软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

DeepTriage:一种基于循环神经网络的缺陷报告分派方法*

席圣渠[1,2], 姚 远[1,2], 徐 锋[1,2], 吕 建[1,2]

[1](南京大学 计算机科学与技术系, 江苏 南京 210023)

[2](南京大学 计算机软件新技术国家重点实验室, 江苏 南京 210023)

通讯作者: 席圣渠, E-mail: nju.cellzero@gmail.com

摘 要: 随着开源软件项目规模的不断增大,人工为缺陷报告分派合适的开发人员（缺陷分派）变得越来越困难.而不合适的缺陷分派往往会严重影响缺陷修复的效率,为此迫切需要一种缺陷分派辅助技术帮助项目管理者更好地完成缺陷分派任务.当前,大部分研究工作都基于缺陷报告文本以及相关元数据信息分析来刻画开发者的特征,忽略了对开发者活跃度的考虑,使得对具有相似特征的开发者进行缺陷报告分派预测时表现较差.本文提出了一个基于循环神经网络的深度学习模型 DeepTriage,一方面利用双向循环网络加池化方法提取缺陷报告的文本特征,一方面利用单向循环网络提取特定时刻的开发者活跃度特征,并融合两者,利用已修复的缺陷报告进行监督学习.在 Eclipse 等四个不同的开源项目数据集上的实验结果表明,DeepTriage 较同类工作在缺陷分派预测准确率上有显著提升.

关键词: 缺陷分派;循环神经网络;深度学习

中图法分类号: TP311

中文引用格式:

英文引用格式:

DeepTriage: A Recurrent Neural Network Based Learning Approach for Bug Triaging

XI Sheng-Qu[1,2], YAO Yuan[1,2], XU Feng[1,2], Lu Jian[1,2]

[1](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

[2](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China)

Abstract: With the increasing size of open source software projects, assigning suitable developers for bug reports (i.e., bug triaging) is becoming more and more difficult. Moreover, the efficiency of bug repairing may be reduced if the bugs are assigned to inappropriate developers. Therefore, it is necessary to provide an automatic bug triaging technique for the project managers to better assign bug reports. Existing work for this task mainly focus on analyzing the text and metadata in bug reports to characterize the relationships between developers and bug reports, while the active level of developers is largely ignored. A shortcoming of these methods is that they may lead to poor performance when developers with different active levels have similar characteristics. This paper proposes a learning model named DeepTriage based on the recurrent neural networks. On the one hand, the ordered natural language text in bug reports is mapped into high-level features by a bidirectional RNN. On the other hand, developer's active level is extracted and transformed into high-level features

* 基金项目: 国家重点研发计划(2016YFB1000802); 国家 863 项目(2015AA01A203); 国家自然科学基金(61702252, 61672274, 61690204)

Foundation item: National Key Research and Development Program of China (2016YFB1000802); National 863 Program of China (2015AA01A203); National Natural Science Foundation of China (61702252, 61672274, 61690204)

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

through a single directional RNN. Then, the features of text and developer's active level are combined and learned from bug reports with known fixers. Experimental results on four different open-source data sets (e.g., Eclipse) show that DeepTriage has significantly improved the accuracy of bug triaging compared with existing work.

Key words: bug triaging; recurrent neural networks; deep learning

大型软件项目常常使用缺陷追踪系统（bug tracking system）.缺陷追踪系统不仅为最终用户提供了一个反馈的渠道,使开发者能够尽早识别软件缺陷或用户建议;而且能够协调开发者之间的工作,记录缺陷的解决方式,并提供归档信息以供查询,为软件的维护提供了便利.当前,Bugzilla、JIRA、mantis 等一系列缺陷追踪系统都得到了广泛的应用.

当一份缺陷报告提交到缺陷追踪系统,管理人员（高级开发者或者项目负责人）需要浏览缺陷报告,并为其选择合适的开发者.这样一个为缺陷报告指定开发者的过程,被称为缺陷分派（bug triaging）.然而,缺陷分派是一项十分耗时耗力的事情.一方面,对于大型项目,每天收到的缺陷报告数量很多,如 Eclipse 项目平均每天收到 91 份缺陷报告、而 Redhat 项目更是有 222 份之多（详见表 1）;另一方面,大型项目的开发与维护往往需要大量的开发者,对于 Eclipse、Mozilla 项目,有超过 1800 名开发者参与到缺陷修复的工作中[5].如果由人工来进行缺陷分派,无疑会消耗大量的时间和人力资源.

