计算机科学
COMPUTER SCIENCE

基于推荐列表的缺陷文件识别

王昭丹, 邹卫琴, 刘文杰

引用本文

王昭丹, 邹卫琴, 刘文杰. 基于推荐列表的缺陷文件识别[J]. 计算机科学, 2024, 51(6A): 230600088-8.

WANG Zhaodan, ZOU Weiqin, LIU Wenjie. Buggy File Identification Based on Recommendation Lists [J]. Computer Science, 2024, 51(6A): 230600088-8.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230600088

基于推荐列表的缺陷文件识别

王昭丹　邹卫琴　刘文杰
南京航空航天大学计算机科学与技术学院　南京 211106
(wangzhaodan@nuaa.edu.cn)

摘　要　缺陷定位是缺陷修复的关键步骤,同时也是一项繁琐的软件活动。现有的静态缺陷定位技术通常将缺陷定位视为一个检索任务,即为每个缺陷报告生成一份按照程序实体与缺陷相关度降序排列的可疑文件推荐列表。然而,开发人员仍需人工一一审查从而找到真正有缺陷的文件,这增加了定位的时间和成本。为解决这个问题,提出了一个相应的解决方案。首先运行主流的基于信息检索的静态缺陷定位技术来获得一个初始的可疑文件推荐列表;然后依据问题特性提出3类领域特征,并基于这3类特征构建一个机器学习模型,尝试从列表中识别出真正有缺陷(Truly Buggy)的源代码文件。在4个开源项目(ZooKeeper,OpenJPA,Tomcat,AspectJ)的2558个bug上进行了实验,结果表明,在最初可疑文件推荐列表上可以获得72.6%～80.7%的真正有缺陷的文件预测准确率。同时探究了3类特征子集及各个特征在预测真正有缺陷的文件上的重要性,发现缺陷报告与源代码的关系特征更重要。

关键词:缺陷报告;缺陷定位;机器学习;信息检索;缺陷文件
中图分类号　TP311

Buggy File Identification Based on Recommendation Lists

WANG Zhaodan,ZOU Weiqin and LIU Wenjie
College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China

Abstract　Bug localization is a key step for bug fixing but also a tedious software activity. Existing static defect location techniques typically treat defect location as a search task,generating a list of recommended documents for each defect report in descending order of program entity relevance to the defect. However,developers still need to manually review each file to find the ones that are actually defective,which increases the time and cost of locating them. To solve this problem,this paper proposes a solution. Firstly,running state-of-the-art information-retrieval-based(IR-based) bug localization techniques to obtain an initial buggy files recommendation list. Then,three domain characteristics are proposed according to the characteristics of the problem,and a machine learning model is built based on these three characteristics,trying to identify the truly buggy files from the list. Preliminary experiments verify that the proposed approach is reasonable and actionable in practice. Experiments are carried out on four open source projects with 2558 bugs(ZooKeeper,OpenJPA,Tomcat,AspectJ) and the results show that it could obtain 72.6%～80.7% prediction accuracy initially recommending the buggy code files in the list. At the same time,we explore the three feature subsets and the importance of each feature in predicting the truly buggy files,and find that the feature of the relationship between the bug report and the source code is more important.

Keywords　Bug Report,Bug localization,Machine learning,Information retrieval,Buggy files

1　引言

缺陷定位是缺陷修复的重要组成部分[1]。为了促进缺陷修复并进一步确保产品质量,研究社区[2-4]提出了一系列缺陷定位技术,根据是否运行测试用例分成两类:静态缺陷定位和动态缺陷定位[5]。静态缺陷定位技术通常将缺陷定位视为一个检索任务,它为每个缺陷报告生成一份按照程序实体与缺陷相关度降序排列的推荐列表,开发者可以通过该列表顺序来一一审查从而找到真正有缺陷的代码文件。虽然静态缺陷定位的定位粒度较粗糙,通常只能从文件或是方法级别来定位缺陷位置,但其使用相对简单,不需要运行代码,成本也会较为低廉。本文主要研究在代码文件粒度上的静态缺陷定位技术。

近年来,静态缺陷定位技术领域的研究工作取得了较好的成果,但在实践过程中应用此类方法仍面临一些严峻的挑战。在实践中,开发人员在面对使用现有的缺陷定位技术生成的源代码文件推荐列表时,即使该文件列表是按照相关度排序,仍需要人工一一审查每个文件是否为真正有缺陷的文件。而当真正有缺陷的代码文件排名靠后甚至未出现在该推荐列表中时,也就意味着开发人员需要花费更多的时间和精力来查询,大大增加了定位真实缺陷文件的代价和成本。

基金项目:国家自然科学基金(62002161);南京航空航天大学前瞻布局科研专项资金;南京航空航天大学人才科研启动基金;
This work was supported by the National Natural Science Foundation of China(62002161),Fund of Prospective Layout of Scientific Research for NUAA(Nanjing University of Aeronautics and Astronautics) and Scientific Research Foundation for the Introduction of Talent for NUAA.
通信作者:邹卫琴(weiqin@nuaa.edu.cn)

例如，基于 6 个流行软件项目（即 Tomcat，AspectJ，Lucene，ZooKeeper，OpenJPA 和 Hibernate-ORM）中的 4 587 个 bug，通过分析它们基于主流之一的基于 IR 的缺陷定位技术（即 BugLocator[2]）对此的定位结果发现，这些真正有缺陷的代码文件（在推荐列表中第一个出现的缺陷文件）在可疑文件推荐列表中的平均检索排名在 7－94 之间。

