小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems

2017 年 12 月 第 12 期
Vol. 38 No. 12 2017

一种针对软件缺陷自动分派的开发者推荐方法

刘海洋,马于涛

(武汉大学 软件工程国家重点实验室,武汉 430072)
E-mail:ytma@ whu. edu. cn

摘 要: 开源软件的缺陷管理是其软件质量保障的一种重要手段,而缺陷的高效分派是大型开源软件缺陷管理的一个棘手问题. 为了提高缺陷分派的效率,本文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,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,本文所提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%,从而验证了其有效性.

关 键 词: 缺陷分派;开源社区;开发者合作网络;主题模型;混合策略
中图分类号: TP311 **文献标识码:** A **文 章 编 号:** 1000-1220(2017)12-2747-07

Developer Recommendation Method for Automatic Software Bug Triage

LIU Hai-yang, MA Yu-tao
(State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

Abstract: Software defect/bug management has become an important means of quality assurance for open source software (OSS), and efficient bug triage is a challenge to bug management in large-scale OSS projects. To improve the efficiency of bug triage, in this paper we present a simple, easy-to-use developer recommendation method for automatic software bug triage. In particular, its core is a hybrid strategy built by using LDA topic modeling (for describing developer's expertise) and a developer collaboration network (for describing the collaboration among developers) based on bug tossing. We conducted an experiment on hundreds of thousands of bugs in the Eclipse and Mozilla projects, and found that when selecting appropriate parameters and tossing strategy, the accuracies of our method for developer recommendation in Eclipse and Mozilla achieved 46.7% and 33.4%, respectively. Moreover, the accuracies of our method for developer recommendation in Eclipse and Mozilla were increased by 209.3% and 131.9%, respectively, compared with the baseline LDA + kNN method, which indicates the effectiveness of our method.

Key words: bug triage; open source community; developer collaboration network; topic model; hybrid strategy

1 引 言

经过多年的发展,许多开源软件的成熟度已经或正在逐步达到企业级应用服务的要求. 例如,截至 2015 年 11 月,全球大约 50% 的活跃网站和 37% 的高端服务器在使用 Apache HTTP Server. 凭借开放、灵活、低成本等诸多优势,开源软件对整个 IT 行业的影响也日益显著. 例如,根据 Gartner 在 2011 年发布的一项调查报告[1],超过一半的被调查公司已经采用了开源软件作为其 IT 战略的组成部分.

Linux 之父 Linus Torvalds 先生认为,Linux 成功的奥秘并不在于源代码本身,而是其开发方式. 分布在世界各地的志愿者通过互联网形成开源社区,利用版本管理工具,通过分享、交互和群体智能,进行协同开发、合作创新和同行评审,从而获得质量不断改进的程序代码[2]. 软件自身的不断优化又会吸引更多的用户和开发者,这种良性循环使得开源软件在很多应用领域都取得了成功,成为软件服务乃至云服务开发的基础.

众所周知,软件缺陷管理是开源软件质量保证的一个重要环节. 目前,大型的开源软件项目大多利用缺陷追踪系统收集缺陷信息,并协助开发者处理这些缺陷. 在软件版本不断更新的过程中,会有大量的缺陷被发现和提交,这导致缺陷的管理变得越来越困难,特别是如何自动地将缺陷分派给合适的开发者去修复[3,4]. 根据文献[5]的统计分析,Eclipse 项目平均需要 40 天将缺陷分派给第 1 个开发者,大约 100 天以后将它再分配给第 2 个开发者.

为了提高缺陷分派的效率,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自的方法. 从使用的数据、算法、再分配图(tossing graph)模型等维度,文献[6]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. 并且,考虑到混合方法在真实开源项目(如 Eclipse 和 Mozilla)中的良好效果,例如 Bhattacharya 等人提出的基于机器学习和再分配图模型的方法[7],研究者也开始尝试在这个方向上提出新的方法,例如 KSAP[8] 和 DevRec[9].

上述这类方法大多认为,虽然最终修复缺陷的开发者只

收稿日期:2016-11-01 收修改稿日期:2016-12-14 基金项目:国家"九七三"重点基础研究发展计划项目(2014CB340404)资助;国家自然科学基金项目(61272111)资助. 作者简介:**刘海洋**,男,1991 年生,硕士研究生,CCF 会员,研究方向为软件工程;**马于涛**,男,1980 年生,博士,副教授,CCF 高级会员,研究方向为软件工程、服务计算.

有一个,但缺陷分派可以看作是由多个开发者协同完成的一个过程.因此,它们通常一次性推荐一群相关的开发者,从而确保获得较高的召回率(recall).与这种思路不同,我们认为相关的开发者虽然在找到最终修复者的过程中发挥了作用,但对于缺陷修复所起的作用是有限的,而非决定性的.因此,推荐的目标应更加明确,即最终的缺陷修复者;同时,为了提高推荐的准确率(accuracy),也可以依次推荐多个候选开发者.这是一种典型的知识服务中专家推荐的思路.

遵循该思路,我们提出了一种用于缺陷自动分派的开发者推荐方法,其主要技术贡献如下:

1)提出了一种基于简单混合策略(主题+度)的开发者推荐方法,一方面利用 LDA 模型分析开发者修复缺陷的主题分布,另一方面利用通过再分配关系构建的开发者合作网络分析开发者(在合作关系网中)的重要性;

2)针对开源项目 Eclipse 和 Mozilla 的缺陷数据,与基于 LDA+kNN 的基准方法进行了对比,验证了本文所提方法的有效性.

