一种基于文本分类和评分机制的软件缺陷分配方法

史小婉 马于涛
(武汉大学计算机学院 武汉 430072)

摘 要 开源软件项目的缺陷管理和修复是保障软件质量及软件开发效率的重要手段,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。文中提出了一种基于文本分类和评分机制的开发者预测方法,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,所提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提升了 17.34%和 10.82%,从而验证了其有效性。

关键词 缺陷分配,文本分类,评分,预测模型,支持向量机
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.11.030

Software Bug Triaging Method Based on Text Classification and Developer Rating

SHI Xiao-wan MA Yu-tao
(School of Computer Science, Wuhan University, Wuhan 430072, China)

Abstract Bug management and repair in open-source software (OSS) projects are meaningful ways to ensure the quality of software and the efficiency of software development, and improving the efficiency of bug triaging is an urgent problem to be resolved. A prediction method based on text classification and developer rating was proposed in this paper. The core idea of building the prediction model is to consider both text classification based on machine learning and rating mechanism 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 the proposed method reach 78.39% and 64.94%, respectively. Moreover, its accuracies 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 the proposed method.

Keywords Bug triage, Text classification, Rating, Prediction model, Support vector machine

1 引言

较低的缺陷修复率会增加软件维护的工作量和软件生产的成本。目前,大多数软件项目已使用缺陷跟踪系统(如 Bugzilla)来管理缺陷的修复过程,以便应用程序实施维护[1]。这些项目每天会接收数以百计的缺陷报告,理想情况下,每一个缺陷都应尽快被分配给可能修复它的开发者。人工分配缺陷是一个很复杂的过程,耗时、耗力且容易出错[2]。例如,Jeong 等的实证研究[3]指出,在 Eclipse 项目中,平均大约需要 40 天时间将缺陷分配给第一个开发者,然后再大约需要 100 天时间将它分配给第二个开发者。

在开源软件开发环境中,随着项目规模的不断增大,大量的缺陷被发现和提交,这导致对缺陷进行人工管理变得越来越困难,特别是如何将缺陷及时分配给合适的开发者进行修复[4-6]。为了提高缺陷分配的效率,降低相应的人力和时间成本,目前已有不少研究者提出了各种自动分配缺陷的方法。从使用的数据、算法、再分配图(tossing graph)模型等维度,文献[7]将现有方法分为 3 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,还是最终修复缺陷,都是有贡献的工作。因此,通常把所有与缺陷相关的开发者都看作缺陷的标签,从而能够在预测时得到较高的召回率。与这种思路不同的是,如果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)将其分配给可以修复它的开发者,将会大大提高缺陷修复的效率[8];同时,为了提高预测的准确率(accuracy),也可以依次

到稿日期:2017-11-30 返修日期:2018-02-02 本文受国家重点基础研究发展计划(973)(2014CB340404),国家自然科学基金(61672387,61702378),武汉市黄鹤英才(现代服务)计划资助。

史小婉(1991-),女,硕士生,主要研究方向为软件工程;**马于涛**(1980-),男,副教授,CCF 高级会员,主要研究方向为软件工程和服务计算,E-mail:ytma@whu.edu.cn(通信作者)。

给出多个候选开发者。

遵循该思路，文中提出了一种用于自动分配缺陷的开发者预测方法，其主要技术贡献如下：

1)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，从而提高了准确率。

2)针对开源项目 Eclipse 和 Mozilla 的缺陷数据，在增量学习模式下将本文方法与基于机器学习(ML)以及机器学习＋再分配图(ML＋TG)的基准方法进行了对比，结果表明所提方法在准确率和缺陷再分配路径长度两个指标上均优于基准方法。

本文第 2 节介绍与本文研究相关的工作；第 3 节详细介绍所提方法的框架及实现细节；第 4 节给出实验设计及结果分析；最后总结全文并展望未来。

2 相关工作

缺陷报告的文本内容主要包括预定义字段(field)、摘要(summary)、描述(description)及评论(comments)、历史修复记录等[5,7]。由于文本内容涵盖了缺陷的主要信息，因此预测缺陷修复者(fixer)的早期研究主要使用的方法为文本分类[9-13]。例如，文献[9]使用从缺陷报告的标题和描述信息中提取的关键字训练了一个朴素贝叶斯(NB)分类器，准确率达到了 30%；文献[10]过滤掉标记为 invalid，wontfix 和 worksforme 的缺陷报告，以及不再活跃和修复缺陷数量少于 9 的开发者，同时使用了朴素贝叶斯(NB)、决策树(C4.5)和支持向量机(SVM) 3 种分类器，预测的准确率达到了 64%。

