可大致分为基于静态分和动态测试两种方式[23]。其中，基于动态测试的软件缺陷定位主要工作是，对源程序执行测试用例后，根据测试用例执的结果以及源程序中的覆盖信息，对软件缺陷的所在位置进行推断。

较为常见的 Tarantula、Jaccard、Ochiai 和 DStar 方法一般统计的指标通常有三方面。首先，测试用例的情况，包括在执行成功和失败两种情况下的用例数量，分别用 N_p 和 N_f 表示，以及测试用例的总数量 N。其次，覆盖语句 s 的用例数量，包括在该条语句上执行成功和失败的用例数量，并分别用 $N_p(s)$ 和 $N_f(s)$ 表示，以及覆盖该条语句的测试用例总数 $N_c(s)$。此外还有未覆盖语句 s 的用例情况，同样包括执行成功和失败的用例数量，分别用 $N_{up}(s)$ 和 $N_{uf}(s)$ 表示，还包括未覆盖语句 s 的测试用例总数 $NU(s)$。以上所说的所有总数均为对应情况下成功和失败用例的和。

在对上述指标进行统计后得到相关原始数据，再根据不同的公式计算程序实体的可疑度，而后根据可疑度对程序实体进行排序等操作，从而达到判断缺陷位置的目的。

2.3 机器学习技术基础

2.3.1 模型构建

利用机器学习解决问题的过程实际上是模型构建的过程。首先，将实际问题抽象理解成成数学问题，并选择相应的机器学习算法解决问题，而后通过学习来训练模型。机器学习可以根据学习方式的不同，分为（有）监督学习、无监督学习和半监督学习。其中有监督学习是本文研究中所涉及的，因此本节将详细描述有监督学习的建模过程。所谓（有）监督学习是指利用已有标签（分类）的样本集进行模型构建，并进行参数优化，从而得到具有较高性能的模型。机器学习中构建模型的流程一般包括如下步骤：

1） 数据收集：一般方式包括从网上下载免费的公开数据，从专业的数据公司购买或者根据实际需求自己主动收集用于问题研究的原始数据集。

2） 数据探索：在收集到数据后，一般需要通过数据可视化操作对数据进行分析，大致了解原始数据集的数据量、特征信息以及数据分布情况等。

3） 数据预处理：在数据探索过程中会发现原始数据可能存在数据缺失、分布不均或者数据中包含大量无用信息的问题。而这些问题数据将会影响所训练模型的性能，因此，需要进行数据清理、数据转换或者数据平衡等预处理工作，用于保证数据质量。

4） 训练模型：首先，选取与待解决问题相关的机器学习算法，通常不存在对所有问题都有很好效果的算法，因此需要根据所研究问题具体分析。然后利用经过预处理的数据对其进行训练。

5） 模型评估：一般在训练前会选定用于衡量模型性能的指标，训练之后用相应的指标判断模型训练结果是否理想。这里的模型性能评估指标也需要根据所研究的问题进行具体分析后再选择，比如对于解决分类问题的模型常用准确率评估，而对于解决回归问题的模型则常用误差来评估，还有许多其它不同的评估指标。

6） 模型优化：根据 5）中模型评估的结果判断所训练的模型性能是否符合预期效果，一般很难直接达到最优效果，因此，对于效果不理想的模型需要通过调整参数进行模型优化，从而提高模型的性能。

2.3.1 决策树

如前文所述，解决分类问题可以用经典的机器学习算法决策树。决策树可以通过 ID3 算法、C4.5 算法或者 CART 算法三种方式实现[24]。在决策树的特征选择过程中，ID3 算法用信息增益度量，C4.5 算法则是利用更加公平的信息增益比来度量。两者的共同之处在于信息增益和信息增益比均基于信息熵，这意味着两者工作过程中都需要进行大量的对数运算，会产生效率问题。而 CART 算法则改善了这种情况，采用基尼指数来度量。用基尼指数来代表模型的不纯度，并且基尼指数的大小与模型不纯度呈正相关。基尼指数的运用不仅简化了模型，同时也保留了熵模型的优点。在用决策树来解决分类问题的时候，如果样本类别有 K 个，并且将第 K 个类别的概率设为 P_k，那么基尼指数的计算方式如公式(2-1)所示：

$$Gini(p) = \sum_{k=1}^{K} p_k(1 - P_k) = 1 - \sum_{k=1}^{K} P_k^2 \tag{2-1}$$

决策树结构中的节点种类分为 3 种，即：根节点、内部节点以及叶子节点。其中，不包含入边且只有一个的被视为根节点；包含出和入两种边，且有多的为内部节点；同样有多个，但没有出边的为叶子节点。另外，利用 CART 算法实现的决策树将是一棵结构十分简洁的二叉树，其每一次决策结果非“是”即“否”。决策树的构建过程中，首先需要用标注好的训练数据集训练生成一颗树，通常也称之为模型。实际上，模型中所包含的是一系列规则，模型的使用则是利用这些规则，将未经标注的特征数据进行分类。对于本文研究中利用 CART 算法实现的二叉树而言，根节点和内部节点都有两个孩子，其每一个节点判断的结果非“是”即“否”，从而形成最终的二叉树。训练数据特征属性所对应的任意特征值，都可被视为决策树的节点，这种划分过程其实就是特征选择的过程，一般情况下会采取基尼指数来衡量划分的好坏，也就是判断具体应该采取哪个属性的哪个类别作为条件。对于每个属性值只能得到是或者否的判断，产生左右两个孩子，最终在判断完全结束后形成一颗二叉树。

2.3.2 SMOTE 算法

在机器学习过程中，训练数据类别不平衡是一种十分常见的现象。不平衡数据集是指在分类问题中，各个类别的样本数据量差距较悬殊。通常情况下，在不平衡数据集中，少数类的数据量占比仅为数据总量的 10-20%，甚至还会更少。用类别不平衡的数据集训练所得的模型，往往会对少数类别的预测精度(例如 precision 和 recall 等)较低，在数据严重不平衡状态下训练出来的模型对于少数类而言有效性极低。在数据类别不平衡的问题上，常用的解决办法大致分为两种，即对原始数据进行过采样或者欠采样。其中，过采法（Oversampling）一般是指增加原始数据中少数类的样本数量，反之，欠采样（Undersampling）一般指通过外部干预，减少原始数据中多数类的样本数量[25]。

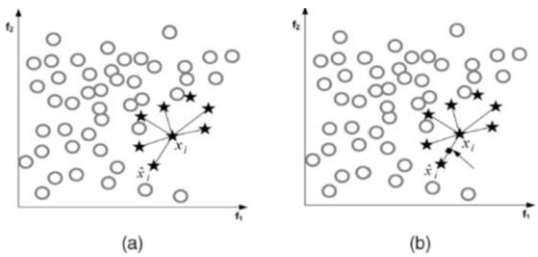

图 2-2 SMOTE 算法样本模拟

SMOTE(Synthetic minority oversampling technique)算法是实现过采样的一种方式，又叫做合成少数过采样算法。其工作原理不同于随机过采样，并不只是对原始样本中的少数类进行复制，而是通过分析该类样本数据的特点，而后根据现有样本特性模拟形成新样本，并将模拟的新样本添加到数据集中。因此 SMOTE 平衡数据也更具有代表性，被广泛用于解决机器学习中训练数据类别不平衡的问题。SMOTE 算法新样本的模拟过程采用了 KNN 技术[26]。如图 2-2 中(a)所示，五角星为少数类别样本，其中一个样本点用 x_i 表示，计算其 K 近邻，根据所需训练数据的平衡需求设置采样倍率 N，从 K 近邻中随机挑选若干样本点 x_j，对于任意近邻 x_j 可根据公式（2-2）进行新样本的创建，其中 rand(0,1)指在 0~1 范围内的随机数。如图 2-2 中(b)所示，x_i 和 x_j 之间的方块即为新样本通过 SMOTE 算法模拟的新样本 x_{new}。

$$x_{new} = x_i + \text{rand}(0,1) * |x_i - x_j| \tag{2-2}$$

2.4 本章小结

本章介绍了本论文研究过程中所使用的技术基础。第 1 和 2 小节分别描述了软件缺陷预测、软件缺陷定位的基本原理与大致流程。第 3 小节主要介绍了本文涉及的机器学习相关内容，重点描述了模型构建过程，常用于解决为分类问题的决策树算法，以及数据预处理过程中平衡数据的 SMOTE 算法原理。