2 相关工作

2.1 基于文本内容的方法

根据文献[4,6]的总结,缺陷报告文本的内容,包括预定义字段、摘要及描述信息、开发者对缺陷的评论、历史修复记录等,是决定分派对象的一个重要因素.因此,早期的研究侧重于使用传统的文本分类方法来预测潜在的缺陷修复者[10,11],即,从缺陷的文本内容中抽取关键字和开发者的信息,利用机器学习技术训练分类模型(如朴素贝叶斯和支持向量机 SVM),然后在测试集中检验其预测效果.例如,针对 Eclipse 项目的小规模数据,Cubranic 等人训练了一个朴素贝叶斯分类器(classifier)[10],其准确率达到了 30%.

虽然这些方法在小数据集上验证了其可行性,但在处理更大规模项目时却常常会遇到缺陷的标注困难和语义缺失问题.为了解决该问题,研究者开始尝试引入主题模型的方法(如 LDA[12], Latent Dirichlet Allocation)对文本进行分类[13-17];并且,针对 Eclipse、Mozilla 等大型项目的实验结果表明,这些方法确实能提高开发者推荐的效果.例如,Xie 等人提出了一种基于主题模型 LDA 的开发者推荐方法 DRETOM[15],针对 Eclipse JDT 和 Mozilla 的实验结果表明,当推荐排名前 5 位和前 7 位的开发者时,其召回率分别达到了 82% 和 50%.

有鉴于此,本文也采用 LDA 模型对缺陷报告进行分析.但与上述方法不同的是,在计算完某个缺陷属于 K 个主题的概率后,不再只选取概率最大的那个主题,而是根据阈值选择 Top-k 个主题,和这个缺陷建立映射关系,然后计算开发者和主题的概率分布.

2.2 基于开发者关系的方法

另一方面,缺陷分派过程中的再分配关系蕴含了开发者之间在修复缺陷时的重要合作信息,由此构造的开发者合作网络(或再分配图)同样被用来辅助进行开发者推荐.Jeong、Bhattacharya 和 Wang 等人之前的工作[5,7,18]也证实,除了缺陷的文本内容,进一步考虑开发者之间的再分配关系,能构造出更好的预测模型,这也催生了近年来(内容+关系)混合类型方法的兴起.例如,Jeong 等人较早利用马尔可夫链构造了一个缺陷再分配图模型[5],在 44.5 万个缺陷报告上的实验结果显示,缺陷的再分配事件数减少了近 72%,而预测准确率比基准方法提高了近 23%.

此外,Wu 等人分析了一些经典的社交网络指标,如入度、出度、度、PageRank、介数(betweenness)和接近性(closeness),对开发者合作网络中开发者技能排名的影响[19].针对 Mozilla 项目的实验结果表明,所提的方法 DREX 在使用出度时的效果最好,当推荐排名前 10 位的开发者时,其召回率可达到 60%.相似地,Xuan 等人也发现了同样的结论,即,出度越大的开发者在排序过程中会获得更高的优先级[20].上述结论为我们进一步研究大规模的开发者合作网络提供了基础.

2.3 混合类型方法

这类方法的实现主要包括两个步骤:首先,根据以往的缺陷修复记录抽取关键字/主题和最终修复者信息,训练出一个分类器,针对给定的缺陷,推荐 Top-k 个开发者,如文献[5,7,20];或者,根据给定的缺陷,搜寻与之最相似的 K 个已修复缺陷,将其中涉及的开发者作为候选人,如文献[18,19].然后,根据构建的开发者合作网络,考虑开发者的链接关系(如再分配、共同评论、邮件通信等)、工作归属(项目或构件)[21,22]等因素,进一步完善和精化上一步得到的结果.

与上述的以往工作不同,最近的研究[8,9]开始尝试将评分机制引入到开发者的排名计算中,即,针对开发者在不同因素上的表现计分(或概率),并对不同的因素赋予相应的权重,然后对潜在的开发者进行排序,再推荐排名靠前的开发者.在 Eclipse、Mozilla 等开源项目的实验结果表明,它们能提升基准方法在缺陷分派方面的召回率.

3 背景知识

3.1 基本概念

1)缺陷分派(Bug triage):在一个缺陷报告被提交并确认后,分派者首次指定一个开发者来进行修复的过程.

2)分派者(Triager):当接收到提交的缺陷报告时,首先验证该缺陷的有效性并负责将其分配给合适的开发者来修复的管理人员.分派者本身也是开发者,同时也可能是缺陷的修复者.

3)开发者(Developer):对于一个开源软件项目,泛指在某个缺陷的修复过程中涉及的所有参与者.并且,这些人员往往也是传统意义上的软件开发人员.

4)修复者(Fixer):最终修复某个缺陷的开发者.

5)缺陷再分配(Bug tossing):一个缺陷被分派给一个开发者后,如果该开发者不能修复,就将该缺陷再分配(或称为传递)给另一个开发者去修复的过程.

6)缺陷再分配的开发者对(Pair of developers in a bug tossing):一次缺陷再分配关联的两个开发者,且该缺陷由其中一人再分配给另外一人.