为了解决这一问题，本文提出了一个解决方案，其核心思想是尝试识别推荐列表中真正有缺陷的代码文件。首先运行现有的主流的基于信息检索的缺陷定位技术来生成一个初始的可疑文件推荐列表，然后，依据问题特性提出 3 类领域特征（包括与缺陷报告相关的特征、与源代码文件相关的特征，以及两者间的关系特征，例如，文本相似性），并基于这 3 类特征构建一个机器学习模型，尝试从列表中识别出真正有缺陷的源代码文件。本文在 4 个有 2 558 个 bug 的开源项目（ZooKeeper，OpenJPA，Tomcat，AspectJ）上进行了实验，结果表明，在最初可疑文件推荐列表上，本文方法可以获得 72.6%～80.7%的缺陷代码文件预测准确率。本文的主要贡献是：1）强调了现有的缺陷定位技术无法很好地预测真正有缺陷的代码文件的问题；2）提出了一个解决方案来解决这个问题，依据问题特性，抽取了 3 类领域特征，并在 4 个项目集上获得 72.6%～80.7%的预测准确率；3）分析了 3 类特征子集及各个特征的重要性，为特征的抽取提供了可解释性。

2 技术框架

本文所提方法的框架图如图 1 所示，包括 3 个部分。第一部分是使用现有的基于信息检索的缺陷定位技术生成一个初始的可疑文件推荐列表；第二部分是对三种实例特征进行抽取；第三部分则建立一个机器学习模型，从初始列表中预测真正有缺陷的文件。

2.1 初始列表生成

如第 1 章所述，本文方法是从使用现有的基于信息检索的缺陷定位技术开始，为一个给定的缺陷得到一个初始的可疑文件推荐列表。具体地，使用 3 种主流的基于 IR 的缺陷定位技术（包括 BugLocator[2]，Blizzard[3] 和 LR[4]）来生成初始列表。

1）BugLocator：使用修订的矢量空间模型（rVSM），根据缺陷报告和源代码之间的文本相似性（余弦相似性）对所有文件进行排序，同时还考虑到以前已经修复的类似缺陷的信息。BugLocator 可以被视为基于经典信息检索的缺陷定位方法的代表。

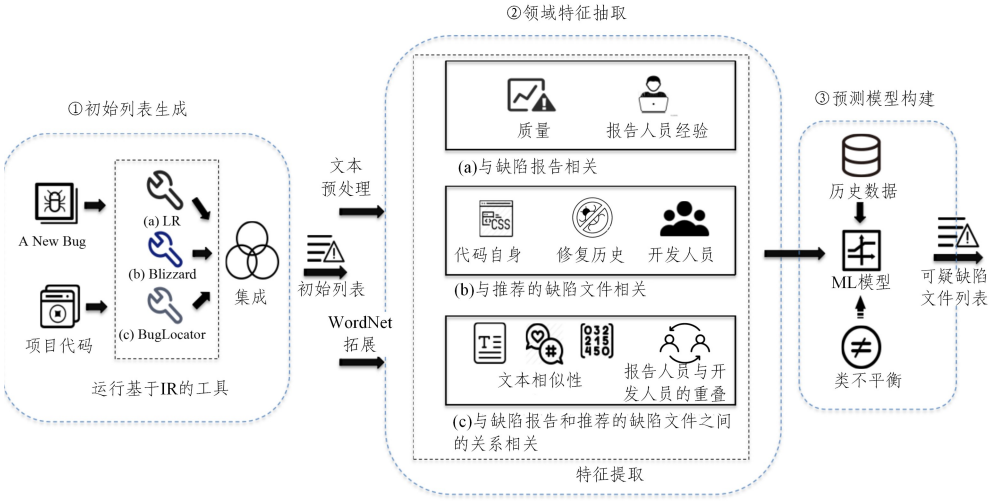


图 1 本文提出方法的框架图

Fig. 1 Framework diagram of the proposed method

2）Blizzard：如果一个缺陷报告缺乏必要的信息或有太多的信息，例如带来噪音的结构化信息，基于信息检索的缺陷定位技术将表现不佳。Blizzard 提议首先重新表述一个缺陷报告，然后使用质量更高的缺陷报告来进行缺陷定位。它改进了经典的基于信息检索的缺陷定位，重点是提高缺陷报告的质量。

3）LR：提出利用领域知识（除了缺陷报告和代码文件之间的表面文本相似性）来提高定位性能，通过考虑 API 描述、代码文件的缺陷修复历史、缺陷报告中出现的程序实体以及历史上类似的缺陷报告等，它进一步引入了一种自适应的排名方法，对这些特征进行加权以达到更好的定位效果。

这 3 种技术是基于信息检索的定位技术的不同种类的代表。为了充分利用它们的优点，本文决定分别运行它们，并使用典型的集成方法，如 AdaBoost[6]，将它们的预测结果合并为初始的可疑文件推荐列表。

本文实验只测试了 BugLocator，包括 BugLocator 在内的 3 种基于信息检索的技术的组合将在下一步的研究工作中进一步探索。本文采用文献[2]推荐的 BugLocator 的默认设置，在 4 个有 2 558 个 bug 的开源项目（即 ZooKeeper，OpenJPA，Tomcat 和 AspectJ）上，针对每个缺陷报告运行 BugLocator 来获得缺陷的初始可疑文件推荐列表。

2.2 领域特征抽取

在第一步中，我们只能确保初始的可疑文件推荐列表中很可能包含一个真正有缺陷的文件，但不知道它的文件名的具体信息。这部分是要精确地从最初的列表中找出真正（或很可能真正）有问题的是哪一个。我们将建立一个机器学习模型来预测初始列表中的文件是否真的有缺陷。为了构建模型，提取了表 1 所列的 3 种实例特征，即与缺陷报告相关的特征、与推荐的可疑文件相关的特征以及它们之间的关系特征（如文本相似性）。