第三章　基于语句复杂度分析的缺陷定位方法

根据本文概述，现有缺陷定位方法中最突出的问题是缺陷定位效率不高，定位结果往往存在大量语句可疑度相同，难以区分的问题。因此本文提出基于语句复杂度分析的方法，旨在区别可疑度相同的程序语句，提高现有缺陷定位方法的定位效率，减少人工审查的代码量，从而降低软件项目的成本。该方法的总体实现情况如下：

1） 分析程序语句复杂度。确定特征属性后从原始数据集中提取特征值。

2） 数据预处理。对 1）中提取的样本进行数据平衡。

3） 训练模型。利用平衡后的样本数据集训练分类决策树。

模型评估。使用模型评估方法对上述步骤的训练结果进行评估检验，同时根据模型性能适当调整参数，从而改进模型的分类性能。

3.1　语句复杂度分析

在软件工程的算法中有复杂度的概念，比如用来定性描述算法运行时间的复杂度，被称为时间复杂度（Time complexity）。本文的研究对象为程序语句，并且将复杂度的概念引入到程序语句中，用语句复杂度来描述执行该语句的难易程度。放到软件缺陷中，先通过分析程序语句的特定属性来确定其复杂程度，而后根据程序语句的复杂度来判断其缺陷情况，实现算法描述如表 3-1 所示。

表 3-1 复杂度分析算法

算法名称：复杂度分析算法
输入：程序语句
输出：程序语句各特征值

```
1   for i 0 to chars.length by 1 do   //chars 为存储语句的字符数组
2   if ch 是空格或换行  then   //ch 为当前字符
3   else if ch 是字母  then
4         while ch 是字母或数字  do
5            ch 加到 arr   //arr 是中间变量
6         end
7         if arr 是关键字  then
8            arr 添加到关键字链表
9         else if arr 是标识符  then
10           arr 添加到标识符链表
11  else if ch 是数字  then
12        while ch 是数字  do
13           ch 加到 arr
14        end
15           arr 添加到数字
16  else switch(ch){
17        case '+'//当前字符是运算符
18        添加至运算符链表
19        //分别识别运算符、界符添加至对应链表
20  Element element = new Element(各链表长度)
21  return element    //返回样本特征值
```

从本文主要研究目的出发，针对现有的基于频谱的软件缺陷定位方法 DStar 定位结果中，可疑度并列第一的程序语句进行了分析。首先从程序语句的组成成分来看，考虑了程序语句中的关键字、标识符、操作符以及分隔符。又考虑到成分类别相同时，不同取值对程序语句复杂度的贡献值也不尽相同，因此也根据成分的取值规划了该值所属的等级，类似于早期研究中为属性赋予权重的思想。最终，结合实际情况以及历史经验总结，本文基于 Java 语言，将标识符数量、关键字数量、关键字等级、操作符数量、操作符等级、分隔符数量以及元素总数确定为衡量程序语句复杂度的特征属性。关键字共包括一般意义上的 47 个关键字以及 11 个保留字，将定性的描述数据类型以及语句类别的关键字级别划分为 1 级，其它关键字则归为 2 级。操作符包括关系运算符、逻辑运算符、算术运算符和赋值运算符，将关系与逻辑运算符划分为 1 级操作符，其它则划分为 2 级运算符。

由于模型构建的过程对数字类型的训练数据相对友好，故而本文在确定特征属性后，利用词法分析器的原理，对原始数据集 Defects4J 中的项目进行了相应特征值提取。首先根据现有频谱定位方法 DStar 的定位结果得到程序语句信息，包括程序语句的可疑度值大小、所在项目文件以及行号等信息。而后配置 Defects4J 数据集，并利用该数据集提供的项目版本控制接口检出对应项目，从而获取到相应的源程序语句集。最后按照特征值所需编写词法分析器，并分析源程序语句集，统计得到所需的特征数据。考虑到并不是每条语句都能取到所有属性值，而训练数据出现空值会产生无效样本的问题影响到后续训练，故而本文在特征数据统计过程中将不存在的特征值记为 0，并且与本文研究所设定的特征属性取值均为数字类型相匹配。

3.2 平衡数据集

在研究软件缺陷的相关课题时，遇到不平衡的数据集是普遍问题，这是由软件缺陷的分布特点造成的。软件缺陷在程序中的分布大致符合“二八原则”，即在 20%的程序中，分布的程序缺陷约占缺陷总数的 80%。由此，有缺陷的程序实体数量通常会远小于无缺陷的程序实体数量，训练数据集普遍存在类别不平衡的问题也就显而易见了。数据不平衡问题带来的直接影响就是模型预测结果的错误率偏高。

表 3-2 真实值与预测值关系

预测值	真实值	
	P(1)	N(0)
P(1)	TP	FP(去真）
N(0)	FN（存伪)	TN

如本文第二章所述，缺陷倾向性预测实际上是解决二分类问题。本文构建的预测模型，在工作过程中产生的错误结果有不同类型，所带来的风险程度也不尽相同，真实值和模型预测值关系如表 3-2 所示，其中 P(1)代表缺陷实体，N(0)代表正确实体，错误类型主要有两种：

1） 将真实值为 P(1)的缺陷实体预测为 N(0)的正确实体。

在缺陷定位工作中，人工审查一般是不会对 FN 类的实体进行代码审查和测试工作的，这意味着软件中仍然存在某些缺陷被遗漏掉，在一定程度上影响了软件的可靠性。发布后造成漏洞再进行修复所带来的损失更远高于开发过程中的修复成本。

2） 将真实值为 N(0)的正确实体预测为 P(1)的缺陷实体。

缺陷定位过程中，人工审查通常会对 FP 类实体进行代码审查和测试工作，若该类数据体量较大，则在无形中增加了软件开发过程的人工成本，背离了自动化预测并定位软件缺陷，从而降低软件开发成本的初衷。

尽管上述两种错误类型都会带来一定的损失，但相比之下，1）中将真实值为 P(1)的缺陷实体预测为 N(0)的正确实体，造成软件可靠性降低所带来的损失，明显要大于 2）类型增加的人工成本。而造成 1）类型错误的一个重要原因就是在训练模型过程中，训练数据类别不平衡，真正缺陷实体的样本是少数类，而且数量相差较为悬殊导致模型训练不充分，影响了分类结果的准确性。

因为本文研究所用的原始样本数量级并不高，所以为了保证数据样本尽量充分的发挥作用，并且客观的反应模型的泛化能力，本文在数据平衡工作中选择了过采样的方法。工作过程中分别尝试了 RandomOverSampler 函数、SMOTE 函数、ADASYN 函数和 SMOTEENN 四种常用的算法。其中，对本文所用数据集平衡效果最好的是 SMOTE 函数实现的过采样，随机模拟并增加缺陷样本，使正负样本类别达到平衡状态。在对原始样本数据进行平衡前，样本中缺陷语句数量仅占约 13%左右，而正确语句数量约为 87%，数据不平衡情况较为严重。采用 SMOTE 方法平衡数据后，构建出了正负类别比例为 1:1 的平衡数据集。本文第二章中已经详细介绍过 SMOTE 算法的原理，在此不再赘述。

3.3 构建决策树分类器

构建决策树分类器的关键是准确定量的选择分类标准，也就是特征选择的过程。本文研究中采用 CART 算法实现分类决策树，如第二章所述，CART 算法是采用基尼指数来度量特征选择的。在分类问题上，相对于 ID3 和 C4.5 算法中的信息增益或者信息增益比，基尼指数的计算方法明显要简单许多。特别是本文所研究的缺陷倾向性预测，在本质上是解决二分类问题，采用基尼指数的效率会更高。其类别数量仅为 2，假设其中一类的概率为 p，那么显然另一类即为（1-p），进而得出简化后的基尼指数计算方式如下列公式（3-1）所示：

$$Gini(p) = 2p(1-p) \tag{3-1}$$

确定算法后，利用 DecisionTreeClassifier 方法构建分类器模型。其中参数 criterion 是用于控制特征选择度量标准的，根据本文研究所需，将其参数值设定为‘gini’，也就是采用 CART 算法实现一个决策树分类器。参数 splitter 用于控制切分点，将其参数值设定为‘best’，即在所有特征中找最好的切分点。由于本文研究过程中共设定了 7 个特征属性，因此对于参数 max_features 和 max_depth 的值均采用了默认的 None。另外，为了保证模型的唯一性，需要设定 random_state 的值，被称为随机数，使模型即使在不同的训练数据集上也能够以同样的规则进行划分。并且，该参数的最终取值需要通过调整参数来实现，也就是模型优化的过程。