7)缺陷再分配路径(Bug tossing path):缺陷再分配过程涉及的所有开发者所构成的一个序列,包含多个缺陷再分配的开发者对.例如,一个缺陷首先分派给开发者 A 来修复,若 A 不能修复则再分配给开发者 B.同样地,若 B 也不能修复就继续再分配下去,直至传给 E.假设 E 最终修复了这个缺陷,

那么这个过程涉及的开发者所形成的序列 A→B→C→D→E,就是一条缺陷再分配路径.

3.2 缺陷报告及缺陷生命周期

大多数的大型软件项目都是利用缺陷追踪系统来管理缺陷的修复过程,这也使得项目的维护更加方便. Bugzilla 就是其中一款十分流行的开源缺陷追踪系统,它可以管理软件开发中缺陷的提交、确认、指派、修复、结束等整个生命周期的各种状态.

Bugzilla 中的软件缺陷报告主要包括三类信息:第一类是用于定义缺陷基本信息的预定义字段,例如,状态(status)、所属产品(product)、所属构件(component)、摘要(summary)、报告者(reporter)等;第二类是缺陷报告的文本描述信息,包括描述(description)和评论(comments),这是以往工作做文本分类的主要信息来源;第三类是缺陷的状态更改历史(history),其中的每条历史记录由 5 个属性"谁、何时、干什么、移除、新增"(who、when、what、removed、added)构成,用来表明"什么人在什么时间干了什么事",其中"removed"和"added"是针对"what"进行的相关操作,如图 1 所示. 根据这些历史记录,我们从中抽取出缺陷再分配的开发者对,然后构建缺陷再分配路径.

Back to bug 11030

Who	When	What	Removed	Added
darin.eclipse	2002-03-08 11:03:28 EST	Assignee	Darin_Wright	jared-eclipse
darin.eclipse	2002-03-08 11:03:35 EST	Status	NEW	ASSIGNED
jared_burns	2002-03-08 17:20:42 EST	Assignee	jared-eclipse	Darin_Wright
		Status	ASSIGNED	NEW
jared_burns	2002-03-08 17:21:22 EST	Status	NEW	RESOLVED
		Resolution	-	FIXED
darin.eclipse	2002-03-12 10:40:42 EST	Status	RESOLBED	VERIFIED

图 1 编号为 11030 的缺陷报告的修改历史

Fig. 1 An example of the history of a bug report (No. 11030)

根据缺陷的处理状态,Bugzilla 管理的缺陷报告主要分为两类:一类是"关闭"的缺陷(closed bugs),即已经被修复或者目前暂时不会再被处理的缺陷;另一类是"开放"的缺陷(open bugs),即正在处理中的缺陷,但其状态会随时发生改变,不利于进行分析. 因此,相关研究工作针对的还是已处理完成的这类缺陷.

总的来说,在 Bugzilla 中第一类缺陷报告都会有一个完整的生命周期:初始提交时处于"未确认"(unconfirmed)状态,在被确认有效后标记为"新提交"(new)状态;当被分派给相关开发者进行处理后,其状态变为"已指派"(assigned),在该状态时缺陷可进行再分配;在被修复(或确认无效、重复、无法再现、不可修复)后,进入到"已解决"(resolved)状态;如果确实被修复并通过了管理员的确认(verified),就进入到最终的"关闭"状态. 当然,因为各种原因有一些缺陷报告也会在"已解决"和"关闭"状态时,转入到"重启"(reopen)状态,从而进入到新的生命周期. 更详细的中文介绍,请参考文献[23].

4 基于混合策略的开发者推荐方法

4.1 整体框架

本文提出的针对软件缺陷自动分派的开发者推荐方法主要分为四步,其整体框架如图 2 所示.

第 1 步. 数据预处理. 从爬虫获取的原始数据经过预处理后,得到缺陷的历史更改记录集和缺陷的文本描述集,构成本文研究的两大数据源.

第 2 步. 构建开发者合作网络. 根据缺陷库所有缺陷的历史更改记录,对于每一个缺陷,我们可以从中抽取出一定数量的缺陷再分配的开发者对. 根据所有缺陷的开发者对集合 Tossing Pair List,可以构成针对缺陷分派的开发者合作网络(具体表现形式为有向图或者无向图).

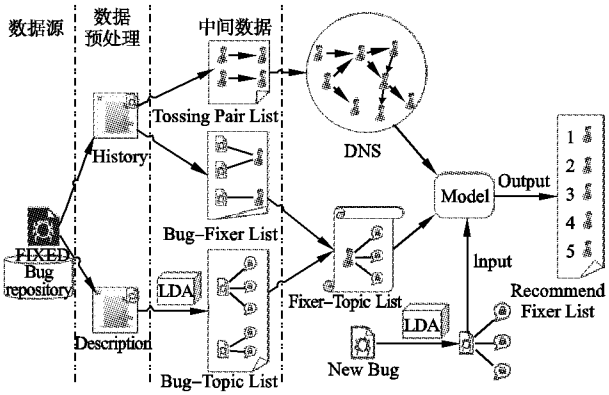

图 2 方法的整体框架

Fig. 2 Framework of our approach

第 3 步. 开发者修复能力建模. 通过给每个缺陷打标签,即确定最终修复缺陷的开发者是谁,得到中间结果 Bug-Fixer List;利用 LDA 对缺陷主题建模,根据设定的阈值,将每一个缺陷对应到一个或者多个主题,得到中间结果 Bug-Topic List;结合这两个相关联的中间结果,建立开发者与主题之间的映射关系 Fixer-Topic List. 这种作为模型重要部分的映射关系,实际上是对开发者修复能力的描述.

