分类号 TP311.5　　密级

U D C　　编号 10486

武漢大學

硕 士 学 位 论 文

一种基于文本分类和评分机制的软件缺陷分派方法研究

研 究 生 姓 名 ：史小婉

学　　号 ：2015202110061

指导教师姓名、职称 ：马于涛 副教授

专 业 名 称 ：计算机软件与理论

研 究 方 向 ：软件工程

二〇一八年五月

Software Bug Triaging Based on Text Classification and Developer Rating

By

SHI Xiaowan

Supervised by

Associate Professor MA Yutao

Wuhan University

May, 2018

论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的研究成果。除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

学位论文作者（签名）：

年　　月　　日

摘要

在过去的二十年中，我们目睹了软件对人类社会日益增长的影响。这一趋势与 Marc Andreessen 在 2011 年发表的著名宣言“软件正在吞噬世界”相呼应。此外，软件定义的任何东西被 Gartner 确定为 2014 年的十大战略技术趋势之一，这意味着我们正在进入一个软件定义的信息社会新纪元。与专有软件相比，开源软件为软件开发和分发方式带来了根本性的转变。由于较低的成本、较强的安全性、不被供应商锁定、更好的质量等因素，越来越多的企业及公司采用开源软件产品作为它们的 IT 系统的重要组成部分。

开源软件项目的缺陷管理和修复是保障其软件质量及软件开发效率的重要手段。目前，大多数大型的开源软件项目使用缺陷跟踪系统来对缺陷进行管理，如 Bugzilla、GNATS 等，缺陷跟踪系统主要完成对缺陷报告的记录、分析和状态更新等管理。软件开发人员、测试人员以及普通用户都能以缺陷报告的形式将缺陷提交至缺陷跟踪系统。但是，随着软件项目规模的不断增大，大量的缺陷被发现和提交，这导致缺陷的人工管理变得越来越困难，特别是如何将大量的缺陷及时分配给合适的开发者去修复。

为了提高缺陷分配的效率、降低相应的人力和时间成本，到目前为止，已有不少的研究者提出了各种缺陷自动分配的方法。从使用的数据、算法、再分配图模型等维度，现有方法大致分为三类：基于文本内容、基于开发者关系和混合类型。但大多数已有方法认为，不管是参与分派和讨论缺陷的开发者还是最终修复缺陷的开发者都是对缺陷分派有贡献的人。因此，它们通常把所有与缺陷相关的开发者都看作是缺陷的标签，从而能够在预测时得到较高的召回率。

针对缺陷的自动分派问题，本文首先提出了一种基于文本分类和评分机制的单一修复者的预测方法，其核心思想是综合考虑基于机器学习的文本分类和基于软件缺陷从属特征的评分机制来构建预测模型。针对大型开源软件项目 Eclipse 和 Mozilla 的十万级已修复软件缺陷的实验表明，在“十折”增量验证模式下，本文方法的最好平均准确率分别达到了 78.39%和 64.94%，比基准方法（机器学习分类+再分配图）的最高平均准确率分别增加了 17.34%和 10.82%，从而验证了其有效性。

另外，我们根据缺陷的历史修复数据分析开发者之间的合作关系，计算分配概率，构建缺陷的分配概率图，然后结合文本分类和评分机制对与缺陷修复过程中涉及的开发者进行推荐。针对 Eclipse 和 Mozilla 两个开源项目的缺陷数据，取得了较高的推荐召回率，分别达到了 78.20%和 71.98%。与 FastText+评分机制+缺陷分配图基准方法相比，平均召回率分别增加了 25.53%和 27.74%，同时也比

ML-KNN+开发者相似度方法的最高平均召回率分别增加了 14.17%和 13.25%，从而验证了该方法对缺陷所涉及的开发者推荐的有效性。

关键词： 开源软件、缺陷分配、文本分类、评分机制、分配图

ABSTRACT

Over the past two decades, we have witnessed the growing effect of software on human society. This trend echoes the famous Marc Andreessen's 2011 proclamation that software is eating the world. Moreover, software-defined anything has been identified by Gartner as one of the top ten strategic technology trends for 2014, implying that we are entering a new era of the information society defined by software. Compared with proprietary software, open-source software (OSS) brings forth a fundamental shift in the manner in which software is developed and distributed. Due to lower cost, superior security, freedom from vendor lock-in, better quality and other primary reasons, an increasing number of enterprise-level companies have adopted OSS offerings as an essential part of their IT systems.

To improve the efficiency of bug triage and reduce the cost of human and time, so far, many researchers have proposed a variety of automatic bug assignment methods. According to the data, algorithms, and tossing graph models, there are three categories of existing methods: text content-based approach, developer relationship-based approach, and hybrid approach. However, most of the existing methods assume that developers who distribute, discuss, or fix bugs are helpful to bug triaging. Therefore, they usually regard all the developers who are involved in the triaging process of a bug as the labels of the bug, so that they can get a higher recall rate when recommending possible developers.

First of all, a prediction method based on text classification and developer rating is proposed in this paper. The core idea of building the prediction model is to consider both text classification based on machine learning and rating based on the source of bugs. According to the experiment on hundreds of thousands of bugs in the Eclipse and Mozilla projects, in the ten-fold incremental verification mode, the best average accuracies of our method reach 78.39% and 64.94%, respectively. Moreover, the accuracies of our method are increased by 17.34% and 10.82%, respectively, compared with the highest average accuracies of the baseline method (machine learning classification + tossing graphs). Therefore, the results indicate the effectiveness of our method.

Also, we analyze the cooperation between developers based on the history of each bug report, calculate the tossing probability, and construct the tossing graph for bugs. We then combine the method mentioned above and the tossing graph to recommend potential developers related to the bug triaging process. Aiming at the defect data of the

two open-source projects of Eclipse and Mozilla, the proposed approach obtains high recall rates which reach 78.20% and 71.98%, respectively. They are increased by 25.53% and 27.74%, respectively, compared with the FastText+developer rating+tossing graphs method. Also, the recall rates of our method are 14.17% and 13.25% higher than those of the ML-KNN+developer similarity method, which demonstrates the effectiveness of our method for recommending multiple possible developers.

Keywords: Open-source software, Bug triage, Text classification, Rating, Tossing graph

目录

图目录

表目录

第一章 绪论

1.1 研究背景

开源软件是一种开放、自由，并可以让公众进行共享的软件，所有人都可对其源码按照自己的需求进行修改。由于其良好的特性，无论是开发人员还是大型的软件公司或企业，都愿意采用开源的方式来展现自己的产品。因此，开源软件在软件行业中发挥着越来越重要的作用[1]。据Gartner调查显示，99%的公司或企业将开源软件加入到了其IT系统中[2]。随着软件行业的发展，开源软件的规模和复杂性也在不断地增加，因此会有更多的缺陷不断产生。由于缺陷数量的不断增大，如果开发人员对其修复不及时，这将会给用户对软件的使用带来更多的不便。大规模缺陷的及时修复成为软件工程研究与实践的一大难题，这也给软件的维护增加了挑战性，严重影响了软件的可靠性和可用性。

软件维护需要投入较高的成本和精力。美国国家标准技术研究所做过一个调查，据估计，为解决软件缺陷每年所耗费的成本大约为 595 亿美元[3]。一些软件维护相关的研究表明：每年用于软件维护所花费的成本至少占与软件产品相关的总成本的 50%，有时甚至高达 90%[4]。有些估计会将软件维护的成本设定为初始软件版本所耗费成本的好几倍[5]。这些调查表明：较低的缺陷修复效率将会增加软件演化过程的工作量，并会增加软件的生产成本。

大多数的软件项目使用缺陷跟踪系统组织管理缺陷的修复过程和促进应用程序维护[6],[7]。例如，Bugzilla[8]是一个非常流行的缺陷跟踪系统，被许多大型的软件项目所使用，如 Mozilla、Eclipse、KDE 和 GNOME。这些应用项目每天都会接收数以百计的缺陷报告。在开源软件的早期，由于项目规模和复杂度较小，整个项目会产生较少的缺陷，人工地将缺陷分配给开发者进行修复完全是可以的。但是，对于现阶段的开源软件项目，缺陷数量大幅度增加，缺陷分配的任务量繁重，很多大型开源项目每天会有 300 多个缺陷被提交来等待分配，缺陷分配的任务大大超过了一个人所承受的工作量[9]。同时，人工的缺陷分配过程是相当复杂的，由于每个开发者所擅长的开发任务、开发经历、团队项目方向和结构都是有差别的，而缺陷分配者对开发者和缺陷报告相关的信息的了解也是有限的，会存在较多的缺陷所分配的开发者不合适修复该缺陷的情况[10]。而且，发现所分配的开发者不适合修复所分配的缺陷以及将缺陷重新分配是需要大量时间的，这将增加缺陷修复所占用的整体时间，从而影响软件开发的周期和效率，同时也会给软件开发和维护增加成本[11]。

在理想的情况下，我们希望每一个缺陷都能在最短时间内被分配给能够修复

它的开发者。缺陷分配是一个很复杂的过程，如果是人工分配，那么分配缺陷的工作将是劳动密集的、耗时的，并且是容易出错的[12]。而且，在软件版本的不断更新过程中，会增加更多的组件、模块、开发人员和测试人员[13],[14]，同时也将会发现和提交更多的缺陷，这将极大地增加项目中缺陷管理的难度，特别是如何高效地为缺陷分配合适的修复者[15],[16],[17]。Jeong 等人的实证研究[18]指出，平均而言，对于 Eclipse 项目，分配者将缺陷分配给第一个开发者一般需要 40 天的时间，若第一个开发者不能够修复所分配的缺陷，则将缺陷分配给第二个开发者会大约再需要 100 天的时间。同样地，在 Mozilla 项目中，需要 180 天的时间进行第一次开发者的分配，如果第一个开发者无法修复缺陷，将会再需要 250 天的时间进行第二次分配。这些数据表明，缺乏有效的缺陷自动分配技术将会导致需要耗费相当大的精力去解决与缺陷修复相关的其他工作。

1.2 研究目的和意义

人类社会经济的发展在很大程度上受软件行业发展的影响，但是随着软件规模和复杂度的增加，软件管理和对软件质量控制的难度也越来越大。根据软件发展的现状来看，软件行业的竞争力在很大程度上依赖于软件质量的好坏，而不是早期人们认为的竞争力完全取决于软件开发和维护中所使用技术的先进性。然而，在软件项目中不断出现的缺陷给软件开发带来了巨大的压力，并且缺陷修复的过程是极其耗时和非常容易出错的。根据多年来的经验，保障开源软件开发效率和降低成本的一个重要手段就是快速地修复软件中所出现的缺陷，因而提高软件缺陷分配的效率是其中亟需解决的一个关键问题。

提高缺陷分配的效率可以大大地降低相应的人力和时间成本，多年以来，相关的研究者一直围绕该问题不断地进行分析和实验，并提出了各自的方法。这些方法主要围绕数据、算法、再分配图（tossing graph）模型等维度开展研究。根据不同的思路可以将现有方法划分为三类：基于文本内容、基于开发者关系和混合类型。当前缺陷自动分配技术的研究主要利用缺陷报告的描述信息、缺陷关联信息、历史分配信息等，但这些方法都没有将缺陷报告信息充分挖掘和利用。本文充分利用缺陷的文本信息、从属特征以及开发者关系等特征，并在增量学习的验证模式下，无论是对缺陷的修复者还是参与缺陷分配的相关开发者的推荐都进行了研究，并获得了较好的效果，其意义体现在以下三个方面：

1、对缺陷修复者的推荐：在新的缺陷报告被提交时，能够一次性（或在尽可能少的次数内）分配给可以修复它的开发者，这将会大大提高缺陷修复的效率，从而降低软件开发的成本以及提高软件的质量。

2、对参与缺陷分配的开发者的推荐：即使对缺陷的修复者推荐有误时，能

够将缺陷推荐给可能参与分配的开发者，加快缺陷分配的速度，尽快地将缺陷分配给能够将其修复的开发者，从而缩短缺陷从被提交到修复的整体时间，进而提高缺陷的修复效率，保证软件的质量。

3、通过本文对缺陷自动分配的研究，可以为开源软件的开发和维护人员在缺陷管理工作上提供一些参考，并希望我们所提出的方法能够应用于缺陷跟踪系统的缺陷分配工作中，以达到提高软件质量、节约软件开发和维护成本的目的。

1.3 国内外研究现状

预测缺陷修复者（fixer）的早期研究主要使用的方法为文本分类[19]~[22]，图 1 展示了该类方法的基本框架。Cubranic 等人首先使用了缺陷的文本信息，利用基于机器学习的文本分类方法来解决缺陷的分配问题[21]，将该问题转换成一个多类别、单标签的监督学习（分类）问题。从缺陷的标题和描述信息中提取出关键字作为缺陷的特征并训练了一个朴素贝叶斯（NB）分类器，在 15,859 个 Eclipse 缺陷报告上的准确率达到了 30%。

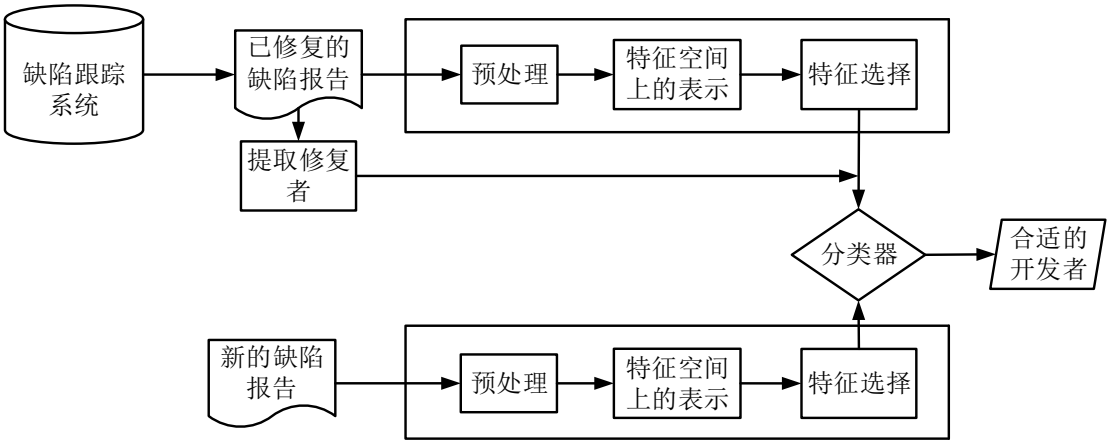


图 1 文本分类基本框架

Anvik 等人对 Cubranic 等人的方法无论是从使用的数据还是分类算法上都进行了改进[22]。对于训练数据集，作者过滤掉了标记为 *invalid*、*wontfix* 和 *worksforme* 以及重复的缺陷报告，同时也去除了包含不再参与缺陷修复工作的开发者信息的缺陷报告，并且对于不再活跃和修复缺陷数量少于 9 个的开发者在推荐过程中不予考虑。在分类算法上，作者使用了 NB、C4.5 和支持向量机（SVM）三种分类器在 Eclipse 和 Firefox 两个项目的数据集上进行实验，准确率分别达到了 57%和 64%。使用该方法对 GCC 项目的缺陷数据进行验证，在推荐一个和三个修复者实验中，精确率分别达到了 16%和 18%，从而验证了其方法的有效性。在后续的工作中，作者又采用了多种机器学习的分类算法，利用 SVM 和基于组件（component）的方法[23]，对于 Eclipse 和 Firefox 两个数据集，准确率提升到

了 97%和 70%。

Lin 等人提出了两种缺陷自动分配的方法[24]：基于文本信息的开发者推荐方法和基于非文本信息（如缺陷类型、缺陷所属类和优先级等）的开发者推荐方法。值得注意的是，该方法是最先使用中文信息来进行缺陷分配的。在试验中，作者采用基于中文文本的 SVM 分类算法以及基于非文本信息的 C4.5 分类算法来实现缺陷的分配。结果表明，基于文本分类方法的准确率达到了 63%，接近于人工缺陷分配的 67%的准确率；基于非文本分类的方法的准确率达到了 77.64%，远远高于人工分配的准确率。

Zhang 等人提出了一种基于最近邻（KNN）查找和开发者合作网络[25]的 KSAP 方法[26]，该方法分为两个步骤：首先，对于新提交的缺陷报告，使用 KNN 方法从已经被修复的历史缺陷报告中搜索出与其相似的缺陷；然后，对于修复这些相似的缺陷报告的开发者，作者使用异构相似度的方法来进行排序，从而对缺陷的修复者进行推荐。在 Mozilla、Eclipse、Apache Ant 和 Apache Tomcat6 项目的数据集上对该方法进行验证，与基准方法相比召回率提高了 7.5~32.25%。同时，发现在对 KNN 的 k 值以及缺陷报告的数量进行调整时，KSAP 方法仍能保持其优势。

Jeong 等人提出了一种基于马尔可夫链的再分配图模型[18]，该模型证明了根据缺陷分配关系所产生的开发者合作网络蕴含对缺陷修复者推荐的重要信息。该方法通过分析历史数据集中开发者之间的缺陷传递关系构建了一个开发者合作网络图，当一个新的缺陷被提交时，可以根据网络图来实现缺陷的传递。作者以 445,000 个缺陷报告作为数据集来进行实验，结果表明，此方法对缺陷的再分配事件减少效果非常明显，减少率接近 72%，而且当使用机器学习分类算法（如朴素贝叶斯和贝叶斯网络[27]）结合再分配图方法的预测准确率比基准方法提高近 23%。

Wu 等人提出了一种称为 DREX（基于 KNN 搜索和开发者专业知识排序）的开发者推荐方法[28]，该方法首先利用 KNN 在已经被修复的缺陷报告数据集中查找与新提交的缺陷最相似的缺陷集合，找出与搜索出的缺陷集合相关的所有开发者，然后根据这些开发者之间的传递关系构建开发者合作网络。通过网络指标入度、出度、频率（frequency）、PageRank、介数（betweenness）和接近性（closeness）等按照所制定的排序函数对开发者进行排序。在 Mozilla 数据上的实验结果表明，在使用出度和频率两个指标进行开发者排序时得到最好的效果，平均召回率达到了 60%。

Xuan 等人提出了一种基于排序方法的开发者推荐模型[29]，该方法的思路是通过分析缺陷的评论者和修复者之间的关系、评论信息的规模以及评论者之间的合作关系，根据开发者对缺陷评论合作关系制定协作排序（Collaboration ranking）以及根据缺陷的特征来衡量开发者的能力而制定能力排序（Capability ranking），然后将以上两种排序方法进行组合对开发者进行最终的排序，从而对缺陷的评论者进行推荐。在

Eclipse IDE、JDT、Firefox 和 Thunderbird 四个项目五年的历史缺陷报告的实验结果表明，当推荐 top10 个评论者时召回率达到了 41%~75%。在后面的研究工作中，作者又提出了一种基于半监督的文本分类方法[30]，该方法的思路是通过训练有标签的缺陷报告来迭代地标记无标签的缺陷报告，然后将标记好的缺陷报告加入到训练数据集来不断地训练新的分类器，使用 EM（期望最大化）算法来不断增强 NB 分类算法的性能，该方法将分类器的准确率提高了 6%，同时与基准方法相比，准确率提高了 11%~43%。

Xie 等人提出了一种称为 DRETOM 的开发者推荐方法[31]。该方法使用斯坦福主题建模工具箱（TMT）将属于同一主题的缺陷报告进行分组从而构建主题模型，当一个新的缺陷报告被提交时，将会很容易知道该缺陷属于哪一个主题，然后根据已经被修复的缺陷的历史数据分析开发者对所对应主题的缺陷的兴趣和经验，从而为缺陷推荐合适的修复者。试验结果表明，该方法要优于机器学习的算法（如 SVM 和 KNN）以及基于社交网络分析的推荐方法。

LDA（Latent Dirichlet Allocation）是一种将离散的数据分类成不同主题的生成概率模型[32]。Naguib 等人[33]使用 LDA 将缺陷报告分为不同的主题，然后通过挖掘历史日志以及缺陷报告的主题模型为缺陷管理系统中的每一个开发者创建一个活动描述文件。该描述信息包括两部分：开发者的角色和与开发者所关联的主题。利用开发者的活动描述和新提交的缺陷的主题来寻找最可能的开发者来修复缺陷。实验结果表明，该方法的平均命中率（即准确率）达到了 88%。

Yang 等人[34]同样使用了 TMT 生成主题模型来验证新提交的缺陷报告属于哪一个主题，然后他们从属于相同主题和相同特征（所属产品和组件、优先级、严重程度）的历史缺陷报告数据中提取出候选开发者。然后，基于这些开发者之间的社交网络关系的分析，该文提出的方法捕获了能够反映缺陷报告状态变化的开发者对缺陷的评论和传递动作的特征，从而实现了缺陷修复者的自动分配。实验结果表明，该方法的效果要好于使用社交网络的出度特征[28]来推荐开发者的方法。

此外，一些考虑缺陷的文本信息以及开发者之间关系的混合方法在近几年被相继提出[17],[35]~[37],[39]。Bhattacharya 等人提出了一种“ML+TG”（基于文本分类和再分配图）的再分配方法[37]，该方法根据缺陷的文本信息使用分类算法将缺陷分配给第一个开发者，若该开发者不能修复缺陷，则根据分配概率图重新分配开发者，涉及开发者之间的分配概率以及开发者的 product 和 component 特征。同时，作者采用增量学习的方法，提高了修复者推荐的准确率。在 Eclipse 的 306,297 个缺陷报告和 Mozilla 的 549,962 个缺陷报告的数据集上准确率分别达到 77.43% 和 77.87%。

Xia 等人提出了一种 DevRec 开发者推荐的方法[17]。该方法是一种基于缺陷

报告分析和开发者合作关系的混合策略，与以往方法不同的是，该方法所推荐的不是缺陷最终的修复者，而是推荐对缺陷修复做出贡献的一群人。分别基于缺陷报告和开发者关系制定了评分函数，其中在对缺陷报告的分析中使用基于 ML-KNN 多标签分类的方法。在 GNU Compiler Collection、OpenOffice、Mozilla、Netbeans 和 Eclipse 五个数据集共 107,875 个缺陷报告的实验中，当推荐 top5 和 top10 个开发者时的召回率分别达到了 48.26%~79.89%和 60.63%~89.24%，与 Bugzie[38]相比，召回率提高了 57.55%和 39.39%；与 DREX[28]方法相比，召回率提高了 165.38%和 89.36%。

Liu 等人提出了一种简单易用的针对软件缺陷自动分派的开发者推荐方法[39]，其核心思想是利用 LDA 主题模型（刻画开发者技能）、开发者合作网络（刻画开发者之间的合作关系）构造（内容+关系）混合策略。针对大型开源软件项目 Eclipse 和 Mozilla 的十万级已修复缺陷的实验表明，在选取合适的参数和分派策略情况下，该方法的开发者推荐的准确率分别达到了 46.7%和 33.4%，比基准的 LDA + KNN 方法的推荐准确率分别提高了 209.3%和 131.9%。