找到最佳的参数取值后，将经过预处理的训练数据集代入模型进行训练。本文采用机器学习中的 fit 方法训练模型，该方法有两个必不可少的参数，对应的正是训练数据的两个组成部分。其一是样本的特征值，另一部分则是对应的标签，也称类别。用 fit 方法训练后的模型即为具有缺陷倾向性预测能力的决策树分类器。

3.4 K 折交叉验证评估模型

利用机器学习方法所构建的缺陷预测模型通常具有缺陷预测能力，但其预测的效果如何还需要通过模型评估方法来检验。机器学习中所使用的样本数据通常被分为训练集、验证集和测试集。训练集用于模型学习，同时模型的学习效果受参数影响，因此在模型构建过程中，通常会运用验证集来进行参数选择。测试集种则包含模型完全未知的样本数据，用于检验模型的学习能力，也称泛化能力。具体做法就是对比测试集中的实际标签和模型预测结果的匹配程度，从而计算模型的精确率、召回率或 F1 等值来判断模型效果。这种将原始数据划分为训练集和测试集来评价模型的方法就是一般的交叉验证方法。

上述的一般交叉验证方法对于数据量较小的机器学习而言，训练数据的利用率极度不充分，学习效果也不理想，并且，分割结果中的类别比例，与原始样本数据中的类别比例，不一定相同或相近。因此，本文研究过程中采用了 K 折交叉验证的方法，其中 K 取值为 10，即十折交叉验证。其原理仍然是交叉验证的思想，具体做法是将原始训练数据，采用不重复抽样的方法随机划分为 k 份，然后在训练模型的过程中，每次随机选其中 1 份数据作为当前训练的测试数据集，其余的 k-1 份数据均模型的学习样本，如此过程需要进行 k 次，直到所有数据都做过训练集和测试集，如此方式也降低了数据划分对模型性能造成的影响。过程中训练一次就产生一套规则，并且会在当次的测试集上对这套规则进行测试，还可以保存模型的评估情况。最后，k 组指标的平均值可作为模型学习效果的评估结果，也指当前的模型在 K 折交叉验证下所表现出来的性能。本文采用 sklearn 库中的 cross_val_score 实现十折交叉验证，即将参数 cv 设定为 10，将本文采集且经过预处理的训练样本，代入模型进行训练与验证。验证过程中还结合模型的性能度量方法实现了模型评估，用到的评估指标有 Precision、Recall 以及 F1 值。

3.5 本章小结

本章主要描述了本文基于语句复杂度分析的软件缺陷定位方法的研究过程。第一节介绍了语句复杂度分析的策略和分析结果。第二节描述了对于不平衡数据集的处理过程。第三和四节则与模型相关，分别描述了构建分类器的具体过程以及对于所构建模型的评估。

第四章　语句复杂度缺陷定位系统的分析设计与实现

本章按照软件项目开发的进程，分别介绍了语句复杂度缺陷定位(Sentence complexity analysis fault location)系统的需求分析、系统设计与系统实现三个阶段的工作情况。本文工作过程中，通过UML 进行了系统的分析与建模，还设计了系统的结构和处理流程。因此，本章也对各阶段具有代表性的图进行了解释。

4.1　语句复杂度缺陷定位系统的需求分析

需求分析阶段的任务是与客户沟通，收集和分析用户的真实需求，并将其通过项目各阶段参与者都能明晰的方式表达出来，比如形成需求规格说明书。需求阶段的工作为后续设计人员进行系统设计提供基础保障，准确获取到用户需求也是项目成功的基石。本文先根据缺陷定位的工作内容，对其业务流程进行了规划和设计，了解整体流程对目标的实现分别做出了哪些贡献，以便找到软件缺陷定位本质的工作流，为后续进行准确的用例分析做准备。

本文所涉及的缺陷定位系统，在完成定位工作时，需要先拥有缺陷预测模型，而后才能开展进一步的缺陷定位工作，得到缺陷语句所在位置。因此，本文根据不同的实现目标，将业务流程划分为训练模型和缺陷定位两个业务流程。并运用 UML 中一种由 Erickson 和 Penker 提出，被称为“Eriksson-Penker 业务扩展模型”的活动图来描述该过程。如图 4-1 所示，模型训练流程的目标在于训练并生成缺陷倾向性预测模型，而缺陷定位流程的目标则是获得缺陷语句所在的位置。在业务流程规划的概念中，流程也就意味着处理，因此也可以说本文的系统有两个处理。其中，对于模型训练的处理需要外部输入训练数据集，缺陷定位的处理需要外部输入待测数据，而整个缺陷定位系统将缺陷分析结果作为输出。

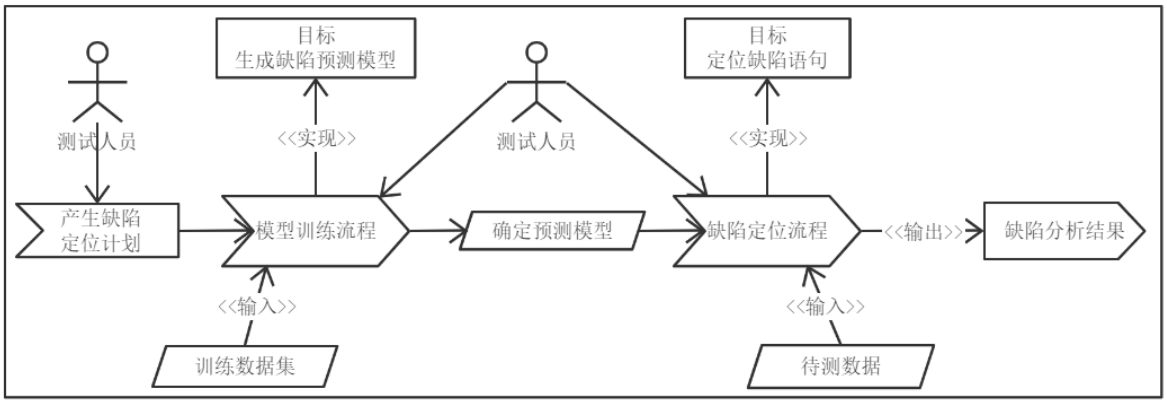

图 4-1 语句复杂度缺陷定位业务流活动图

运用 Eriksson-Penker 业务扩展模型进行业务流程规划后，本文对其中的每一个处理进行分析，得到了最本质的工作流。工作流明晰之后进行了需求分析，并通过 UML 中的用例图描述了用户与系统的交互情况。如图 4-2 所示，本文所实现语句复杂度软件缺陷定位系统，参与者为需要进行缺陷定位工作的测试人员，而系统主要有构建模型和定位缺陷两部分工作。

模型训练部分需要管理训练数据、预处理数据和训练模型，并且根据业务目标，这一部分需要产生缺陷预测模型。其中，管理数据需要包括导入本地数据和删除样本数据功能。预处理数据需要包括平衡数据、查看数据信息以及删除平衡数据的功能。训练模型需要包括一键训练与保存、

查看模型信息以及删除模型的功能。缺陷定位部分需要管理待测数据和分析缺陷倾向，需要得到分析结果。定位缺陷部分包括管理待测数据和分析缺陷倾向，根据业务目标，这一部分需要得到缺陷分析结果。其中，管理待测数据包括上传和删除待测数据功能。分析缺陷倾向性则需要一键分析、查看结果以及删除结果的功能。