Table 1 Number of Bug Reports in Large Project

表 1 大型项目中的缺陷报告数量

项目名称	启用时间（年）	缺陷报告数	平均缺陷报告数量（/天）
Eclipse	15(2002~)	500,000	91
Netbeans	16(2001~)	270,000	46
Mozilla	19(1998~)	1,380,000	198
Redhat	16(2001~)	1,300,000	222

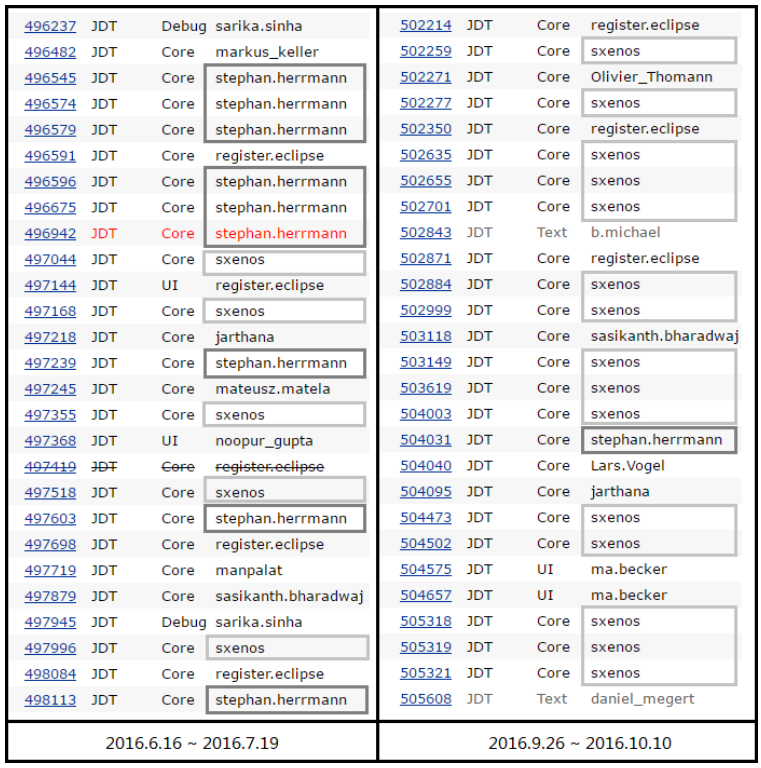
统计了截至 2017 年 6 月底,四个开源项目的大致缺陷报告数量,以及启用时长,以估计平均每天收到的缺陷报告数量

为解决这一问题,自动缺陷分派应运而生.自动缺陷分派通过统计、学习历史数据,自动地为新的缺陷报告推荐开发者.在以往研究中,缺陷报告中的文本数据是一项很重要的信息,文献[8,9,10,11,12]等使用向量空间模型（Vector Space Model）来表示缺陷报告,并将缺陷报告的文本数据表示为单词计数的向量;文献[5,6,13]等则使用了主题模型,利用单词在文本中的出现关系将文档表示为在不同主题下的分布情况.另外,缺陷报告还包含着以标签形式存在的元数据信息.每一种元数据都可以视为缺陷报告的一个分类字段,如产品、组件、操作系统类型、平台等字段,这些字段可以由报告者在提交缺陷报告时,通过列表的方式选择.文献[5,6]利用了这些信息进一步提高了自动缺陷分派的准确率.

然而,目前的自动缺陷分派技术依然存在着一定的不足,主要体现在如下几个方面.首先,文本数据更强调文字序列中前后元素之间的相互影响,元素之间次序的不同会导致文本含义的变化,而现有的缺陷分派方法都没有考虑单词的次序信息.其次,已有工作大多忽略了对开发者活跃度的考虑,使得对具有相似特征的开发者进行缺陷报告分派预测时表现较差.图 1 给出了一个数据中的真实案例.如图所示,Xenos 和 Stephan 是 Eclipse 项目中的两名开发者,两人都有 JDT 相关内容的修复记录.那么,当一份 JDT 相关的缺陷报告到来时,通常难以选择由 Xenos 还是 Stephan 修复.观察 Xenos 和 Stephan 的修复历史,可以发现在 2016 年 6 月中旬至 7 月中旬,Stephan 相对活跃并且修复了多个 JDT 的缺陷;而在同年 9 月至 10 月, Xenos 则更加活跃.因此,一个有效的缺陷分派方法应该在 6 至 7 月将 JDT 领域的缺陷分派给 Stephan,而 9 至 10 月则分派给 Xenos.