1.4 研究内容

本文的研究工作受国家重点基础研究发展计划（973）（2014CB340404）、国家自然科学基金（61272111）和武汉市黄鹤英才（现代服务）计划资助。

如果我们将机器学习的技术用于解决缺陷分配问题，则可定义为：将缺陷报告的一系列特征作为分类算法输入的特征向量，修复缺陷的开发者作为输出的类别标签。因此，当新的缺陷报告被提交时，我们可以根据训练好的分类器来对缺陷的开发者进行预测。

缺陷分配问题的解决方案可以分为两种：第一种是将缺陷的修复者作为其标签，由于缺陷最终的修复者只有一个，并且在缺陷管理系统中参与缺陷修复的开发者有成百上千，简而言之，有多个类别，但是每个缺陷只能分配一个开发者，即，只属于一个类别，因此，这类缺陷分配问题被称为单标签多类分类问题；另一种是将对缺陷修复有贡献的开发者都作为缺陷的标签，由于在缺陷的生命周期中，参与其修复工作的开发者会有多个，也就是每个缺陷属于多个类别，因此，这类问题可被定义为多标签分类问题。

因此，本文的主要研究内容是开源软件的缺陷自动分配。对于缺陷的分配，为了提高缺陷的修复效率，我们可以对其修复者进行推荐，但是仍然存在修复者推荐错误的缺陷，因此我们也可以对从缺陷被提交到修复过程中对缺陷分派有贡献的开发者进行推荐，从而加快缺陷被分配给正确的修复者的速度，达到提高缺陷修复效率的目的。因此，本文对这两方面的工作都做了相应的研究。

对于缺陷的单一修复者推荐，我们提出了一种文本分类+评分机制的方法（本文第三章），使用缺陷相关的文本信息进行文本分类，并根据缺陷所属组件的产品信息制定评分机制，将两种方法进行结合以实现对缺陷的修复者进行推荐。对于缺陷分派有贡献的相关开发者的推荐，我们提出了一种文本分类+评分机制+缺陷分配图的方法（本文第四章），该方法考虑了在缺陷修复过程中参与缺陷传递的开发者之间的合作关系，通过开发者之间的合作关系构建缺陷分配图，并结合文本分类+评分机制，自动推荐与缺陷相关的多个可能的开发者。

1.5 本文的结构框架

本文一共分为五个章节，各个章节的具体内容安排如下：

第一章为绪论，主要介绍了开源软件缺陷跟踪系统中的缺陷分配的研究背景，本文的研究目的和意义，同时总结了国内外对缺陷自动分配方法的研究现状以及本文的主要研究内容。

第二章为相关概念和方法的介绍，详细阐述了缺陷报告所包含的各类信息以及缺陷从被提交到被修复的整个生命周期。然后，介绍了文中所使用的词向量化方法以及在对比实验中所使用的快速文本分类算法，同时对主流的机器学习分类算法进行了简单介绍。

第三章介绍了本文提出的一种基于文本分类和评分机制的软件缺陷单一修复者分配方法，着重介绍了评分机制和候选人排序方法，最后通过与基准方法的对比分析验证了所提方法的有效性。

在第四章中，针对与缺陷分配有关的开发者的推荐问题，考虑了在缺陷修复过程中参与缺陷传递的开发者之间的合作关系，通过分析开发者之间的合作关系构建缺陷分配图，并提出了一种基于文本分类+评分机制+缺陷分配图的改进方法，进一步提高缺陷的修复效率。

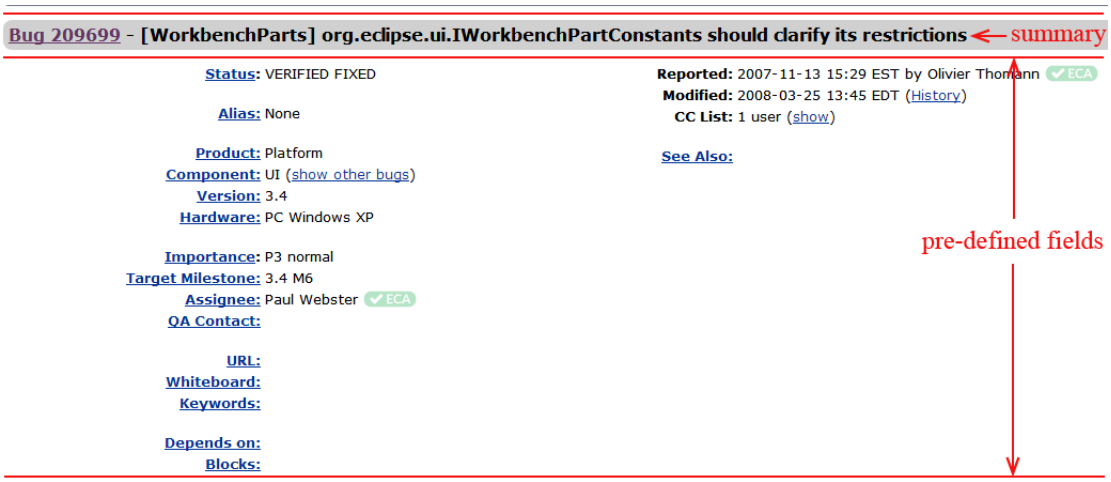
第五章是对本文的总结以及对未来工作的展望。

第二章 相关概念和方法

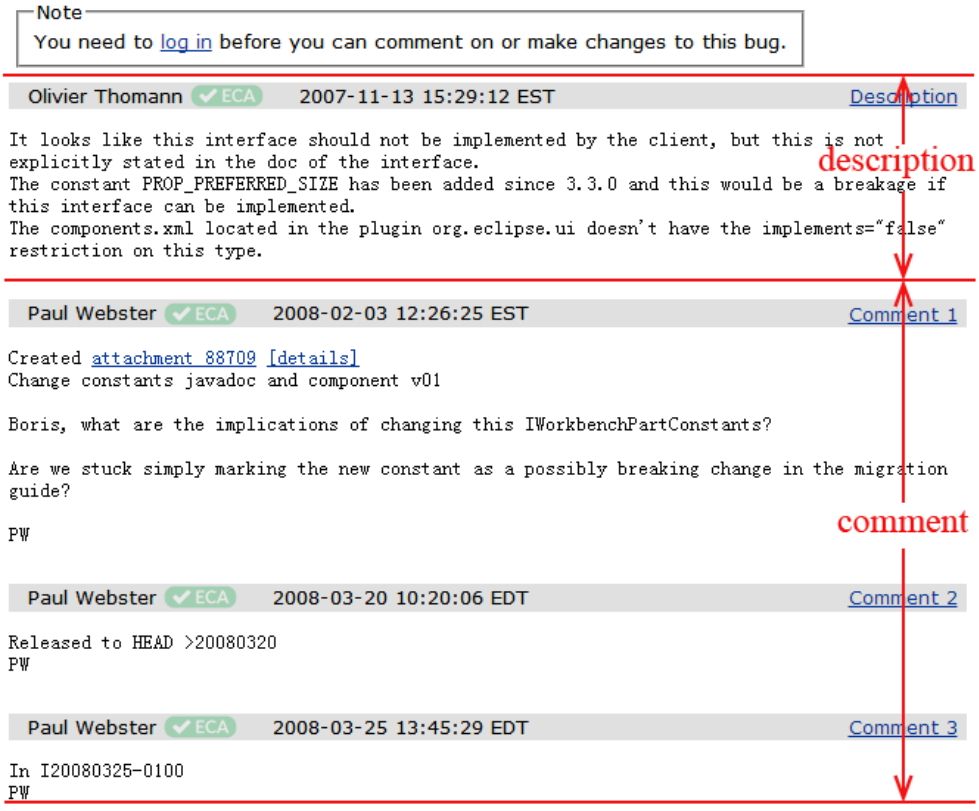
2.1 缺陷报告及生命周期

2.1.1 缺陷报告

软件缺陷报告主要包括以下四类信息（以 Bugzilla 为例）：



（a）摘要和预定义字段



（b）description 和 comment

Back to ~~bug 209699~~

Who	When	What	Removed	Added
Tod_Creasey	2007-11-14 08:25:54 EST	Assignee	Platform-UI-Inbox	pwebster
		Summary	org.eclipse.ui.IWorkbenchPartConstants should clarify its restrictions	[WorkbenchParts] org.eclipse.ui.IWorkbenchPartConstants should clarify its restrictions
pwebster	2008-01-07 14:34:59 EST	Status	NEW	ASSIGNED
		Target Milestone	---	3.4
pwebster	2008-01-08 09:57:21 EST	Target Milestone	3.4	3.4 M5
pwebster	2008-02-03 12:24:50 EST	CC		Boris_Bokowski
pwebster	2008-02-11 14:20:21 EST	Target Milestone	3.4 M5	3.4 M6
pwebster	2008-03-20 10:20:06 EDT	Status	ASSIGNED	RESOLVED
		Resolution	---	FIXED
pwebster	2008-03-25 13:45:29 EDT	Status	RESOLVED	VERIFIED

（c）修改历史

Attachments

Change constants javadoc and component v01 (1.49 KB, patch) 2008-02-03 12:26 EST, Paul Webster ECA	*no flags*	Details | Diff
Add an attachment (proposed patch, testcase, etc.)		View All

（d）附加信息

图 2 示例：编号为 209699 的缺陷报告的内容

第一类是缺陷报告的预定义字段，主要描述了缺陷的基本特性，主要包括：状态、提交时间、修改时间、所属产品、所属构件、版本号、邮件抄送列表以及重要性（优先级和严重程度）等信息。例如，状态（Status）为 “resolved fixed”表示该缺陷已经被开发者所修复；重要性（Importance）为“P3 normal”表示该缺陷的优先级为 P3，严重程度为正常（normal）。但有的信息在缺陷的生命周期中是有可能发生变化的，如从缺陷被提交到被开发者修复这个过程中，其状态是会改变的，以及邮件抄送列表中的邮件地址也是会发生变化的。

第二类是缺陷的文本信息，是由摘要（summary）、描述（description）、评论（comment）三部分信息所组成。摘要和描述信息提供了所报告的缺陷的详细信息，评论是由参与或对项目感兴趣的开发者所发布，是用来讨论如何修复该缺陷的。

第三类是缺陷的状态更改历史，每条记录都由 5 个属性“谁，什么时间，做什么，移除增加”（who，when，what，removed，added）所构成，主要说明某个开发者在某个时间对缺陷做了相关的操作，其中“removed”和“added”为所对应的开发者针对“what”动作所添加或删除的内容。这些记录描述了在缺陷的生命周期中缺陷状态的改变以及所对应的开发者的活动，从这些数据中我们可以抽取出缺陷的修复时间和修复者，以及在缺陷生命周期中所涉及的所有开发者。

同时，开发者也可以对缺陷报告附加一些非文本的信息，如补丁和测试用例等。

2.1.2 缺陷生命周期

一个缺陷报告从被提交到最终被关闭都会有一个完整的生命周期，如图 3 所示。

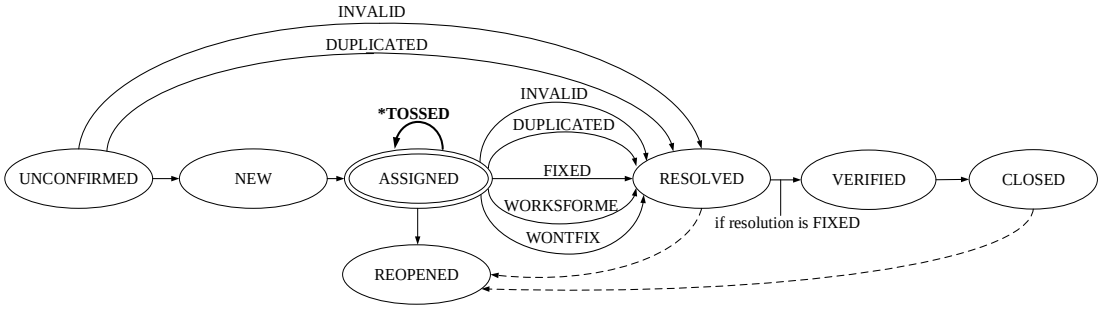


图 3 缺陷报告的生命周期

缺陷报告在被首次提交时被设定的状态为“未确认”（Unconfirmed），在被管理人员确认为有效（不重复且信息完整的缺陷）后，会被标记为“新提交”（New）状态；管理人员或者开发人员会根据缺陷报告对缺陷的描述将其分配给合适的开发者，此时缺陷的状态变为“被分配”（Assigned）。在该状态下，若所分配的开发者无法修复该缺陷，可以将该缺陷进行多次再分配；直到它被修复后，其状态会被标记为“已解决”（Resolved）。特别地，若缺陷被开发者确认无效、与其他缺陷重复或无法修复时，也会被标记为该状态；之后会经过管理人员或其他开发人员的验证，通过验证之后，该缺陷会进入到“关闭”（Closed）状态。但是，如果缺陷的修复方案不被管理员认可或缺陷没有被正确修复，则会被重新分配给新的开发者；需要注意的是，缺陷被关闭之后，若有开发者不满意修复结果或是在修复之后依然出现同样的问题，那么管理人员可以把缺陷的状态标记为“重新打开”（Reopened），再次分配开发者进行修复，意味着该缺陷进入到新的生命周期。

需要注意的是，开发人员在缺陷处理的过程中会扮演不同的角色。提交缺陷报告的人称为报告者或提交者；决定缺陷报告是否具有实际意义以及将缺陷分配给哪个开发者去修复的人称为分配者；最终修复缺陷的人称为修复者，同时修复者还可以对缺陷报告增加一些附加的信息，如修复缺陷的方法；一个开发者可以在缺陷生命周期中的任何时候承担任何的角色。例如，一个开发者可以提交一个缺陷报告并将其分配给自己，贡献一个缺陷的解决方案，然后将缺陷修复。那么对于该缺陷报告而言，该开发者承担了上述所有的角色。

2.2 词向量化和快速文本分类

2.2.1 词向量化

将一个单词映射到新的维度空间，并由一个多维连续的实数向量对它进行表示的方法叫做“Word Representation”或“Word Embedding”[40],[41],[42]，也就是我们所谓的词向量化。在以前的研究中，研究者往往会采用稀疏表示法来进行词向量化，但是在实际问题中，稀疏表示法存在很大的局限性.当单词量较大时，词向量化后的维度太大而导致维数灾难，同时单词间的语义信息不能够被准确表示出来。现阶段的研究逐渐过渡到了低维空间的密集表示法，该方法用低维向量表示单词，解决了维数灾难的问题，同时还可以挖掘单词之间的关联关系，从而能够较大程度地提高单词向量语义上的准确度[43]。在本文中，我们使用了两种词向量化的方法：TF-IDF[44]和 Word2vec[45]。

2.2.1.1 TF-IDF

TF-IDF，即“词频-逆文本频率”，是一种用于文本处理的统计方法，该方法通过计算来对一个字或词在一个语料库中某个文件的重要程度进行评估[46],[47]。当字/词在文件中出现的次数越多，其重要性会越大，即字/词的重要性与其在文件中出现的频率成正比。但是，若该字/词在语料库中出现的频率越高，其重要程度反而会降低，与语料库中出现的频率成反比。该方法的核心思想为：对于一个类别的文本，若某个单词或短语在其中出现的频率较高，而在其他类别的文本中出现的次数较少或不出现，则说明该单词或短语能够较好地区分文本类别，可以用来进行文本分类。其中 TF 为词频，即文本中各个词出现的频率。IDF 为逆文本频率，它反应了一个词在所有文本中出现的频率，如果一个词在很多的文本中出现，那么它的 IDF 值会较低；而反过来如果一个词在比较少的文本中出现，那么它的 IDF 值应该较高[48]，比如一些专业的名词。一个极端的情况，如果一个词在所有的文本中都出现，那么它的 IDF 值应该为 0。

一个词 x 的 TF 为该词在某个文本中出现的次数，即：

$$TF(x) = n(x). \tag{2-(1)}$$

一个词 x 的 IDF 的基本公式如下：

$$IDF(x) = log\frac{N}{N(x)}, \tag{2-(2)}$$

其中，N代表语料库中文本的总数，而$N(x)$代表语料库中包含词 x 的文本总数。但是，上面的 IDF 公式在一些特殊的情况会有一些小问题，比如某一个生僻词在语料库中没有，会导致公式 2-(2)的分母为 0，这样 IDF 就没有意义了。为了使语料库中没有出现的词也可以得到一个合适的 IDF 值，我们使用如下的 IDF 平滑

后的公式：

$$IDF(x) = log\frac{N+1}{N(x)+1} + 1. \quad 2\text{-}(3)$$

有了 IDF 的定义，我们就可以计算某一个词的 TF-IDF 值，公式如下：

$$TF - IDF(x) = TF(x) * IDF(x). \quad 2\text{-}(4)$$

从公式 2-(4)可以看出，词 x 在语料库中出现次数越多，其权重会越小。虽然 TF-IDF 方法只是对单词在文本信息中的重要程度进行计算，但是没有考虑词汇之间的语义信息和关联关系。当要进行分类的文本较短时，该方法的效果往往很差，这就是 IDF 的不足之处。

2.2.1.2 Word2vec

Word2vec 是 Google 公司五年前开发的一款可以进行词向量训练的工具，该工具能够通过对文本中单词之间的关联关系进行学习，从而对词汇以分布式向量的方式进行表示[42],[49],[50]。该方法不仅能够解决之前的词向量稀疏表示法的高维稀疏特征问题，而且能够引入低维度密集表示法中不能够表示的单词之间的语义特征，有助于更好地进行文本分类。Word2vec 从算法上来讲是一种神经网络概率语言模型，我们可以用其计算单词的词向量。同时，Word2vec 与传统的高维词向量方法相比，不仅减少了在词向量化过程中计算的复杂度，也不会由于所表示的向量维度较高而造成向量维数灾难。更有意义的是，Word2vec 词向量是根据词汇所在上下文分析单词之间的关联关系而计算出的，并且能够充分捕捉上下文中词汇之间的语义信息，因此，我们很容易通过该方法计算两个词汇的相似程度。根据算法中对词向量训练的方式，Word2vec 可以分为两种模型：CBOW（Continuous Bag-of-words Model）[51]和 Skip-gram[52]。

对于 CBOW 模型，它是根据上下文中的词汇来预测给定词，我们可以将其表达式归纳如下：

$$P(w_t|\tau(w_{t-k}, w_{t-k+1}, \cdots, w_{t+k-1}, w_{t+k})). \quad 2\text{-}(5)$$

其中，假设w_t为语料词典中的一个词，根据 CBOW 模型我们可以通过与w_t相邻的上下文中窗口大小为k的词$w_{t-k}, w_{t-k+1}, \cdots, w_{t+k-1}, w_{t+k}$来对该词所出现的概率进行预测。对于表达式中的$\tau$运算符，其表示将词$w_t$上下文窗口中相邻的词汇进行相加运算。进一步地，CBOW 模型图如图 4 所示。

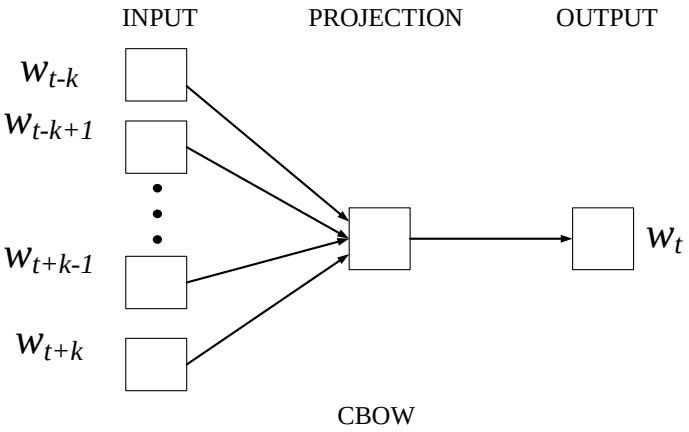


图 4 CBOW 模型图

图中输入层为给定词w_t的上下文中窗口大小为k的词$w_{t-k}, w_{t-k+1}, \cdots, w_{t+k-1}, w_{t+k}$，输出层为$w_t$。

从数学来看，CBOW 模型等价于一个 embedding 矩阵乘以一个词袋模型的向量，可以得到一个连续向量 embedding。这也正是该模型被称为 CBOW 的原因。

Skip-gram 则是根据给定词来预测其上下文，即通过所给定的词汇w_t经过训练去预测其上下文相邻窗口k内所有词汇$w_{t-k+1}, \cdots, w_{t+k-1}, w_{t+k}$的概率，其数学表达式如下：

$$P(w_{t-k}, w_{t-k+1}, \cdots, w_{t+k-1}, w_{t+k} | w_t). \quad 2\text{-}(6)$$

进一步地，Skip-gram 的模型图如下图所示。

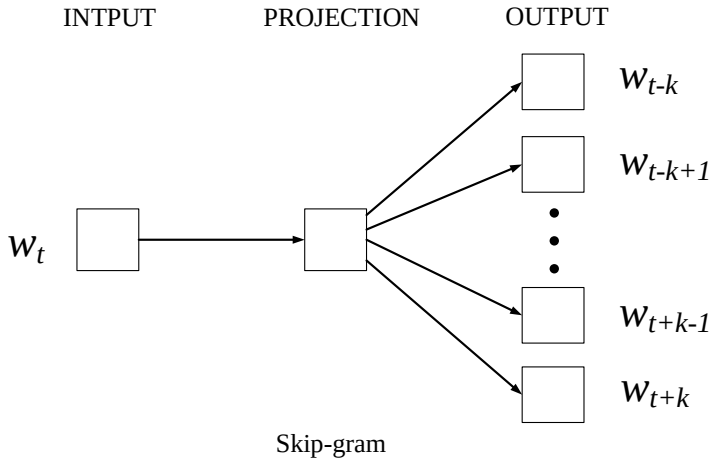


图 5 Skip-gram 模型图

图中的输入层为给定词w_t，输出层为w_t的上下文相邻窗口k内所有词汇$w_{t-k+1}, \cdots, w_{t+k-1}, w_{t+k}$。

如果将 Skip-gram 模型的前向计算过程表示为数学形式，我们得到：

$$P(w_t | w_i) = \frac{e^{U_t \cdot V_i}}{\sum_j e^{U_j \cdot V_i}}. \quad 2\text{-}(7)$$

其中，U_t为词w_t所对应的列向量，w_i为词w_t的上下文窗口中的其中一个词，V_i是词w_i的输入向量，为模型中 embedding 层矩阵中的列向量，U_j是词w_t上下文窗口中的词w_j所对应的输出向量，为模型中 softmax 层矩阵里的行向量。