在缺陷被提交至修复的过程中，开发者能通过多种方式进行合作，如评论缺陷、分配缺陷和传递缺陷等，根据这些合作信息可以构造开发者之间的合作关系网络[14]。之前的工作[3,15]也表明，在使用缺陷文本内容的基础上，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。例如，Jeong 等较早利用马尔可夫链构造了一个缺陷再分配图模型[3]，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近 72%，而预测准确率比基准方法提高近 23%。Wang 等根据开发者对缺陷的评论以及所修复缺陷的来源等信息构建了一个复杂的开发者合作网络[15]，准确率相比基准方法提高了 5%～15%。此外，Wu 和 Xuan 等分析了一些经典的社交网络指标对开发者合作网络中开发者技能排名的影响[16-17]。

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。这类方法主要包括两个步骤[18-19]：首先，根据缺陷报告的信息，使用分类算法或缺陷之间的相似度来预测可能的开发者；然后，使用开发者合作网络进一步明确候选人。Zhang 等提出了基于最近邻(KNN)查找和开发者合作网络的 KSAP 方法[20]，与基准方法相比，召回率提高了 7.5%～32.25%。Xia 等提出了一种基于缺陷报告和开发者关系的混合方法 DevRec[21]，该方法在召回率方面明显优于 DREX 方法[16]。Bhattacharya 等提出了一种“ML＋TG”的方法[22]，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该缺陷，则根据再分配概率图重新分配开发者。虽然该方法在 Eclipse 和 Mozilla 数据集上取得了良好的预测效果，但其时效性有待验证，也即需要给定一个时限来判断第一个开发者是否可以修复缺陷，当判定其无法修复时，再采用再分配策略去分配其他的开发者。

3 方法框架

3.1 整体流程

本文所提出的用于自动分配缺陷的开发者预测方法的整体流程如图 1 所示。

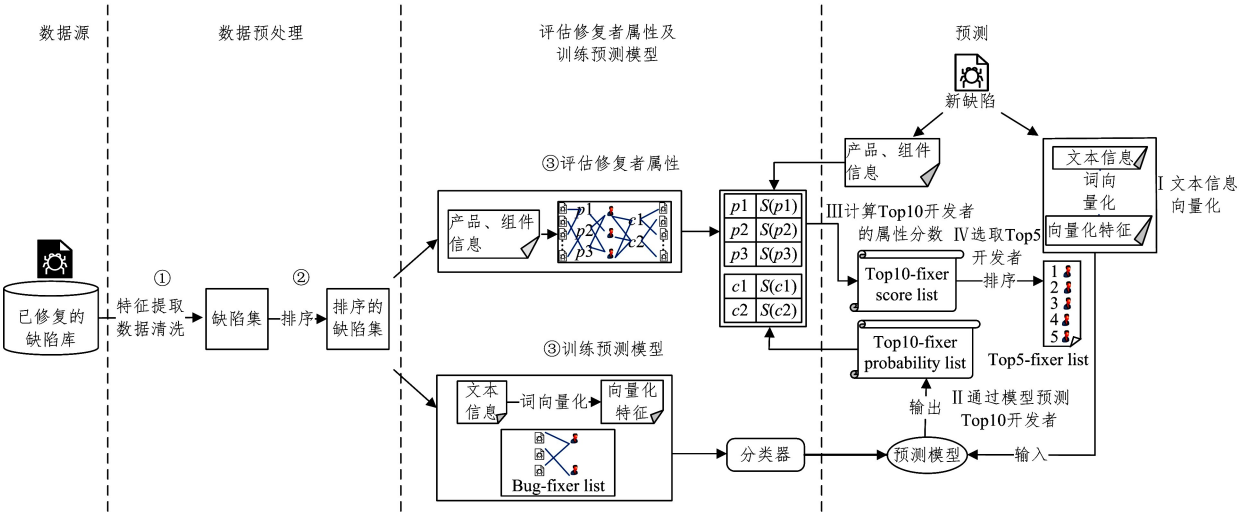


图 1 整体框架

Fig. 1 Framework of our approach

该流程主要包括如下 3 个主要步骤。

1)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[23] 和 TF-IDF[24]（Term Frequency-Inverse Document Frequency）词向量化等自然语言处理。按照修复时间对预处理后的数据集进行排序，得到实验数据，并均分成 11 份。