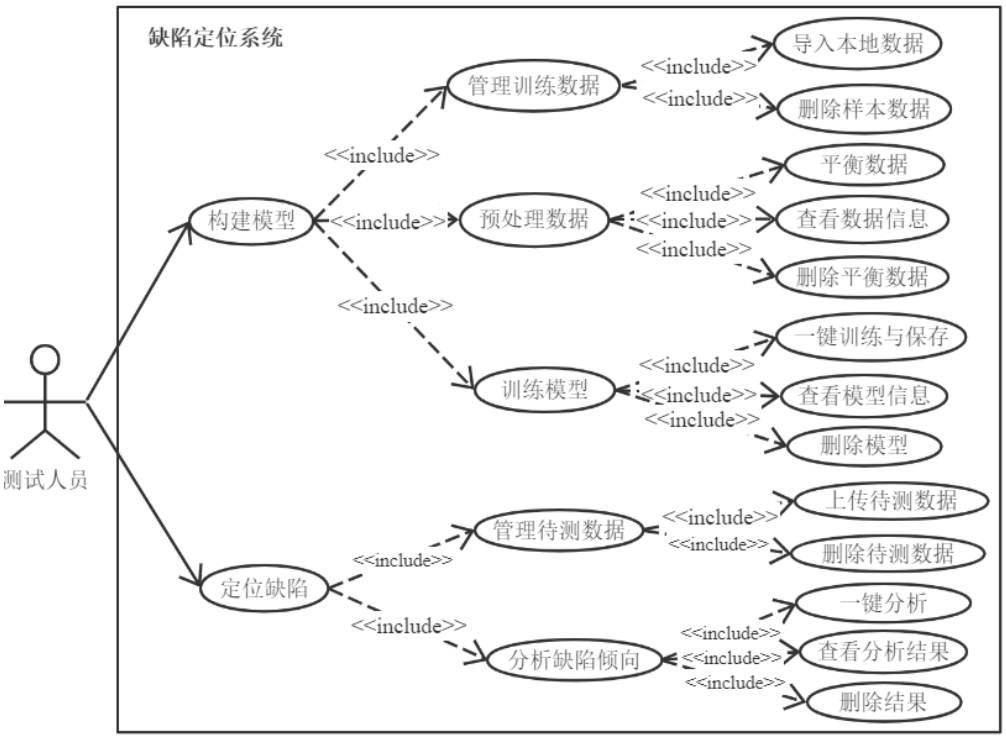

图 4-2 语句复杂度缺陷定位系统用例图

4.2 语句复杂度缺陷定位系统设计

在完成了业务流规划与分析，并且明确了需求之后，本文对系统进行了结构设计。根据软件缺陷定位工作中的两大需求，本文将语句复杂度缺陷定位系统分为构建模型和定位缺陷两大模块。又根据详细需求分别对这两大模块进行了细化，并且通过模块结构图进行表达。

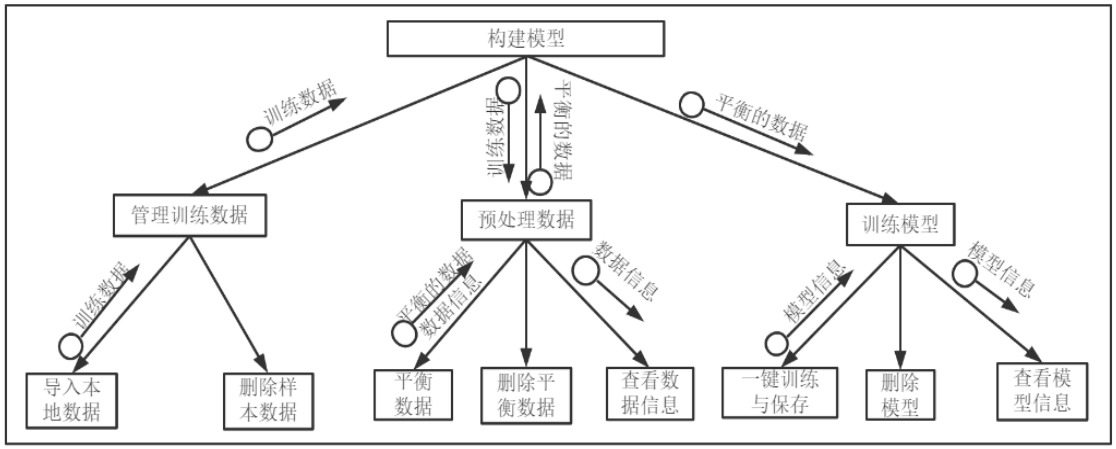

图 4-3 构建模型模块结构图

如图 4-3 所示，本文在设计过程中考虑到了同一模块内所包含的各子模块之间的相关性。并且在设计系统的时候，根据缺陷定位工作的特性考虑了不同模块的可扩展性。构建模型包括管理

训练数据模块、预处理数据模块和模型训练模块。其中，管理训练数据模块实现导入本地数据和删除样本数据功能。除了本地数据的上传，本文还考虑到后续改进工作中可以在此模块中增加其它数据导入形式。预处理数据模块包括平衡数据、查看数据信息和删除平衡数据功能。本文设计预处理数据模块实现平衡数据的功能，是为了消除机器学习中，类别不平衡数据对模型性能的不良影响，为训练模型工作提供优质数据。对于预处理数据模块的扩展，本文考虑到初平衡数据之外的其它预处理工作，如数据变换、数据规约等一系列预处理功能。训练模块包括一键训练与保存，查看模型信息和删除模型功能。

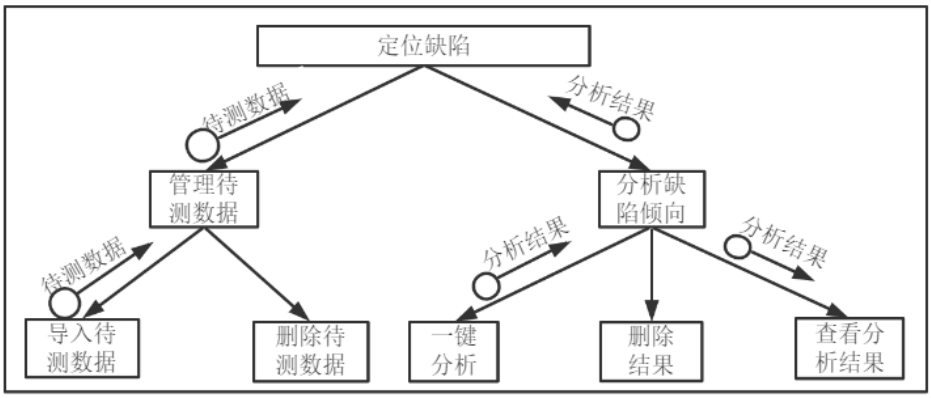

图 4-4 定位缺陷模块结构图

如图 4-4 所示，定位缺陷包括管理待测数据模块和分析缺陷倾向性模块。其中，管理待测数据模块包括导入和删除待测数据功能。对于此模块的可扩展性，同样考虑了除从本地导入待测数据以外的方式，甚至不同类型。分析缺陷倾向性模块包括一键分析缺陷，查看分析结束和删除结果功能。此模块的可扩展性在于对结果的处理，比如保存到本地或者上传到云服务以及分享至第三方等。虽然在本文的研究过程中不涉及上述的扩展需求，但为了系统更好的服务于软件缺陷测试工作，本文在设计过程中也考虑到了系统各模块的可扩展性。

本文在进行系统结构设计的同时还对系统的处理流程进行了规划，并通过流程图描述了语句复杂度缺陷定位系统各功能模块的处理流程。系统完整的流程图篇幅过长，在此仅截取系统构建模型部分的流程图，并在相应功能下对系统的处理流程进行必要的设计说明。其中构建模型子系统的管理训练数据、预处理数据和训练模型流程分别如图 4-5、图 4-6 以及图 4-7 所示。