因此，Skip-gram 模型实质上是对输入单词的输入向量与目标单词的输出向量之间的余弦相似度进行计算，并采取 softmax 归一化的方法对向量进行归一化。

2.2.2 快速文本分类

FastText 是 FaceBook 公司开发的一款用于快速文本分类的分类器，它可以为文本分类和表征学习提供一种简单而有效的方法[53]。与 SVM、Logistic Regression 和 neural network 等模型相比，FastText 不仅能够大大地缩短训练时间，而且能够保持较好的分类效果。

FastText 使用了 N-gram 袋表征语句和词袋，同时也充分利用了子字（subword）中所包含的信息，可以通过模型中的隐藏层来进行不同类别间的信息共享。所以说，FastText 集结了机器学习和自然语言处理中最有效和最成功的理念[54]。同时，为了缩短运行时间，FastText 模型使用了层次 Softmax（利用了类别不均衡分布的优势）技巧。该技巧能够通过哈夫曼编码的方式对标签进行编码，使得模型的预测目标的数量极大地缩小，从而大大缩短模型的训练时间。因此，FastText 可以进行两种文本相关的工作——学习词向量表征和文本分类。

2.2.2.1 模型架构

FastText 的核心思想为：对于训练好的一个模型，我们可以输入一个词序列，那么输出结果为这个词序列属于不同文本类别的概率。模型的主要工作流程为：首先，同一个序列中的词和词组会组成一个特征向量，该特征向量通过线性变换映射到模型的中间层，然后将中间层映射到标签。其中，在该映射过程中，在预测文本标签时会使用非线性激活函数，而在中间层不使用[55]。FastText 模型架构与 CBOW 模型具有很大的相似性，它们都是基于 Hierarchical Softmax 的，同时，在训练过程中都具有输入层、隐藏层、输出层三层架构。但是，两种模型也有不同之处，对于 FastText，其最终的结果为预测文本的标签，而 CBOW 模型则会根据上下文预测中间词。FastText 模型图如下图所示。

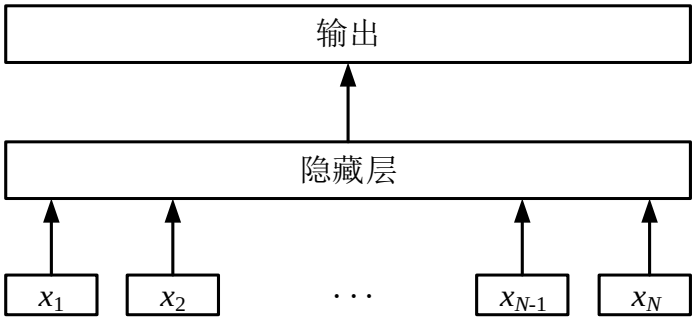


图 6 FastText 模型图

层次之间的映射过程包括：首先，将词和词组输入到输入层，所输入的词和词组会被以特征向量的方式进行表示；然后，经过线性变换，该特征向量被映射到隐藏层，而隐藏层需要对最大似然函数进行求解；然后，根据每个类别的模型参数和权重构建 Huffman 树；最后，将构建好的 Huffman 树作为输出。

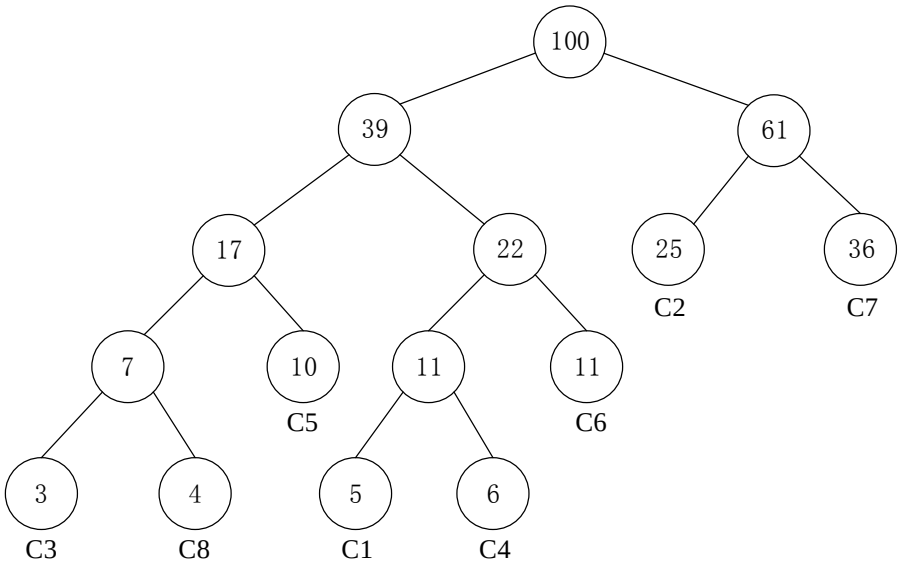


图 7 哈夫曼树示例

2.2.2.2 模型训练

Huffman 树中每一叶子结点代表一个 label，在每一个非叶子节点处都需要做一次二分类，我们将该节点的左边定义为正类别，右边定义为负类别，选择该节点左分枝的概率和选择右分枝的概率，这里以逻辑回归的公式为例进行说明。

正类别的概率：

$$\sigma(X_i\theta) = \frac{1}{1+e^{-x_i\theta}}, \tag{2-(8)}$$

负类别的概率：

$$1 - \sigma(X_i\theta), \tag{2-(9)}$$

其中，θ为实例X_i的权值向量。

每一个 label 都会有一条路径，对于训练样本的特征向量 X_i 和对应的 label

Y_i，预测出来 X_i 的样本属于所对应的 label 是 Y_i 的概率为：

$$P(Y_i|X_i) = \prod_{j=2}^{l} P(d_j|X_i, \theta_{j-1}), \quad 2\text{-}(10)$$

其中，d_j代表对 Huffman 树中的非叶子节点左右分枝的选择，其值为 1 代表选择非叶子节点的左分枝，0 为选择该节点的右分枝。

$$P(d_j|X_i, \theta_{j-1}) = f(x) = \begin{cases} \sigma(X_i\theta), \ if \ d_j = 1 \\ 1 - \sigma(X_i\theta), \ if \ d_j = 0 \end{cases}. \quad 2\text{-}(11)$$

当模型为条件概率分布时，可以使用对数函数来对损失函数进行表示，那么，该对数函数的极大似然估计即为经验风险最小化。

对数似然函数为：

$$L(Y, P(Y|X)) = -log\ P(Y|X). \quad 2\text{-}(12)$$

目标函数为：

$$l = \frac{1}{n}\sum_{i=1}^{n} log\ P(Y_i|X_i). \quad 2\text{-}(13)$$

2.2.2.3 FastText 的词向量表征

2.2.2.3.1 FastText 的 N-gram 特征

词袋模型是 FastText 经常使用的特征，但单词之间的顺序不会被词袋模型所考虑。为了弥补该特征的不足，FastText 加入了 N-gram 特征[56]。例如，“他 打 我”若只使用词袋模型，这句话中的特征（词）就是 “他”、“打”、“我”。这些特征和句子 “我 打 他”的词袋模型特征是一样的。 但是，如果加入 2-Ngram 特征，则效果会有很大的不同。第一句话的特征就会还有 “他-打”和“打-我”，那么这两句话“他 打 我”和“我 打 他”这两句话就能区别开来了。同时，在实际的应用中，我们需要将低频的 N-gram 过滤掉以提高效率。

在 FastText 中，词袋模型能使所有的文本信息在不同的类别中都能够被共同使用，这是因为其中的低维度向量与文本中的每一个单词都会有所关联，而这些隐藏的表征在分类器中的不同类别中都能够被共享。但是为了能够更多地考虑词之间的语义关系，FastText 又加入了将局部词序考虑在内的 N-gram，这在解决很多文本分类问题时起到了很大的作用。

例如，FastText 能够学会“小猫”、“小狗”、“男孩”、“女孩”指代的是特定的人或动物，然后经过训练对这些词进行数值标记，并生成相关文档将这些数值存储起来。若程序要对一个用户提出的请求进行响应，如这个小朋友是男生还是女生？FastText 能够马上对生成的文档进行查找，并通过分析，理解该用户的问题是有关人的问题。

2.2.2.3.2 FastText 词向量优势

（1）适合规模较大数据+训练时间短：能够训练较为复杂的模型，如在使用标准多核 CPU 的硬件前提下，1 亿个词汇可以在 1 分钟内被处理完。与深度学习模型相比，对于一个类似复杂度的模型，当 FastText 的训练时间只需要几秒钟时，深度学习则需要几天甚至十几天的时间，并且 FastText 的训练效果往往不会比深度学习差。

（2）支持处理多种语言：FastText 利用其语言形态结构，能够被用来设计对汉语、英语、法语、德语等多种语言的文本进行分析。它还加入了 N-gram 特征能够充分利用子字信息，在对于汉语这种语义复杂、词态丰富的语言时，能够取得较好的效果。

（3）FastText 擅长于文本分类，例如，对文本的情感分析和标签预测方面表现较好。

（4）与 Word2vec 相比，FastText 考虑了单词之间的相似性，比如对于单词 sportsman 和 sportswoman， FastText 的词嵌入学习能够考虑这两个单词包含相同前缀有可能会表达相似的意思，而 Word2vec 却不具备这种能力。

2.2.2.3.3 FastText 词向量与 Word2vec 对比

FastText 的训练过程相当于对 Word2vec 中的 CBOW 模型以及 h-softmax 的灵活运用，主要体现在以下两个方面：

1、模型的输入层：FastText 的输入层是整个文本，如一个句子或词序，但同时也可以进行单词和 N-gram 特征的输入，而 Word2vec 中 CBOW 模型的输入层是给定词的上下文窗口中相邻的词汇。

2、模型的输出层：FastText 的输出层所对应的是每个文本分类的标签，而 Word2vec 的输出层是计算哪一个单词的概率最大，其对应的主体是每一个单词。但是无论是对于 FastText 还是 Word2vec，他们输出层所对应的向量都不会被保留。

同时，这两种模型也有一些不同之处，其本质上的区别体现在对 h-softmax 的使用上：

1、Word2vec 模型训练的最终的目的是得到词向量，该模型中的词向量是在输入层而不是在输出层得到的，输出层的 h-softmax 也会产生一些中间向量，但是这些向量都不会被保留和使用。

2、FastText 算法在输出层充分利用了 h-softmax 所具备的分类功能，在分类过程中会生成一个分类树，在对文本进行标签预测时，会对分类树的所有叶节点进行遍历，从而找到概率最大的标签作为文本的类别。

2.3 用于缺陷分类的机器学习方法

分类是一种用于从训练数据集中推导出一般趋势的机器学习技术。训练数据集是由一组输入对象（称为特征向量）及它们各自所对应的目标输出（称为类别标签）所组成。有监督学习（或者是分类）的任务就是在使用训练数据集训练之后，根据跟定的一组输入对象（称为测试数据集）来预测它们所对应的输出类别标签。同时，测试数据集中类别的真正标签是已知的，因此，可以用来测试该分类器的准确率、召回率以及精确率。

2.3.1 涉及的机器学习算法

不管是单标签多类分类问题还是多标签分类问题，机器学习的技术都广泛运用于缺陷的开发者分配中，下面介绍一下本文解决这两种分类问题所涉及的机器学习算法。

（1）朴素贝叶斯算法（Naive Bayes）

朴素贝叶斯算法是依据贝叶斯定理而形成的一种分类算法，实现该算法需要具备一个前提条件：特征条件独立假设。若给定的训练数据集，我们要首先假设特征之间是相互条件独立的，对输入/输出的联合概率分布进行学习；然后，基于学习到的此模型，对于给定的输入实例 x，利用贝叶斯定理对每一类别都求出后验概率，从中选择概率最大的标签 y 作为输出。朴素贝叶斯算法实现过程简单，并且有着较高的学习效率和较好的分类效果，是一种经常被选择作为基准算法的分类方法。

朴素贝叶斯算法在对模型学习之前，首先要有一个特征条件独立性的假设，即数据集的每个属性之间都是相互独立的。朴素贝叶斯也由于该较强的假设而得名。条件独立假设的具体定义如下：

$$P(X=x|Y=c_k)=P\left(X^{(1)}=x^{(1)},\cdots,X^{(n)}=x^{(n)}\middle|Y=c_k\right)$$

$$=\prod_{j=1}^{n}P\left(X^{(j)}=x^{(j)}\middle|Y=c_k\right). \qquad 2\text{-}(14)$$

在使用朴素贝叶斯算法对数据集进行分类时，首先对训练数据集进行学习得到分类模型，对给定的输入实例x，计算其后验概率，然后从所有类别的后验概率中选择最大的类作为x的类标签进行输出，后验概率的计算公式根据贝叶斯定理来进行定义，如下所示：

$$P(Y=c_k|X=x)=\frac{P(X=x|Y=c_k)P(Y=c_k)}{\sum_k P(X=x|Y=c_k)P(y=c_k)}. \qquad 2\text{-}(15)$$

将式 2-(14)代入式 2-(15)，得到朴素贝叶斯法分类的基本公式：

$$P(Y=c_k|X=x)=\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^j|Y=c_k)}{\sum_k P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)},(k=1,2,\cdots,K) \quad 2\text{-}(16)$$

根据后验概率最大化的准则，朴素贝叶斯分类器可用下面的式子来进行表示：

$$y=f(x)=arg\max_{c_k}\frac{P(Y=c_k)\prod_j P\left(X^{(j)}=x^{(j)}\middle|Y=c_k\right)}{\sum_k P(Y=c_k)\prod_j P\left(X^{(j)}=x^{(j)}\middle|Y=c_k\right)}$$

$$=arg\max_{c_k}P(Y=c_k)\prod_j P\left(X^{(j)}=x^{(j)}\middle|Y=c_k\right). \quad 2\text{-}(17)$$

（2）K 近邻算法

K 近邻算法可谓是分类算法中分类过程最简单直观的方法，实现该算法的主要思路为：在给定一个训练数据集的前提下，对于一个新的输入实例，根据某种计算距离的算法，计算出该实例与训练数据集中的每一个实例的距离，然后从中选择出与该实例最近邻的k个实例，使用少数服从多数的方法，若被选择出的k个实例的多数属于其中一个类，那么我们就用该类标签对该实例进行标记。

K 近邻算法是一种计算过程化的算法，不会产生中间模型，其本质上是对特征空间的划分，该算法有三个基本要素：k值的选择、距离度量以及分类的决策规则。

1）k 值大小的选择对算法的分类效果影响较大。如果选择的k值较小，则会导致与新输入的实例距离比较靠近的少数训练数据集中实例对类别选择有作用，容易发生过拟合现象；如果选择的k值较大，则会导致与新输入的实例有着较远距离的并且数量较多的训练集中的实例也会对类别的选择起作用，从而增大了近似误差。在实际的应用中，一般会采用交叉验证的方法来进行k的选择。

2）距离度量一般采用 Lp 距离，当 p=2 时，即为欧氏距离。例如，样本$x=(x^{(1)},x^{(2)},\cdots,x^{(n)})$与样本$y=(y^{(1)},y^{(2)},\cdots,y^{(n)})$之间的欧式距离为：

$$L_2(x,y)=(\sum_{i=1}^{n}(x^i-y^i)^2)^{\frac{1}{2}}. \quad 2\text{-}(18)$$

需要注意的是，在距离度量之前，需要将每个属性所对应的值标准化或规范化，这种做法可以避免初始值域较大的属性的权重远远大于初始值域较小的属性，而导致分类过程中产生较大的误差。

3）该算法最终分类的决策规则采用的是“少数服从多数”的原则，即由输入实例的k个最近邻的训练数据集实例中所具有的多数类别作为输入实例的类标签。

（3）决策树（Decision Tree）

决策树一般用于分类和预测。决策树从实例出发，能够从一组无规律、无次序的实例数据中推理出一种分类规则，是一种归纳学习算法。构造决策树的过程就是寻找属性和类别之间关系的过程，决策树构建好后，给定一个新的（未知类

别）实例，我们可以对其类别进行预测。它采用的方式为自顶向下的递归，不断地在所构建的决策树的内部节点进行属性的比较，然后根据比较的结果走节点向下的分枝，直至决策树的叶子节点，从而实例的类别也得以确定。

现阶段，流行的决策树算法主要有 ID3、C4.5(C5.0)、CART，同时 PUBLIC、SLIQ 和 SPRINT 等算法也经常会被研究者所使用。这些算法在选择测试属性时采用的方法都是不同的，导致所生成的决策树的结构也会有所差异；此外，决策树构建之后对树的剪枝所选择的方法及时间点也都有各自的不同之处。这里，我们从三个方面详细介绍一下本文中使用的 CART 算法。

1)特征选择。就是从训练数据集中选择对其具有最好分类能力的属性，CART 算法在选择属性过程中所采用的技术指标为基尼指数。在分类问题中，假设数据集一共有 K 个类，且p_k为样本点属于第k类的概率，则用于特征选择的基尼指数定义为：

$$Gini(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} {p_k}^2. \quad 2\text{-}(19)$$

对于给定的样本集合 D，其基尼指数为

$$Gini(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|}\right)^2, \quad 2\text{-}(20)$$

其中，K 是整个样本集中类别的个数，C_k为样本集合 D 中属于第k类的样本子集。若根据特征 A 的某一取值a可以将样本集合 D 分为两个子集 D_1 和 D_2，则 $D_1 = \{(x,y) \in D | A(x) = a\}$和$D_2 = D - D_1$。因此，考虑特征 A 对数据集的分割，数据集 D 的基尼指数可被定义为：

$$Gini(D,A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2). \quad 2\text{-}(21)$$

其中，基尼指数$Gini(D)$表示集合 D 的不确定性，基尼指数$Gini(D,A)$表示经$A = a$分割后集合 D 的不确定性。基尼指数的数值越大，样本集合的不确定性也就越大。

2）决策树的生成。采用基尼指数最小化准则，对训练数据集进行特征选择。根据给定的训练数据集，从决策树的根节点开始，递归地对每个节点进行下面的 4 步操作来构建二叉决策树：

1、计算节点的训练数据集 D 所有的特征对该数据集的基尼指数(Gini)。此时，对每一个特征 A，其取值个数可能有多个，我们对其可能取得每一个值a进行操作：根据“样本$A = a$的结果是‘是’或‘否’”将数据集 D 分割成 D_1 和 D_2 两部分，并对$A = a$时的 Gini 进行计算；

2、在对所有的特征以及特征所对应的可能取值的 Gini 计算完成后，从中选出 Gini 最小的特征作为最优特征，而该特征所对应的切分点作为最优切分点，

然后根据选出的最优特征与最优切分点从当前节点中生成两个子节点，最后根据“是”或“否”两种结果将训练数据集分配到两个不同的子节点中去；

3、对两个子节点递归地调用上面两步，直至满足停止条件；

4、生成 CART 决策树。

3）决策树的剪枝。CART 剪枝算法是从一个通过决策树算法构建的完整的决策树的底端剪去一些分枝或子树，使决策树分枝减少模型变简单，从而当输入一个新的实例时，模型可以对其进行更准确地预测。剪枝的主要工作分为两步：首先，对于已经构建好的决策树 T_0，不断地从其底端进行剪枝，直至该决策树的根节点，从而得到一个子树序列$\{T_0, T_1, \cdots, T_n\}$；然后，采用交叉验证的方式，使用验证数据集对得到的子树序列进行测试，最后根据测试结果从所有的子树序列中选择最优的子树作为最终的模型。

（4） 随机森林（Random forest）

随机森林采用了多个决策树的投票机制，改善了决策树容易产生过拟合的问题。其核心思想如下：通过自助法（Bootstrap）重采样技术，新的训练样本集是从原始的训练样本集 N 经过有放回地重复随机的方式抽取 k 个样本而得到的，然后将新的训练样本构建成 k 个分类树而组成了随机森林。对于一个新的实例，使每个分类树都对其进行预测得到分类结果，然后采用投票的方式（少数服从多数）来确定该实例的类别。随机森林算法本质上就是将多个决策树合并在一起，所抽取的每个新的训练样本都可以构建一棵分类树，每棵树的分布都是相同的。但是，每棵树对新实例的分类结果会有所差别，这是由它们的分类能力和相关性决定的。一个决策树的分类能力很小，但是当多棵树对同一个实例进行分类时，就可以在较大额程度上减少误差，其结果的准确率就会大大提升。训练过程中会随机产生大量的决策树，那么一个实例的类别可以通过每棵分类树的分类结果经过统计后来确定。

随机森林的生成方法如下：

1、从样本集中通过重采样（有放回地重复随机抽）的方式产生 n 个新样本；

2、假设样本特征数目为a，对于新产生的样本集合，从a中选择 k 个特征，建立决策树；

3、对第 1 步和第 2 步重复 m 次，最终产生 m 棵决策树；

4、对于一个新实例，从每棵决策树中得到其分类结果，然后通过多数投票机制来确定实例类别；

其中，m 是指抽取样本和生成决策树所循环的次数，n 是指一次从训练数据集中所抽取出的样本的数目，所抽取的 n 个样本构成一个新的训练样本集，通过 m 次循环抽取会产生 m 个训练样本集，从而建立 m 棵决策树组成随机森林。

（5）逻辑斯蒂（Logistic）回归

在实际应用中，我们常用的 Logistic 回归是二分类的，但是该算法也可以被用来进行多分类。与二分类不同的是，多类问题所采用的处理方法为 softmax。下面就对这两种分类模型进行介绍。

1）二项 Logistic 回归模型

二项 Logistic 回归模型是一种由条件概率分布$P(Y|X)$来进行表示的分类模型，其形式是进行参数化后的 Logistic 分布，模型中的参数是通过监督学习的方式来估计的，其中变量x取值为实数，变量y取值为 0 或 1。二项 Logistic 回归模型的条件概率分布如下：

$$P(Y=1|x)=\frac{\exp(w\cdot x+b)}{1+\exp(w\cdot x+b)}, \quad 2\text{-}(22)$$

$$P(Y=0|x)=\frac{1}{1+\exp(w\cdot x+b)}, \quad 2\text{-}(23)$$

其中，w称为权值向量，b称为偏置，$w\in R^n$且$b\in R$，它们都是参数；$x\in R^n$为模型的输入，$Y\in\{0,1\}$是输出，$w\cdot x$为二者的内积。