最后,为新缺陷推荐合适的开发者. 当一个新的缺陷到来时,根据获得的 LDA 主题模型,利用新缺陷的描述信息,可以推断出其主题概率分布,从而确定它的主题类别,作为下一步的输入. 然后,参考开发者修复能力快照及开发者合作网络,选择推荐策略,依次推荐一定数目的潜在开发者.

4.2 开发者再分配策略

再分配策略对推荐的准确率有直接影响,本文提出了 3 种不同策略,分别是基于度、基于主题以及二者混合的策略.

4.2.1 再分配策略描述

1)基于度的策略

把每个开发者作为节点,两个开发者之间的传递关系看成一条边,从而得到开发者合作网络. 这个网络既可以是有向图,也可以是无向图.

在有向图中,节点有入度和出度之分. 开发者节点的入度越大,即开发者接收到其他开发者传递给他的缺陷数目越大,可能意味着该开发者修复缺陷能力越强、越有知名度;开发者的出度越大,意味着遇到修复不了的缺陷时有更多的缺陷再传递人选. 选择出度/入度大的节点作为缺陷传递的目标,似乎都有利于缺陷的修复. 所以,当一个开发者遇到修复不了的缺陷的时候,倾向于传递给网络中与之相邻且入度、出度最大的开发者节点.

在无向图中,基于同样的思想,我们考虑的是节点的度.

缺陷分派时,上一个节点每次传递给度最大的下一个节点.

2)基于主题的策略

根据开发者修复的缺陷历史记录,对这些缺陷的主题建模,得到开发者与主题之间的映射关系,最终体现为开发者-主题概率矩阵.根据这个矩阵可知,某一个开发者修复某种主题类别的概率.当开发者修复不了某个缺陷(根据 LDA 可知其主题类别)时,把开发者合作网络中与之相邻的节点作为候选集,结合开发者-主题概率矩阵,选择这种主题类别的概率最大的开发者作为其传递的对象.

3)基于度和主题的混合分配策略

综合考虑以上两种策略,理想条件下,每次选择候选开发者时,都要满足两个条件:1)(出/入)度最大;2)修复主题缺陷的数目和概率最大.但是当这两个条件冲突时,这种策略又细分为以度为主和以主题为主的两种混合策略.

冲突时的处理情形:假设与 A 相邻的开发者有 B 和 C,若 B 的(出/入)度最大,而 C 修复主题缺陷的数目和概率最大.优先满足条件 1)传递给 B,叫以度为主的混合策略,优先满足条件 2)传递给 C,叫以主题为主的混合策略.

4.2.2 混合策略实现算法

首先,给出用 Java 语言描述的开发者的缺陷修复能力,在本文中表示为一个类.

```
Class Developer{
String name;          //开发者名字
Boolean isFixer;   //是否是修复者
LinkedHashMap < String,Double >  topicMap;   //修复的所有缺陷的主题概率分布
int fixedCount;//修复的缺陷数
String nodeNo;   //在开发者网络中的编号
int degree;       //度
ArrayList < Developer >  adjoint;   //与之相邻的节点列表
}
```

然后,给出算法的伪代码描述.

算法 1. 基于混合策略的开发者推荐

输入: 待分配缺陷的主题编号 *topicId*、开发者集合 *depList*

输出: 最合适的修复者 *candidate*

```
01. BEGIN
02.     candidateList = ∅,candidate = null;
03.     FOR EACH Developer d in depList
04.       IF d. topicMap contains topicId
05.          candidateList = candidateList∪{d};
06.       FOR EACH Developer f in candidateList
07.          IF f. fixCount * f. Pro(topicId)  >
08.             candidate. fixCount  * candidate. pro(topicId)
09.             && f. pro(topicId)  >  candidate. pro(topicId)
10.             && f. degree > candidate. degree
11.                candidate = f;
12.       END FOR
13.     END FOR
14.     RETURN candidate;
15. END
```

4.3 方法中其他重要步骤的实现

4.3.1 数据预处理

1)对历史数据的过滤

不同缺陷报告中的缺陷状态的更改日志包含了很多不同信息,例如编号(ID)为 252431 的缺陷包含了邮件抄送(CC)、受托人(Assignee)、所属构件、目标里程碑(Target Milestone)、状态(Status)、标记(Flags)等信息,而图 1 中 ID 为 11030 的缺陷仅包含了受托人、状态、决议(Resolution)信息.

从 3.2 节介绍的缺陷报告可知,虽然缺陷的“History”字段记录了缺陷修复过程中的所有活动,但并非所有的信息都是需要的.根据本文实验的特点,即为了找出开发者之间的传递关系,构建开发者合作网络,我们从中过滤出“What”为受托人、状态、决议三类关键信息来做进一步分析.

2)对描述数据的过滤

每个缺陷报告包含的标题、描述、评论都是文本格式,信息详略程度不一,所以我们将上述所有文本信息统称为(文本)描述信息,并放在一起作分析.由此,每个缺陷报告都对应一个描述文本.对于每一个缺陷描述文本,我们依次进行去停词、词干提取等英文的自然语言处理,精简并改善描述文本的质量.