综合上述两个问题,本文提出了一种考虑文本次序信息和开发者活跃度的自动缺陷分派方法 DeepTriage. 首先,对于文本中的次序信息, DeepTriage 使用了双向循环神经网络结合池化的方法 oh-2LSTMp[7].该方法可以从自然语言文本中抽取高层特征,并且在文本分类问题上得到了较好的表现.其次,我们观察到开发者在不同时间区间有着不同的活跃程度（即对项目缺陷修复的参与程度有所差别）.当基于文本信息依然难以进一步鉴别合适的修复者时,将缺陷报告交给其中最活跃的开发者,对既快又好地修复缺陷能够起到一定的帮助作用.基于

以上考虑, DeepTriage 收集历史数据中开发者的修复活动记录,并将这些记录输入到单向循环神经网络中,以得到开发者活跃程度的高层特征信息.最后, DeepTriage 将两部分信息进行联结,输入到分类器中,共同对适合修复新缺陷报告的开发者进行预测.



496237	JDT	Debug	sarika.sinha
496482	JDT	Core	markus_keller
496545	JDT	Core	stephan.herrmann
496574	JDT	Core	stephan.herrmann
496579	JDT	Core	stephan.herrmann
496591	JDT	Core	register.eclipse
496596	JDT	Core	stephan.herrmann
496675	JDT	Core	stephan.herrmann
496942	JDT	Core	stephan.herrmann
497044	JDT	Core	sxenos
497144	JDT	UI	register.eclipse
497168	JDT	Core	sxenos
497218	JDT	Core	jarthana
497239	JDT	Core	stephan.herrmann
497245	JDT	Core	mateusz.matela
497355	JDT	Core	sxenos
497368	JDT	UI	noopur_gupta
~~497419~~	~~JDT~~	~~Core~~	~~register.eclipse~~
497518	JDT	Core	sxenos
497603	JDT	Core	stephan.herrmann
497698	JDT	Core	register.eclipse
497719	JDT	Core	manpalat
497879	JDT	Core	sasikanth.bharadwaj
497945	JDT	Debug	sarika.sinha
497996	JDT	Core	sxenos
498084	JDT	Core	register.eclipse
498113	JDT	Core	stephan.herrmann
2016.6.16 ~ 2016.7.19			

502214	JDT	Core	register.eclipse
502259	JDT	Core	sxenos
502271	JDT	Core	Olivier_Thomann
502277	JDT	Core	sxenos
502350	JDT	Core	register.eclipse
502635	JDT	Core	sxenos
502655	JDT	Core	sxenos
502701	JDT	Core	sxenos
502843	JDT	Text	b.michael
502871	JDT	Core	register.eclipse
502884	JDT	Core	sxenos
502999	JDT	Core	sxenos
503118	JDT	Core	sasikanth.bharadwaj
503149	JDT	Core	sxenos
503619	JDT	Core	sxenos
504003	JDT	Core	sxenos
504031	JDT	Core	stephan.herrmann
504040	JDT	Core	Lars.Vogel
504095	JDT	Core	jarthana
504473	JDT	Core	sxenos
504502	JDT	Core	sxenos
504575	JDT	UI	ma.becker
504657	JDT	UI	ma.becker
505318	JDT	Core	sxenos
505319	JDT	Core	sxenos
505321	JDT	Core	sxenos
505608	JDT	Text	daniel_megert
2016.9.26 ~ 2016.10.10			

Fig.1 Developers' active levels may be different during different periods in the same domain

图 1 同一个领域,开发者在不同时间活跃程度不同

为了验证方法的有效性,本文选取了 SVM[9],Yang[6],TopicMiner[5]作为对比方法,并在 Eclipse[1]、Mozilla[3]、Redhat[4]、Netbeans[2]四个大规模数据集上进行了实验.本文共收集了 602902 份缺陷报告,并参照[5,12]以 Top-1、Top-5 准确度（accuracy）对结果进行评估.实验结果显示,本文的方法 Top-1 准确度达到 42.96%,相较于对比方法(SVM, Yang, TopicMinerMTM),分别提升了 60%,23%与 12%.