对于一个新的输入实例x，将其代入式 2-(22)和 2-(23)可求得$P(Y=1|x)$和$P(Y=0|x)$。通过比较所得到的两个条件概率值的大小，Logistic 回归会将实例x的类标签标记为概率值较大的那一个类。为了对参数进行更方便的表示，将权值向量和输入向量的表示形式都加以扩充，但w和x的内积仍被记作$w\cdot x$，即$w=(w^{(1)},w^{(2)},\cdots,w^{(n)},b)^T$。这时，二项 Logistic 回归模型如下：

$$P(Y=1|x)=\frac{\exp(w\cdot x)}{1+\exp(w\cdot x)}, \quad 2\text{-}(24)$$

$$\mathrm{P}(Y=0|x)=\frac{1}{1+\exp(w\cdot x)}. \quad 2\text{-}(25)$$

Logistic 回归模型的特点：一个事件发生的概率与不发生的概率的比值定义为该事件的几率。如果事件发生的概率是p，那么根据定义，该事件的几率是$\frac{p}{1-p}$，取对数之后，该事件的对数几率也被称为 logit 函数，表示如下：

$$\mathrm{logit}(p)=log\frac{p}{1-p}. \quad 2\text{-}(26)$$

对二项 Logistic 回归而言，由式 2 - (24)与式 2 -(25)可得

$$log\frac{P(Y=1|x)}{1-P(Y=1|x)}=w\cdot x. \quad 2\text{-}(27)$$

公式 2-(27)表明，二项 Logistic 回归模型是输出$Y=1$的事件的对数几率是输入实例x的线性函数表示的模型。从另一个角度来看，由于$x\in R^{n+1}$，$w\in R^{n+1}$，所以对输入实例x进行分类的线性函数$w\cdot x$的值域为实数。这里，该线性函数的

值越大，模型中的概率值就会越小；线性函数的值越小，概率值就越大。若线性函数的值接近正无穷或负无穷时，则该概率会更接近 0 或 1。

在模型的学习过程中，若所使用的训练数据集为$T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中，$x_i\in R^n$，$y_i\in\{0,1\}$，其模型参数的估计一般是采用极大似然估计法，在不断进行学习和优化后最终得到二项 Logistic 回归模型。假设有$P(Y=1|x)=\pi(x)$和$P(Y=0|x)=1-\pi(x)$，则似然函数为

$$\prod_{i=1}^{N}[\pi(x_i)]^{y_i}\,[1-\pi(x_i)]^{1-y_i}, \tag{2-(28)}$$

那么对数似然函数为

$$\begin{aligned}L(w)&=\sum_{i=1}^{N}[y_i\,log\,\pi(x_i)+(1-y_i)\,log(1-\pi(x_i))]\\&=\sum_{i=1}^{N}\left[y_i\,log\frac{\pi(x_i)}{1-\pi(x_i)}+log(1-\pi(x_i))\right]\\&=\sum_{i=1}^{N}[y_i(w\cdot x)-log(1+exp(w\cdot x))].\end{aligned} \tag{2-(29)}$$

为了得到w的估计值，我们可以对$L(w)$进行极大值的求解。

经过转换，原始问题就变成了目标函数为对数似然函数的最优化问题。在一般的应用中，我们通常采用梯度下降法也称为拟牛顿法来求解该问题。求解后，若得到$\hat{w}$为w的极大似然估计值，那么二项 Logistic 回归模型可被表示为

$$P(Y=1|x)=\frac{\exp(\hat{w}\cdot x)}{1+\exp(\hat{w}\cdot x)}, \tag{2-(30)}$$

$$P(Y=0|x)=\frac{1}{1+\exp(\hat{w}\cdot x)}. \tag{2-(31)}$$

2）多项 Logistic 回归模型

上面所介绍的 Logistic 回归模型是用于二类分类的二项分类模型，我们可以将其推广为用于多分类问题的多项 Logistic 回归模型。假设离散型随机变量Y的取值集合是$\{1,2,\cdots,K\}$，即训练数据集中所包含的类标签有多个，那么多项 Logistic 回归模型可定义为

$$P(Y=k|x)=\frac{\exp(w_k\cdot x)}{1+\sum_{k=1}^{K-1}\exp(w_k\cdot x)},k=1,2,\cdots,K-1, \tag{2-(32)}$$

$$P(Y=K|x)=\frac{1}{1+\sum_{k=1}^{K-1}\exp(w_k\cdot x)}. \tag{2-(33)}$$

其中，$x\in R^{n+1}$，$w_k\in R^{n+1}$。

类似地，二项 Logistic 回归的参数估计法也可以推广到多项 Logistic 回归。

（6）支持向量机（SVM）

支持向量机可被定义为特征空间上的间隔最大的线性分类器，是一种二类分类模型。它通过间隔最大化的学习策略，将原始问题转化为一个凸二次规划（convex quadratic programming）问题。根据训练数据集是否为线性以及是否线性可分，可以将模型分为三种：线性可分支持向量机、线性支持向量机、非线性支持向量机。

1）线性可分支持向量机

算法思想：给定一个线性可分的训练数据集，首先通过间隔最大化等价地将原始问题的求解转化为对凸二次规划问题的求解，然后经过不断地学习和训练确定参数w和b，从而得到分离超平面$w^* \cdot x + b^* = 0$，并根据分离超平面构建分类决策函数$f(x = sign(w^* \cdot x + b^*))$。

定义 1 超平面(w, b)对于训练数据集T中所有样本点(x_i, y_i)都具有函数间隔，从所有的函数间隔中选择最小值，将其定义为超平面(w, b)关于训练数据集的函数间隔，即

$$\hat{\gamma} = \min_{i=1,\cdots N} \hat{\gamma_i} = \min_{i=1,\cdots N} y_i(w \cdot x_i + b). \tag{2-(34)}$$

定义 2 超平面(w, b)对于训练数据集T中所有样本点(x_i, y_i)都具有几何间隔，从所有的几何间隔中选择最小值，将其定义为超平面(w, b)关于T的几何间隔，即

$$\gamma = \min_{i=1,\cdots,N} \gamma_i = \min_{i=1,\cdots,N} y_i \left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}\right). \tag{2-(35)}$$

根据定义 1 和定义 2 对函数间隔和几何间隔的定义，我们可以发现这两者之间存在以下关系：

$$\gamma = \frac{\hat{\gamma}}{\|w\|}. \tag{2-(36)}$$

那么，求解几何间隔最大的分离超平面的问题可以转化为下面的约束最优化问题：

$$\max_{w,b} \gamma$$

$$s.t. \left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}\right) \geq \gamma \ (i = 1,2, \cdots, N) \tag{2-(37)}$$

根据上面所述的几何间隔与函数间隔之间的关系，可将该约束最优化问题改写为：

$$\max_{w,b} \frac{\gamma}{\|w\|}$$

$$s.t. y_i(w \cdot x_i + b) \geq \hat{\gamma} \tag{2-(38)}$$

由于函数间隔$\hat{\gamma}$的取值大小并不会影响所求得的最优化问题的解，我们可以取$\hat{\gamma} = 1$。将$\hat{\gamma} = 1$代入式 2-(38)中，并且$\frac{1}{\|w\|}$和$\frac{1}{2}\|w\|^2$相对于变量 w 有着相同的单调性，所以对$\frac{1}{\|w\|}$进行最大化和对$\frac{1}{2}\|w\|^2$进行最小化是等价的。因此，最优化问题可以被转化为下面的形式:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$s.t. y_i(w \cdot x_i + b) \geq 0 \tag{2-(39)}$$

由于需要优化的目标函数是二次的，且约束条件是线性的，所以该问题是一个凸二次规划问题。我们可以通过给每一个约束条件加上一个 Lagrange multiplier（拉格朗日乘值）来求解该问题，即引入拉格朗日乘子α，那么约束条件可通过拉格朗日函数融合到目标函数中。由此，我们可构建如下的拉格朗日函数：

$$L(w,b,\alpha)=\frac{1}{2}\|w\|^2-\sum_{i=1}^{N}\alpha_i y_i(w\cdot x_i+b)+\sum_{i=1}^{N}\alpha_i. \qquad 2\text{-}(40)$$

由于拉格朗日函数具有对偶性，将该拉格朗日函数与原始问题综合起来进行分析，原始问题的对偶问题就变成了极大极小问题，如公式 2-(41)所示：

$$\max_{\alpha}\min_{w,b}L(w,b,\alpha). \qquad 2\text{-}(41)$$

所以，根据对偶问题的特点，我们需要先对$L(w,b,\alpha)$中的w和b这两个变量求极小，再对α变量求极大，从而求出对偶问题的解。将拉格朗日函数$L(w,b,\alpha)$分别对变量w和b求偏导，并令求偏导后的式子等于 0，求解后可得：

$$w=\sum_{i=1}^{N}\alpha_i\, y_i x_i. \qquad 2\text{-}(42)$$

$$\sum_{i=1}^{N}\alpha_i y_i=0. \qquad 2\text{-}(43)$$

将式 2-(42)代入拉格朗日函数 2-(40)，并利用式 2-(43)，可得：

$$\min_{w,b}L(w,b,\alpha)=-\frac{1}{2}\sum_{i=1}^{N}\sum_{i=1}^{N}\alpha_i\,\alpha_j y_i y_j\left(x_i\cdot x_{ji}\right)+\sum_{i=1}^{N}\alpha_i. \qquad 2\text{-}(44)$$

然后求$\min_{w,b}L(w,b,\alpha)$对α的极大，此问题也是对偶问题，最终求得α的解$\alpha^*=(\alpha_1{}^*,\alpha_2{}^*,\cdots,\alpha_N{}^*)^T$。

2）线性支持向量机

假定存在一个训练数据集$T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中，$x_i$ 为数据集的第 i 个特征向量，y_i 为 x_i 的类标签，并且 x_i 和 y_i 满足条件：$x_i\in X\in R^n$，$y_i\in Y=\{-1,+1\}\ (i=1,2,\cdots,N)$。在训练数据集中往往会存在一些特异点（outlier），这些特异点会导致训练数据集不可以被线性可分。但将特异点去除之后，训练数据集将变为线性可分的。

线性可分的约束条件为所有样本点到超平面的函数间隔大于等于 1，当数据集中存在的某些样本点(x_i,y_i)不满足该条件时，就意味着该训练数据集为线性不可分的。因此，我们可以引入松弛变量来对函数间隔进行调整来解决该问题，做法为：对训练数据集中每一个样本点(x_i,y_i)都引入一个松弛变量ξ_i使其满足$\xi_i\geq 0$，从而使每个样本点的函数间隔加上松弛变量之后大于等于 1。则约束条件转化为如下形式：

$$y_i(w\cdot x_i+b)\geq 1-\xi_i. \qquad 2\text{-}(45)$$

在引入松弛变量ξ_i之后，目标函数的形式发生了变化，由原来的$\frac{1}{2}\|w\|^2$变成：

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i. \qquad 2\text{-}(46)$$

其中，C 称为惩罚参数（$C>0$），C 值越大时对误分类的惩罚就越大，C 值越小时对误分类的惩罚就会越小，C 值大小的选择一般由所要解决的问题来决定。通过对目标函数的分析，我们可以采用两种方式来对其进行最小化：使间隔尽可能大，即$\frac{1}{2}\|w\|^2$尽可能小；同时，尽量减少误分类点的个数。因此，C 是调和函数间隔与误分点个数的系数。

根据上面的思路，可以通过对线性支持向量机的学习来解决训练数据集线性不可分的问题。线性可分支持向量机的间隔最大化称为硬间隔最大化，类似地，此问题被称为软间隔最大化。因此，根据支持向量机的学习来解决训练数据集线性不可分的问题通过转化可变成凸二次规划问题，如下所示：

$$\begin{aligned} &\min_{w,b,\xi}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t.\, y_i(w\cdot x_i + b) \geq 1-\xi_i \qquad 2\text{-}(47) \\ &\xi_i \geq 0 \end{aligned}$$

使用拉格朗日函数求解线性不可分问题的方法及过程可以参考线性可分支持向量机的求解方法和过程。

3） 非线性支持向量机

定义 3 通过核函数与软间隔最大化，从非线性可分的训练集中学习得到的分类决策函数

$$f(x) = sign\left(\sum_{i=1}^{N}\alpha_i^* y_i K(x, x_j) + b^*\right). \qquad 2\text{-}(48)$$

称为非线性支持向量机，其中$K(x,z)$是正定核函数。

核函数的基本思想是：通过一个非线性变换将输入空间的超平面模型转化为特征空间的超平面模型，也就是支持向量机模型，研究中常用的转化方式为从欧式空间R^n或离散集合映射到希尔伯特空间 H。数据量庞大的高维度空间的数据集分类问题同样可以使用线性支持向量机来解决。核函数有如下的定义：如果存在一个映射关系$\emptyset(x): X \to H$，使得对所有的变量$x, z \in X$，函数$K(x,z)$满足条件$K(x,z) = \emptyset(x)\cdot\emptyset(z)$，其中$\emptyset(x)\cdot\emptyset(z)$为二者的内积，则称$K(x,z)$为核函数，即为$\emptyset(x)$映射函数。

低维空间的数据常常存在着线性不可分的现象，根据模式识别理论，我们可以将低维空间的数据映射到高维特征空间来进行处理，从而实现数据集的线性可分。在映射过程中，确定非线性映射函数的参数和形式一般是较为困难的，并且很容易发生维度灾难。然而，采用核函数技术可以更加有效地解决此类问题。

常用的核函数有：

① 线性核函数：$K(x,z)=x^t\cdot z$；

② 多项式核函数：$K(x,z)=(ax^t\cdot z+c)^d$；

③ 高斯核函数：$K(x,z)=\exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$；

④ 指数核函数：$K(x,z)=\exp\left(-\frac{\|x-z\|}{2\sigma^2}\right)$；

⑤ 拉普拉斯核函数：$K(x,z)=\exp\left(-\frac{\|x-z\|}{\sigma}\right)$；

⑥ Sigmoid 核函数：$K(x,z)=\tanh(ax^t\cdot z+c)$。

（7）多标签 *K* 近邻算法(ML-KNN)

ML-KNN 的核心思想为：对于一个新的实例，通过某种距离算法从训练数据集中选择出与它最近邻的 k 个实例，得到这些被选出的实例所对应的标签集合，然后根据最大后验概率准则，我们可以确定所给定的新实例的标签集合。

新实例 x 所对应的 k 个最近邻实例的标签信息为：

$$\vec{C}_x(l)=\sum_{x_a\in N(x)}\vec{y}_{x_a}(l), l\in Y, \qquad 2\text{-}(49)$$

其中，$\vec{C}_x$是一个$1*n$的行向量，它的元素$\vec{C}_x(l)$是指对于标签l，在实例 x 的 k 个近邻的实例中拥有该标签的实例个数；$N(x)$为 x 的 k 个最近邻的实例集合，$\vec{y}_x$为 x 的标签向量，也是一个$1*n$的行向量，它的元素$\vec{y}_x(l)$若为 1，代表$l\in Y_x$，若为 0，则$l\notin Y_x$；Y_x为实例x 的标签集合，$Y_x\in Y$，Y 为数据集中所有标签的集合，标签的总数为 n。

那么，对于新的实例 x，首先得到其 k 个近邻的实例集合 $N(x)$，定义事件H_1^l为 x 有标签l，事件H_0^l为 x 无标签l，定义事件$E_j^l(j\in\{0,1,\cdots,k\})$为对于标签$l$，$k$ 个近邻实例中有 j 个包含这个标签。基于向量$\vec{C}_x$，可通过最大后验概率准则和贝叶斯定理得到：

$$\vec{y}_x(l)=arg\max_{b\in\{0,1\}}P(H_b^l)P(E^l_{\vec{C}_x(l)}|H_b^l), \qquad 2\text{-}(50)$$

其中，$\vec{y}_x(l)$为我们所要的结果，代表实例 x 是否有标签l；$P(H_b^l)$代表 x 是否有标签l的先验概率，可以用标签l在整个训练集出现的次数除以标签总次数来求出，公式如下：

$$P(H_1^l)=\frac{\sum_{i=1}^m\vec{y}_{x_i}(l)}{m}, \qquad 2\text{-}(51)$$

即样本中拥有标签l的向量数目除以向量总数。后验概率$P(E^l_{\vec{C}_x(l)}|H_b^l)$的计算方法如下：

$$P\left(E^{l}_{\vec{C}_x(l)}\middle|H^{l}_{b}\right)=\frac{c[j]}{\sum_{p=0}^{k}c[p]}. \quad 2\text{-}(52)$$

其中，$c[j]$中的 j 等于$\vec{C}_x(l)$，即 x_i 的 k 近邻实例中有标签l的个数；若x_i的 k 近邻实例中有标签l的个数为δ，且x_i也有标签l，则$c[\delta]+1$，$i\in\{1,2,\cdots,m\}$；由此，$\sum_{p=0}^{k}c[p]$为该向量有l标签，且其 k 近邻中有 0~m 个拥有标签l的向量总个数。

然后对$P\left(H^{l}_{b}\right)$与$P(E^{l}_{\vec{C}_x(l)}|H^{l}_{b})$的乘积进行计算，其中，$b$ 的取值有两种，分别为 0 和 1，若$b=1$时该乘积值最大，则向量 x 包含l标签，反之则不包含。

2.3.2 评价标准

（1）单标签多类分类效果评价

缺陷分配的单标签多类分类的目的是为每个新缺陷分配一个合适的开发者来进行修复，本文以整个测试集缺陷分配的准确率（命中率）作为方法的评价指标（用于第三章）。例如，针对每个缺陷返回 Top-k 个开发者，如果其中的任意 1 人与缺陷真实的修复者（标签）一致，就认为预测命中了有效的开发者，否则视为无效。准确率（accuracy）的计算公式为：

$$Accuracy=\frac{n}{N}. \quad 2\text{-}(53)$$

其中，n 为预测命中的缺陷个数，N 为测试集中缺陷的总数。

（2）多标签分类效果评价

缺陷分配的多标签分类的目的是为每个新的缺陷分配多个开发者，这些开发者都是对缺陷的修复有贡献的人，本文以整个测试集缺陷分配的精确率和召回率作为方法的评价指标（用于第四章）。精确率（Precision）的计算公式为：

$$Precision=\frac{1}{N}\sum_{I=1}^{N}\frac{|Y_i\cap Z_i|}{|Z_i|}, \quad 2\text{-}(54)$$

召回率（Recall）的计算公式为：

$$Recall=\frac{1}{N}\sum_{I=1}^{N}\frac{|Y_i\cap Z_i|}{|Y_i|}. \quad 2\text{-}(55)$$

其中，N 为测试集中缺陷的总数，Y 为缺陷真正的标签集合，Z 为预测出的缺陷的标签集合，| |为计数函数，例如$|Y_i\cap Z_i|$为第 i 个缺陷真正的标签集合与对其所预测的标签集合的交集的规模。

2.4 小结

本章首先对缺陷报告和缺陷的生命周期进行了较为详细地介绍，为后面缺陷的分配工作进行缺陷特征的选择和使用奠定了基础；其次详细介绍了词向量化的

方法，包括本文所使用的两种向量化方法（TF-IDF 和 Word2vec）所涉及的公式以及模型图，此外也对快速文本分类方法（FastText）进行了介绍；然后，针对两种分类问题，简单介绍了本文进行缺陷分配所涉及的机器学习算法的原理和实现步骤；最后，介绍了两种缺陷分类问题所使用的评价指标。

第三章 单一修复者推荐方法

3.1 方法框架

3.1.1 整体流程

本章提出了一种用于缺陷自动分配的修复者预测方法，该方法是一种考虑多种缺陷特征的混合型方法，针对缺陷包含的文本和缺陷来源（产品 product 和组件 component），使用词向量化（word embedding）技术和评分机制，得到开发者（与给定缺陷报告）适合度的排序，从而提高推荐的准确率。

本章所提出的用于缺陷自动分配的开发者预测方法的整体流程如图 8 所示，主要包括如下三个主要步骤：

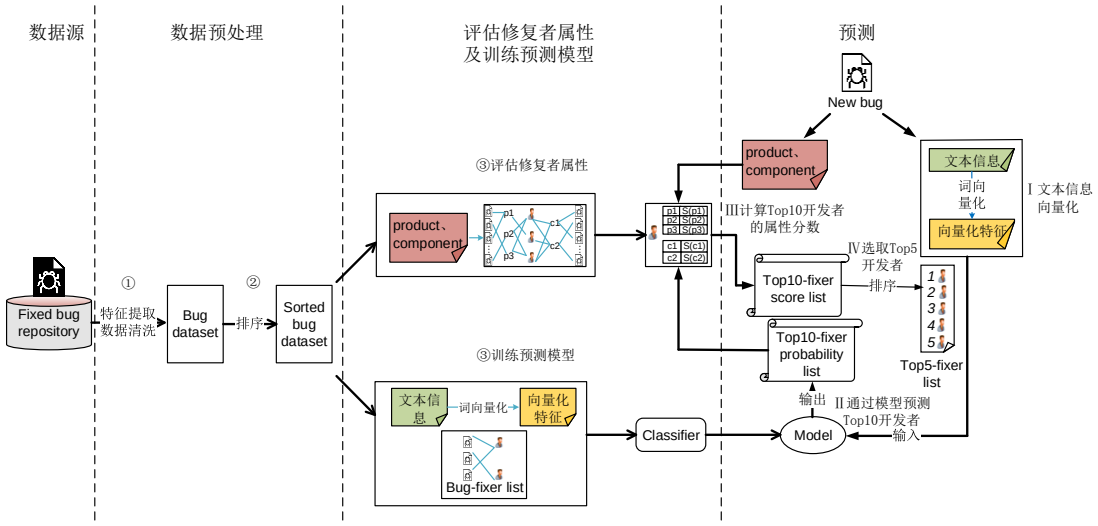


图 8 单一修复者推荐方法的整体框架图

第一步，数据预处理。针对获得的原始数据，抽取缺陷的修复时间、修复者、产品、组件、摘要、描述和评论信息，过滤掉修复者和修复时间为空的缺陷报告。将每个缺陷的摘要、描述和评论看作是一个文本文件，进行内容的清洗（如去数字、去标点符号和去非字母字符）、分词、去停用词、词干提取和（基于 Word2vec[45] 和 TF-IDF[44]（Term Frequency–Inverse Document Frequency））词向量化等自然语言处理。按照修复时间将预处理后的数据集进行排序，得到实验数据，并均分成 11 份。

第二步，“十折”增量学习（10-fold incremental learning）与验证。在第 i 轮时，前 i 份数据作为训练集，第 i+1 份数据作为测试集，并包含如下主要的步骤：

1. 训练预测模型：使用训练集中所有缺陷的词向量化的文本信息训练 ML 分类算法，得到一个基础的文本分类模型。

2. 评估修复者的属性（产品和组件）：根据训练集中缺陷的产品和组件特征，对涉及的修复者进行评分。本步骤可与预测模型训练并行执行。

3. 预测给定缺陷的修复者。为了模拟真实的缺陷分配场景，测试集中每个缺陷的文本信息都不包含评论，且处理方式与训练集的一样。利用预测模型，可计算得到测试集中每个缺陷被分配给指定开发者的概率 $P(d_i)$，返回排名前 N 的开发者及其对应的概率值。