2）“十折”增量学习（10-fold incremental learning）与验证。在第 i 轮时，将前 i 份数据作为训练集，第 $i+1$ 份数据作为测试集，进行如下主要的操作。

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。

3）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。

3.2　词向量化

Word2vec 是一种使用分布式向量来表示文本的方法[25]。与传统的文本向量空间模型相比，使用 Word2vec 模型表示文本既能解决高维稀疏特征问题，又能引入语义特征。Word2vec 包含两种训练模型：CBOW（Continuous Bag-of-Words Model）和 Skip-gram。另外，TF-IDF 是一种常用的统计方法[24]，用以评估一个字/词对文件集或语料库中的一份文档或一个类别的重要程度。

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 $t_i=\langle w_1,w_2,\cdots,w_j\rangle$（$w_j$ 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，将加权过的词向量累加，得到文本 t_i 的向量表示 $weight\_R(t_i)$，如式(1)所示：

$$weight\_R(t_i)=\sum_{w\in t_i} word2vec(w)*\alpha_w \tag{1}$$

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

3.3　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的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的计算式如下：

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

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）的缺陷的概率计算式如下：

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

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 及其可能的修复者 d_v，如果 $p_{b_l}\in P_{d_v}$，则 d_v 在该产品特征上获得的分数（概率）为 $s_{d_v}(p_{b_l})$，否则为 0，如式(4)所示：

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

类似地，d_v 在组件特征 c_{b_l} 上获得的分数如式(5)所示：

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

其中，C_{d_v} 为 d_v 修复的所有缺陷所属的组件集合。

3.4　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$\{d_i\mid 1\leqslant i\leqslant \text{Top-}N\}$。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，并使用式(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。

$$R(d_i)=\alpha P(d_i)+(1-\alpha)(S(p\mid d_i)+S(c\mid d_i)) \tag{6}$$

其中，$\alpha(\alpha\in[0,1])$为权重参数。

4　实验及结果分析

4.1　数据收集和预处理

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，下面以 Eclipse 为例介绍数据集的收集过程。首先，设置查询条件为“状态为 VERIFIED 且决议为 FIXED”；其次，依次搜集了（2001.10－2011.09 共 10 年）编号为 1－357573 的 20 万条缺陷数据；再次，利用爬虫工具爬取了每个缺陷对应的状态更改历史、描述、评论、所属产品及所属构件等信息；最后，确定这些缺陷的修复时间和修复者。限于篇幅，实验数据的相关详细信息请参考文献[8]。Eclipse 数据集包含 199960 个缺陷，共有 2794 个修复者；类似地，Mozilla 数据集包含 219874 个缺陷，共有 5104 个修复者。

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；软件环境的配置如下：Ubuntu64 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（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，并使用 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 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[25,27-28]的经验，针对本文所使用的文本数据的特点以及 Skip-gram 模型的特点进行赋值的。

4.2 实验设计

4.2.1 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。分别将按时间先后顺序排列的 Eclipse 和 Mozilla 数据平均分成 11 份，在第 i 次实验时，将前 i 份数据作为训练集(TDS)，第 $i+1$ 份数据作为测试集(VDS)。

4.2.2 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(RF)和 LibSVM[26] 6 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：MLonly)；同时，结合针对缺陷的产品和组件特征的评分机制，形成了 6 种符合本文方法思想的混合模型(MLS：ML+Score)。对于这 6 种分类算法，输入为一个(新的)缺陷向量化后的文本特征，输出为 k 个可能修改这个缺陷的开发者。