4.3.2 开发者合作网络的构建

对修改历史过滤的结果,我们准备针对每个缺陷包含“What”为受托人、状态、决议三类信息的所有记录过滤,得到一条缺陷再分配路径[5].通过实验发现,事实上很难得到一条较精确的结果,特别是当传递次数很多时,甚至人工分析也不能得到一条完整路径.所以,我们挑选出有明确传递关系的缺陷再分配的开发者对,来构建开发者合作网络.根据过滤后得到的开发者再分配对集合,对于每一个开发者再分配对(A→B),将开发者 A 和 B 作为网络节点、A→B 作为网络的一条有向边,以此来构建开发者合作网络.

4.3.3 开发者修复能力建模

1)确定修复缺陷的开发者

如前所述,一种观点认为缺陷修复是众多开发者协同的过程,因而一个缺陷对应一个开发者集合,这个集合由参与到这个缺陷修复活动的所有开发者构成[15,19];另一种观点认为一个缺陷是由一个修复者最终修复的[3,5,7,10,11],我们也是这样理解的,一个缺陷对应一个修复者.因为,在一个缺陷修复的过程中,默认最终的修复者已经理解并掌握了协助他完成该工作的开发者所具备的能力,在今后遇到同种类型的缺陷时,这个修复者是最理想的缺陷修复人选.

表 1 缺陷对应的主题概率分布(阈值为 0.1)

Table 1 An example of the topic probability distribution of three bugs when the threshold is set to 0.1

ID	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
001	**0.15**	0.05	**0.1**	0.05	0.05	**0.6**
002	0.06	0.06	**0.7**	0.06	0.06	0.06
003	**0.2**	0.075	**0.5**	0.075	0.075	0.075

然而,缺陷报告并没有在预定义字段给我们指出谁是最终的修复者.在以前的工作中,虽然文献[10]提出了 6 种启发式的策略,但也存在一些问题;文献[11]随机抽样检查缺陷的活动日志,采用启发式的策略总结了 4 条规则.类似地,我们也是通过观察分析缺陷的修复活动日志,得出一个较为通用的规则:不管缺陷是否曾重启(REOPEN),只要缺陷最终的状态为 VERIFIED,决议为 FIXED,我们找到最后一条状态更改的记录,把其中涉及的开发者作为缺陷的最终修复者,如图 1 中的 jared_burns.

2)LDA 结果分析及处理

假设开发者 Tom 修复的所有缺陷编号分别为 001、002 和 003. 对这三个缺陷报告的文本描述信息进行过滤(参见 4.3.1),然后,利用 LDA 方法可以得到这三个缺陷各自的主题概率分布,假设如上页表 1 所示.

因为每个缺陷的描述信息质量不同,我们可以设置概率阈值 λ,只有概率大于 λ 时,才认为缺陷属于对应的主题. 这样,一个缺陷可以属于多个主题. 所以,每个缺陷所属的主题类别及主题个数可能会各不相同. 这里我们取 $\lambda=0.1$. 此时,表 1 中编号 001 的缺陷既属于主题 0,也属于主题 2 和主题 5,而编号 002 的缺陷仅属于主题 2,编号 003 的缺陷属于主题 0 和主题 2. 由此,可以得到如图 3 所示的映射关系. 值得注意的是,有的缺陷报告描述质量太差,经过主题建模后很可能不属于任何一个主题,因为各个主题概率值均小于阈值.

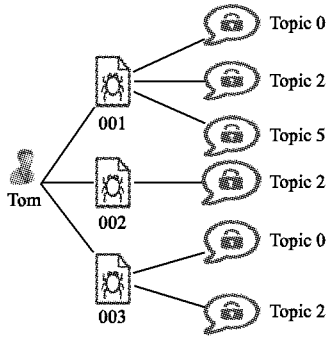

图 3 Fixer-Bug-Topic 关系图

Fig. 3 An example of a Fixer-Bug-Topic diagram

3)计算开发者对应的主题概率分布

根据开发者-缺陷、缺陷-主题两种关联关系,可以得到开发者-主题的关系,如图 3 所示. 再利用公式(1),可以计算得到一个开发者对应的修复缺陷的主题概率分布,即开发者的缺陷主题修复能力. 例如,图 3 中 Tom 修复的缺陷的主题概率分布为:$P(\text{topic}_0)=0.33$,$P(\text{topic}_2)=0.50$,$P(\text{topic}_5)=0.17$.

$$P(topic_i)=\frac{N_{bug}topic_i}{\sum_i N_{bug}topic_i}. \quad (1)$$

5 实验及结果分析

5.1 数据收集

Eclipse 和 Mozilla 两个开源软件项目都是使用 Bugzilla 来管理缺陷的. 从缺陷的生命周期的可知,缺陷的状态是不断在变化的,已经修复的缺陷还可以再打开. 所以应该选取状态较为稳定的历史缺陷来分析,同时数据量也应足够大.

下面我们以 Eclipse 为例来介绍数据集的收集过程. 首先,通过设置查询条件状态为 VERIFIED、决议为 FIXED 得到缺陷列表,依次搜集了缺陷编号从 1 到 357573 的 200,000 条缺陷数据;然后,利用爬虫工具爬下了每个缺陷对应的状态更改历史、描述、评论等信息. 同样地,在 Mozilla 中也依次搜集了缺陷编号从 37 到 660036 的 220,000 条数据.