4. 计算 Top-N 的开发者的属性分数。该分数表示开发者对不同来源的缺陷的熟悉程度。

5. 返回 Top-k 名候选者。综合考虑步骤 3 和步骤 4 的结果，根据排序函数得到 Top-k（$k < N$）名可能的修复者。

6. 计算评价指标值。

第三步，评价方法的预测效果。以 10 轮预测结果的均值来验证方法的有效性。

3.1.2 词向量化

Word2vec 是一种词向量化的方法[45]，其使用分布式向量来对文本进行表示。与传统的词向量化模型相比，Word2vec 既能解决高维稀疏特征问题，还能引入单词的语义特征。Word2vec 根据训练方式的不同可分为两种模型：CBOW（Continuous Bag-of-Words Model）和 Skip-gram。另外，TF-IDF 是一种常用的用来评估一个字或词对某个语料库中的某个文件的重要程度的统计方法[44]。

本文的词向量化使用 Word2vec 的 Skip-gram 模型。首先，使用训练集中缺陷的文本信息来构建模型的语料词典；其次，使用 Skip-gram 模型来训练训练集和测试集中缺陷的文本信息，得到所对应的文本的单词向量，其中每个缺陷的文本信息可以表示为 $t_i = <w_1, w_2, \ldots, w_j>$（$w_j$ 为文本中的单词）；然后，根据 TF-IDF 计算每个单词的重要程度，并以此作为 Skip-gram 模型中词向量的权重；最后，将所有的词向量进行加权累加，从而通过计算可以将得到文本 t_i 的向量表示为 $weight\_R(t_i)$，如公式 3-(1)所示。

$$weight_{R(t_i)} = \sum_{w \in t_i} word2vec(w) * \alpha_w. \qquad 3\text{-}(1)$$

其中，$word2vec(w)$ 表示单词 w 的 Word2vec 词向量，α_w 表示 w 的 TF-IDF 权重。

3.1.3 评分机制

缺陷的来源（产品和组件特征）可作为开发者修复技能的一个重要属性，在以往的研究中被经常使用。以产品特征为例，根据训练集中缺陷所属的产品以及缺陷对应的修复者，可以统计出每一个开发者修复属于给定产品的缺陷的个数，

以及属于给定产品的缺陷占该修复者所修复缺陷总数的比例。

具体地，假设一个开发者 d_i 修复的所有缺陷所属的产品集合为 P_{di}（$|P_{di}| = m$，即共有 m 种产品），属于每种产品 p_j 的缺陷数量记为 $N(p_j)$，该开发者修复来自给定产品的缺陷的概率计算如下：

$$s_{d_i}(p_j) = \frac{N(p_j)}{\sum_{j=1}^{m} N(p_j)}, \tag{3-(2)}$$

其中，$N()$为计数函数。类似地，该开发者修复属于给定组件（隶属于某个产品 p_j）的缺陷的概论计算如下：

$$s_{d_i}(c_k|p_j) = \frac{N(c_k)}{\sum_{k=1}^{n} N(c_k)}, \tag{3-(3)}$$

其中，n 为 d_i 修复的所有缺陷所属组件的个数。

对于一个新缺陷 b_l 和它可能的修复者 d_v，如果 $p_{bl} \in P_{dv}$，那么 d_v 在该产品特征上获得的分数（概率）为 $s_{dv}(p_{bl})$，否则为 0，如公式 3-(4)所示。类似地，d_v 在组件特征 c_{bl} 上获得的分数如公式 3-(5)所示。

$$S(p_{b_l}|d_v) = \begin{cases} s_{d_v}(p_{b_l}) & \text{if } p_{b_l} \in P_{d_v}, \\ 0 & \text{otherwise.} \end{cases} \tag{3-(4)}$$

$$S(c_{b_l}|d_v) = \begin{cases} s_{d_v}(c_{b_l}|p_{b_l}) & \text{if } c_{b_l} \in C_{d_v}, \\ 0 & \text{otherwise.} \end{cases} \tag{3-(5)}$$

其中，C_{dv} 为 d_v 修复的所有缺陷所属的组件集合。

3.1.4 候选人排序

对于一个新的缺陷报告 b，假设其所属产品为 p、所属构件为 c，使用本文所提文本分类算法预测得到的候选开发者集合为$\{d_i|1 \le i \le \text{Top-}N\}$。对于每一个候选开发者 d_i，根据缺陷所属的 p 和 c 计算该开发者在产品和组件特征上的分数，并使用公式 3-(6)所示的排序函数对候选开发者进行排序，然后按值的大小返回 Top-k 个最终的候选人。

$$R(d_i) = \alpha P(d_i) + (1-\alpha)\big(S(p|d_i) + S(c|d_i)\big). \tag{3-(6)}$$

其中，α 为权重参数（$\alpha \in [0,1]$）。

3.2 “十折交叉”增量学习模式

增量学习（Incremental Learning）的学习模式类似于人类对知识的学习能力，既能够学习新的知识，同时也可以存储以前所学到的知识。增量学习的主体一般为一个学习系统，它能够不断地从新样本或新数据中学习到新的知识，并且能够将以前已经学到的知识保存在系统中。早期的缺陷分配的方法一般是将数据集分成两部分：80%作为训练数据集，20%作为测试数据集。Bettenburg 等人已经使

用增量学习的方法（类似于机器学习中的分离样本验证技术）来进行缺陷报告重复的检测[37]。在基于增量学习的训练和验证方法（也称交叉验证，如图 9 所示）中，该算法首先收集用于实验的数据，按照时间顺序（根据缺陷报告的修复时间）将缺陷报告的数据集进行排序，然后将数据平均分成 *n* 份。在第一次实验中，第 1 份数据作为训练数据集，第 2 份数据作为测试数据集；在第二次实验中，前两份数据作为训练数据集，第 3 份数据作为测试数据集；在第 *i* 次实验的时候，用前 *i* 份的数据作为训练集，第 *i*+1 份的数据作为测试集；以此迭代下去。类似于 Bettenburg 等人的实验方式，在本文中我们取 *n*=11 来进行 10 次迭代的验证实验。

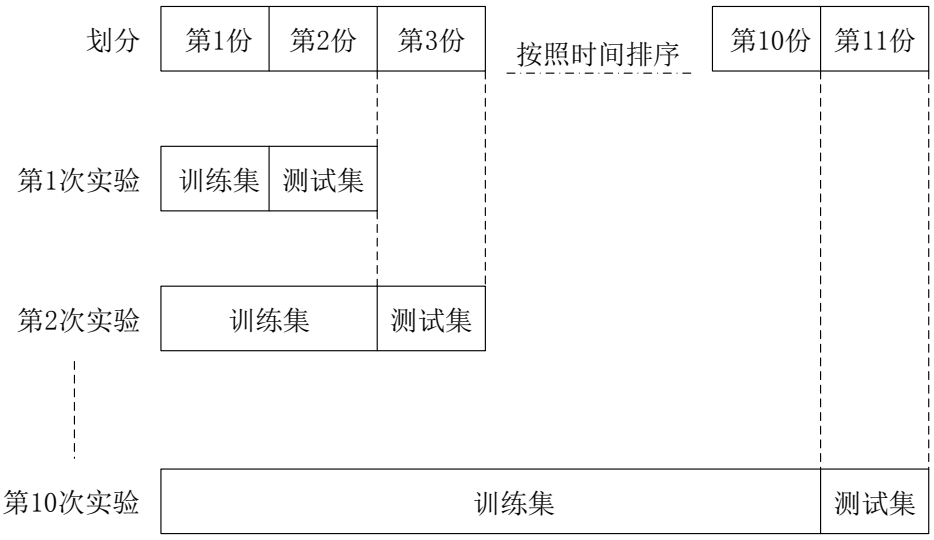


图 9 “十折交叉”增量学习模式

3.3 实验及结果分析

3.3.1 数据收集和与处理

（1）数据收集

通过 Bugzilla，本文收集了 Eclipse 和 Mozilla 两个开源软件项目的历史缺陷数据。以 Eclipse 数据集为例，首先，将缺陷的查询条件设置为“状态为 VERIFIED 且决议为 FIXED”，如图 10 所示；其次，依次搜集了（在 2001.10 至 2011.09 这 10 年期间）编号从 1 到 357573 的共 20 万条缺陷数据如图 11 所示；再次，利用爬虫工具在网站中爬下了每个缺陷所对应文本信息（摘要、描述、评论）、生命周期中状态更改的历史、所属产品及所属构件等信息。

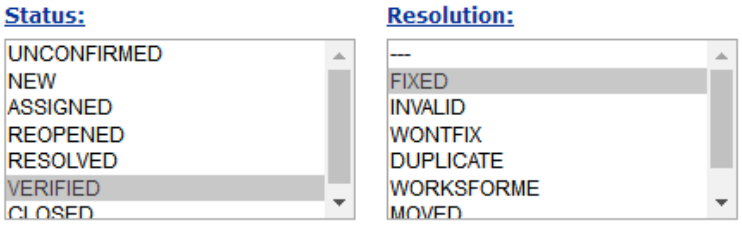


图 10 缺陷查询条件

1603	JDT	Debug	eclipse	VERI	FIXE	Evaluate actions of the Display view evaluate on empty selection (1GENSR6)
1605	JDT	Debug	darin.eclipse	VERI	FIXE	Feature: Watch List (1GEPJ0Z)
1611	JDT	Debug	Darin_Swanson	VERI	FIXE	Console preferences (1GETAZ6)
1613	JDT	Debug	Darin_Swanson	VERI	FIXE	Debugger VM launching error never displayed (1GETDGZ)
1614	JDT	UI	kai-uwe_maetzel	VERI	FIXE	Current line in Debugger should start at center of source editor (1GETH7J)
1617	JDT	Debug	Darin_Swanson	VERI	FIXE	type names not always shown in inspector (1GEULC4)
1621	JDT	Debug	eclipse	VERI	FIXE	Debugger doesn't come to front when breakpoint is hit (1GEUZEX)
1623	JDT	Debug	eclipse	VERI	FIXE	StackFrame selected but toolbar actions disable (1GEV0L7)
1628	JDT	Debug	Darin_Swanson	VERI	FIXE	README: packages appear when doing eval (1GEYFPN)
1629	JDT	Debug	jared_burns	VERI	FIXE	Terminating debug process in processes view (1GEYK0A)
1633	JDT	Debug	darin.eclipse	VERI	FIXE	Scrapbook imports
1635	JDT	Debug	darin.eclipse	VERI	FIXE	README: Attach launching not working with debug element selection (1GF5ZQC)
1636	JDT	Debug	eclipse	VERI	FIXE	Copy/paste across console docs (1GF61GB)
1637	JDT	Debug	eclipse	VERI	FIXE	Rollover icon the same as flat icon (1GF64JZ)
1638	JDT	Debug	jared_burns	VERI	FIXE	The case for IBreakpoints (1GI3JYO)
1640	JDT	Debug	darin.eclipse	VERI	FIXE	USABILITY - Inspect and Display reset selection (1GF6CH0)
1643	JDT	Debug	Darin_Swanson	VERI	FIXE	README: Scrapbook does not clean up its temp files. (1GF808S)
1646	JDT	Debug	jared_burns	VERI	FIXE	Feature: Resolution failure notification (1GF9OG5)
1648	JDT	Debug	eclipse	VERI	FIXE	Inspect, Display, and Run to line, are enabled in all editors (1GFBIAG)
1651	JDT	Debug	jared_burns	VERI	FIXE	(minor) Method entry breakpoints are shown in the javadoc comment (1GFCYC9)
1654	JDT	Debug	darin.eclipse	VERI	FIXE	USABILITY - Can't look at client server output in one perspective (1GFX2ZU)
1655	JDT	Debug	darin.eclipse	VERI	FIXE	BUG - Timing out on toString; total redraw of debug view (1GG8SQJ)
1660	JDT	Debug	Darin_Swanson	VERI	FIXE	Do not prompt for source when no source attachment
1663	JDT	Debug	darin.eclipse	VERI	FIXE	Missing toString() feature makes debugging painful (1GH3W9P)
1665	JDT	Debug	darin.eclipse	VERI	FIXE	Drop to frame hangs if after invoke (1GH3XDA)

图 11 搜集的 20 万条缺陷的部分缺陷示例

（2）文本数据预处理

每个缺陷都包含着摘要、描述和评论的文本信息，这三种特征几乎涵盖了开发者对缺陷的认知信息。因此，我们将这三种文本合并在一起进行分析，并将上述所有文本信息统称为描述文本。由此，每个缺陷报告都具有一个文本特征。对于缺陷的文本信息，我们依次进行内容的清洗、分词、去停用词、词干提取等自然语言的处理，其中，文本内容的清洗工作包括去数字、去标点符号和去非字母字符。

（3）确定缺陷的修复时间和修复者

由于我们所使用的缺陷都是已经被修复的，因此，我们可根据缺陷的状态更改历史，找出缺陷的修复时间和修复者，如图 12 所示，编号为 1667 的缺陷，其修复时间为 2002-01-13 16:29:54 EST，修复者为 Darin_Swanson。由于有些缺陷 History 中部分数据的缺失而无法查找到其修复时间或修复者，因此，我们过滤掉修复者或修复时间为空的缺陷报告。根据统计，Eclipse 数据集包含 199,960 个缺陷，共有 2,794 个修复者。类似地，Mozilla 数据集包含 219,874 个缺陷，共有 5,104 个修复者。

Back to ~~bug 1667~~

Who	When	What	Removed	Added
jeffmcaffer	2001-10-11 00:08:48 EDT	Assignee	Darin_Swanson	Darin_Wright
darin.eclipse	2001-10-12 23:27:47 EDT	CC		Darin_Swanson
Darin_Swanson	2002-01-11 20:59:15 EST	Assignee	Darin_Wright	Darin_Swanson
Darin_Swanson	2002-01-11 20:59:24 EST	Status	NEW	ASSIGNED
Darin_Swanson	2002-01-13 16:29:37 EST	Assignee	Darin_Swanson	jared-eclipse
		Status	ASSIGNED	NEW
Darin_Swanson	2002-01-13 16:29:54 EST	Status	NEW	RESOLVED
		Resolution	---	FIXED
jared_burns	2002-01-14 11:58:59 EST	Status	RESOLVED	VERIFIED

图 12 示例：编号为 1667 缺陷的 History

（4）实验环境及参数设置

本文实验所使用的计算机硬件为 DELL T5810 Precision 塔式图形工作站，其配置如下： Intel Xeon E5-1620V3 处理器，8 核 3.5GHz CPU，16GB 内存，华硕 GTX 1080。另外，所使用的计算机软件环境配置如下：Ubuntu 64 位操作系统（版本号为 16.04.3），编程语言为 Python（版本号为 2.7.12），自然语言处理工具箱 NLTK[1]（Natural Language Toolkit，版本号为 3.2.2）。

进一步地，使用 NLTK 库中的 WordPunct tokenizer()和 regexp_tokenize()方法对文本进行分词，以及使用 Porter stemmer 来提取词干。在本文的实验中，Word2vec 中的 Skip-gram 模型参数的配置如下：词向量的维度（size）为 300，最小词频阈值（min-count）为 10，当前词与预测词在一个句子中的最大距离（window）为 5，采用了 Hierarchical Softmax 进行加速（$hs=1$），高频词汇的随机降采样的配置阈值（sample）为 1e-3。这些参数是根据以往研究的经验[42],[49],[50]，并针对本文所使用的文本数据的特点以及 Skip-gram 模型的特点来赋值的。

3.3.2 实验设计

3.3.2.1 验证模式

考虑到本文研究数据的时序性，在实验中采用了类似“十折交叉”验证的增量学习模式。我们分别将按时间先后排列的 Eclipse 和 Mozilla 数据平均分成 11 份，在第 i 次实验的时候，用前 i 份的数据作为训练集（TDS），第 i+1 份的数据作为测试集（VDS）。

3.3.2.2 分类算法

选取了 NB、KNN、分类与回归树 CART、逻辑斯特回归（LR）、随机森林

[1] http://www.nltk.org/

（RF）和 LibSVM[57]六种经典的分类算法（参数值为默认设置），作为基准方法来训练开发者预测模型（MLO：ML only），相关介绍见 2.3.1 小节内容；同时，结合针对缺陷的产品和组件特征的评分机制，形成了六种符合本节方法思想的混合模型（MLS：ML+Score）。对于这六种分类算法，输入是一个（新的）缺陷向量化后的文本特征，输出是 k 个可能修改这个缺陷的开发者。

3.3.3 实验结果

针对一个给定的缺陷报告，以往研究[17],[18],[20],[28]通常推荐最多 5 个可能的开发者。表 1 和表 2 分别展示了使用基准分类算法和本文所提方法在 Eclipse 和 Mozilla 数据集上的最好结果（$k=5$）。其中，MLO 表示只用机器学习分类算法得到的结果，而 MLS 表示本文所提的方法。实验结果表明：无论使用哪种分类算法，在分类算法的基础上加上评分机制所得到的准确率都高于只使用分类算法所得到的结果，其中 NB 的提升效果最为明显。并且，对于 Eclipse 和 Mozilla 数据集，在只使用分类算法时，当 k = 5 时，LibSVM 在所有的分类算法中获得了最高的平均准确率，分别为 57.05%和 49.68%；在分类算法的基础上使用评分机制，LibSVM 仍然能获得最高的平均准确率，分别达到了 78.39%和 64.94%。进一步地，用箱线图展示了对这两个数据集进行的 10 次预测结果，如图 11 所示。

表 1 针对 Eclipse 的实验结果（α = 0.6, Top-k = 5）

算法	方法	准确率（测试集编号）										准确率	增幅
		2	3	4	5	6	7	8	9	10	11	（均值）	（%）
NB	MLO	20.79	21.54	20.49	20.43	19.02	17.28	17.02	18.42	17.81	18.17	19.10	-
	MLS	45.23	39.51	38.68	37.12	38.79	40.76	41.24	41.17	40.94	38.70	**40.21**	110.52
KNN	MLO	26.44	32.38	35.15	36.17	35.25	37.36	37.90	38.36	39.68	40.00	35.87	-
	MLS	44.32	52.69	55.08	56.72	58.45	58.83	59.34	59.94	61.04	61.81	**55.82**	55.62
CART	MLO	44.35	47.54	52.38	50.97	55.50	55.01	57.93	59.98	57.23	51.86	53.28	-
	MLS	47.17	51.26	53.76	53.94	57.23	57.28	60.16	61.47	60.71	54.39	**55.74**	4.62
RF	MLO	34.32	35.04	37.50	36.33	40.46	39.90	40.81	41.45	42.47	41.25	38.95	-
	MLS	54.75	56.98	62.53	60.89	66.57	67.42	67.81	69.19	70.45	69.17	**64.58**	65.80
LR	MLO	41.40	45.41	46.53	48.33	55.16	55.77	57.74	57.82	57.81	51.74	51.77	-
	MLS	58.46	63.15	66.76	69.00	78.27	77.34	77.56	78.49	75.52	71.67	**71.62**	38.34
LibSVM	MLO	48.43	50.23	52.26	54.33	56.32	58.34	61.24	63.23	63.93	62.23	57.05	-
	MLS	69.64	71.76	73.85	75.68	78.28	80.19	82.41	84.36	83.97	83.72	**78.39**	37.41

表 2 针对 Mozilla 的实验结果（α = 0.6, Top-k = 5）

算法	方法	准确率（测试集编号）										准确率	增幅
		2	3	4	5	6	7	8	9	10	11	（均值）	（%）
NB	MLO	13.82	9.11	12.10	13.91	13.21	14.30	14.82	15.02	15.93	15.78	13.80	-
	MLS	29.76	30.29	33.47	31.56	31.98	33.79	33.81	33.43	34.16	34.65	**32.69**	136.88
KNN	MLO	22.19	29.58	22.11	23.76	22.87	24.36	25.38	26.74	28.94	29.00	25.49	-
	MLS	32.87	41.69	32.57	33.76	35.59	34.71	35.84	35.81	35.96	37.48	**35.63**	39.78
CART	MLO	21.31	30.76	29.79	33.09	35.43	38.11	45.30	47.46	47.08	35.58	36.39	-
	MLS	24.48	33.37	33.02	36.41	38.92	40.61	47.57	49.96	50.64	40.36	**39.53**	8.63
RF	MLO	18.51	26.37	23.12	25.52	36.23	39.45	43.24	44.35	42.43	41.52	34.07	-
	MLS	33.64	40.37	37.83	40.19	50.28	53.46	56.79	57.32	56.43	54.91	**48.12**	41.24
LR	MLO	29.93	40.54	36.47	41.61	42.60	47.27	56.30	56.39	56.57	56.60	46.43	-
	MLS	41.36	51.48	47.69	51.72	57.07	62.31	73.59	70.13	69.42	71.34	**59.61**	28.39
LibSVM	MLO	43.21	44.23	46.55	48.44	49.46	50.00	51.24	54.25	55.32	54.12	49.68	-
	MLS	58.34	58.65	60.03	60.37	62.19	65.78	68.41	71.45	72.90	71.27	**64.94**	30.72

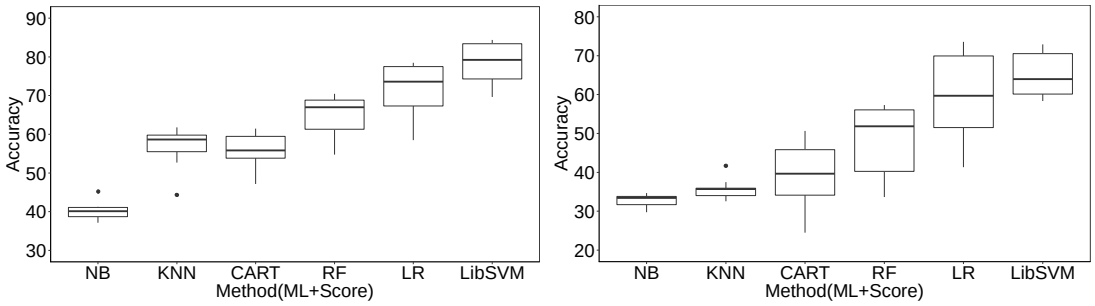


图 13 针对 Eclipse（左）和 Mozilla（右）的实验结果（框线图）

表 3 k 值变化对实验结果的影响（α = 0.6, k = 1~4）

算法	方法	平均准确率（Eclipse）				平均准确率（Mozilla）			
		k = 1	2	3	4	k = 1	2	3	4
NB	MLO	14.26	15.93	16.41	18.37	8.23	9.31	10.14	12.25
	MLS	29.76	31.25	36.43	39.87	31.38	35.49	27.13	30.74
KNN	MLO	26.75	29.69	32.05	33.47	17.61	19.34	22.46	24.77
	MLS	43.29	48.73	51.80	53.64	24.85	26.93	30.14	33.27
CART	MLO	25.35	26.41	29.08	32.84	27.26	28.55	31.69	34.74
	MLS	31.54	33.67	35.92	37.16	31.54	33.67	35.92	37.16
RF	MLO	28.68	30.05	33.42	36.15	25.35	26.41	29.08	32.84
	MLS	41.69	45.71	60.29	63.41	37.08	39.94	41.37	46.82
LR	MLO	28.68	30.05	33.42	36.15	36.05	40.32	41.58	44.16
	MLS	41.69	30.05	33.42	36.15	48.47	51.26	54.30	55.79
LibSVM	MLO	40.27	48.04	53.01	55.52	40.54	44.43	46.79	48.55
	MLS	**57.31**	**65.49**	**72.63**	**74.17**	**52.19**	**54.33**	**59.21**	**60.73**