本文的主要贡献总结如下：

- 采用 oh-2LSTMp 的方式对文本进行了特征抽取.缺陷报告的文本信息属于自然语言描述,单词间存在着次序先后的关系.本文方法考虑了文本中单词的次序关系,使用了更合适的方式对文本信息进行了特征抽取.所得到的特征,一方面降低了文本信息的维度,另一方面便于同其他特征进行联结,共同应用到缺陷分派问题.
- 提出对开发者活跃程度的建模.在某些情况下,只通过缺陷报告内容难以确定最合适的开发者.此时,将开发者之前一段时间的活跃程度作为额外信息往往可以提升缺陷分派的效果.
- 在大规模真实数据集上的实验验证了本文方法的有效性.本文从 4 个大规模、并且持续维护的开源项目的缺陷追踪系统中获取了大量缺陷报告.在这些缺陷报告上的实验显示本文方法显著优于已有方法.

本文组织如下.首先介绍相关工作,从机器学习和信息检索两个方面描述当前缺陷分派工作的研究进展;随

后,简要介绍循环神经网络相关的背景知识;再次,将介绍本文的研究思路和采取的方法,以及取得的实验结果;最后,对当前方法的不足,未来研究可能的方向以及面临的挑战进行了展望.

1 相关工作

针对自动缺陷分派问题,研究者们提出了一系列基于机器学习或是信息检索的方法[5,6,8,9,10,11,,13].机器学习方法通常将开发者视为标签,缺陷报告的信息(主要为文本)视为特征,以分类的方式学习和预测合适的开发者.信息检索的方式则同时对开发者与缺陷报告建模,通过相似性的方式查找合适的开发者.在信息使用的类别上,缺陷报告的文本信息是最重要的组成部分,在各类方法中都得到了使用;另外,元数据也是源于缺陷报告内容的重要信息,在近年的一些研究中得到了应用;还有一些使用其他信息的方法,例如关注开发者之间的关系,开发者在修复过程中扮演的角色,或是结合项目的代码注释、提交等信息,以获得更好的结果.本文工作更侧重于使用缺陷报告自身的信息,获得更好的分派结果.

Anvik 等人首先提出了文本分类的方式,并尝试使用 Naive Bayes(朴素贝叶斯)方法[8].Cubranic 等人则在文本分类基础上,尝试了 Naive Bayes、SVM(支持向量机)、C4.8(一种决策树方法)来解决这一问题[9].Tamrawi 等人提出了名为 Bugzie 的方法,采用基于模糊集和开发者缓存的方法[12].Naguib 等人[14]则使用语言模型 LDA,在主题空间比较缺陷报告同开发者的相似性,并且利用开发者的历史行为划分开发者的角色,共同的为缺陷报告推荐开发者.Yang 等人[6]也使用了主题模型,以获得缺陷报告的主题分布.Xia 等人[5]提出了 TopicMinerMTM 模型,在 LDA 的基础上加入了监督信息,使缺陷报告的主题受其特征(即本文所提元数据)监督,以获得更紧密的主题分布.本文的方法同上述方法不同,在文本处理上采用了更为合适的方法,考虑了文本间次序的关系,通过神经网络的方式,抽取出文本信息的特征,最终用于合适开发者的推荐.

在以往的工作中,元数据也被使用在自动缺陷分派中[5,6].Yang 等人的方法[6],首先使用 LDA 决定缺陷报告的主题,其次使用元数据信息过滤掉元数据同当前文档不一致的缺陷报告,剩余一致的缺陷报告被视为相似的缺陷报告,并且基于这些相似的缺陷报告,推荐合适的开发者修复.Xia 等人的工作[5]提出了 TopicMinerMTM,在使用 MTM 获取报告的主题后,更新开发者在当前元数据下各个主题占据整体的比例,并依据元数据和主题信息共同推荐开发者.Yang 的方法中,通过过滤的方式选择候选者,可能忽略更为合适的开发者,并且难以区分相似的开发者.Xia 的方法,在多名开发者拥有相似能力时,倾向于将缺陷报告分派给所有修复记录中修复最多的开发者(修复越多的人,占有的比例越高).本文的方法,提出了开发者活跃度的概念,在多名开发者能力相近时,更倾向于选取当前最活跃的开发者.不仅能够帮助区分能力相近的开发者,而且有助于又快又好的修复缺陷报告.