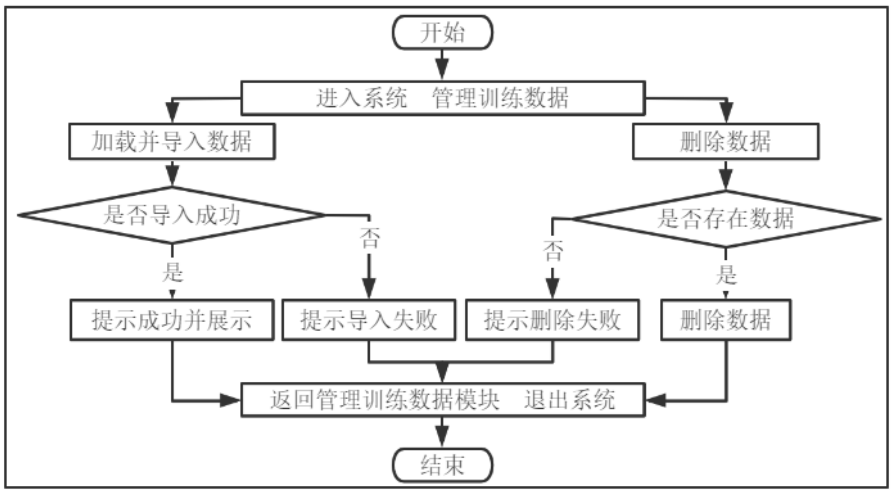

图 4-5 管理训练数据流程图

用户进入语句复杂度缺陷定位系统的构建模型模块，选择管理训练数据，此时由系统的处理流程如图 4-5 所示。当用户选择导入训练数据时，系统加载用户选择的本地数据并上传至系统，并且系统需要判断是否成功导入。若用户导入的数据合法并已经成功导入，系统提示用户导入成功，并向用户展示其所导入的数据，否则提示用户导入失败。无论导入是否成功，处理结束后均停留在管理训练数据功能模块内。当用户选择模块内的删除数据时，系统需要判断是否存在数据，若存在则删除数据并清空展示列表，否则提示用户无数据可删除。同样，无论删除与否，处理结束后均停留在管理训练数据模块内，等待用户选择进入其它模块或退出系统。

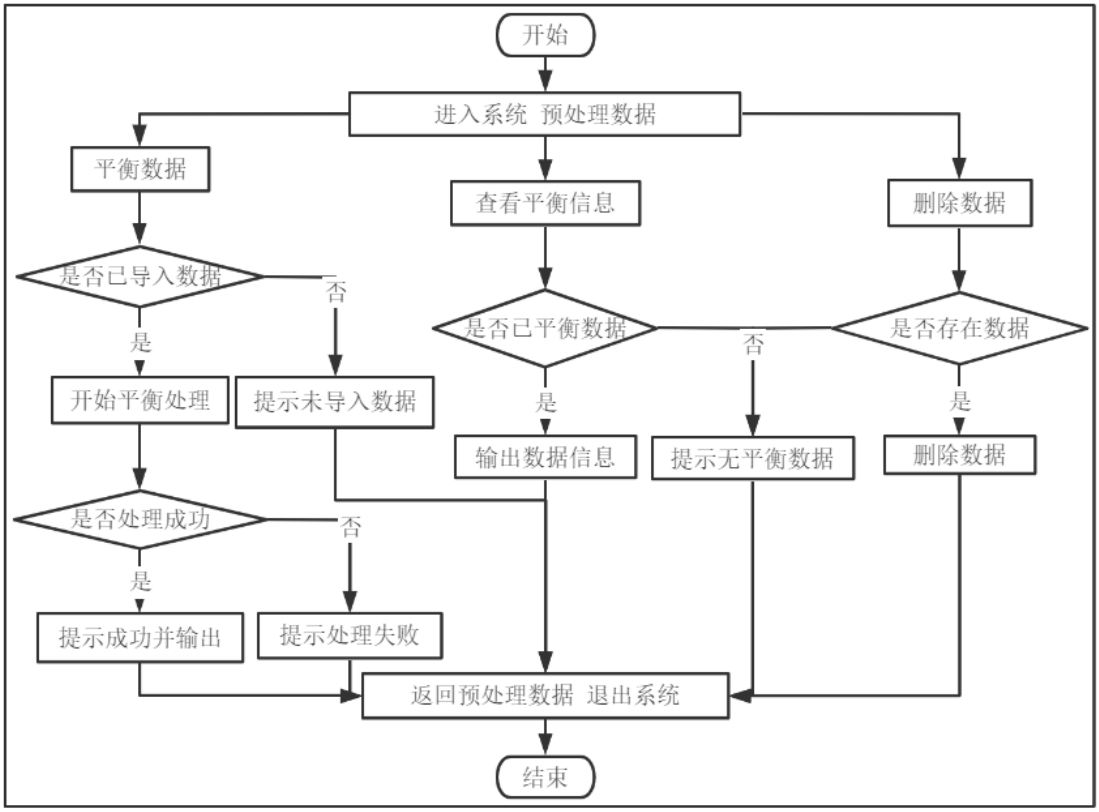

图 4-6 预处理数据流程图

用户进入语句复杂度缺陷定位系统的构建模型模块，选择预处理数据，此时由系统处理流程如图 4-6 所示。当用户选择平衡数据时，根据业务需求，在机器学习的模型训练过程中，需要依次完成数据的导入，数据预处理以及模型训练的工作。所以在系统进行预处理的平衡操作之前，首先需要判断是否存在样本数据，如果不存在可供使用的样本数据，也就是在没有导入数据的情况下，系统将提示用户未导入样本数据，反之则正常执行平衡处理。在平衡处理结束后，系统需要判断是否处理成功，处理成功时提示用户平衡成功并展示平衡后的数据列表，否则提示用户处理失败。无论成功与否，系统处理结束后，均停留在预处理数据功能模块内。当用户选择预处理数据模块内的查看平衡信息时，系统需要判断是否存在平衡数据记录，若存在则输出数据平衡前后的变化信息，否则将提示用户无平衡数据，处理结束后同样停留在预处理数据功能模块内。当用户选择模块内的删除数据时，系统需要判断是否存在平衡数据，若存在则删除数据并清空平衡数据展示列表，并且同时删除该组数据的平衡信息，否则提示用户无数据可删除。同样，无论删除与否，处理结束后均停留在预处理数据模块内，等待用户选择进入其它模块或退出系统。

用户进入语句复杂度缺陷定位系统的构建模型模块，选择训练模型，此时由系统处理流程如图 4-7 所示。用户选择一键训练并保存，根据业务需求，系统在开始训练之前需要先判断是否存在可用的训练数据，如果不存在，系统提示缺少训练数据，反之则开始训练模型，在训练结束后系统判断是否训练成功，若成功则系统给出相应提示并保存模型至本地供用户复用，否则提示用户训练失败。无论成功与否，系统处理结束后均停留在训练模型模块内。当用户选择模块内的查看模型信息时，系统需要判断是否存在训练模型的记录，若存在则输出模型的基本性能信息，否则提示用户无模型，处理结束后同样停留在训练模型模块内。当用户选择模块内的删除模型时，系统同样需要判断是否存在模型，若存在则删除模型，并且删除模型性能信息，否则提示用户无模型可删除。同样，无论删除与否，处理结束后均停留在训练模型模块内，等待用户选择进入其它模块或退出系统。

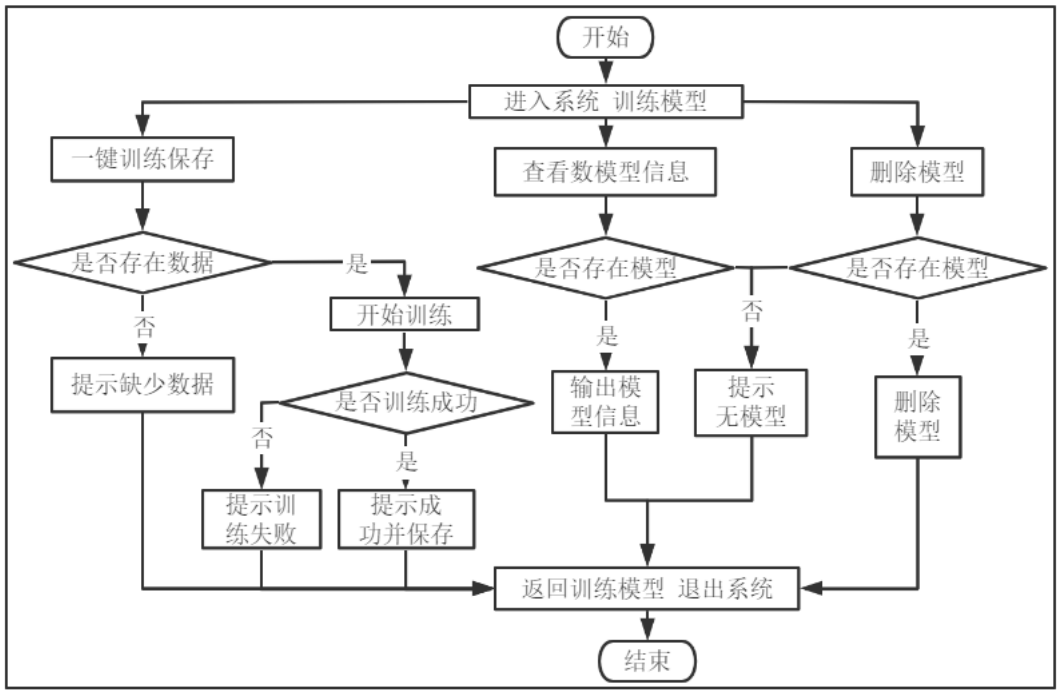

图 4-7 训练模型流程图

4.3 语句复杂度缺陷定位系统实现

4.3.1 开发环境及配置

本文所涉及语句复杂度缺陷定位系统的开发环境为 windows 10 操作系统，Intel(R) Core(TM) i7-8550U 1.80GHz 处理器，8G 内存。开发工具前端为 VS Code 1.45.1，后端为 IDEA 2019.1.1 并配置 JDK 8 和 python 3。数据库采用 MySQL 8.15.1，浏览器为 Google Chrome。