对于我们在 Eclipse 中选取的 2001.10 至 2011.9 这 10 年期间的共 20 万个缺陷,Mozilla 中选取的 1998.4 至 2011.5 这 13 年期间的共 22 万个缺陷,经过前文所述的预处理,为这些缺陷确定了修复者,由此构成后面实验的数据集.

5.2 实验设计

5.2.1 实验流程

本文提出的缺陷分派方法,其效果主要取决于 3 个方面:一是开发者-主题概率分布,主要受 LDA 主题模型[12,14]的参数影响;二是开发者合作网络,无向图或者有向图;三是再分配策略.

首先,通过选取不同的 LDA 参数、开发者合作网络(DCN)、再分配策略(TS),形成不同的组合模型;然后以原始数据集的前 90% 部分作为训练集,分别训练开发者分派模型;最后用原始数据集的后 10% 部分作为测试集,来测试各种组合模型的分派准确率.

最终,可以将这种最优结果下的参数组合(LDA + DCN + TS)视为缺陷分派的最终模型. 同时,在这种最优的LDA参数下,使用同一数据集利用 LDA + kNN 的方法[13],用其产生的结果作为本文实验的对比,以验证本文提出的缺陷分派方法的有效性.

表 2 Eclipse 实验结果

Table 2 Experimental results for the Eclipse project

alpha	#topic	beta	Indegree_Directed (%)				Outdegree_Directed (%)				Undirected (%)			
			Degree	Topic	Mix_t	Mix_d	Degree	Topic	Mix_t	Mix_d	Degree	Topic	Mix_t	Mix_d
0.05	20	0.1	34.3	37.5	38.7	32.4	36.3	37.1	38.6	37.4	34.8	39.2	40.3	39.7
0.1	20	0.1	34.8	37.4	37.0	34.1	37.5	37.8	38.5	38.6	35.2	39.3	39.1	40.0
0.2	20	0.1	**37.7**	40.4	**41.1**	36.3	39.5	40.9	**44.1**	**43.7**	**38.7**	**42.6**	**43.8**	**46.7**
0.5	20	0.1	36.3	39.5	40.0	35.0	37.5	37.6	39.0	38.7	36.6	41.5	40.2	40.7
1.0	20	0.1	36.2	41.1	**41.1**	**36.4**	**40.0**	40.9	42.2	42.0	36.8	40.9	40.5	41.1
0.05	50	0.1	33.0	38.0	36.0	33.4	36.5	38.3	37.4	38.8	33.6	41.8	39.6	38.9
0.1	50	0.1	28.6	37.6	35.0	29.6	32.0	39.6	39.8	38.3	29.8	35.6	35.1	34.6
0.2	50	0.1	32.1	39.5	37.0	34.5	35.8	38.8	38.8	40.9	33.4	38.5	36.3	38.2
0.5	50	0.1	30.0	35.2	31.7	30.7	34.3	36.9	38.3	36.4	32.8	37.3	37.3	36.8
1.0	50	0.1	27.1	34.4	34.1	32.0	30.2	35.2	33.8	34.4	30.0	33.4	32.4	33.9
0.05	100	0.1	27.8	31.8	31.6	28.0	28.5	32.1	31.7	31.1	28.3	32.7	30.2	29.9
0.1	100	0.1	30.5	36.3	34.0	31.4	34.9	36.7	35.5	38.0	32.3	42.1	39.5	37.8
0.2	100	0.1	33.8	37.8	36.3	35.4	35.9	**41.4**	41.1	37.6	35.7	40.0	37.0	35.9
0.5	100	0.1	30.8	**42.6**	35.7	32.2	33.3	41.0	39.1	35.6	31.6	35.5	34.9	35.3
1.0	100	0.1	25.4	30.2	28.6	28.2	30.1	31.8	31.4	37.3	28.0	36.8	35.3	33.8

关于 LDA 的参数设置. 基于吉布斯抽样的 LDA 主题模

型开源实现,例如 GibbsLDA + +、JGibbLDA 等,使用时需要设置 4 个参数:主题数目#*topic*、迭代次数 t、参数 α 和 β. 因为 LDA 参数的设置对缺陷主题分类有较大影响,所以,本文我们采用控制变量法,先固定取值 $\beta=0.1$、$t=2000$,然后 α 分别取 0.05、0.1、0.2、0.5、1.0,主题数目#*topic* 分别取 20、50、100 来分组讨论.

5.2.2 评价标准

缺陷分派的目的是为每一个缺陷分配一个合适的开发者来进行修复,测试集的每一个缺陷都对应一个修复者. 例如,可以对每个缺陷最多推荐 3 位开发者,如果其中的任意 1 人和标签(修复者)一致,就认为推荐命中了有效的开发者,此次分派有效,否则无效. 我们以整个测试集缺陷分派的准确率(命中率)作为方法的评价指标,其计算见公式(2).

$$Accuracy=\frac{n}{N}, \tag{2}$$

其中,n 表示推荐命中的缺陷个数,N 表示缺陷的总数.