研究者们还尝试使用其他信息进行自动缺陷分派.传递图(Tossing Graph)描述了当前开发者无法修复缺陷,并将缺陷报告传递给其他开发者的有向图.Jeong 等人利用传递图,以优化分派的精度[10].Bhattacharya 等人在 Jeong 等人的基础上进行优化,通过分析包含多种属性的传递图,进一步优化了推荐结果[11].还有一部分工作,结合了项目的代码注释和提交信息.这些方法通过建立程序片段同缺陷信息的联系,进一步挖掘版本控制系统的提交文本、代码注释,以找到合适的开发者[15,16,17].本文工作更侧重于使用缺陷报告本身的信息,获取更好的缺陷分派结果.

2 预备知识

2.1 循环神经网络

循环神经网络(RNN)是专门用于处理序列数据的深度学习模型,相比与普通神经网络的各计算结果之间相互独立的特点,RNN 的每一次隐含层的计算结果都与当前输入以及上一次的隐含层结果相关.通过这种方法,RNN 的计算结果便具备了记忆之前几次结果的特点.

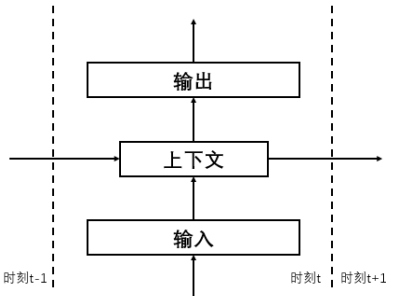


Fig.2 RNN Structure[21]

图 2 RNN 的结构[21]

RNN 是在有序的数据上进行学习的,但是 RNN 的结构决定了它只能对距离比较近的时刻的记忆更加强烈,而对距离久远的时间记忆较为模糊.长短期记忆网络（LSTM,Long Short-Term Memory）[22]模型是一种 RNN 的变型,通过刻意的设计来避免长期依赖问题,记住长期的信息在实践中是 LSTM 的默认行为. LSTM 的特点是在 RNN 结构以外添加了各层的门节点.阀门有 3 类:遗忘门（forget gate）,输入门（input gate）和输出门（output gate）.这些门可以打开或关闭,用于将判断模型网络的记忆态（之前网络的状态）在该层输出的结果是否达到阈值从而加入到当前该层的计算中.

文本分类方法:oh-2LSTMp

结合池化的双向 RNN 方法 oh-2LSTMp 是当前一种效果很好的文本分类方法.该方法采用 LSTM 记忆长期依赖,并使用了正向、逆向两个 LSTM,从不同的方向接受文本的输入,避免了单向 RNN 中,越靠后的单词对结果的影响越大的特点.图中红色代表了正向输入,蓝色代表了逆向输入.输入的文本按照一位有效码（one-hot,详见 3.3）的方式进行编码,减少了预先对单词进行嵌入（embedding）消耗的时间.除此以外,为了缓解从较长文本抽取特征时,LSTM 需要记忆整个文本中关键信息的压力,方法结合了池化（pooling）操作.每一个输入 LSTM 单元的单词,LSTM 都会产生一个输出,通过收集文本中每一个单词的输出,并通过池化的方式将这些结果聚集到同一个描述文本的向量,一方面获取了对文本整体更加全面的描述,另一方面也减轻了 LSTM 的学习负担.

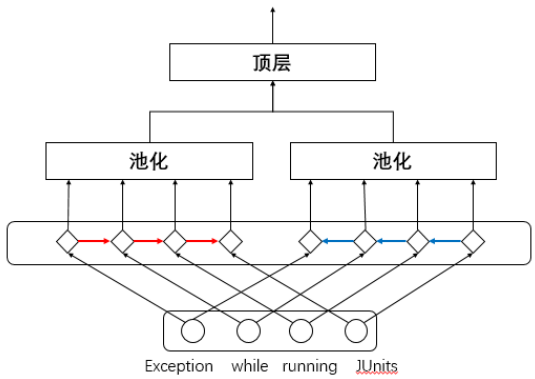


Fig.3 oh-2LSTMp Structure[7]

图 3 oh-2LSTMp 的结构[7]

3 基于循环神经网络的缺陷分派

3.1 方法框架

本文所采用的缺陷分派方法的流程如图 4 所示.方法分为训练和推荐两个阶段.在训练阶段,需要通过学习历史已经包含修复开发者的缺陷报告建立模型.在推荐阶段,向模型输入新的、未分派的缺陷报告,模型将输出推荐的开发者列表.