4.3.2 系统功能的实现

本系统的开发致力于使用本文所研究的语句复杂度分析模型进行缺陷定位工作，其核心算法在于语句复杂度分析方法的实现，包括分析语句复杂度和构建模型两部分。尽管在进行机器学习构建模型之前涉及到很多数据采集工作的实现，但均服务于模型构建，在此不宜赘述。另外，语句复杂度分析方法的实现思想已经在第三章中进行了描述，所以，在此仅对缺陷定位系统开发过程中应用该方法构建模型，实现缺陷定位的关键步骤进行阐述。实现算法描述如表 4-1 所示。

表 4-1 模型构建算法

算法名称：模型构建算法
输入：训练数据
输出：模型

```
1   import pandas, sklearn, seaborn……   //加载数据处理库
2   def decision_tree_classifier():   //定义构建类
3         pd.read()   //读取数据
4         oversampling_SMOTE.fit_sample(features, target)   //平衡数据
5         for best_r in range():   //寻找最优参数
6               DecisionTreeClassifier(random_state=best_r)   //创建决策树
7         tree.fit(features_os, target_os)   //以平衡数据训练决策树
8         cross_val_score(..., scoring='f1')   //以 f1 值评估模型
9         plt.show()   //结果可视化
```

首先，定义一个训练类，导入训练数据。通过 pandas 的 read 方法实现读取数据，并且对获取的数据进行预处理，本文主要实现的预处理工作是数据平衡，将待平衡数据的特征值与对应标签分别作为 SMOTE 函数的参数，平衡处理后得到可用于模型训练的数据集，同样由特征值和标签两部分构成。得到训练数据后的下一个步骤是构建分类器并进行参数优化。本文系统实现过程中，运用 DecisionTreeClassifier 方法实现决策树分类器（模型）的构建，range 规划区间，根据模型性能对其进行参数调整。最终，采用最优参数构建模型，并传入训练数据供其学习。模型训练结束后还需对其性能进行评估，以确定是否采用该模型进行后续缺陷定位工作。本系统在实现过程中采用 cross_val_score 方法完成十折交叉验证的工作。其中，代表评估指标的参数分别设定了 recall、precision 和 F1 值。并且运用 plt.show()方法将结果进行可视化输出，观察和记录实验结果。语句复杂度分析模型的构建服务于缺陷定位工作，系统的终极需求是实现缺陷定位。本系统对于定位工作的实现，首先通过 pandas 的 read 方法获取到待测数据，而后用过 predict 方法实现对所构建模型的调用，并将其分类结果输出，为后续工作提供参考依据。

在语句复杂度分析缺陷定位系统的开发过程中，后端机器学习模型构建的部分用 python 编写，其它逻辑处理通过 Java 编写，采用 Spring boot 和 MyBatis 框架，前端采用 Vue 编写。前端界面整体风格如图 4-8 所示。

图 4-8 语句复杂度缺陷定位系统前端整体风格

根据本文语句复杂度缺陷定位系统设计详情，构建模型模块中实现了管理训练数据、预处理

数据以及训练模型的功能。其中，管理训练数据中的导入训练数据功能由后端编写接口importRawData 实现从本地上传数据文件，供前端调用。平衡数据功能，由后端编写 balance_data 类，其原理第二章中已经详细描述过，此处不再赘述。并在接口 balanceRawData 中执行该 python 类，供前端调用，实现平衡数据向用户展示。训练模型中的一键训练并保存，后端编写 trainModel 接口供前端调用，该接口执行 decision_tree_classifier 类实现模型的训练，并且将训练模型保存至本地供后续复用。同时，该类执行后返回模型评估信息，仍然可通过 Java 获取，并将相应参数传给前端，为用户提供查看模型信息的功能。构建模型子系统中的删除功能，分别编写对应的后端接口供前端调用。其中 deleteRawData 用于删除用户导入的本地样本数据，deleteBalanceData 用于删除平衡后的数据，deleteModel 用于删除模型。在删除操作前要根据各自的功能需求进行相应判断。如在删除源数据之前判断是否存在平衡后的数据，若存在则需要一并删除。

缺陷定位子系统包括管理待测数据和分析缺陷情况功能。其中，管理待测数据中的导入功能实现方式与构建模型模块中的导入训练数据功能实现方式相似，以接口调用的方式进行数据传输。定位结束后其结果为 ndarray 类型数组，将该结果存入数据库供后续查看与使用。查看结果的实现则通过后端接口从数据库读取行号、程序源码以及预测结果（缺陷倾向性），通过接口getLocationResultListByPage 反馈到前端，并以表格形式呈现给用户。定位结果以及数据的删除同理于构建模型子系统中的删除功能，此处不再赘述。

本文工作过程中还实现了对现有缺陷定位方法定位结果的提取转化，以及从 Defects4J 中采集原始样本数据，并进行特征值提取。其中对于现有定位结果的转化和提取通过 python 数据处理库 pandas 和 numpy 中相应的的数据转化方法实现。而特征值的提取过程结合本文研究所使用的特征属性，通过编写一个对能够对 Java 程序进行识别的词法分析器实现，将 Java 程序按照本文所需提取的内容进行分析，并将分析结果以链表形式存储，最终通过计数得出相应的特征值。

4.4 本章小结

本章按照软件开发流程中的各个阶段划分，对本文语句复杂度缺陷定位系统的分析设计与实现过程进行了描述。第一节是对系统的需求分析并通过 UML 表达，形成了用例图。第二节是根据第一节中需求分析的结果进行具体的功能模块设计。第三节描述了编程实现缺陷定位系统的环境及过程，并对关键功能的流程和实现进行了解释。

第五章　基于语句复杂度的缺陷定位方法实验评估与分析

上一章中介绍了缺陷定位工具的设计与实现。本章介绍研究过程中的实验情况。首先介绍实验对象的来源，首先介绍我们所选取的实验对象，然后介绍实验评估指标，最后与现有缺陷定位方法的结果进行对比试验，从而验证本文提出的基于语句复杂度分析缺陷定位方法的高效与实用性。

5.1　实验数据集

本文研究的目的是降低缺陷定位工作中需要人工审查的代码量，从而提高现有缺陷定位方法的定位效率，为了保证方法的实用性，对本文所构建缺陷预测模型的有效性以及语句复杂度缺陷定位方法的定位效果进行了实验评估。所使用的实验对象来均来源于 Defects4J 数据集。该数据集是软件缺陷研究中最为广泛使用的公共数据集之一，可从 http://defects4j.org 公开获得。Defects4J 是一款真实故障数据库，用于对 Java 程序进行受控测试研究，并且该数据集可扩展。本文研究过程中使用的 Defects4J 版本为 2.0.0，其中包括个 17 个项目共计 835 个错误版本，错误数量统计详情如表 5-1 所示。

表 5-1 Defects4J(V2.0.0)包含的真实错误

项目 ID	项目名称	错误数量
Chart	jfreechart	26
Cli	commons-cli	39
Closure	closure-compiler	174
Codec	commons-codec	18
Collections	commons-collections	4
Compress	commons-compress	47
Csv	commons-csv	16
Gson	gson	18
JacksonCore	jackson-core	26
JacksonDatabind	jackson-databind	112
JacksonXml	jackson-dataformat-xml	6
Jsoup	jsoup	93
JxPath	commons-jxpath	22
Lang	commons-lang	64
Math	commons-math	106
Mockito	mockito	38
Time	joda-time	26