此外，表 3 展示了在这两个数据集中 k 值变化（从 1 增长到 4）对平均准确率的影响。总的来说，推荐的开发者越多，MLO 和 MLS 的平均准确率会越高，但 MLS 的效果会比对应的 MLO 的更好。LibSVM 在所有的分类算法中也获得了最高的平均准确率。而且，当 k 值取 1 到 5 时，无论对于数据集 Eclipse 还是 Mozilla，在 α 取值为 0.6 时所得到的平均准确率最高。在求解参数 α 时，取值区间为 0 到 1，值每次增加 0.1。

同时，我们还计算了（参数设置同表 1 和表 2）只使用 LibSVM 分类算法和使用 LibSVM 加上评分机制时 Eclipse 和 Mozilla 数据集的缺陷再分配路径长度，如图 14 所示。在缺陷分配过程中，当某个缺陷无法被分配的开发者修复时，会一直在开发者之间被传递下去，直到被最终分配的开发者修复。从图中可以看出，与原始的再分配路径长度相比，虽然两种方法都能有效地缩短再分配路径的长度，但是使用评分机制的 LibSVM 方法所产生的缺陷再分配路径长度要明显小于只使用 LibSVM 分类算法的。例如，Eclipse 数据集中原始长度为 10 的再分配路径，在使用 LibSVM 分类算法后被平均降低到 5 以下，而使用本文方法后被平均降低到 3 以下。这也说明了本文方法对于缺陷修复者的预测更加有效。

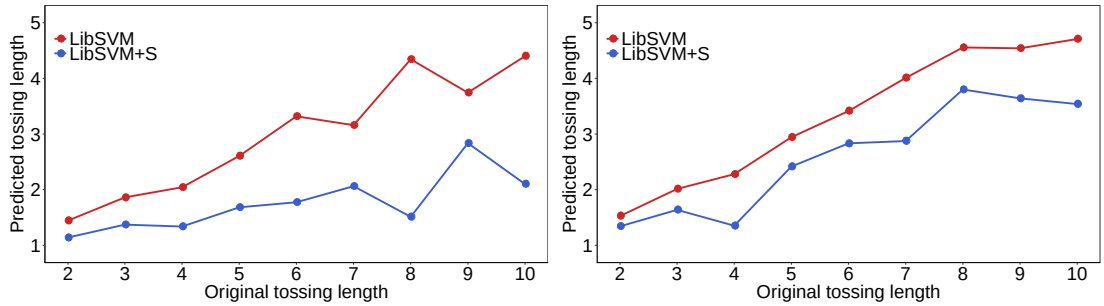


图 14 Eclipse（左）和 Mozilla（右）在 LibSVM 和 LibSVM+Score 方法下缺陷再分配路径长度的变化

3.3.4 对比分析

3.3.4.1 与 ML+TG 方法的对比分析

表 4 是在相同的数据集下，以文献[37]中的 ML+TG 作为基准方法得到的“十折”增量学习的平均结果，其中 ML 包括 NB、KNN、CART、LR、RF、SVM 这六种分类算法。由表 4 可知，Eclipse 和 Mozilla 数据集都是在使用 SVM 分类算法时的平均准确率最高，分别为 61.05%和 54.12%，而对应的本章所提方法的预测准确率分别为 78.39%和 64.94%，提升效果明显。此外，与基准方法相比，本章所提方法在 NB、KNN、CART 等其他五种分类算法下的平均准确率均有不同程度的提高，仅在使用 CART 时的提升效果不明显。另一方面，该基准方法采用的是开发者的再分配策略，而本章所提的方法为开发者的一次性分配策略，同时

不需要计算复杂的再分配图。因此，综合开发者分配的准确率以及时效性，本文所提的方法更有效。

表 4 基于 ML+TG 的基准方法结果（Top-*k* = 5）

	平均准确率（%）					
	NB	**KNN**	**CART**	**RF**	**LR**	**SVM**
Eclipse	20.40	38.21	54.45	52.43	55.73	**61.05**
Mozilla	22.43	29.75	37.51	38.89	51.18	**54.12**

与文献[37]的结果相比，基准方法在本节实验中的平均准确率要低一些。一方面，这是因为本文所使用的实验数据与文献[37]中使用的数据在规模上存在差异。虽然我们使用的数据量要小一些，但是从实验结果可以看出，使用本节提出的方法得到的平均准确率反而更高。另一方面，这也可能是因为大家使用了不同的文本处理技术或者参数配置，导致在基础的文本分类上就存在差异。

3.3.4.2 与 FastText+S 方法的对比分析

表 5 显示的是在与本章实验相同的数据集下，以 FastText+评分机制作为基准方法得到的“十折”增量学习的结果。该方法的流程为：在数据与处理之后，使用 FastText 方法训练分类模型，并采取 3.1.3 节中所提出的评分机制来评估修复者的属性（产品和组件），在预测给定缺陷的修复者时，与本章所提出的方法类似，首先使用 FastText 模型得到测试集中每个缺陷被分配给 Top-*N* 的开发者及对应概率，然后计算 Top-*N* 个开发者的属性分数，最终根据排序函数得到 Top-*k* 可能的修复者。由表 5 中的结果数据可知，Eclipse 和 Mozilla 数据集在使用 FastText+S 方法时平均准确率最高分别为 33.54%和 25.44%，而本章所提出方法的准确率分别为 78.39%和 64.94%，远远高于该基准方法的准确率，提升效果非常明显。

表 5 基于 FastText+S 的基准方法结果（α = 0.7，Top-*k* = 5）

	准确率（测试集编号）										准确率
	2	3	4	5	6	7	8	9	10	11	（平均）
Eclipse	28.84	37.69	27.34	30.17	31.26	34.13	33.79	35.86	37.72	38.58	33.54
Mozilla	6.80	27.15	12.50	22.75	24.58	24.16	27.63	32.91	40.38	35.58	25.44

3.4 小结

本章针对软件缺陷库中的缺陷报告，利用机器学习方法进行文本分类，并根

据缺陷所属的产品和组件信息制定候选开发者的评分机制，通过对缺陷修复者的预测，形成了基于文本分类及评分机制的软件缺陷自动分配方法。针对知名开源项目 Eclipse 和 Mozilla 中约 44 万个缺陷历史数据集，采用“十折交叉”的增量学习模型进行了实验，其推荐效果优于给定的多种基准方法，从而验证了本章所提方法的有效性。

第四章 多个可能开发者推荐方法

尽管在缺陷的修复工作中，只有一个开发者是缺陷的最终修复者。但是，缺陷从被提交到被修复的过程是一个开发者合作的过程。每一个相关的开发者对缺陷的修复都会贡献他们的专业知识和解决思路，并且他们还可以根据自己的合作关系以及所熟知的开发者的专业知识，将未修复的缺陷传递给其他开发者，即使所推荐的开发者不能够修复传递来的缺陷，这些开发者也可以根据自己的合作关系将该缺陷进行再次的分配，从而加快缺陷的修复速度。因此，在这个过程中，我们把这些开发者以及缺陷的最终修复者都定义为对缺陷修复有贡献的人。

在第三章的实验中，我们使用了一种基于文本分类及评分机制的缺陷分配方法，即，利用机器学习的分类算法进行文本分类并结合缺陷所属的产品和组件特征进行缺陷修复者的预测，虽然取得了较高的预测准确率，但仍然存在一部分缺陷无法对其修复者进行准确预测。这些缺陷有可能给软件维护带来很大的负面影响，但我们无法预估这些缺陷所带来的危害。虽然根据前面的工作我们不能一次性地为这些缺陷指定正确的修复者，但是可以对缺陷修复过程中所涉及的开发者进行预测，从而加快缺陷的修复工作。

一般来讲，对于一个缺陷，对其有贡献的开发者会有一群人而不止一个。因此，这种缺陷分配问题是一种多标签分类的问题，即一个缺陷会对应多个标签。根据这种思路，我们提出了混合策略来解决这种缺陷的多标签分类问题：根据开发者之间的合作关系构建缺陷分配图，针对缺陷文本信息的文本分类、所属产品和组件的评分机制以及分配概率图，提出了一种混合策略来对推荐的开发者进行排序和选择。

4.1 缺陷修复的多标签分类问题

在多标签分类学习的框架中，每个样本可能同时属于多个类别。多标签分类的核心思想为：通过对给定的多标签训练数据集进行学习，训练出多标签分类的模型，通过该模型可以有效地对未知类别的新样本所属的类别标签的集合进行预测。例如，在文本分类问题中，一篇文档可能同时与多个预先定义的主题相关。多标签分类的图形表示如下：

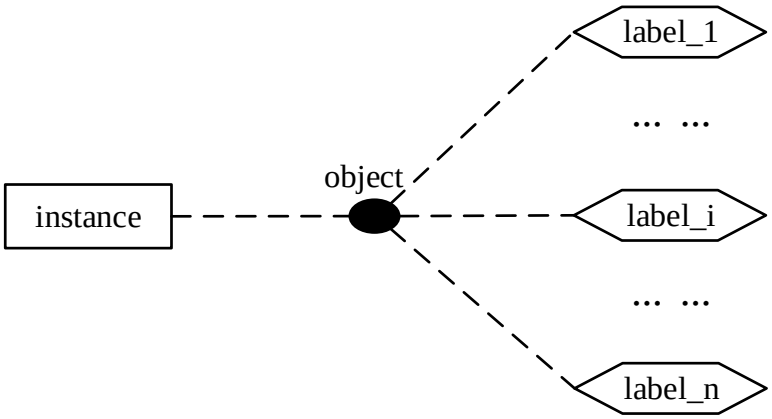


图 15 多标签分类问题示意图

在缺陷跟踪系统中，从缺陷的提交到修复是一个非常复杂的过程，在该过程中缺陷的状态不断地发生着变化，而开发者之间的分工合作正是导致缺陷状态变化的原因之一。从一个新的缺陷报告被提交开始，开发者就不断地对其进行分配直至该缺陷被修复。根据对已经被修复的缺陷的历史数据分析，我们发现，有的缺陷在被提交后能够一次性地分配给正确的修复者，但也有很大一部分缺陷中间会经过多个开发者的传递才会最终分配给正确的修复者。所以对于一个缺陷来讲，会有多个开发者对其修复做出相应的贡献，我们可以把这些开发者都定义为缺陷的标签。那么，对缺陷进行对其修复能够做出贡献的开发者的推荐工作就成为一个多标签的分类问题。

在缺陷报告中，缺陷的修改历史日志记录了缺陷从提交到最终修复的状态的改变以及开发者参与缺陷分配的过程，以下图编号为 10215 的缺陷为例进行说明。首先，该缺陷被分配给 Erich_Gamma，它的状态从“NEW”转变为“ASSIGNED”；然后，Erich_Gamma 将缺陷传递给 Kevin_Haaland，Kevin_Haaland 传递给 Eduardo_Pereira，Eduardo_Pereira 传递给 Erich-Gamma，Erich-Gamma 传递给 Kai-Uwe_Maetzel；最后，Kai-Uwe_Maetzel 最终将缺陷传递给 jdt-text-inbox。在缺陷的传递过程中，该缺陷的状态一直保持为“ASSIGNED”，直到开发者 jdt-text-inbox 将其修复。修复之后，缺陷的状态被更新为“FIXED”。由此可见，该缺陷的修复工作一共有 6 个开发者参与，即可以认为该缺陷有 6 个标签。

Back to ~~bug 10215~~

Who	When	What	Removed	Added
john.arthorne	2002-02-25 16:39:18 EST	Keywords		accessibility, performance
erich_gamma	2002-03-11 08:57:07 EST	CC		nick_edgar
		Assignee	Erich_Gamma	Kevin_Haaland
		Product	JDT	Platform
nick_edgar	2002-03-11 15:57:15 EST	Assignee	Kevin_Haaland	Eduardo_Pereira
Tod_Creasey	2002-03-14 11:54:40 EST	Keywords	accessibility	usability
eduardo_pereira	2002-05-30 17:41:59 EDT	Assignee	Eduardo_Pereira	Erich_Gamma
		Product	Platform	JDT
dirk_baeumer	2002-08-05 12:11:39 EDT	Assignee	Erich_Gamma	Kai-Uwe_Maetzel
kai-uwe_maetzel	2002-08-20 11:49:59 EDT	Component	UI	Text
kai-uwe_maetzel	2002-09-12 05:59:45 EDT	CC		Adam_Kiezun
kai-uwe_maetzel	2002-09-12 13:16:57 EDT	Assignee	Kai-Uwe_Maetzel	jdt-text-inbox
kai-uwe_maetzel	2002-09-12 13:22:23 EDT	Status	NEW	ASSIGNED
kai-uwe_maetzel	2002-12-11 11:06:24 EST	Status	ASSIGNED	RESOLVED
		Resolution	---	FIXED
		Target Milestone	---	2.1 M4

图 16 编号为 10215 的缺陷的分配过程

在 Eclipse 和 Mozilla 项目中，已修复的缺陷报告所涉及的不同规模的开发者所对应的缺陷报告数量的分布情况如下图所示：

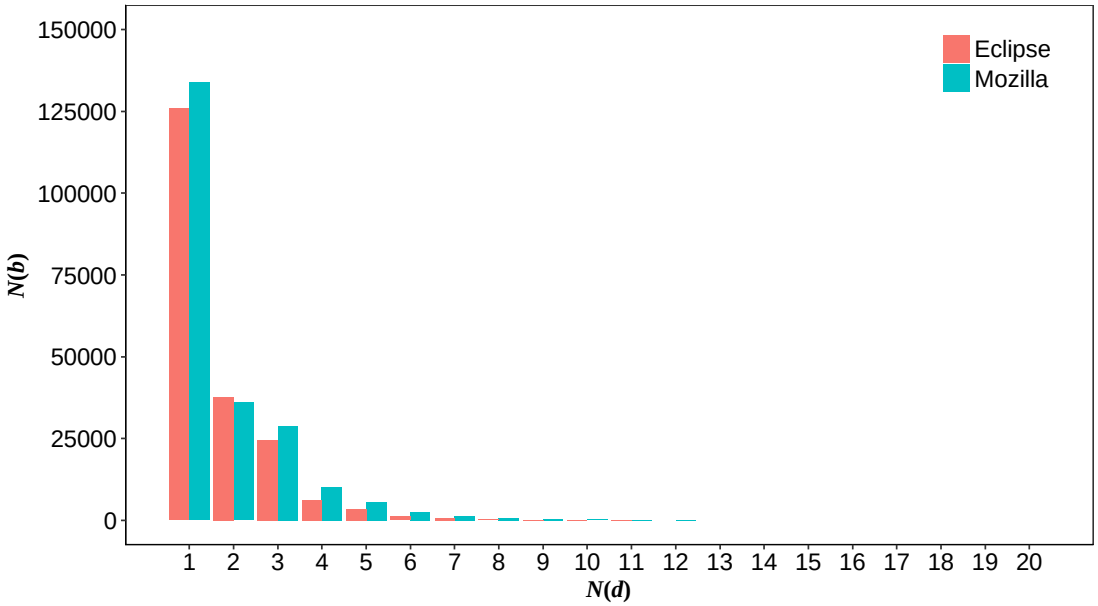


图 17 Eclipse 和 Mozilla 中不同规模的开发者所对应的缺陷数量

从图 17 的横坐标中我们可以看出，Eclipse 项目的缺陷报告在修复过程中所涉及的开发者的数量最大为 14，Mozilla 项目的缺陷报告所涉及的开发者的数量最大为 20。无论是 Eclipse 还是 Mozilla，都大约有 40%的缺陷报告对应多个开发者（开发者数量大于等于 2），并且在这部分缺陷报告中，所涉及的开发者数量为 2~10 的缺陷报告所占的比例高达 99%以上。从分析结果中我们可以看出，

在缺陷修复过程中，一般会有多个开发者参与分配和修复工作。因此，对缺陷有贡献的开发者进行推荐可以加快缺陷修复工作的进程，提高修复的效率。

4.2 方法框架

4.2.1 整体流程

第一步，数据预处理。首先从原始的缺陷报告中过滤掉修复者和修复时间为空的报告，然后抽取出修复时间、修复者、所属产品和组件、摘要、描述和评论信息，其中摘要、描述和评论组成缺陷的文本信息。文本的清洗、分词、去停用词、词干提取和词向量化方法同第三章。同样按照修复时间将预处理后的数据集进行排序，得到实验数据，并均分成 11 份。

第二步，“十折”增量学习与验证。在第 i 轮时，前 i 份数据作为训练集，第 i+1 份数据作为测试集，并包含如下主要的步骤：

1．训练预测模型。根据训练数据集中的词向量化的文本数据选择合适的机器学习分类算法，得到文本分类模型。

2．评估开发者的属性。分析训练数据集中的缺陷的所属产品和组件属性，根据所制定的评分公式，对所涉及的开发者进行评分。

3．构建缺陷分配图。根据训练数据集中的缺陷的修复历史，分析缺陷从被提交到被修复过程中开发者对缺陷的传递关系（开发者合作关系），根据所制定的分配概率的计算公式计算开发者之间的分配概率，最后根据开发者之间的合作关系以及分类概率构建缺陷分配图。此步骤可与训练预测模型以及评估开发者属性并行进行。

4．预测给定缺陷的开发者。对于新提交的缺陷报告，按照与训练数据集相同的方式处理其文本信息，根据已经训练完成的分类模型进行开发者预测，得到排名前 N 的开发者及其对应的概率值 $P(d_i)$。

5．计算 Top-N 的开发者的属性分数。

6．返回 Top-k 名候选者。根据第 4 步和第 5 步的结果使用排序函数得到 Top-k 的开发者。

7．得到新的候选开发者列表。对于 Top-k 名中的每一个候选开发者，根据缺陷分配图，选择分派概率最高的 m 个开发者加入到候选开发者列表中。

第三步，使用 precision（精确率）和 recall（召回率）评价方法的预测效果。以 10 轮预测结果的均值来验证方法的有效性。

该方法的整体框架图如下所示：

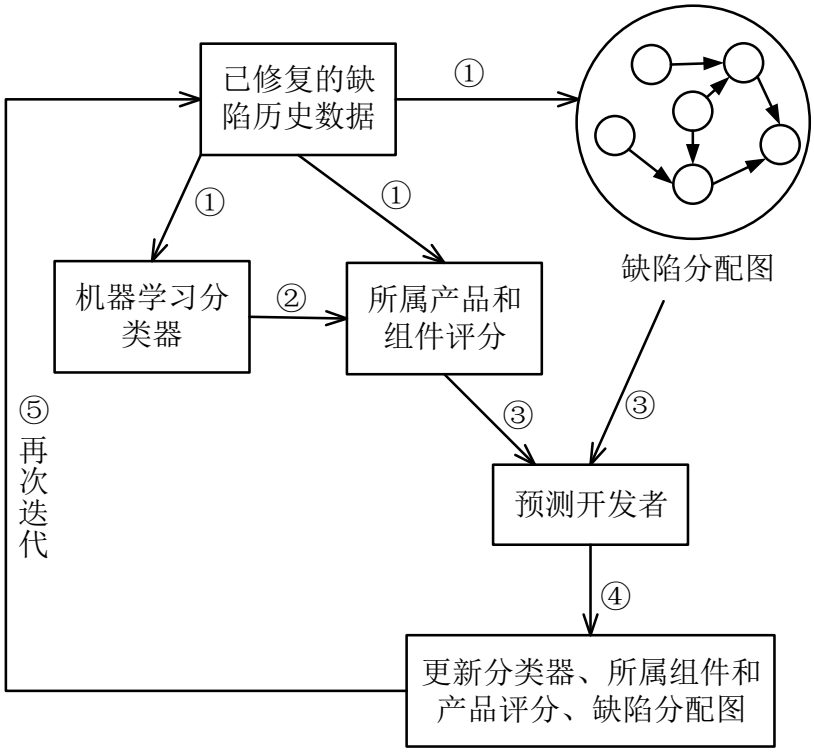


图 18 多个可能开发者推荐方法的整体框架图

其中，每次在测试数据集更新时，分类器、所属产品和组件评分以及缺陷分配图的更新过程如图 19 所示。

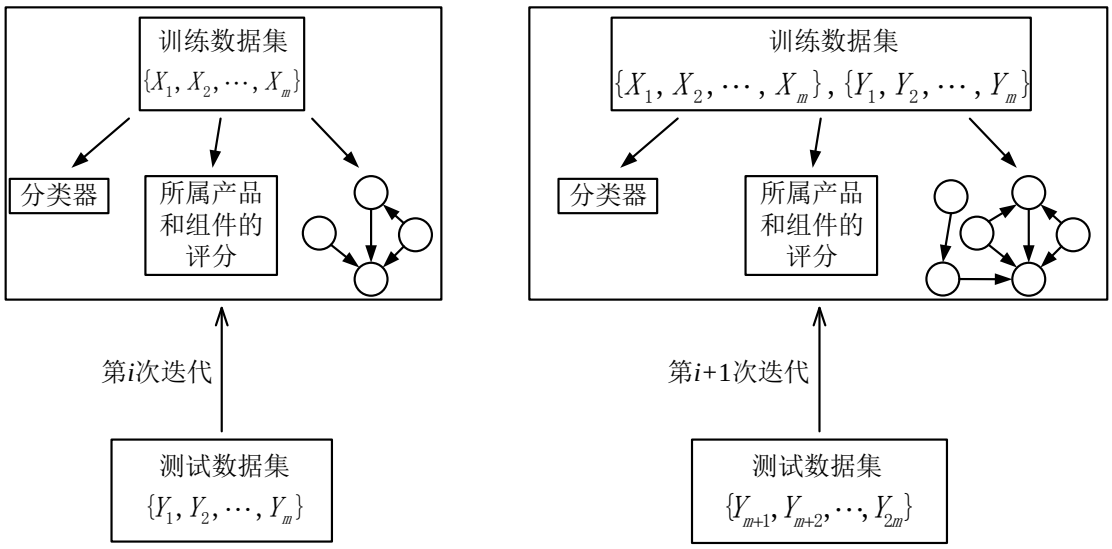


图 19 模型更新图

4.2.2 缺陷分配图

当缺陷被分配给第一个开发者并且该开发者不能将其修复时，缺陷会被再次分配给另一个开发者，若第二个开发者还不能修复该缺陷，则一直分配下去直到缺陷被修复。基于缺陷的传递路径，Jeong 等人提出了缺陷分配图的概念。缺陷