与缺陷报告相关的特征包括两类：报告质量特征和报告人员经验特征。缺陷报告质量是一个基本因素，对于一个

新的缺陷报告，它在很大程度上影响了一个缺陷定位工具能否成功地检索一个真正有缺陷的代码文件[2]。

表 1　用于识别真正有 bug 的文件的实例特征

Table 1　Instance features for identifying truly buggy file

(s)

维度	特征名	描述
与 bug 报告相关	hasItemizations	Bug 报告中是否包含明细
	Keyword Completeness	通过检查关键字是否出现在 bug 报告中来检查 bug 报告的完整性
	hasCodeSamples	Bug 报告中是否包含代码示例
	hasStackTraces	Bug 报告中是否包含堆栈跟踪
	hasPatches	Bug 报告中是否包含补丁
	hasScreenshots	Bug 报告中是否包含截图
	Report Readability	根据文献[7]，使用 Kincaid，Automated Readability Index(ARI)，Coleman-Liau，Flesh，Fog，Lix 和 SMOG Grade 来衡量可读性
	Reporter Expertise	由一个报告人员提交/修复的 bug 数量，以及其修复提交的 bug 比例
与源代码文件相关	Bug-Fixing Recency	代码文件的当前 bug 报告与最近一次修复的 bug 报告之间的(被报告的时间)时间距离
	Bug-Fixing Frequency	在当前 bug 报告之前，源代码文件被修复的次数
	NDEV	在所有以前的代码版本中更改过文件的开发人员的累计数量
	Code Complexity	由 CK 指标[8]、代码行数和语句数度量
	Comments-Codes Consistency	代码注释和代码之间的一致性，用 LDA 主题相似度来衡量
	Code Readability	代码可读性衡量工具与 bug 报告可读性相同
	DAF(Developer Attention Focus)	度量开发人员的活动有多集中[9]
	MAF(Module Attention Focus)	度量活动对模块(本文中指代码文件)的关注程度[9]
	Developer's Structural Scattering	根据从一个代码文件到另一个代码文件所需要跨越的子系统数量，度量开发人员修改的代码文件在结构上的距离[10]
	Developer's Semantic Scattering	根据开发人员修改的代码文件的文本相似性来衡量代码文件的实现职责的分散程度[11]
	Bug fix dependencies	与当前源代码文件有调用/被调用关系的过去 Buggy 文件的数量
与 Bug 报告和源代码文件之间的关系相关	Surface Lexical Similarity	在 bug 报告和源代码文件之间用余弦相似度(使用 VSM)度量
	API-Enriched Lexical Similarity	在 bug 报告和由 API 文档扩展的代码文件之间用余弦相似度(使用 VSM)度量
	Collaborative Filtering Score	给定的 bug 报告与在报告 bug 之前修复了代码文件的 bug 报告之间的余弦相似度(使用 VSM)
	Class Name Similarity	Bug 报告摘要中包含的源代码类名的长度
	Topic Similarity	Bug 报告和源代码文件之间的主题相似性(使用 LDA)
	Semantic Similarity	Bug 报告和源代码文件的单词嵌入向量的余弦相似度
	Developers and Reporters	在 Bug 报告中出现和为源代码文件做出贡献的重叠人数

受文献[7]的启发，本文考虑了与缺陷报告质量相关的 7 个特征，包括是否存在明细、代码示例、堆栈跟踪、补丁和屏幕截图，以及缺陷报告的关键字完整性和文本可读性。

本文还考虑了报告人员的经验特征。我们相信提交或修复缺陷越多的报告人员，其经验也越丰富，则其提交的缺陷报告就会越专业；如果报告人员提交的缺陷被修复的百分比更高，那么他/她提交的缺陷报告将更有可能是有效的和高质量的。因此，对于报告人员的专业知识，本文主要计算报告人员提交的缺陷数量、报告人员修复的缺陷数量，以及报告人员提交的所有缺陷中被修复的缺陷数量。

与推荐的可疑缺陷文件相关的特征包括代码复杂性、缺陷修复历史、代码可读性、注释代码一致性以及代码文件涉及的开发人员特征。越复杂的代码文件越容易产生缺陷，本文主要使用 CK 指标[12]、代码行(LoC)和语句数量来度量代码文件的复杂性。同时，代码的大小也可能与缺陷的发生有关(如代码段越长，就越容易出现错误)。本文使用代码行数和语句总数来度量代码大小。在缺陷修复历史方面，对于每个代码文件，主要考虑了 4 个特征：缺陷修复频率、缺陷修复临近性、缺陷修复依赖关系以及接触该文件的开发人员数量。根据文献[4]，经常修复的源代码文件可能是容易出现缺陷的文件(缺陷修复频率)，最近修复的源代码文件越可能与新出现的缺陷有关(缺陷修复临近性)。缺陷修复依赖关系考虑了当前源代码文件和过去的有缺陷的文件之间的依赖性。我们相信，如果当前源文件和过去有缺陷的文件之间存在依赖关系，那么当前源文件很可能存在缺陷。本文使用 Java-callgraph 套件(它可以从 Java 系统中生成静态和动态调用图[1])来获得当前源代码文件的完整调用链，并计算调用链中存在的过去的有缺陷的文件的数量。接触源代码文件的开发人员的数量也与代码文件是否有缺陷倾向有关[8]。如果参与源代码文件的开发人员的累积数量越多，则该源代码文件产生问题的可能性就越大。

代码可读性[13]和代码注释一致性[11]在某种程度上也揭示了代码文件的质量，并进一步与缺陷的发生有关。可读性相对较低的代码文件(在本文中通过使用 Style 工具[7]计算)，或者其代码与注释不一致(在本文中通过 LDA 主题相似性度量)可能会阻碍开发人员很好地理解代码，因此更容易引入缺陷。