5.3 实验结果

上页表 2 和表 3 分别是本文所提方法在不同参数下在 Eclipse 和 Mozilla 上的结果. 其中,Degree、Topic、Mix_t、Mix_d 分别表示基于度的再分配策略、基于主题的再分配策略,以及以度为主、以主题为主的混合策略.

实验结果显示:对于 Eclipse 数据集,在 $\beta=0.1$、$\alpha=0.2$、#*topic* =20 的 LDA 参数条件下,利用训练集构建无向图结构的开发者合作网络,在分派缺陷时,采用混合策略,分派的准确率达到了 46.7%;对于 Mozilla 数据集,在 $\beta=0.1$、$\alpha=0.5$、#*topic* =20 的 LDA 参数条件下,利用训练集构建有向图结构的开发者网络,在分派缺陷时,采用基于主题的策略,分派的准确率达到了 33.4%.

5.4 对比分析

表 3 Mozilla 实验结果

Table 3 Experimental results for the Mozilla project

alpha	#topic	beta	Indegree_Directed (%)				Outdegree_Directed (%)				Undirected (%)			
			Degree	Topic	Mix_t	Mix_d	Degree	Topic	Mix_t	Mix_d	Degree	Topic	Mix_t	Mix_d
0.05	20	0.1	16.2	20.1	22.3	20.4	17.7	17.8	20.0	18.3	16.7	18.6	21.2	21.5
0.1	20	0.1	16.4	20.0	23.8	23.0	18.3	17.7	21.4	17.8	17.9	17.8	22.2	21.7
0.2	20	0.1	19.4	30.3	26.5	27.7	**24.3**	33.0	31.8	**27.8**	**23.2**	**32.6**	31.2	**31.9**
0.5	20	0.1	**19.7**	**31.1**	29.5	**28.4**	23.5	**33.4**	**32.1**	25.5	22.6	32.4	**31.8**	30.8
1.0	20	0.1	19.3	29.1	**29.9**	26.8	**24.3**	31.5	29.5	27.0	23.1	30.6	30.7	29.2
0.05	50	0.1	12.4	15.4	14.8	15.5	13.1	14.7	14.4	14.2	13.0	15.0	14.2	14.7
0.1	50	0.1	10.2	11.4	12.3	12.7	10.3	9.7	10.5	10.5	10.8	10.2	11.1	11.9
0.2	50	0.1	18.6	29.2	25.1	22.6	21.0	27.8	28.1	26.5	20.7	27.5	28.6	26.5
0.5	50	0.1	14.7	21.1	20.2	17.8	18.1	21.1	24.1	22.5	18.2	20.3	25.5	22.0
1.0	50	0.1	14.9	24.0	24.4	21.5	17.8	23.6	23.4	22.5	17.8	24.2	25.5	24.5
0.05	100	0.1	13.6	17.5	16.1	16.2	13.5	14.7	14.9	15.4	16.7	18.6	21.2	21.5
0.1	100	0.1	15.1	19.8	17.7	18.8	12.2	13.7	13.5	13.4	17.9	17.8	22.2	21.7
0.2	100	0.1	14.9	19.6	18.6	16.8	15.0	19.2	18.4	17.6	**23.2**	**32.6**	31.2	**31.9**
0.5	100	0.1	14.6	25.2	22.5	20.8	16.7	26.3	26.4	25.0	22.6	32.4	**31.8**	30.8
1.0	100	0.1	17.3	29.8	29.6	26.9	23.5	30.0	30.8	26.5	23.1	30.6	30.7	29.2

1)表 4 是在相同的数据集下,以 LDA + kNN 作为基准方法得到的结果. 由表可知,Eclipse 数据集在 LDA 参数 $\beta=0.1$、$\alpha=0.2$、#*topic* =20 下,分派的准确率最高为 15.0%;Mozilla 数据集在 LDA 参数 $\beta=0.1$、$\alpha=0.5$、#*topic* =20 下,分派的准确率最高为 14.4%. 而本文的方法 LDA + DCN + TS,准确率分别达到了 46.7% 和 33.4%. 经过对比,推荐准确率的提升效果明显,验证了本文提出的缺陷分派方法是有效的.

表 4 LDA + kNN 基准实验结果(%)

Table 4 Results of the baseline method LDA + kNN (%)

	$k=1$	3	10	20	50	100	200
Eclipse	8.1	9.6	12.2	13.6	14.6	**15.0**	14.9
Mozilla	7.5	8.9	11.9	13.3	**14.4**	14.3	14.0

2)相对于文献[13],观察发现实验的准确率的值虽然相对偏低,但提高效果却很明显(分别为 209.3% 和 131.9%),这是因为本文的实验数据集更大(Eclipse 数据集 20 万个缺陷对应的开发者数量为 2794 个、Mozilla 数据集 22 万个缺陷对应的开发者数量更是达到了 5104 个). 例如,虽然对于 Eclipse 项目文献[13]利用 LDA + kNN 的准确率达到 35% 左右,但在本文的数据集上准确率只有 15% 左右.

3)同等条件下,一般无向图比有向图的准确率高. 这说明:历史上具有缺陷传递关系的两个开发者 A 和 B,当有一方 A 向另一方 B 传递缺陷后,在将来的某个时刻,B 遇到解决不了的缺陷时,也会将缺陷传递给 A.

4)同等条件下,在有向图中,基于出度的分配策略推荐准确率比入度的准确率高,这与文献[19,20]的结论是吻合的.