4.2.3 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合适的开发者以进行修复，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。例如，针对每个缺陷返回 Top-k 个开发者，如果其中的任意一人和缺陷真实的修复者(标签)一致，则认为预测命中了有效的开发者，否则视为无效。准确率的计算如式(7)所示：

$$Accuracy=\frac{n}{N} \tag{7}$$

其中，n 为预测命中的缺陷个数，N 为缺陷的总数。

4.3 实验结果

针对一个给定的缺陷报告，以往研究[3,6,15-16]通常推荐最多 5 个可能的开发者。表 1 和表 2 分别列出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 表示只使用机器学习分类算法得到的结果，而 MLS 表示本文所提方法。

表 1 针对 Eclipse 的实验结果($\alpha=0.6, k=5$)

Table 1 Experimental results on Eclipse project($\alpha=0.6, k=5$)

算法	方法	准确率(测试集编号)										准确率(均值)	增幅/%
		2	3	4	5	6	7	8	9	10	11		
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

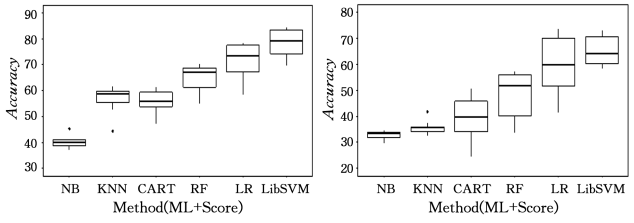
表 2 针对 Mozilla 的实验结果($\alpha=0.6, k=5$)

Table 2 Experimental results on Mozilla project($\alpha=0.6, k=5$)

算法	方法	准确率(测试集编号)										准确率(均值)	增幅/%
		2	3	4	5	6	7	8	9	10	11		
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

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，用箱线图展示了当 $k=5$ 时

对这两个数据集进行 10 次预测的结果，具体如图 2 所示。



(a)Eclipse　　(b)Mozilla

图 2　针对 Eclipse 和 Mozilla 的实验结果(箱线图)

Fig. 2　Experimental results on Eclipse and Mozilla by using box plots

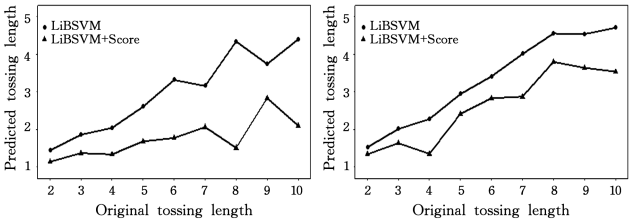
此外，表 3 还列出了在这两个数据集中 k 值的变化(从 1 增长到 4)对平均准确率的影响。总的来说，推荐的开发者越多，MLO 和 MLS 的平均准确率会越高，但 MLS 的效果比对应的 MLO 更好。LibSVM 在所有的分类算法中也获得了最高的平均准确率。而且，当 k 在 1 到 5 之间取值时，无论对于数据集 Eclipse 还是 Mozilla，在 α 取值为 0.6 时所得到的平均准确率均最高。在求解参数 α 时，取值区间为 0 到 1，其值每次增加 0.1。

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＋Score 方法时 Eclipse 和 Mozilla 数据集的缺陷再分配路径长度，如图 3 所示。

表 3　k 值的变化对实验结果的影响($\alpha=0.6$，$k=1,2,3,4$)

Table 3　Eeffect of change of k on experimental results for two projects($\alpha=0.6$，$k=1,2,3,4$)

算法	方法	平均准确率(Eclipse)				平均准确率(Mozilla)			
		$k=1$	$k=2$	$k=3$	$k=4$	$k=1$	$k=2$	$k=3$	$k=4$
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**



(a)Eclipse　　(b)Mozilla

图 3　在 LibSVM 和 LibSVM＋Score 方法下缺陷再分配路径长度的变化

Fig. 3　Changes in tossing path length obtained by LibSVM and LibSVM＋Score

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，其会一直在开发者之间传递下去，直到被最终分配的开发者修复。例如，一个缺陷首先分配给开发者 A 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，就继续进行再分配，直至传给开发者 E。假设 E 最终修复了该缺陷，那么这个过程涉及的开发者所形成的序列 $A\rightarrow B\rightarrow C\rightarrow D\rightarrow E$ 就是一条缺陷再分配路径，其长度为 5。从图 3 可以看出，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短于只使用 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。以上说明了本文方法对于缺陷修复者的预测更加有效。