受文献[9]和文献[10]的启发，本文相信源代码文件的开发人员也是引入缺陷的一个重要因素。与专注于一个模块的开发人员相比，致力于多个模块的开发人员更容易在开发和测试过程中出错。在本研究中，主要计算 4 种开发人员特征，包括开发人员注意焦点(DAF)、模块注意焦点(MAF)[9]、开发人员结构散射(Structural Scattering)和语义散射(Semantic Scattering)[10]。

与缺陷报告和推荐的可疑缺陷文件之间的关系相关的特征主要关注缺陷报告和代码文件之间的文本相似性和开发人

1) https://github.com/gousiosg/java-callgraph

员与报告人员之间的重叠。考虑到与缺陷报告文本相似性更大的代码文件更有可能与缺陷相关联[2]，对于每个缺陷报告，检索其与缺陷的初始可疑文件推荐列表中的这些源代码文件之间的文本相似性。计算3种类型的相似度，分别是原始文本相似度(使用LR中提到的4种相似度，即表面词汇相似度、API丰富词汇相似度、协同过滤得分和类名相似度[4])、主题相似度(使用潜狄利克雷分配模型LDA在主题级别捕获两个文档更高的抽象相关性)和单词嵌入相似度(使用单词嵌入技术，如Word2Vec，将缺陷报告和源代码文件转换为单词嵌入向量并计算它们的余弦相似度)捕获上下文语义。这3种类型的文本相似度都是基于缺陷报告和源代码文件中出现的词汇术语计算的。考虑到开发人员(或用户)可能使用不同的术语来表达相同的意图，本文还引入了WordNet[14]同义词网络，以扩展缺陷报告和源代码文件中的词汇术语，避免潜在的威胁。

本文还认为，缺陷报告和源代码中涉及的开发人员/报告人员的重叠也会影响结果。例如，如果缺陷报告的报告人员与列表中推荐的错误源文件的开发人员相同，那么该文件更有可能与缺陷报告相关。在重叠计算过程中，从缺陷报告的CCList和Comments中提取报告人员和开发人员，以及那些对代码文件做出贡献的人(通过挖掘Git提交历史)，以检查其中有多少人重叠。

2.3 预测模型构建

在前面的步骤中，对于每个缺陷报告，本文使用BugLocator，LR和Blizzard集成的推荐列表作为初始可疑文件推荐列表。对于所有对，cf表示给定bug b的初始可疑文件列表中的每个代码文件，从中提取如表1所列的特征。

将这些特征作为实例特征，并确定每个实例的标签如下：对于每对〈bug b，code file cf〉，如果代码文件cf确实与该bug有关，则标签为1，否则标签为0。然后，为了构建模型，提取了3个实例特征，分别是关于缺陷报告的特征、关于源代码文件的特征以及关于两者之间关系的特征。

将所有实例划分为一个训练数据集和一个测试数据集，在训练数据集上训练预测模型，并用测试数据集来验证模型的有效性。通过建立模型进行5次交叉验证，主要测试了3种机器学习算法，即随机森林(RF)、支持向量机(SVM)和决策树(DT)。在模型构建过程中，还使用过采样来处理类的不平衡问题(一个类比另一个类拥有更多的实例)。

3 实验分析

为了验证本文方法的合理性，进行了相关实验。首先构造一个包含4个开源项目的数据集，然后，建立一个模型来预测这4个项目中真正有缺陷的文件。

3.1 数据集构造

本文选择了6个开源项目进行实验，分别是Tomcat，AspectJ，Lucene，ZooKeeper，OpenJPA，Hibernate-ORM。这6个项目来自不同的领域，代码大小也不同。对于每个项目，需要将它们的bug与相应的bug代码文件链接起来，这样就有了评估的基本事实。在这6个项目中，Tomcat和AspectJ来自文献[4]共享的数据集。对于ZooKeeper和OpenJPA，抓取了在2018年11月之前它们的代码和bug报告，并通过手动分析提交日志获得的启发式规则将bug与buggy的代码文件链接起来。分析发现，这两个项目的开发人员通常在提交日志中使用ProjectName-BugID将bug修复提交链接到bug报告(例如，ZooKeeper-4123表示针对bugID 4123的bug修复提交)。在修正错误的提交中修改的代码文件被认为是有错误的代码文件。此外，还过滤了其链接提交导致多个错误或链接到多个提交的错误，只添加了代码文件的提交也会在数据集构造中被删除。在使用启发式规则链接bug和buggy的代码文件之后，我们还手动检查链接的结果并删除发现的任何噪声。表2列出了6个实验项目的基本统计信息。

表2 实验项目的基本数据

Table 2 Basic data of experimental project

Project	Time Range	# Linked Bugs	Loc
ZooKeeper	2008-06-09—2018-11-12	470	140175
OpenJPA	2006-08-11—2018-11-16	533	723284
Tomcat	2002-07-06—2014-01-18	992	573208
AspectJ	2002-03-13—2014-01-10	563	690560
Hibernate	2004-08-22—2018-11-16	1285	1068498
Lucene	2007-07-12—2018-11-14	1454	1762563

3.2 实验结果

为了研究所提方法能否有效地预测真正有缺陷的代码文件子集，调查了3个研究问题。

3.2.1 RQ1：本文方法能否有效识别真正有缺陷的代码文件？

对于每个缺陷报告，运行BugLocator来获得缺陷的初始可疑文件推荐列表。BugLocator是基于信息检索的缺陷定位技术的一个强有力的基线，并且在工业应用中是有效的。在本文实验中，只测试了BugLocator，包括BugLocator在内的3种基于信息检索的技术的组合将在下一步的研究工作中进一步探索。本文采用文献[2]推荐的BugLocator的默认设置。在4个有2558个bug的开源项目(即ZooKeeper，OpenJPA，Tomcat和AspectJ)上的实验表明，从最初的可疑文件推荐列表中预测真正有缺陷的代码文件，可以获得72.6%～80.7%的准确性。