分配图是一个带有权重的有向图，它的每一个节点都代表一个开发者，其中的每一条有向边$D_1 \rightarrow D_2$表示：存在一个缺陷被分配给开发者D_1，但是该开发者不能够修复，又把缺陷分配给开发者D_2进行修复。两个开发者之间的有向边的权重是指基于缺陷分配历史数据，两个开发者之间的分配概率。我们把从开发者 D 到D_i之间的分派事件定义为$D \rightarrow D_i$，那么从开发者 D 到D_i的分配概率（也称为转换概率）可用下面的公式表示：

$$Pr(D \rightarrow D_i) = \frac{\#(D \rightarrow D_i)}{\sum_{i=1}^{k} \#(D \rightarrow D_i)}. \quad 4\text{-}(1)$$

其中，$\#()$为计数函数，如$\#(D \rightarrow D_i)$为$D \rightarrow D_i$分配路径的条数，k 为 D 所参与的缺陷分配路径上所有的开发者的个数。

缺陷分配路径

$A \rightarrow B \rightarrow C \rightarrow D$
$A \rightarrow E \rightarrow D \rightarrow C$
$A \rightarrow B \rightarrow E \rightarrow D$
$C \rightarrow E \rightarrow A \rightarrow D$
$B \rightarrow E \rightarrow D \rightarrow F$

图 20 缺陷分配路径

表 6 缺陷分配概率表

分配者	分配缺陷个数	修复者					
		C		*D*		*F*	
		#	*Pr*	#	*Pr*	#	*Pr*
A	4	1	0.25	3	0.75	0	0
B	3	0	0	2	0.67	1	0.33
C	2	-	-	2	1.00	0	0
D	2	1	0.50	-	-	1	0.50
E	4	1	0.25	2	0.50	1	0.25

在这个公式中，如果对于开发者 D，$k = 0$表明在分配图中 D 没有出边（D 是修复者或 D 没有将缺陷抛出）。在图 20 和表 6 中，我们提供了一个缺陷分配路径的例子来对分配概率的计算方式进行解释说明。例如，开发者 A 一共分配了 4 个缺陷，其中有 3 个是被 D 修复的，有一个是被 C 修复的，所以，$Pr(A \rightarrow D) = 0.75$，$Pr(A \rightarrow C) = 0.25$，$Pr(A \rightarrow F) = 0$。如果一个开发者没有参与任何缺陷的分配工作（例如：F），他不会出现在表 6 的第一列；如果一个开发者没有修复任何缺陷（例如：A），那么他不会有分配概率。图 21 展示了使用计算出的分配

概率来构建的最终的缺陷分配图。在开源软件项目的一个模块中，如果一个新的缺陷报告被提交，在默认情况下，与这个模块相关的开发者都会被加入到该缺陷的分配者列表中。

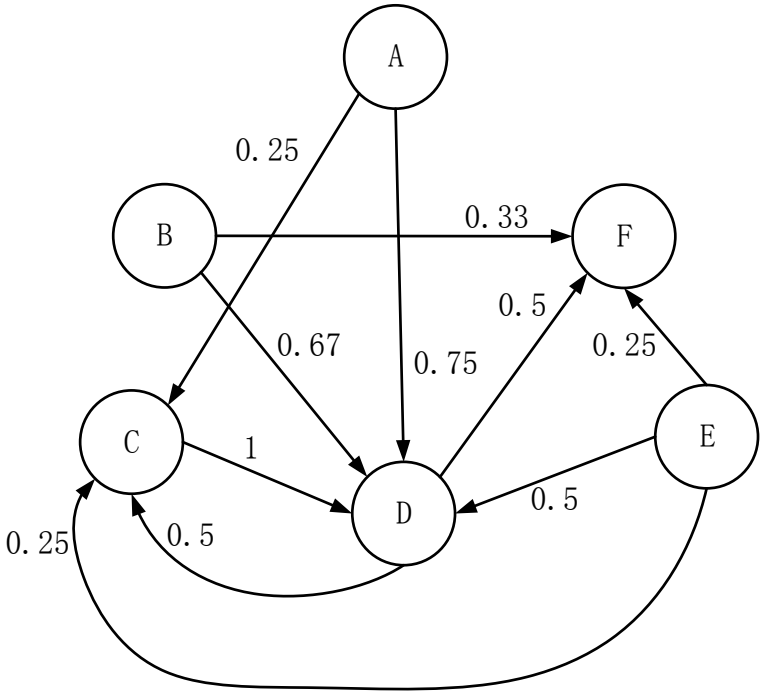


图 21 根据图 20 的分配路径所构建的缺陷分配图

4.3 实验设计

4.3.1 验证模式

本实验所使用的数据以及文本信息的处理方法与第三章相同。考虑到数据的时序性，本实验同样采用“十折交叉”增量学习模式进行验证，分别将按缺陷的修复时间先后排列的 Eclipse 和 Mozilla 数据平均分成 11 份，同样按照第三章的实验方式：在第 *i* 次实验的时候，用前 *i* 份的数据作为训练集（TDS），第 *i*+1 份的数据作为测试集（VDS）。

4.3.2 分类算法

类似于第三章的算法选择，本实验选取了朴素贝叶斯 NB、KNN、分类与回归树 CART、逻辑斯特回归 LR、随机森林 RF 和 LibSVM 六种经典的分类算法，结合针对缺陷的产品和组件特征的评分机制以及缺陷分配图，形成了六种符合本文方法思想的混合模型（ML+Score+Tossing Graphs）。其中，以 KNN 作为基本分类算法的混合模型是本实验的基准方法之一。

4.3.3 评价标准

考虑到缺陷分配的目的是为每个新缺陷分配一个开发者集合（对缺陷有贡献的开发者），本实验以整个测试集缺陷分配的召回率（recall）和精确率（precision）作为方法的评价指标。具体的计算公式可参考 2.3.2 节中的多标签分类问题的评价标准。

4.4 实验结果

表 7~表 14 分别展示了本文所提方法在 Eclipse 和 Mozilla 数据集上的结果，其中以 KNN 作为基本的文本分类方法的混合模型是本实验的一个基准方法。从结果中可以看出，除了 NB 分类算法，CART、LR、RF 和 LibSVM 无论是召回率还是精确率都高于 KNN 的基准方法；但是，当 α = 0.7、Top-k = 5、m=2 时，对于 Eclipse 的数据，NB 算法的精确率是高于 KNN 的。总的来说，LibSVM 在所有的分类算法中获得了最高的平均召回率和平均精确率：当 m=1，$\alpha = 0.6$时，所对应的召回率的平均值最高为 54.57%和 51.16%，所对应的平均精确率为 25.10%和 24.12%；当 m=2，$\alpha = 0.7$时，所对应的召回率的平均值最高为 78.20%和 71.98%，所对应的平均精确率为 20.97%和 19.03%。进一步地，用箱线图展示了这两个数据集在两个参数取不同值时的 10 次预测结果的召回率和精确率。

表 7 针对 Eclipse 的召回率（α = 0.6, Top-k = 5, m=1）

算法	召回率（测试集编号）										召回率（均值）	增幅（%）
	2	3	4	5	6	7	8	9	10	11		
KNN	37.56	40.78	41.69	43.73	44.27	45.19	46.83	47.26	49.15	49.03	44.55	-
NB	37.54	37.92	39.71	38.46	40.04	41.17	42.36	43.44	44.10	45.32	41.00	-7.97
CART	39.29	38.33	39.61	43.70	44.47	45.62	46.58	47.37	49.83	51.99	44.68	0.29
LR	40.18	43.24	44.07	46.48	47.91	48.88	49.84	51.43	52.94	55.61	48.06	7.88
RF	39.43	41.27	42.51	44.74	45.99	47.37	49.27	50.51	52.43	52.95	46.65	4.71
LibSVM	46.59	47.73	48.81	50.57	53.26	56.61	58.48	59.56	60.72	63.37	54.57	22.49

表 8 针对 Eclipse 的召回率（α = 0.7, Top-k = 5, m=2）

算法	召回率（测试集编号）										召回率（均值）	增幅（%）
	2	3	4	5	6	7	8	9	10	11		
KNN	60.28	62.41	63.72	64.25	65.74	66.93	68.56	69.15	70.07	72.48	66.36	-
NB	58.75	59.58	60.71	62.49	63.57	65.18	67.27	66.35	67.81	68.63	64.03	-3.51
CART	63.27	64.54	65.77	66.18	67.74	68.58	70.39	69.96	72.81	73.46	68.27	2.88
LR	65.44	67.35	69.24	70.62	72.38	74.57	75.21	77.44	78.83	79.41	73.05	10.08
RF	64.17	65.23	66.74	68.57	69.08	72.29	74.52	75.39	76.44	77.25	70.97	6.95
LibSVM	69.43	70.87	71.69	75.44	78.57	80.24	82.43	84.65	83.55	85.17	78.20	17.84

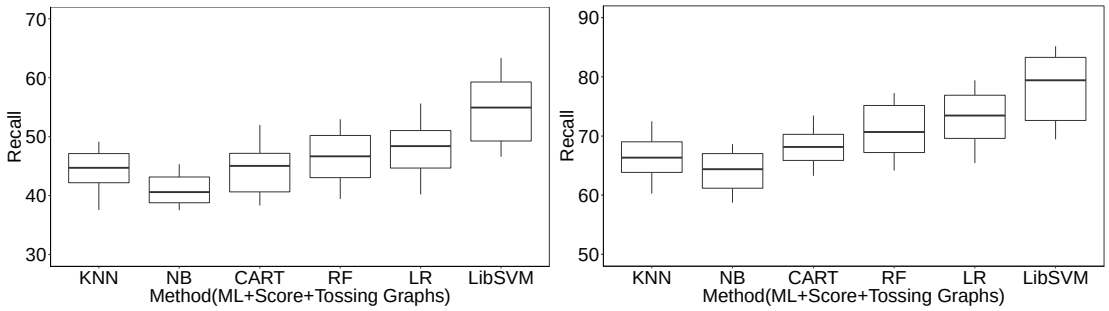


图 22 针对 Eclipse 实验结果的召回率（左：α = 0.6, Top-k = 5, m=1；右：α = 0.7, Top-k = 5, m=2）

表 9 针对 Eclipse 的精确率（α = 0.6, Top-k = 5, m=1）

算法	精确率（测试集编号）										精确率（均值）	增幅（%）
	2	3	4	5	6	7	8	9	10	11		
KNN	16.76	17.37	18.93	18.47	19.05	20.48	20.21	21.53	21.65	22.04	19.65	-
NB	15.43	16.17	16.51	18.44	18.07	19.62	20.19	21.32	21.68	21.96	18.94	-3.61
CART	16.47	17.31	18.75	19.05	19.73	20.33	21.45	22.58	23.19	23.46	20.23	2.95
LR	19.37	20.16	21.43	21.77	22.65	22.27	23.71	24.29	25.01	25.59	22.63	15.17
RF	18.74	19.37	20.45	21.31	22.29	22.76	23.53	24.61	25.72	26.14	22.49	14.45
LibSVM	20.93	21.84	23.69	24.51	24.87	25.36	26.24	27.39	28.47	27.61	25.10	27.74

表 10 针对 Eclipse 的精确率（α = 0.7, Top-k = 5, m=2）

算法	精确率（测试集编号）										精确率	增幅
	2	3	4	5	6	7	8	9	10	11	（均值）	（%）
KNN	13.24	14.36	15.79	15.03	15.94	17.21	16.57	14.33	16.76	18.34	15.76	-
NB	14.39	16.72	16.98	17.47	17.83	18.91	16.59	15.99	16.93	17.21	16.90	7.23
CART	12.45	13.74	14.05	14.83	15.71	16.82	17.17	18.39	18.75	19.13	16.10	2.16
LR	15.51	16.37	17.68	17.92	18.27	19.01	19.39	20.22	21.36	21.8	18.75	18.97
RF	15.77	15.69	16.35	18.21	19.87	20.94	19.33	20.41	21.52	22.43	19.05	20.88
LibSVM	16.79	17.32	17.44	19.59	21.67	20.45	23.76	24.82	23.69	24.15	20.97	33.06

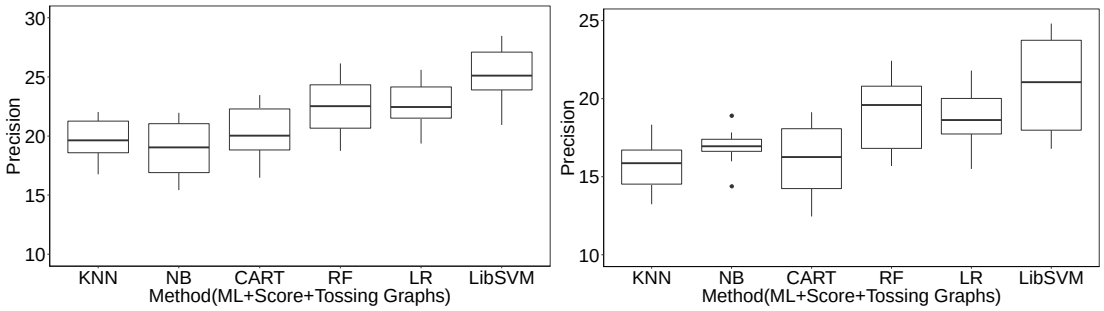


图 23 针对 Eclipse 实验结果的精确率（左：α = 0.6, Top-k = 5, m=1；右：α = 0.7, Top-k = 5, m=2）

表 11 针对 Mozilla 的召回率（α = 0.6, Top-k = 5, m=1）

算法	召回率（测试集编号）										召回率	增幅
	2	3	4	5	6	7	8	9	10	11	（均值）	（%）
KNN	37.47	38.61	39.06	41.48	42.5	43.79	44.81	46.36	47.94	46.73	42.88	-
NB	35.24	36.45	37.67	39.75	40.29	38.58	41.64	42.88	44.76	45.17	40.24	-6.16
CART	39.51	41.37	42.69	43.84	44.73	46.29	47.45	46.87	47.72	46.23	44.67	4.17
LR	41.71	43.52	45.63	44.84	45.68	49.90	48.13	49.07	50.54	52.74	47.18	10.03
RF	52.74	40.57	42.38	42.29	45.53	47.05	49.41	50.74	51.36	52.31	46.08	7.46
LibSVM	52.31	45.64	47.39	49.92	52.73	51.67	52.57	54.81	55.75	57.83	51.16	19.31

表 12 针对 Mozilla 的召回率（α = 0.7, Top-k = 5, m=2）

算法	召回率（测试集编号）										召回率（均值）	增幅（%）
	2	3	4	5	6	7	8	9	10	11		
KNN	53.74	56.35	57.27	59.09	60.18	62.33	63.65	64.37	65.26	67.81	61.01	-
NB	51.21	53.73	54.69	56.82	57.44	64.25	63.47	58.66	59.34	60.41	58.00	-4.93
CART	56.37	58.94	60.21	61.73	62.45	63.71	65.57	67.42	66.79	69.38	63.26	3.69
LR	61.27	63.86	65.94	67.47	67.16	68.46	70.23	69.04	72.85	73.39	67.97	11.41
RF	58.73	59.14	61.25	63.68	64.16	65.47	67.07	68.53	69.11	68.52	64.57	5.84
LibSVM	65.43	67.27	70.84	71.46	69.35	72.18	73.07	75.47	76.21	78.52	71.98	17.98

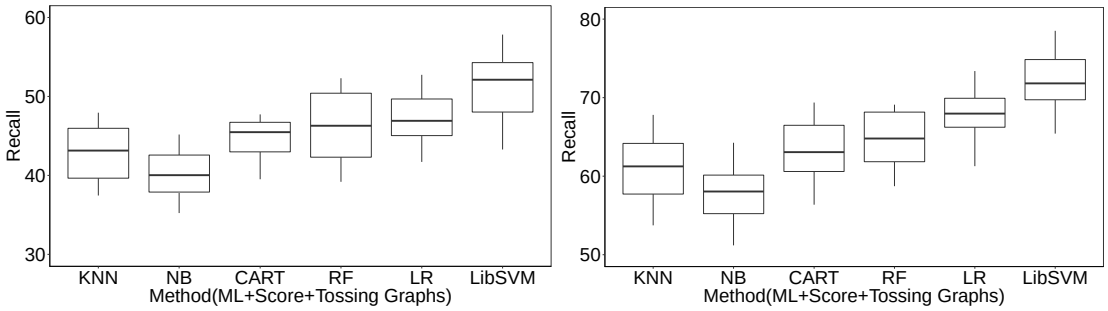


图 24 针对 Mozilla 实验结果的召回率（左：α = 0.6, Top-k = 5, m=1；右：α = 0.7, Top-k = 5, m=2）

表 13 针对 Mozilla 的精确率（α = 0.6, Top-k = 5, m=1）

算法	精确率（测试集编号）										精确率（均值）	增幅（%）
	2	3	4	5	6	7	8	9	10	11		
KNN	15.43	16.04	16.85	17.21	18.46	19.17	19.72	20.26	21.05	21.34	18.55	-
NB	21.34	14.15	14.72	15.23	16.04	17.53	17.94	18.26	19.37	20.02	16.70	-9.97
CART	20.02	17.24	17.85	18.72	19.17	20.88	21.47	23.93	22.76	23.05	20.16	8.68
LR	17.33	18.27	21.01	23.36	24.25	22.42	25.68	27.17	26.93	26.06	23.25	25.34
RF	16.21	17.34	18.92	19.57	20.28	22.46	23.71	25.87	24.64	24.93	21.39	15.31
LibSVM	18.76	19.35	21.57	22.63	24.82	25.49	27.31	26.26	27.05	27.93	24.12	30.03

表 14 针对 Mozilla 的精确率（α = 0.7, Top-k = 5, m=2）

算法	精确率（测试集编号）										精确率（均值）	增幅（%）
	2	3	4	5	6	7	8	9	10	11		
KNN	11.43	11.97	13.84	14.25	14.72	15.94	13.51	15.03	13.82	14.69	13.92	-
NB	11.37	11.04	12.51	12.83	13.04	13.68	14.72	15.17	16.23	14.56	13.52	-2.87
CART	12.53	13.76	14.27	14.05	15.33	16.14	17.23	17.97	18.19	18.45	15.79	13.43
LR	13.22	14.36	17.84	17.74	16.15	18.03	19.76	21.77	22.08	21.64	18.26	31.18
RF	12.73	13.64	15.87	17.61	14.88	18.57	19.93	20.15	21.76	22.12	17.73	27.37
LibSVM	14.73	15.26	16.07	16.48	19.65	18.71	20.16	21.44	23.06	24.75	19.03	36.71

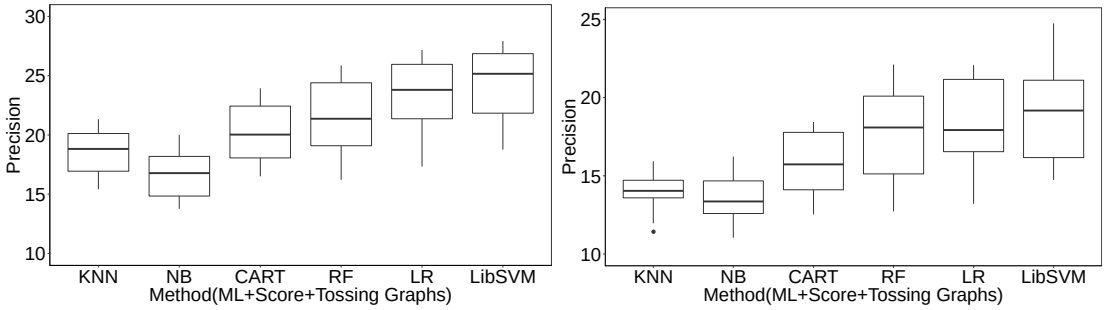


图 25 针对 Mozilla 实验结果的精确率（左：α = 0.6, Top-k = 5, m=1；右：α = 0.7, Top-k = 5, m=2）

4.5 对比分析

4.5.1 与 FastText+S+TG 方法的对比分析

表 15 为在相同的数据集下，以 FastText+评分机制+缺陷分配图作为基准方法得到的“十折”增量学习的结果。从表中可以看出，Eclipse 和 Mozilla 数据集在该方法下的平均召回率分别为 52.67%和 44.24%，平均精确率分别为 24.81%和 21.96%，而对应的本章中所提方法的平均召回率最高分别为 78.20%和 71.98%，精确率分别为 20.97%和 19.03%。从召回率的结果来看，与基准方法相比，本章中我们所提出的方法对潜在开发者的推荐效果更好，从而验证了该方法的有效性。

表 15 基于 FastText+S+TG 的基准方法结果（α = 0.7，Top-k = 5）

评价标准	数据集	测试集编号 2	3	4	5	6	7	8	9	10	11	均值
召回率	Eclipse	39.43	49.46	48.17	51.77	42.84	55.29	54.91	67.28	58.41	59.16	52.67
	Mozilla	26.93	38.24	43.46	34.11	45.75	44.87	49.38	53.12	50.69	55.81	44.24
精确率	Eclipse	20.16	22.94	21.72	24.35	27.36	23.06	25.41	27.57	28.26	27.32	24.81
	Mozilla	18.43	19.16	20.68	19.74	20.55	21.49	23.37	24.62	26.40	25.18	21.96

4.5.2 与 ML-KNN+开发者相似度方法的对比分析

4.5.2.1 ML-KNN+开发者相似度方法的整体流程

第一步，数据处理。此步骤对原始数据的处理以及向量化方法与第三章和第 4.2 节使用的方法相同，同样将数据按照修复时间进行排序然后平均分成 11 份。

第二步，“十折”增量学习与验证。该验证模式与第三章和第 4.2 节使用的模式相同，即在第 i 轮时，前 i 份数据作为训练集，第 i+1 份数据作为测试集，并包含如下主要的步骤：

1. 训练预测模型。使用 ML-KNN 多标签分类算法对训练数据集中的缺陷的文本数据及标签进行训练，得到多标签分类模型。

2. 根据分类模型预测给定缺陷的开发者。按照训练集的处理方式对测试数据集的缺陷文本信息进行处理，根据已经训练完成的分类模型进行开发者预测。

3. 为分类模型预测的开发者寻找 m 个相似的开发者。根据测试数据集中缺陷所属的产品和组件以及开发者所修复的属于不同产品和组件的缺陷的数量，为分类模型所推荐的每一个开发者寻找 m 个最相似的开发者，将分类模型推荐的开发者以及其 m 个相似的开发者都作为候选的开发者。