5)开发者合作网络(DCN)的运用对分配的准确率并没有很明显的提升. 分析其原因,一是我们仅仅利用历史记录中开发者间具有明显的传递信息构建的网络,同时删除了重复的边,这样构建的网络可能本身就遗漏了一些信息;二是没有考虑开发者社交网络是随着时间不断在演化的,某些开发者可能由积极的参与者变为不活跃的状态等;三是仅考虑了网络度对结果的影响,按节点度大的原则分配,然而度越大的节点更可能遭遇工作量过载的情况,这样可能抵消了本身具备的优势.

6)结果可能是局部最优的. 我们观察到在 Mozilla 有向

图的结果中,当#*topic* = 100、β = 0.1 时,推荐的准确率是随着 α 的增大在不断地提升的,有可能超过 33.4 % 的最高准确率. 所以,确定 LDA 的参数以达到全局最优还有待解决.

7)与本文研究相关的最新文献[8,9]均采用的是基于评分机制的开发者排名推荐,且利用的信息不同,并不适合直接与本文所提的方法进行对比. 因此,在文本中没有将它们作为基准方法.

6 总 结

在本文中,我们通过对软件缺陷库的缺陷报告信息的综合运用,包括描述信息对开发者缺陷修复能力建模,根据状态更改记录构建开发者合作网络,提出了基于度、主题以及二者混合的三种分配策略,共同组成了本文的针对软件缺陷自动分配的开发者推荐方法. 通过对开源软件项目 Eclipse 和 Mozilla 部署在 Bugzilla 上的十万级缺陷历史数据集进行实验,验证了本文的方法在缺陷分派领域的有效性.

对实验结果的分析,也发现方法的局限性,例如缺乏对时间维度的考虑,忽视了开发者活跃度的改变,节点度大的开发者可能会工作过载,LDA 参数的调优等. 由于开源软件社区开发者都是志愿贡献者,缺乏统一的机构或组织对其进行管理. 所以,对开发者的活跃度、工作量等缺乏准确、定量的描述,这就为现有缺陷修复的很多方法带来了不利影响. 在未来的工作中,我们会对社区开发者的情况进一步挖掘,例如,寻找更好的开发者合作网络,分析开发者的工作量对缺陷分派的影响等等. 并且,我们也会将本文所提的方法运用在其他的开源软件数据集上进行验证.

References:

[1] Wurster L F, Igou B, Babat Z. Survey analysis: overview of preferences and practices in the adoption and usage of open-source software [R]. Stamford: Gartner, Inc., Technical Report: G00210068, 2011.

[2] Aberdour M. Achieving quality in open source software [J]. IEEE Software, 2007, 24(1): 58-64.

[3] Anvik J. Automating bug report assignment [C]. Proceedings of the 28th International Conference on Software Engineering (ICSE'06), 2006: 937-940.

[4] 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.

[5] 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 (ESEC/FSE'09), 2009: 111-120.

[6] 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.

[7] 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.

[8] 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.

[9] Xia X, Lo D, Wang X, et al. Dual analysis for recommending developers to resolve bugs [J]. Journal of Software: Evolution and Process, 2015, 27(3): 195-220.

[10] Cubranic D, Murphy G C. Automatic bug triage using text categorization [C]. Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE'04), 2004: 92-97.

[11] Anvik J, Hiew L, Murphy G C. Who should fix this bug [C]. Proceedings of the 28th International Conference on Software Engineering (ICSE'06), 2006: 361-370.

[12] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation [J]. Journal of Machine Learning Research, 2003, 3: 993-1022.

[13] Huang Xiao-liang, Yu Shu-si, Guan Ji-hong. Software bug triage method based on LDA topic model [J]. Computer Engineering, 2011, 37(21): 46-48.

[14] Somasundaram K, Murphy G C. Automatic categorization of bug reports using latent dirichlet allocation [C]. Proceedings of the 5th Annual India Software Engineering Conference (ISEC'12), 2012: 125-130.

[15] 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 (PROMISE'12), 2012: 19-28.

[16] Nagwani N K, Verma S, Mehta K K. Generating taxonomic terms for software bug classification by utilizing topic models based on latent dirichlet allocation [C]. Proceedings of the 11th International Conference on ICT and Knowledge Engineering (ICT&KE'13), 2013: 1-5.

[17] 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 (COMPSAC'14), 2014: 97-106.

[18] Wang S, Zhang W, Yang Y, et al. DevNet: exploring developer collaboration in heterogeneous networks of bug repositories [C]. Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'13), 2013: 193-202.

[19] 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 (APSEC'11), 2011: 389-396.

[20] Xuan J, Jiang H, Ren Z, et al. Developer prioritization in bug repositorie [C]. Proceedings of the 34th International Conference on Software Engineering (ICSE'12), 2012: 25-35.

[21] 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 (ISSRE'14), 2014: 122-132.

[22] 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.

[23] Pan Xing-liang. Study on present situation and characteristic analysis of the mozilla bug repository [D]. Dalian: Dalian University of Technology, 2013.

附中文参考文献:

[13] 黄小亮,郁抒思,关佶红. 基于 LDA 主题模型的软件缺陷分派方法[J]. 计算机工程,2011,37(21):46-48.

[23] 潘兴亮. Mozilla 缺陷报告仓库现状研究及特征分析[D]. 大连:大连理工大学,2013.