表3列出了提取真正有缺陷的文件的结果，从表中可以发现，本文的模型可以实现一个有希望的预测精度和真正有缺陷的文件提取。4个实验项目的所有模型都获得了约为75%～80%的精度。表4列出了每个预测模型取得最佳性能的参数设置。

表3 识别真正有bug的文件集的预测精度

Table 3 Prediction accuracy of identifying truly buggy file sets

(%)

Classifier	Projects			
	Zookeeper	OpenJPA	Tomcat	AspectJ
RF	78.1	80.0	80.7	76.8
SVM	75.6	78.7	76.7	74.3
DT	72.6	76.0	73.1	73.1

表4 各预测模型取得最佳性能的参数设置

Table 4 Parameter settings for each prediction model to achieve the best performance

Classifier	Parameter
RF	# Trees=100
SVM	Classification Technique=SVMRadial Sigma=0.9 Cost=0.5
Decision Tree	Classification Technique=LMT # Iterations=21

3.2.2 RQ2:与基于所有特征构建的预测模型相比,基于单个维度特征构建的预测模型效果如何?

本研究从与 bug 报告相关、与源代码文件相关以及 bug 报告和源代码文件间的关系 3 个维度提取了 26 种特征。在这个研究问题中,本文研究了预测模型是否通过使用所有特征(与其子集相比)受益。

当随机森林在 RQ1 中达到最佳性能时,本文构建了随机森林模型(参数 # Tree 设置为 100) 基于维度特征的子集。对于每个维度,本文有两种实验设置:一种是只根据其中的特征构建随机森林模型,另一种是根据其他 3 个维度的特征构建随机森林模型。本文使用与 RQ1 相同的设置来评估预测模型的性能,报告的 AUC 是 5 次交叉验证运行的平均值。然后,将基于所有特征构建的随机森林模型的 AUC 与基于各类特征子集构建的模型的 AUC 进行比较。

表 5 列出了识别真正有缺陷的代码文件从特征子集构建的随机森林模型的平均 AUC 结果。与基于某一维特征构建的预测模型相比,基于所有特征构建的随机森林模型实现了更高的 AUC 值。考虑到所有特征子集,AUC 中的增量都很大,与基于 3 个维度特征构建的预测模型相比,基于所有特征构建的随机森林模型取得了更好的性能。因此,结合 3 个维度的特征有利于提高随机森林模型的有效性。

表 5 不同维度特征子集构建的随机森林模型的 AUC 值

Table 5 AUCs of random forest models based on individual features subsets

(%)

Classifier	Projects			
	Zookeeper	OpenJPA	Tomcat	AspectJ
All	69.6	72.9	75.6	68.8
Bug Reports	54.0	63.1	61.7	60.4
Source Files	64.5	65.4	55.5	61.6
Bug Reports & Source Files	66.9	67.3	69.7	64.2

3.2.3 RQ3:哪些特征在预测真正有缺陷的文件中更具影响力?

尽管有多种因素可能会影响预测真正有缺陷的源代码文件集,但有些因素可能比其他因素更有影响力。在这个研究问题中,本文调查了以下这些因素。

随机森林在 RQ1 中已被证明在识别缺陷文件的精确度上表现最佳。因此,本文只研究考虑随机森林模型的最重要的特征。

第 1 步 相关性分析。利用变量聚类分析构建了特征之间相关性的分层概览,该分析在名为 Hmisc 的 R 包中实现。在分层概览中,相关特征被分组到子层次结构中。按照先前研究[15-16]中执行的程序排除相关特征——随机选择一个特征并从特征相关性大于 0.7 的子层次结构中删除其他特征。在这一步结束时,剩余 28 个特征。

第 2 步 冗余分析。通过相关分析降低特征之间的共线性后,识别出相比其他特征没有独特信号的冗余因素。解释模型中的冗余因素会相互干扰,进而扭曲因素和预测变量之间的模型关系。本文使用 rms R 包中的 redun 函数来进行冗余分析,但是在这一步没有发现冗余,因此不需要删除任何特征。

第 3 步 识别重要特征。在删除冗余特征后,还有 28 个剩余特征,然后使用 bigrf R 包构建随机森林模型。为了评估特征的重要性和模型的有效性,使用 10 折交叉验证的方法。在训练过程中,我们使用 bigrf 包中的 varimp 函数来计算一个因素的重要性,该因子基于 OOB 估计值。OOB 是随机森林分类器的内部误差估计,其基本思想是检查当特征逐个随机排列时,OOB 估计值是否会显著减小。为了避免实验中的随机性,重复此步骤 100 次,因此,我们构建并运行了 1 000 次预测模型。在每次验证运行中,都会为每个特征获得一个重要性值,应用 ScottKnott ESD 测试[17],使用来自所有 1 000 次运行的重要性值来确定对整个数据集最重要的特征。该测试将一组分布(每个特征变量对应一个分布)作为输入,并识别在统计上彼此显著不同的变量组。

第 4 步 重要特征的影响。为了了解这些特征的影响,应用 Wilcoxon 秩和检验[18]和 Bonferroni 校正[19]来分析缺陷源文件与非缺陷源文件差异的统计显著性,使用 Cliff's delta 来衡量两组之间差异的影响大小。

表 6—表 9 分别列出了真正缺陷文件预测在 4 个项目集的前 5 个最重要的特征。在这些表中,其重要性值在统计上显著不同于其他特征组的不同特征组(即,$p\text{-}value < 0.05$)。最后一栏 δ 显示 $p\text{-}value$ 和 Cliff's delta。如果 $p\text{-}value < 0.01$ 且 $|\delta| > 0.147$,即至少具有小效应量的统计显著差异,则最后一列的文本以粗体显示。