4.4　与 ML＋TG 方法的对比分析

表 4 是在相同的数据集下，以文献[22]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 6 种分类算法。由表 4 可知，Eclipse 和 Mozilla 数据集都是在使用 SVM 分类算法时平均准确率最高，分别为 61.05％和 54.12％，而对应的本文所提方法的预测准确率分别为 78.39％和 64.94％，效果提升明显。此外，与基准方法相比，本文所提方法在 NB，KNN，CART 等其他 5 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，而本文所提方法为开发者的一次性分配策略，且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，本文所提方法更有效。

表 4　基于 ML＋TG 的基准方法的结果($k=5$)

Table 4　Results of benchmark method based on ML＋TG ($k=5$)

数据集	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[22]的结果相比，基准方法在本文实验中的平均准确率略低。一方面，这是因为本文所使用的实验数据与文献[22]中使用的数据在规模上存在差异。虽然我们使用的数据量更小，但是由实验结果可以看出，使用本文所提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存在差异。

4.5 效度分析

通过对本文实验结果的详细分析,我们也发现了存在的一些可能影响本文结论正确性(validity)的潜在威胁和局限性,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 6 种经典的分类算法(参数均为默认配置),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。

结束语 针对软件缺陷库中的缺陷报告信息,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,通过对潜在开发者的预测,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。文中还对知名开源项目 Eclipse 和 Mozilla 中的十万级缺陷历史数据集进行了实验,结果验证了本文方法在自动分配缺陷方面的有效性。

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,并不断优化算法的参数;2)进一步挖掘缺陷报告和开发者的特征,如开发者的活跃度、开发者的工作量或专业知识和技能等;3)在其他开源软件项目缺陷数据集上对本文方法进行验证。

参 考 文 献

[1] ZIMMERMANN T,PREMRAJ R,SILLITO J,et al. Improving bug tracking systems[C]// Proceedings of the 31st International Conference on Software Engineering. New York:IEEE Press,2009:247-250.

[2] 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.

[3] 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. New York:ACM Press,2009:111-120.

[4] ANVIK J. Automating Bug Report Assignment [C] // Proceedings of the 28th International Conference on Software engineering. New York:ACM Press,2006:937-940.

[5] 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.

[6] XIA X,LO D,WANG X,et al. Accurate developer recommendation for bug resolution[C]// Proceedings of the 20th Working Conference on Reverse Engineering. New York:IEEE Press,2013:72-81.

[7] 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.

[8] LIU H Y,MA Y T. Developer Recommendation Method for Automatic Software Bug Triage [J]. Journal of Chinese Computer Systems,2017,38(12):2747-2753. (in Chinese)
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.

[9] CUBRANIC D,MURPHY G C. Automatic Bug Triage Using Text Categorization[C]// Proceedings of the 16th International Conference on Software Enginee-ring and Knowledge Engineering. Pittsburgh:KSI Research Inc.,2004:92-97.

[10] ANVIK J,HIEW L,MURPHY G C. Who Should Fix This Bug?[C]// Proceedings of the 28th International Conference on Software Engineering. New York:ACM Press,2006:361-370.

[11] 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. New York:IEEE Press,2009:451-455.

[12] 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. New York:IEEE Press,2014:345-355.

[13] 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. New York:ACM Press,2014:53-63.

[14] 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.

[15] WANG S,ZHANG W,YANG Y,et al. DevNet:Exploring Developer Collaboration in Heterogeneous Networks of Bug Repositories[C]// Proceedings of the 7th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. New York:IEEE Press,2013:193-202.

[16] 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. New York:IEEE Press,2011:389-396.

[17] XUAN J,JIANG H,REN Z,et al. Developer Prioritization in Bug Repositories [C]// Proceedings of the 34th International Conference on Software Engineering. New York:IEEE Press,2012:25-35.

(下转第 219 页)

Neighbor Queries[C]// Proceedings of the Vldb Endowment. 2010:1231-1242.

[10] JENSEN C S,LIN D,OOI B C. Query and update efficient B+-tree based indexing of moving objects[C]// Proceedings of the Thirtieth International Conference on Very Large Data. 2004:768-779.