Defects4J 中的所有项目均是采用 Java 语言编写的，并且每个错误都是独立的，不包括功能或者重构等更改，并且每个程序版本中只包含一个错误。此外，Defects4J 为了便于研究人员重现错误，还在数据库抽象层提供了一系列接口，其中之一是提供了用于检出 V_{bug} 和 V_{fix} 的统一界面，使用户无需了解该项目的版本控制系统即可访问数据库中任何项目的 V_{bug} 和 V_{fix}，本文用于实验的项目源码便是通过项目检出采集而来。

另外，本文实验过程中使用的是 windows 10 操作系统，配置 Intel(R) Core(TM) i7-8550U 1.80GHz 处理器，8G 内存的计算机。VMware Workstation 10.0 搭载 Ubuntu16.04 并配置 Defects4J

2.0.0。

5.2 实验设计

本文选取 Defects4J 中 JFreeChart 、Apache commons-Lang 和 Joda-Time3 个项目，共计 116 个错误进行了实验。首先基于现有缺陷定位方法 DStar 在上述 3 个项目上的定位结果，从 Defects4J 中检出对应项目，而后提取结果中可疑度值排名第一并且包含真实错误的源程序语句，共计 371 条原始样本数据，详细统计信息如表 5-2 所示。

表 5-2 实验数据来源统计

项目名称	项目版本数量	样本数量
jfreechart	10	152
commons-lang	21	190
joda-time	4	29
合计	35	371

首先对于本文所构建缺陷预测模型的有效性，采用精确度(Precision)衡量模型将无缺陷的实体错误的预测为有缺陷实体的误报程度，采用召回率(Recall)衡量模型将缺陷实体预测为正确实体的漏报程度。这两者之间存在矛盾，呈现此消彼长的态势，模型的预测效果应当全面考虑，其中某一指标很高只是假象，因此，同时采用介于两者之间的 F1 值(F1)，综合衡量本文缺陷预测模型误报程度和漏报程度的一个指标进行评估。三种评价指标的计算公式如下：

$$Precision\ (P) = \frac{TP}{TP+FP} \tag{5-1}$$

$$Recall\ (R) = \frac{TP}{TP+FN} \tag{5-2}$$

$$F1 = \frac{2PR}{P+R} \tag{5-3}$$

其中 *TP* 的含义为真正例(True Positive)，即存在缺陷的实体被正确预测到的样本数量；*FP* 代表的是假正例(False Positive)，即正确实体被错误预测为有缺陷实体的样本数；*FN* 的含义则为假负例(False Negative)，即有缺陷的实体而被错误的预测为无缺陷实体的样本数量。

本文提出语句复杂度分析方法的初衷是服务于软件缺陷定位工作，减少人工审查代码量从而提升现有定位方法的定位效率。因此，本文在真实故障数据集 Defects4J 上对该方法的实用性进行了验证。软件缺陷定位方法的评价指标中，WET-N(Wasted Effort at Top-N)指标代表前 *N* 条可疑语句中，出现第一条真正缺陷语句所需审查的语句数量，侧面体现的是定位到错误所需的人工成本。根据本文研究内容，仅考虑在可疑度最高的 *N* 条语句列表中，找到真正缺陷语句时需要审查的程序语句数量。

实验选取 jfreechart、commons-lang 和 joda-time 三个项目，共计 35 个版本进行了定位工作。定位工作基于现有缺陷定位方法定位结果中，排名并列第一且包含真正缺陷的样本。通过对比本文方法与现有缺陷定位方法 DStar，在定位过程中找到第一条缺陷语句时需要人工审查的代码量，判断本文方法是否能够提升定位效率，以及在有效提升的情况下其提升程度是多少，并通过对比实验进行验证。

5.3 结果分析

如前文所述，本文通过十折交叉验证的方法，对所提出的缺陷预测模型进行了预测效果评估，评估指标分别采用 Precision、Recall 以及 F1 值，实验结果统计如表 5-3 所示。

表 5-3 十折交叉验证结果

序号	Precision	Recall	F1
1	0.69	0.63	0.66
2	0.73	0.6	0.66
3	0.87	0.81	0.84
4	0.78	0.91	0.84
5	0.87	0.84	0.86
6	0.9	0.81	0.85
7	0.77	0.84	0.81
8	0.85	0.69	0.76
9	0.85	0.88	0.86
10	0.81	0.91	0.85
均值	0.81	0.79	0.8

根据计算公式（5-1）可以看出，Precision 值越高，代表模型将无缺陷的实体预测为有缺陷实体的误报程度越低，模型的预测效果越好。在表 5-3 预测模型的 10 组评估结果中，Precision 值超过 0.8 的比例占 60%，无 0.6 以下的结果，并且有 1 组超过 0.9，平均 Precision 达到 0.81。以上结果表明，本文预测模型正确判断无缺陷语句的能力较强，体现了其在减少人工审查代码量这一目标上的有效性。

根据计算公式（5-2）可以看出，Recall 值越高代表模型将缺陷实体预测为正确实体的漏报程度越低，模型预测效果越好。在表 5-3 预测模型的 10 组评估结果中，Recall 超过 0.8 的占比 70%，无 0.6 以下的结果，并且有 2 组超过 0.9，最终平均 Recall 达到 0.79。以上结果表明，本文所构建模型正确判断有缺陷语句的能力较强，能够尽量避免真实错误语句被漏报，体现出其是有利于保障软件可靠性的。

根据计算公式（5-3）可以看出，F1 值是由 Precision 与 Recall 共同决定，代表一种平衡状态，其值越高意味着模型的误报和漏报率的平衡程度越好，也就是模型的综合预测能力越强。在表 5-3 预测模型的 10 组评估结果中，F1 高于 0.8 的占比 70%，无 0.6 以下结果，最终均值达到 0.80。表明本文所构建模型对于有缺陷语句和无缺陷语句均具有良好的判断能力，能够在尽可能多检测到缺陷语句的同时，减少人工审查代码量，能够消耗尽量少的成本同时保证软件的可靠性。

综上所述，本文通过十折交叉验证的方法评估模型性能，验证结果表明本文所构建基于语句复杂度分析的模型具有良好的预测性能。

表 5-4 语句复杂度分析方法有效改进的版本量

项目名称	版本数量	有效改进版本量	改进版本占比
jfreechart	10	8	80.0%
commons-lang	21	18	85.7%
joda-time	4	2	50.0%

本文在定位的过程中，通过语句复杂度方法对Defects4J三个项目人工审查代码量的降低程度，判断该方法在减少人工审查代码量，提升缺陷定位工作效率上的有效性。本文统计了语句复杂度

分析方法对 Defects4J 中三个项目各版本的改进数量，以及改进的版本数量占实验数据总版本数量的比例，详情如表 5-4 所示。

实验过程中，还通过对比本文提出的语句复杂度分析方法与现有 DStar 方法，在缺陷定位过程中有效减少人工审查语句的数量，来检验其对缺陷定位工作效率的提升程度。本文基于 Defects4J 中 jfreechart、commons-lang 和 joda-time 三个项目的各个版本进行实验，分别统计了语句复杂度分析方法与现有缺陷定位方法 DStar，找到第一条错误语句所需要审查的代码行数，同时计算了语句复杂度分析方法减少审查的代码量占比。由于各版本的统计数据篇幅过长，在此仅以合并版本的项目为单位展示统计结果，详情如表 5-5 所示。

表 5-5 SCAFL 与 DStar 方法的代码审查情况对比

项目名称	DStar 方法	SCAFL 方法	SCAFL 有效提升效率
jfreechart	80	22	72.5%
commons-lang	154	34	77.9%
joda-time	7	4	42.9%

根据表 5-5 中数据还发现本文提出的语句复杂度分析方法，对于数据量越大的项目定位效率提升程度越明显。另外，在细化统计每个版本情况的过程中发现，本文提出的语句复杂度分析方法对于其中约 1/3 的项目版本，可以直接定位到第一条错误语句，不需要额外审查正确语句。由此可见，本文提出的语句复杂度方法能够有效减少缺陷定位工作中人工审查代码的量，并且对于现有方法的定位效率有明显提升。