表 6 前 5 个最重要的特征(Zookeeper)

Table 6 Top 5 most important features(Zookeeper)

Feature	δ
Surface Lexical Similarity	**−0.393(medium)** **
Semantic Similarity	**−0.346(medium)** **
Topic Similarity	**−0.293(small)** **
Bug fix dependencies	−0.139(small) **
Class Name Similarity	−0.075(negligible) *

表 7 前 5 个最重要的特征(OpenJPA)

Table 7 Top 5 most important features(OpenJPA)

Feature	δ
Surface Lexical Similarity	**−0.297(medium)** **
Topic Similarity	**−0.205(medium)** **
Semantic Similarity	**−0.189(medium)** **
Collaborative Filtering Score	**−0.177(small)** **
Class Name Similarity	−0.083(negligible) *

表 8 前 5 个最重要的特征(Tomcat)

Table 8 Top 5 most important features(Tomcat)

Feature	δ
Surface Lexical Similarity	**−0.385(medium)** **
Semantic Similarity	**−0.357(medium)** **
Topic Similarity	**−0.295(small)** **
Bug fix dependencies	**−0.184(small)** **
Bug-Fixing Recency	-0.067(negligible) *

表 9 前 5 个最重要的特征(AspectJ)

Table 9 Top 5 most important features(AspectJ)

Feature	δ
Surface Lexical Similarity	**−0.373(medium)** **
Semantic Similarity	**−0.258(medium)** **
Topic Similarity	**−0.166(small)** **
Bug fix dependencies	−0.058(small) **
Class Name Similarity	−0.031(negligible) *

在这 4 个项目中,有 7 个特征属于前 5 个最重要的特征,即 Surface Lexical Similarity,Semantic Similarity,Topic Simi-

larity, Bug fix dependencies, Class Name Similarity, Collaborative Filtering Score, Bug-Fixing Recenc, 可以发现有 3 个特征(Surface Lexical Similarity, Semantic Similarity 和 Topic Similarity)在 4 个项目集中都有出现。

在这些特征中, Surface Lexical Similarity, 即词汇相似度, 是影响随机森林模型在所有 4 个项目中区分缺陷文件和非缺陷文件的最重要的特征。统计测试表明, 对于某个缺陷报告, 其对应的缺陷文件拥有更高的词汇相似度。缺陷报告与缺陷文件间的词汇相似度越高, 两者就越匹配, 该源代码文件也越可能是报告对应的缺陷文件。Semantic Similarity, 即语义相似度, 其对预测缺陷文件也有较大的影响。缺陷报告与缺陷文件间的语义相似度越高, 该源代码文件就越可能是报告对应的缺陷文件。语义相似性在一定程度上弥补了词汇相似性存在的语义鸿沟问题。

4 相关工作

近年来, 静态缺陷定位研究工作的一类主流方法是 IRBL(基于信息检索的缺陷定位)。静态缺陷定位技术的研究阶段有以下 4 个侧重点:

1)更换检索模型。早在 2010 年以前, 信息检索技术刚刚被应用到缺陷定位领域, 研究者的研究重点是尝试使用不同的信息检索(Information Retrieval, IR)模型对缺陷定位任务进行建模。在 IRBL 领域现有的方法中, 向量空间模型[20](Vector Space Model, VSM)及其变体是使用最广泛的 IR 模型。2009 年, Gay 等[21]提出一个使用相关反馈(Relevance Feedback)机制的方法来提升概念定位的结果, 在他们的方法中最先明确使用 VSM 模型来进行检索。2012 年, Zhou 等[2]指出传统 VSM 模型未考虑文档越长则越有可能包含缺陷的因素, 倾向于将长度较短的文档排在前列, 因此他们对 VSM 模型进行改进, 提出了 rVSM 模型, 其中将文档长度作为一个权重参数。后续有多项研究工作(Wong 等[22]、Rahman 等[23]、Youm 等[24])均是在 rVSM 模型之上构建新的 IRBL 方法。潜在语义索引模型[25](Latent Semantic Index, LSI)是一种实用的主题模型, 它是在传统 VSM 模型的基础上进行改良后得到的。在 IRBL 领域, LSI 是最早被用来进行特征缺陷定位的 IR 模型, 并且只活跃在研究早期(2010 年以前), 近年来几乎没有方法使用 LSI 作为 IR 模型来进行缺陷定位。隐狄利克雷分布模型[26](Latent Dirichlet Allocation, LDA)是一个生成式的统计主题模型, LDA 模型也是在早期研究中就被应用到 IRBL 领域中, 并且在近年来仍有部分研究关注于改良 LDA 模型。

2)扩充领域特征。大约从 2011 年开始, 研究者们发现单纯使用 IR 模型定位缺陷的性能低下, 很难满足实际的应用需求, 并且发现除了文本相似度之外, 软件中的其他特征也可以帮助定位缺陷, 研究人员开始挖掘与缺陷有关的特有特征(如版本历史、相似报告、堆栈信息等), 并结合特征与 IR 模型来提升缺陷定位的性能。2012 年, Zhou 等[2]在他们的方法 BugLocator 中使用了相似缺陷报告特征, 并且该方法在后续多个研究工作中得到了应用。2014 年, Moreno 等[27]考虑将堆栈踪迹信息应用于他们的方法 Lobster 中, 并且使用缺陷报告中的堆栈踪迹和软件源码中的程序依赖图来寻找在结构上相似的代码元素。2016 年, Wang 等[28]在他们的改良方法 AmaLgam+中使用了堆栈踪迹特征, 设计一个堆栈分数来度量堆栈中出现的每个文件的可疑程度。2018 年, Rahman 等[3]利用堆栈踪迹来重新查询, 从堆栈踪迹中提取代码实体(类名和方法名), 根据执行顺序构造出权重图, 并在图上面应用 PageRank 算法计算出每个代码实体的权重来重新构造查询语句。