[11] PAPADIAS D,ZHANG J,MAMOULIS N,et al. Query Processing in Spatial Network Databases[J]. Vldb,2003,29:802--813.

[12] KOLAHDOUZAN M,SHAHABI C. Voronoi-based k nearest neighbor search for spatial network databases[C]// Proceedings of the Thirtieth International Conference on Very Large Data Bases. 2004:840-851.

[13] HUANG X,JENSEN C S,ŠALTENIS S. The Islands Approach to Nearest Neighbor Querying in Spatial Networks[C]// International Conference on Lecture Notes in Computer Science. 2006,3633:73-90.

[14] HUANG X,JENSEN C S,LU H,et al. S-GRID:A versatile approach to efficient query processing in spatial networks[M]// Advances in Spatial and Temporal Databases. Springer Berlin Heidelberg,2007:93-111.

[15] HONG S,CHANG J. A new k-NN query processing algorithm based on multicasting-based cell expansion in location-based services[J]. Journal of Convergence,2013,4(4):1-6.

[16] WANG H,ZIMMERMANN R. Location-based query processing on moving objects in road networks[C]// Proceedings of International Conference on Very Large Data Bases (VLDB 2007). 2007:321-332.

[17] PATROUMPAS K,SELLIS T. Monitoring Orientation of Moving Objects around Focal Points[C]// Symposium on Large Spatial Databases. 2009:228-246.

[18] LI G,FENG J,XU J. Desks:Direction-aware spatial keyword search[C]// IEEE 28th International Conference on Data Engineering (ICDE). 2012:474-485.

[19] YI S,RYU H,SON J,et al. View field nearest neighbor:A novel type of spatial queries[J]. Information Sciences,2014,275(3):68-82.

[20] LEE M J,CHOI D W,KIM S Y,et al. The direction-constrained k nearest neighbor query[J]. GeoInformatica,2016,20(3):471-502.

[21] LU B L,CUI X Y,LIU N. Bichromatic Reverse Nearest k Neighbor Query Processing on Road Network[J]. Journal of Chinese Mini-Micro Computer Systems,2015,36(2):266-270. (in Chinese)
卢秉亮，崔晓玉，刘娜. 路网中双色反向 k 近邻查询处理[J]. 小型微型计算机系统，2015,36(2):266-270.

[22] LEE K W,CHOI D W,CHUNG C W. Dart:An efficient method for direction-aware bichromatic reverse k nearest neighbor queries[M]// Advances in Spatial and Temporal Databases. Springer Berlin Heidelberg,2013:295-311.

[23] LI G L,FENG J H,XU J. DESKS:Direction-aware spatial keyword search[C]// 2012 IEEE 28th International Conference on Data Engineering. Washington:IEEE,2012:474-485.

[24] BRINKHOFF T. A framework for generating network-based moving objects[J]. GeoInformatica,2002,6(2):153-180.

(上接第 198 页)

[18] 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. New York:IEEE Press,2014:122-132.

[19] YAN M,ZHANG X H,YANG D,et al. A Component Recommender for Bug Reports Using Discriminative Probability Latent Semantic Analysis[M]. Butterworth-Heinemann,2016,73:37-51.

[20] 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.

[21] 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.

[22] 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.

[23] MIKOLOV T,SUTSKEVERI,CHEN K,et al. Distributed Representations of Words and Phrases and their Compositionality [C]// Proceedings ofthe 27th Annual Conference on Neural Information Processing Systems. La Jolla:Neural Information Processing Systems Foundation,2013:3111-3119.

[24] GAN J,CHEN L C. Research of improved IF-IDF Weighting algorithm[C]// Proceedings of the 2nd International Conference on Information Science and Engineering. New York:IEEE Press,2011:2304-2307.

[25] 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. New York:IEEE Press,2015:136-140.

[26] CHANG C C,LIN C J. LIBSVM:a library for support vector machines[J]. ACM Transactions on Intelligent Systems and Technology,2011,2(3):1-27.

[27] RONG X. word2vec parameter learning explained [EB/OL]. https://arXiv.org/abs/1411.2738.

[28] GOLDBERG Y,LEVY O. word2vec explained:deriving mikolov et al. negative-sampling word-embedding method[EB/OL]. https://arXiv.org/abs/1402.3722.