第三步，使用 precision（精确率）和 recall（召回率）评价方法的预测效果，并以 10 轮预测结果的均值作为最终结果。

4.5.2.2 开发者相似度计算

开发者所修复的缺陷有多种来源（属于多种产品和组件），并且缺陷的来源可以作为评估开发者技能的重要属性，每一个开发者都会有自己所擅长修复的缺陷种类。当一个开发者被分配多个属于不同产品的缺陷时，若其修复的属于某个产品 c 的缺陷数量最多，那么我们可以认为该开发者较擅长修复属于产品 c 的缺陷。同理，也适用于所属组件特征。那么，对于两个开发者，只有他们所擅长修复的缺陷的所属产品和组件均相同时，他们之间才有相似度可言。

对于开发者 d_i，其修复的缺陷所属产品的集合为 P_{di}（$|P_{di}| = m$，即共有 m 种产品），其修复的缺陷所属组件的集合为 C_{di}（$|C_{di}| = n$，即共有 n 种组件），属于每种产品 p_j 的缺陷数量记为 $N(p_j)$，属于每种组件 c_k 的缺陷数量记为 $N(c_k)$，那么其擅长修复的缺陷的所属产品和组件为：

$$p_{d_i} = arg\max_{p_j} N(p_j), \tag{4-(2)}$$

$$c_{d_i} = arg\max_{c_k} N(c_k), \tag{4-(3)}$$

其中，$N()$为计数函数。

那么，开发者 d_i 所擅长修复的所属产品和所属组件的缺陷个数占其修复的缺陷总数的比例分别为：

$$P(p_{d_i}) = \frac{N(p_{d_i})}{\sum_{j=1}^{m} N(p_j)}, \tag{4-(4)}$$

$$P(c_{d_i}) = \frac{N(c_{d_i})}{\sum_{k=1}^{n} N(c_k)}. \tag{4-(5)}$$

对于所擅长修复的缺陷的所属产品和组件都相同的开发者 d_1 和开发者 d_2，他们的相似度定义如下：

$$Sim(d_1, d_2) = \sqrt{(P(p_{d_1}) - P(p_{d_2}))^2 + (P(c_{d_1}) - P(c_{d_2}))^2}. \tag{4-(6)}$$

4.5.2.3 结果对比

表 16 和表 17 分别展示了 ML-KNN+开发者相似度方法在 Eclipse 和 Mozilla 数据集上的召回率和精确率的结果，其中，对于 ML-KNN 算法的参数 k（k 为最近邻个数），这里取 5 和 10；对于为每个算法所推荐的开发者寻找相似开发者的个数参数 m，这里取 1 和 2。从结果中我们可以看出，当 k=5，m=1 时，Eclipse 和 Mozilla 数据集的平均召回率为 44.21%和 40.21%，所对应的平均精确率为 22.73%和 22.18%；当 k=5，m=2 时，Eclipse 和 Mozilla 数据集的平均召回率为 53.29%和 50.06%，所对应的平均精确率为 20.55%和 19.65%；当 k=10，m=1 时，Eclipse 和 Mozilla 数据集的平均召回率为 51.46%和 48.26%，所对应的平均精确率为 21.01%和 20.14%；当 k=10，m=2 时，Eclipse 和 Mozilla 数据集的平均召回率为 64.03%和 58.73%，所对应的平均精确率为 18.92%和 17.67%。本章中所提出的文本分类+评分机制+缺陷分配图方法的平均召回率最高分别为 78.20%和 71.98%，平均精确率分别为 20.97%和 19.03%，在召回率方面占优，精确率也非常接基准方法的最好结果。上述结果表明，本章中我们所提出的方法能准确预测更多可能的开发者。

表 16 针对 Eclipse 和 Mozilla 的实验结果的召回率

数据集	*k*	*m*	召回率（测试集编号）										召回率（均值）
			2	3	4	5	6	7	8	9	10	11	
Eclipse	5	1	36.98	38.21	40.69	43.73	42.61	45.17	48.73	47.25	49.36	49.34	44.21
	5	2	43.46	45.37	49.78	48.82	49.26	56.48	58.74	59.89	58.73	62.34	53.29
	10	1	41.77	43.69	45.12	47.36	49.47	52.45	55.53	57.61	60.38	61.20	51.46
	10	2	50.72	51.37	54.69	59.82	62.93	67.05	69.41	72.43	74.59	77.26	64.03
Mozilla	5	1	30.15	35.43	39.37	35.72	40.69	40.21	44.83	45.49	42.46	47.72	40.21
	5	2	39.56	42.25	45.43	47.59	49.76	50.28	53.74	55.44	57.35	59.17	50.06
	10	1	38.77	40.12	43.76	45.34	46.71	48.32	50.58	54.92	56.83	57.26	48.26
	10	2	47.72	50.64	54.81	56.47	54.29	54.53	59.13	66.5	69.76	73.47	58.73

表 17 针对 Eclipse 和 Mozilla 的实验结果的精确率

数据集	*k*	*m*	精确率（测试集编号）										精确率（均值）
			2	3	4	5	6	7	8	9	10	11	
Eclipse	5	1	20.31	19.46	21.38	22.41	23.07	23.49	24.17	23.14	25.37	24.46	22.73
	5	2	18.42	18.97	19.31	20.07	20.76	21.44	22.27	22.58	20.34	21.29	20.55
	10	1	18.76	19.37	20.15	21.43	21.60	22.58	23.61	20.32	20.74	21.51	21.01
	10	2	15.49	16.81	17.35	19.4	18.87	17.14	21.36	20.17	21.05	21.52	18.92
Mozilla	5	1	19.77	20.43	21.76	21.05	23.46	25.18	24.37	22.34	21.47	21.95	22.18
	5	2	16.42	17.74	18.38	19.07	20.46	19.37	20.51	21.15	22.03	21.4	19.65
	10	1	17.19	17.83	18.07	19.14	20.35	20.72	21.47	22.26	22.64	21.73	20.14
	10	2	14.75	16.32	15.46	14.89	16.16	17.37	19.41	20.53	21.04	20.76	17.67

4.6 小结

本章的主要研究问题为缺陷修复的多标签分类问题。根据开发者在缺陷分配过程中的合作关系构建了缺陷分配图，与第三章所提出的方法相结合，形成了文本分类+评分机制+缺陷分配图的缺陷修复相关开发者推荐的混合方法。并与 FastText+评分机制+缺陷分配图以及 ML-KNN+开发者相似度方法进行对比，验证了本章所提出的方法在缺陷修复的多标签分类问题上的有效性。

第五章 总结与展望

5.1 本文的主要贡献

开源软件的质量在很大程度上影响着软件行业的发展，而软件维护是软件质量保证的重要环节。在开源软件中，软件缺陷的分派是软件维护过程中的重要工作之一，因此，开源软件中的缺陷分配的效率严重影响着开源软件的质量。在缺陷从被提交到被修复的过程中，无论是最终的修复者还是参与缺陷分派的相关开发者都对缺陷的修复工作做出了不同的贡献。因此，本文对开源软件中缺陷的修复者推荐以及参与缺陷分派的相关开发者的推荐都做了较为细致的研究。

本文的主要贡献主要有两点：

一是提出了一种基于文本分类和评分机制的单一修复者的预测方法，其核心思想是综合考虑基于机器学习的文本分类和基于软件缺陷从属特征的评分机制来构建预测模型。首先根据文本分类选出排名靠前的开发者，然后根据所制定的评分机制，对开发者进行排序，最终选出排名前 5 的开发者作为缺陷可能的修复者。在“十折”增量学习的验证模式下，通过对大型开源软件项目 Eclipse 和 Mozilla 的十万级已修复软件缺陷的实验验证了该方法的有效性。

二是提出了一种基于混合策略（文本分类、基于缺陷所属产品和组件的评分机制、缺陷分配图）的多个开发者推荐方法。根据缺陷的历史修复数据分析开发者之间的合作关系，计算分配概率，构建缺陷分配图，然后结合第三章所介绍的缺陷的文本分类和缺陷所属产品和组件的评分机制。针对 Eclipse 和 Mozilla 两个开源项目的缺陷数据，与基于 FastText+基于所属产品和组件的评分机制+缺陷分配概图以及 ML-KNN+开发者相似度两种基准方法进行对比，验证了所提方法的有效性。

5.2 下一步工作展望

通过对本文所述实验及实验结果的详细分析，我们也发现存在一些可能影响本文结论正确性（validity）的潜在威胁/局限性，主要包括：

（1）数据集与方法的通用性：本文只使用了 Eclipse 和 Mozilla 两个数据集，所提方法在其他数据集上的通用性有待验证。

（2）核心算法的选择与优化：在本文的文本分类实验中，我们只使用了 Word2vec 词向量化中的 Skip-gram 方法和六种经典的分类算法（参数均为默认配置），当然还存在效果更好的算法和更优的算法参数配置。

（3）开发者特性的考虑：在开源软件社区中，开发者是承担缺陷分配任务的主体。除了缺陷报告的各种特征，在缺陷分配时还可以考虑开发者具有的一些

特性，例如，开发者的活跃度（有些开发者退出后不再参与新缺陷的修复工作，以及在新缺陷的修复过程中会有新的开发者加入）、给定时间窗口内的工作量（当开发者被分配过多的缺陷时，可能会发生任务过载的情况，导致效率降低或者在较短时间内会快速再分配缺陷），等等。

针对上述问题，未来的工作主要包括：1）尝试更多的文本分类和自然语言处理算法，并对算法的参数不断加以优化；2）进一步挖掘缺陷报告和开发者的特征，例如，开发者的活跃度、开发者的工作量或专业知识/技能等；3）将本文方法运用在其他的开源软件项目缺陷数据集上进行验证。

参考文献

[1] Mockus A, Fielding R T, Herbsleb J D. Two case studies of open source software development: Apache and Mozilla [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2002, 11(3): 309-346.

[2] Wurster L F, Igou B, Babat Z. Survey Analysis: Overview of Preferences and Practices in the Adoption and Usage of Open-Source Software [J]. G00210068. Gartner Group, 2011: 1-28.

[3] Gallaher M P, Kropp B M. Economic Impacts of Inadequate Infrastructure for Software Testing [EB/OL]. arXiv preprint, arXiv: 7007.011, 2002.

[4] Seacord R C, Plakosh D, Lewis G A. Modernizing legacy systems: software technologies, engineering processes, and business practices [M]. Addison-Wesley Professional, 2003.

[5] Sommerville I. Software engineering [M]. New York: Addison-Wesley, 2010.

[6] Lu Y, Ma L. Bug track management system design [J]. JOURNAL-DALIAN INSTITUTE OF LIGHT INDUSTRY, 2005, 24(3): 215.

[7] Zimmermann T, Premraj R, Sillito J, et al. Improving bug tracking systems [C]// Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada: IEEE, 2009: 247-250.

[8] Bugzilla User Database, http://www.bugzilla.org/installation-list/, 2010.

[9] Zou W, Hu Y, Xuan J, et al. Towards training set reduction for bug triage [C]// Proceedings of the 35th Annual Computer Software and Applications Conference. Munich, Germany: IEEE, 2011: 576-581.

[10] Zhang T, Lee B. A hybrid bug triage algorithm for developer recommendation [C]// Proceedings of the 28th annual ACM symposium on applied computing. New York, USA: ACM, 2013: 1088-1094.

[11] Park J, Lee M, Kim J, et al. Costriage: A cost-aware triage algorithm for bug reporting systems [C]// Proceedings of the National Conference on Artificial Intelligence. California, USA: AAAI, 2011: 139.

[12] Xuan J, Jiang H, Hu Y, et al. Towards Effective Bug Triage with Software Data Reduction Techniques [J]. IEEE Transactions on Knowledge & Data Engineering, 2014, 27(1): 264-280.

[13] Increase in Open Source Growth, http://software.intel.com/enus/blogs/2009/08/04/idc-reports-an-increase-in-open-source-growth/, 2009.

[14] Canfora G, Cerulo L. Supporting change request assignment in open source development [C]// Proceedings of the 2006 ACM symposium on Applied computing. Tübingen, Germany: ACM, 2006: 1767-1772.

[15] Akila V, Zayaraz G, Govindasamy V. Bug triage in open source systems: a review [J].

International Journal of Collaborative Enterprise, 2014, 4(4): 299-319.

[16] Zhang T, Jiang H, Luo X, et al. A Literature Review of Research in Bug Resolution: Tasks, Challenges and Future Directions [J]. The Computer Journal, 2016, 59(5): 741-773.

[17] Xia X, Lo D, Wang X, et al. Accurate developer recommendation for bug resolution [C]// Proceedings of the 20th Working Conference on Reverse Engineering. Koblenz, Germany: IEEE, 2013: 72-81.

[18] Jeong G, Kim S, Zimmermann T. Improving bug triage with bug tossing graphs [C]// Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Amsterdam, The Netherlands: ACM, 2009: 111-120.

[19] Saha R K, Lease M, Khurshid S, et al. Improving bug localization using structured information retrieval [C]// Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. Silicon Valley, CA, USA: IEEE, 2014: 345-355.

[20] Wang S, Lo D. Version history, similar report, and structure: putting them together for improved bug localization [C]// Proceedings of the 22nd International Conference on Program Comprehension. Hyderabad, India: ACM, 2014: 53-63.

[21] Cubranic D, Murphy G C. Automatic Bug Triage Using Text Categorization [C]// Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering. Banff, Alberta, Canada: KSI Research Inc., 2004: 92-97.

[22] Anvik J, Hiew L, Murphy G C. Who Should Fix This Bug? [C]// Proceedings of the 28th International Conference on Software Engineering. Shanghai, China: ACM, 2006: 361-370.

[23] Anvik J, Murphy G C. Reducing the effort of bug report triage: Recommenders for development-oriented decisions [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2011, 20(3): 10.

[24] Lin Z, Shu F, Yang Y, et al. An empirical study on bug assignment automation using Chinese bug data [C]// Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement. Orlando, Florida, USA: IEEE, 2009: 451-455.

[25] Chen L, Wang X, Liu C. An Approach to Improving Bug Assignment with Bug Tossing Graphs and Bug Similarities [J]. Journal of Software, 2011, 6(3): 421-427.

[26] Zhang W, Wang S, Wang Q. KSAP: An Approach to Bug Report Assignment Using KNN Search and Heterogeneous Proximity [J]. Information and Software Technology, 2016, 70: 68-84.

[27] Friedman N, Nachman I, Peér D. Learning bayesian network structure from massive datasets: the sparse candidate algorithm [C]// Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc, 1999: 206-215.

[28] Wu W, Zhang W, Yang Y, et al. DREX: Developer Recommendation with K-Nearest-Neighbor Search and Expertise Ranking [C]// Proceedings of the 18th Asia Pacific Software Engineering Conference. Ho Chi Minh, Vietnam: IEEE, 2011: 389-396.

[29] Xuan J, Jiang H, Ren Z, et al. Developer Prioritization in Bug Repositories [C]// Proceedings of the 34th International Conference on Software Engineering. Zurich, Switzerland: IEEE, 2012: 25-35.

[30] Xuan J, Jiang H, Ren Z, et al. Automatic Bug Triage using Semi-Supervised Text Classification[J]. Software Engineering and Knowledge Engineering, 2017:209-214.

[31] Xie X, Zhang W, Yang Y, et al. Dretom: Developer recommendation based on topic models for bug resolution [C]// Proceedings of the 8th international conference on predictive models in software engineering. ACM, 2012: 19-28.

[32] Wei X, Croft W B. LDA-based document models for ad-hoc retrieval [C]// Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2006: 178-185.

[33] Naguib H, Narayan N, Brügge B, et al. Bug report assignee recommendation using activity profiles [C]// Proceedings of the 10th IEEE Working Conference on Mining Software Repositories. San Francisco, California, USA: IEEE, 2013: 22-30.

[34] Yang G, Zhang T, Lee B. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports [C]// Proceedings of the 38th Annual Computer Software and Applications Conference. Vasteras, Sweden: IEEE, 2014: 97-106.

[35] Hu H, Zhang H, Xuan J, et al. Effective Bug Triage Based on Historical Bug-Fix Information [C]// Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering. Naples, Italy: IEEE, 2014: 122-132.

[36] Yan M, Zhang X H, Yang D, et al. A Component Recommender for Bug Reports Using Discriminative Probability Latent Semantic Analysis [J]. Information and Software Technology, 2016, 73: 37-51.

[37] Bhattacharya P, Neamtiu I, Shelton C R. Automated, Highly-Accurate, Bug Assignment Using Machine Learning and Tossing Graphs [J]. Journal of Systems and Software, 2012, 85(10): 2275-2292.

[38] Tamrawi A, Nguyen T T, Al-Kofahi J M, et al. Fuzzy set and cache-based approach for bug triaging [C]// Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. ACM, 2011: 365-375.

[39] 刘海洋, 马于涛. 一种针对软件缺陷自动分派的开发者推荐方法[J]. 小型微型计算机系统, 2017, 38(12): 2747-2753.

[40] Levy O, Goldberg Y. Neural word embedding as implicit matrix factorization [C]// Proceedings of Neural Information Processing Systems 27. Montreal, Quebec, Canada: NIPS, 2014: 2177-2185.

[41] Tang D, Wei F, Yang N, et al. Learning sentiment-specific word embedding for twitter sentiment classification [C]// Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. Baltimore, Maryland, USA: ACL, 2014, 1: 1555-1565.

[42] Goldberg Y, Levy O. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method [EB/OL]. arXiv preprint arXiv:1402.3722, 2014.

[43] Ganguly D, Roy D, Mitra M, et al. Word embedding based generalized language model for information retrieval [C]// Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval. ACM, 2015: 795-798.

[44] Gan J, Chen L C. Research of improved IF-IDF Weighting algorithm [C]// Proceedings of the 2nd International Conference on Information Science and Engineering. Hangzhou, China: IEEE, 2011: 2304-2307.

[45] Mikolov T, Sutskever I, Chen K, et al. Distributed Representations of Words and Phrases and their Compositionality [C]// Proceedings of the 27th Annual Conference on Neural Information Processing Systems. Lake Tahoe, Nevada, USA: Neural Information Processing Systems Foundation, Inc., 2013: 3111-3119.

[46] Ramos J. Using tf-idf to determine word relevance in document queries [C]// Proceedings of the first instructional conference on machine learning. Piscataway, NJ, USA: ICML, 2003, 242: 133-142.

[47] Aizawa A. An information-theoretic perspective of tf–idf measures [J]. Information Processing & Management, 2003, 39(1): 45-65.

[48] Wu H C, Luk R W P, Wong K F, et al. Interpreting tf-idf term weights as making relevance decisions [J]. ACM Transactions on Information Systems (TOIS), 2008, 26(3): 13.

[49] Rong X. word2vec parameter learning explained [EB/OL]. arXiv preprint, arXiv:1411.2738, 2014.

[50] Lilleberg J, Zhu Y, Zhang Y. Support vector machines and word2vec for text classification with semantic features [C]// Proceedings of the 14th IEEE International Conference on Cognitive Informatics & Cognitive Computing. Beijing, China: IEEE, 2015: 136-140.

[51] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space [EB/OL]. arXiv preprint arXiv:1301.3781, 2013.

[52] Guthrie D, Allison B, Liu W, et al. A closer look at skip-gram modelling [C]// Proceedings of the 5th international Conference on Language Resources and Evaluation. Genoa, Italy: LREC, 2006:

1-4.

[53] Mahoney M V. Fast Text Compression with Neural Networks [C]// Proceedings of the 13th International Florida Artificial Intelligence Research Society Conference. Orlando, Florida, USA: AAAI. 2000: 230-234.

[54] Forman G, Kirshenbaum E. Extremely fast text feature extraction for classification and indexing [C]// Proceedings of the 17th ACM conference on Information and knowledge management. ACM, 2008: 1221-1230.

[55] Joulin A, Grave E, Bojanowski P, et al. Fasttext. zip: Compressing text classification models [EB/OL]. arXiv preprint arXiv:1612.03651, 2016.

[56] Zolotov V, Kung D. Analysis and Optimization of fastText Linear Text Classifier [J]. arXiv preprint arXiv:1702.05531, 2017.

[57] Chang C-C, Lin C-J. LIBSVM: a library for support vector machines [J]. ACM Transactions on Intelligent Systems and Technology, 2011, 2(3): 27.

硕士期间研究成果

[1]史小婉, 马于涛. 一种基于文本分类和评分机制的软件缺陷分配方法[J]. 计算机科学（已录用，2018 年 12 月刊）.

致谢

时光荏苒，在论文完成之际，我的硕士生涯也已经接近尾声。这三年的时光既漫长又短暂，其中有着酸甜苦辣，但更多的是收获和成长。现在心中充盈最多的是感谢和感激。感谢陪我度过这三年美好时光的各位老师和同学，正是由于你们的指导和帮助，我才能不断的克服各种困难，顺利的完成自己的学业。

首先要感谢的就是我的导师马于涛老师，马老师对我生活和学业上有着无微不至的关心和帮助，在硕士学习期间，马老师不仅传授了我做学问的技巧，还传授了我做人的准则，这些必将使我受益终身。马老师治学严谨、学识渊博，品德高尚、待人和善，在我们平时的学习中，马老师经常给我们分享一些前沿技术以及对我们的读写论文有帮助的方法和资料，并且总是鼓励我们多问问题，经常对我们进行一对一的辅导，无论是在学习上和生活上遇到问题，马老师总是很热心的帮助我们一起去解决。无论是对于自己的资格论文还是毕业论文，从课题的选择、实验的实施，直至论文的最终完成，马老师都始终给予我耐心的指导和支持。同时，马老师又是我们的“小马哥”，生活中和我们做朋友，和我们一起分享着生活中的喜怒哀乐，最喜欢的就是，只要马老师那里有零食，总是分给我们吃，喜欢那种一起吃着零食讨论问题的感觉。

同时，也要感谢课题组的王健老师、何扬帆老师、王翀老师、冯在文老师等老师们，感谢他们在学习和生活上对我的指导和帮助。还要感谢已经毕业了的三位师兄：刘海洋、尤国安、陈明明，感谢他们在学习上对我的帮助。还要感谢和我一起上下课、一起考试、一起吃饭、将要和我一起毕业的宋化志和许涛，感谢他们两个对我的关心和照顾。同时，还要感谢课题组中的师弟师妹们，他们分别是：刘君超、王枫、黄金筱、丁胜男、刘烨、黄亦薇，他们让我在硕士期间充满了开心和快乐。

感谢武汉大学，感谢您为我的成长提供一方乐土！

我也要感谢我的父母和男朋友对于我这么多年来的支持，他们的支持和关怀不仅给与我心灵上的蕴藉，还成为我坚持奋斗的不竭动力，他们让我更加乐观地面对生活中的一切，他们对我如此无私的付出，我会用一生去回报。

最后，我要向百忙之中参与审阅、评议本论文的各位老师、向参与本人论文答辩的各位老师表示由衷的感谢!

感谢所有关心帮助过我的人，祝你们一切顺利，幸福美满！