3)重构缺陷报告质量。在一定程度上来说, 由于大量研究已经对代码库和缺陷报告中的相关特征进行了比较充分的挖掘, 很难再提取新的特征来改良 IRBL 方法。研究者从改善查询的角度提出了查询重构的策略来提升性能。2013 年, Sisman 等[29]首先将查询重构策略应用到缺陷定位领域中, 他们先进行一次初始查询得到相关的文件排名列表, 然后从排名靠前的文件中抽取附加的词汇来扩充查询语句。2018 年, Rahman 等[3,30]提出使用一种上下文感知的查询重构方法 BLZZARD, 该方法对缺陷报告中的两类重要信息堆栈踪迹和程序元素分别构造踪迹图和文本图模型, 对这两部分中的词汇进行建模。在构造好的图模型基础上, 使用 PageRank 算法对图中的词汇进行加权, 并根据权重来选取加入查询语句的词汇。2019 年, Kim 等[31]提出一个包含 3 个组件的自动查询重构方法: 缺陷报告扩展、候选词提取和查询扩展。首先, 使用缺陷报告中的附件信息扩展缺陷报告; 然后, 在提取候选词时通过将情感词汇加入到停用词表来将它们移除, 并且使用伪相关反馈的方式从源码中检索文件并移除噪音文件; 最后, 对检索到的文件中的词汇加权并选择权重较高的部分来扩展查询语句。

4)应用深度学习。随着深度神经网络的崛起, 从 2017 年开始, 越来越多的研究人员开始尝试使用深度学习模型(如 CNN)来自动提取特征并结合 IR 技术对缺陷进行定位。与基于信息检索的缺陷定位只考虑了缺陷报告和源代码文本上的特征相比, 基于深度学习的缺陷定位方法可以提取缺陷报告和源代码文件更深层和更高维的抽象特征。这类方法可以减少人为设计特征的工作量, 同时提高了缺陷定位的准确率。2017 年, Lam 等[32]将深度神经网络(DNN)与 IR 技术中的 rVSM 相结合进行缺陷定位。实验证明, 深度神经网络和信息检索技术可以很好地相互补充, 能够实现比单个模型更高的缺陷定位精度。2019 年, Xiao 等[33]提出使用词嵌入和强化的卷积神经网络来提升缺陷定位性能。他们使用词嵌入来表示缺陷报告和源文件中保留语义信息的单词, 并使用不同的 CNN 模型来提取它们的特征。2020 年, Cao 等[34]提出了一种基于修正图的深度学习来定位错误方法的新方法 BugPecker, 通过分析代码、提交过去的缺陷报告来构建修订图, 以此提取语义匹配分数、修订的协同过滤分数、缺陷修复临近性和缺陷修复频率特征, 并输入 DNN 模型获得可疑的错误方法列表。2021 年, Huo 等[35]提出了一种跨项目缺陷定位的深度迁移学习方法 TRANP-CNN, 从源项目中提取可转移的语义特征, 并充分利用目标项目中的标记数据进行有效的跨项目错误定位。2022 年, Meng 等[36]提出了 TRANSFER 的新方法, 该方法利用深层语义特征和从开源数据中转移的知识来改进故障定位和程序修复。2022 年, Liang 等[37]提出了一种称为 FLIM 的新框架, 该框架可以在功能级别提取程序的语义特征, 然后通过聚合功能级别的交互来计算自然语言和编程语言之间的相关性。利用微调的语言模型将错误定

位任务视为代码检索任务，并使用学习排序模型将功能级语义特征与 IR 特征融合，以计算最终相关性。2023 年，Yousofvand 等[38]建议将缺陷定位问题视为节点分类问题。首先使用 Gumtree 算法，通过将错误图与其对应的修复图进行比较来标记节点，然后使用一种图神经网络 GraphSAGE 来进行分类。

结束语 针对现有的缺陷定位技术无法很好地定位 Truly Buggy 文件的问题，本文提出了一种解决方案，其核心思想是尝试从目前主流的基于 IR 的缺陷定位技术的推荐列表中精确地识别出真正有缺陷的文件。本文进行了一些实验，初步验证了该方法的有效性。但目前仍存在 Bug 报告和源代码间存在文本语义差距的实际问题，下一步的工作是全面实施和完善本文方法，使其在实践中得到应用。

参 考 文 献

[1] ZOU W，LO D，CHEN Z，et al. How practitioners perceive automated bug report management techniques[J]. IEEE Transactions on Software Engineering，2018，46(8)：836-862.

[2] ZHOU J，ZHANG H，LO D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports[C]// International Conference on Software Engineering. IEEE，2012：14-24.

[3] RAHMAN M M，ROY C K. Improving ir-based bug localization with context-aware query reformulation[C]// Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018：621-632.

[4] YE X，BUNESCU R，LIU C. Learning to rank relevant files for bug reports using domain knowledge[C]// Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014：689-699.

[5] XUAN J，MONPERRUS M. Learning to combine multiple ranking metrics for fault localization[C]// International Conference on Software Maintenance and Evolution. IEEE，2014：191-200.

[6] ZHOU Z H. Ensemble methods：foundations and algorithms [M]. CRC Press，2012.

[7] ZIMMERMANN T，PREMRAJ R，BETTENBURG N，et al. What makes a good bug report? [J]. IEEE Transactions on Software Engineering，2010，36(5)：618-643.