5.4 本章小结

本章对本论文研究过程中的实验部分进行了描述。第一节对实验所使用的数据集 Defects4J 以及实验环境进行了介绍。第二节重点描述了研究过程中的实验所设计的方式以及设计理由。第三节是对实验结果的分析和总结。

第六章　总结与展望

6.1　本文工作总结

典型的基于频谱的缺陷定位方法 DStar，定位结果中往往存在大量可疑值相同的语句，人工审查代码成本相对较高。针对这一问题，本文提出了基于语句复杂度分析的缺陷定位方法，首先，根据 DStar 方法在 Defects4J 数据集里 3 个项目上的定位结果，提取可疑值并列第一的程序语句，对其进行特征提取与类别标记，构造原始训练数据集。随后运用预处理后的数据集训练产生分类决策树模型，用于对程序语句进行缺陷倾向性预测，并对所构建的模型进行评估。最后，将理论研究过程所构建的缺陷预测模型，用于软件缺陷定位系统的开发，最终实现了一款用于缺陷定位工作的系统。综上所述，本文主要贡献如下：

1） 提出了一组用于衡量程序语句复杂度的特征属性。能够有效分析现有缺陷定位方法中，具有相同可疑度的程序语句复杂度。

2） 构建出了通过程序语句复杂度进行缺陷倾向性预测的模型。能够根据程序语句的复杂度信息，对程序语句进行分类，给出有效的缺陷分析结果。

3） 设计并实现了一款用于缺陷定位工作的系统。将上述研究应用于实践，运用语句复杂度分析的方法辅助软件缺陷定位工作，减少了人工审查代码过程的工作量，提升了现有缺陷定位方法的精确度，一定程度上降低了软件项目消耗的成本。

6.2　未来工作展望

本文在研究过程中主要关注了基于频谱的缺陷定位方法 DStar，致力于提升该方法的定位效率。基于现有真实故障数据库 Defects4J 中的项目，对 DStar 定位结果中可疑度值并列第一，且包含缺陷的语句进行了特征分析。通过构建缺陷倾向性预测模型再次精准定位缺陷，实现缩小人工审查代码范围，从而提高现有缺陷定位的效率。由此可见，本文研究的后续工作还可以进行以下几点改进：

1） 训练数据集来源：本文研究过程所用到的训练数据集是单一的，在后续研究中可以尝试多元化。使用其它的现有公开数据集或者针对特定项目进行训练数据提取。

2） 可疑度相同的语句：本文主要关注了可疑度并列第一并且其中包含真正缺陷的语句，后续研究中还可以考虑将可疑度按梯队划分，如能够分开考虑并列第一、并列第二或其它情况。

3） 模型的普适性：本文提出的模型只在 DStar 方法上进行了实验，后续研究可以考虑将模型迁移至其它已有的缺陷定位方法中，判定对于除 DStar 以外的现有软件缺陷定位方法是否具有提升效率的作用。

结束语

随着毕业设计的告一段落，我的本科生活也要结束了，在此感谢母校信息科大对我专业能力以及综合素质的培养。回首这四年大学时光，首先要感谢大学期间的每一位班主任以及任课老师。在日常的学习和生活中，老师们悉心耕耘，教会了我各科专业知识，并且老师们严谨治学和认真工作的态度也在潜移默化的影响着我，使我在大学学习过程中受益良多。本次毕业设计是在副教授崔展齐老师的指导下完成的，从毕业设计工作启动到论文完成的各阶段，崔老师都耐心地指导我完成相应工作。指导过程中，崔老师针对我的计划方案具体问题具体分析，给出相应的意见和建议，引导我面对问题不断分析和探索，最终完成本文。十分感谢崔老师的培养，使我在本科学习的最后阶段还能提升一定的科研能力，遇到崔老师是我的幸运。另外，还要感谢在本科期间所有为我生活和学习提供过帮助的同学们，以及一直在我身边支持我的家人。我会一直记得和同学们一起自习，一起进步的日子。家人为我提供了情感和物质上的双重保障，是我在遇到问题时继续前进和坚持的动力。最后，向所有参加论文评阅和答辩评审的老师们致以最诚挚的谢意。

参考文献

[1] Hailpern B, Santhanam P. Software debugging, testing, and verification[J]. IBM Systems Journal, 2002, 41(1): 4-12.

[2] 邱宝鑫. 基于条件分类执行切片谱的多缺陷定位方法研究[D]. 湘潭: 湘潭大学, 2019.

[3] 蔡蕊，张仕，余晓菲，等. 基于程序频谱的缺陷定位方法[J]. 计算机系统应用, 2019, 28(01): 188-193.

[4] Jones JA, Harrold MJ. Empirical evaluation of the Tarantula automatic fault-localization technique[C]. Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA: Association for Computing Machinery, 2005: 273–282.

[5] Baudry B, Fleurey F, Le TY. Improving test suites for efficient fault localization[C]. Proceedings of the 28th International Conference on Software Engineering, Shanghai, China: Association for Computing Machinery, 2006: 82–91.

[6] Wen W, Li B, Sun X, et al. Program slicing spectrum-based software fault localization[C]. Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering, Miami, FL, UAS: KSI Research Inc, 2011: 213–218.

[7] 陈娣. 基于动态切片和关联规则的软件故障诊断系统的设计与实现[D]. 武汉: 华中科技大学, 2016.

[8] Lourenço F, Lobo V, Bacao F. Binary-based similarity measures for categorical data and their application in Self-Organizing Maps[J]. Proceedings of the JOCLAD, 2004: 1-18.

[9] 丁晖，陈林，钱巨，等. 一种基于信息量的缺陷定位方法[J]. 软件学报, 2013, 24(07): 1484-1494.

[10] Gonzalez A. Automatic error detection techniques based on dynamic invariants[D]. Delft City: Delft University of Technology, 2007.

[11] Wong WE, Debroy V, Gao RZ, et al. The DStar method for effective software fault localization[J]. IEEE Transactions on Reliability, 2014, 63(1): 290-308.

[12] Wong WE, Vidroha D, Yihao L, et al. Software fault localization using DStar (D*)[A]. 2012 IEEE Sixth International Conference on Software Security and Reliability[C]. Gaithersburg, MD: IEEE, 2012: 21-30.

[13] 曹鹤玲，姜淑娟，鞠小林. 软件错误定位研究综述[J]. 计算机科学, 2014, 41(2): 1-6, 14.

[14] Lian L, Kusumoto S, Kikuno T, et al. A new fault localizing method for the program debugging process[J]. Information and Software Technology, 1997, 39(4): 271-284.

[15] 陆凯. 基于图挖掘的错误定位方法研究[D]. 徐州：中国矿业大学, 2019.

[16] Xiaoyuan X, Eric WW, Tsong YC, et al. Metamorphic slice: An application in spectrum-based fault localization[J]. Information and Software Technology, 2013, 55(5): 866-879.

[17] 许高阳，李必信，孙小兵，等. 一种基于层次切片的软件错误定位方法[J]. 东南大学学报（自然科学版）, 2010, 40(4): 692-698.

[18] 董俊华. 基于频谱的程序切片缺陷定位研究[D]. 西安: 西安理工大学, 2018.

[19] 陈理国，刘超. 基于高斯过程的缺陷定位方法*[J]. 软件学报, 2014, 25(6): 1169-1179.

[20] 何海江. 基于线性分类算法的软件错误定位模型[J]. 计算机工程与应用, 2017, 53(21): 42-48.

[21] 解铮，黎铭. 基于代价敏感间隔分布优化的软件缺陷定位[J]. 软件学报, 2017, 28(11): 3072-3079.

[22] 陈媛. 基于数据挖掘的软件缺陷预测技术研究[D]. 北京: 中国科学院大学;中国科学院研究生院, 2012.

[23] 张灿，赵逢禹. 基于特征相似的软件缺陷排除方法[J]. 计算机应用与软件, 2017, 34(11): 13-19.

[24] 张桂，曹健. 面向大数据分析的决策树算法[J]. 计算机科学, 2016, 43(S1): 374-379.

[25] 吴方君. 静态软件缺陷预测研究进展[J]. 计算机科学与探索, 2019, 13(10): 1621-1637.

[26] 翟云，杨炳儒，曲武. 不平衡类数据挖掘研究综述[J]. 计算机科学, 2010, 37(10): 27-32.