[8] OSTRAND T J，WEYUKER E J，BELL R M. Programmer-based fault prediction[C]// Proceedings of the 6th International Conference on Predictive Models in Software Engineering. 2010：1-10.

[9] POSNETT D，D'SOUZA R，DEVANBU P，et al. Dual ecological measures of focus in software development[C]// International Conference on Software Engineering. IEEE，2013：452-461.

[10] DI NUCCI D，PALOMBA F，DE ROSA G，et al. A developer centered bug prediction model[J]. IEEE Transactions on Software Engineering，2017，44(1)：5-24.

[11] JARMAN D，BERRY J，SMITH R，et al. Legion：Massively composing rankers for improved bug localization at adobe[J]. IEEE Transactions on Software Engineering，2021，48(8)：3010-3024.

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

[13] BUSE R P L，WEIMER W R. Learning a metric for code readability[J]. IEEE Transactions on Software Engineering，2009，36(4)：546-558.

[14] MILLER G A. WordNet：a lexical database for English [J]. Communications of the ACM，1995，38(11)：39-41.

[15] BAO L，XING Z，XIA X，et al. Who will leave the company? A large-scale industry study of developer turnover by mining monthly work report[C]// 2017 IEEE/ACM 14th International Conference on Mining Software Repositories. IEEE，2017：170-181.

[16] TIAN Y，NAGAPPAN M，LO D，et al. What are the characteristics of high-rated apps? A case study on free android applications[C]// International conference on software maintenance and evolution. IEEE，2015：301-310.

[17] CHAKKRIT T. The Scott-Knott Effect Size Difference (ESD) Test[EB/OL]. (2018-05-08). https：//cran. r-project. org/web/packages/ ScottKnottESD/ScottKnottESD. pdf.

[18] WOLPERT D H，MACREADY W G. An efficient method to estimate bagging's generalization error[J]. Machine Learning，1999，35：41-55.

[19] ABDI H. Bonferroni and Šidák corrections for multiple comparisons[J]. Encyclopedia of Measurement and Statistics，2007，3(1)：2007.

[20] SALTON G，MCGILL M. Introduction to modern information retrieval[M]. McGraw-Hill，1983.

[21] GAY G，HAIDUC S，MARCUS A，et al. On the use of relevance feedback in IR-based concept location[C]// International Conference on Software Maintenance. IEEE，2009：351-360.

[22] WONG C P，XIONG Y，ZHANG H，et al. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis[C]// International Conference on Software Maintenance and Evolution. IEEE，2014：181-190.

[23] RAHMAN S，GANGULY K K，SAKIB K. An improved bug localization using structured information retrieval and version history[C]// International Conference on Computer and Information Technology. IEEE，2015：190-195.

[24] YOUM K C，AHN J，LEE E. Improved bug localization based on code change histories and bug reports[J]. Information and Software Technology，2017，82：177-192.

[25] DEERWESTER S，DUMAIS S T，FURNAS G W，et al. Indexing by latent semantic analysis[J]. Journal of the American Society for Information Science，1990，41(6)：391-407.

[26] BLEI D M，NG A Y，JORDAN M I. Latent dirichlet allocation [J]. Journal of Machine Learning Research，2003，3(Jan)：993-1022.

[27] MORENO L，TREADWAY J J，MARCUS A，et al. On the use of stack traces to improve text retrieval-based bug localization [C]// International Conference on Software Maintenance and Evolution. IEEE，2014：151-160.

[28] WANG S，LO D. Amalgam+：Composing rich information sources for accurate bug localization[J]. Journal of Software：Evolution and Process，2016，28(10)：921-942.

[29] SISMAN B，KAK A C. Assisting code search with automatic query reformulation for bug localization[C]// 2013 10th Working Conference on Mining Software Repositories. IEEE，2013：

309-318.

[30] RAHMAN M M, ROY C. Poster: improving bug localization with report quality dynamics and query reformulation[C]// International Conference on Software Engineering: Companion. IEEE, 2018: 348-349.

[31] KIM M, LEE E. A novel approach to automatic query reformulation for ir-based bug localization[C]// Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. 2019: 1752-1759.

[32] LAM A N, NGUYEN A T, NGUYEN H A, et al. Bug localization with combination of deep learning and information retrieval [C] // International Conference on Program Comprehension. IEEE, 2017: 218-229.

[33] XIAO Y, KEUNG J, BENNIN K E, et al. Improving bug localization with word embedding and enhanced convolutional neural networks[J]. Information and Software Technology, 2019, 105: 17-29.

[34] CAO J, YANG S, JIANG W, et al. Bugpecker: Locating faulty methods with deep learning on revision graphs[C]// Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. 2020: 1214-1218.

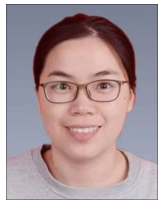
[35] HUO X, THUNG F, LI M, et al. Deep transfer bug localization [J]. IEEE Transactions on Software Engineering, 2019, 47(7): 1368-1380.

[36] MENG X, WANG X, ZHANG H, et al. Improving fault localization and program repair with deep semantic features and transferred knowledge[C]// Proceedings of the 44th International Conference on Software Engineering. 2022: 1169-1180.

[37] LIANG H, HANG D, LI X. Modeling function-level interactions for file-level bug localization[J]. Empirical Software Engineering, 2022, 27(7): 1051-1076.

[38] YOUSOFVAND L, SOLEIMANI S, RAFE V. Automatic bug localization using a combination of deep learning and model transformation through node classification[J]. Software Quality Journal, 2023, 31(4): 1045-1063.

WANG Zhaodan, born in 1999, postgraduate. Her main research interests include software static bug localization.



ZOU Weiqin, born in 1988, Ph.D, professor, is a member of CCF (No. D3300M). Her main research interests include bug localization and software repository mining.