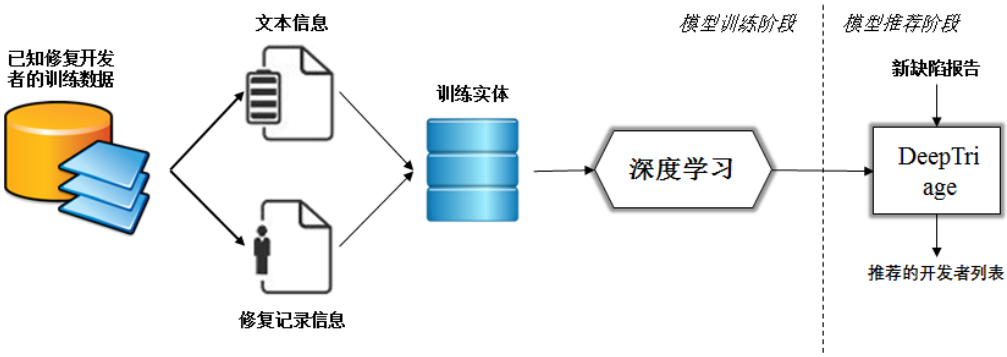


Fig.4 The Overall Workflow of DeepTriage

图 4 DeepTriage 的整体工作流程

在该流程中,对于缺陷报告,首先需要从中抽取原始信息.本文方法需求的原始信息包括：保留原始次序的文本信息,以及按时间顺序排列的开发者修复记录信息.随后,这些原始信息构成了训练实体,并将被输入到一个

深度神经网络中.在这个神经网络中,原始信息转换为高层特征,已知的修复者为当前特征的标签通过神经网络的训练会得到一个预测模型.在推荐阶段,给定一个新的缺陷报告时,将其输入到模型当中便可以得到一个该缺陷报告的推荐修复者列表.

在整个流程中,最为核心的技术就是模型的构造方法,下面将介绍模型的构造与学习方式.

3.2 基于循环神经网络的缺陷分派模型

缺陷分派的目标是为缺陷报告推荐合适的开发者.因此,模型的主要目标是完成从缺陷报告信息到开发者的映射.本文方法所使用的信息包括文本信息和修复记录信息.其中,文本是缺陷报告的重要组成部分,以自然语言的形式描述了缺陷报告的相关信息,可以从缺陷报告内容中获得;修复记录信息则是同一组件、模块下,一段时间内修复缺陷开发者的序列,用来建模开发者的活跃程度,可以通过获取当前缺陷报告的组件、模块信息,并查询缺陷追踪系统获得.假设 N 为训练样本中的缺陷报告总数,令$\mathrm{T}=\{\mathrm{t}_1,\mathrm{t}_2,\ldots,\mathrm{t}_\mathrm{N}\}$表示文本信息的集合,其中$\mathrm{t}_\mathrm{i}$表示第 i 个缺陷报告所对应的词语序列,又令$\mathrm{H}=\{\mathrm{h}_1,\mathrm{h}_2,\ldots,\mathrm{h}_\mathrm{N}\}$表示修复记录信息的集合,其中$\mathrm{h}_\mathrm{i}$表示第 i 个缺陷报告出现的时候之前一段时间内的修复行为,最后令$\mathrm{D}=\{\mathrm{d}_1,\mathrm{d}_2,\ldots,\mathrm{d}_\mathrm{N}\}$表示每个缺陷报告的真实修复的开发者,其中多个缺陷报告可能被同一个人修复.那么,若采用分类的方式构造模型,可以理解为构造一个预测函数$\mathrm{f}:\mathrm{f}(\mathrm{t}_\mathrm{i},\mathrm{h}_\mathrm{i})\rightarrow d_i$,而此函数可以通过最小化目标函数的方式学习得到,目标函数表示为:

$$\min_f \sum_i L(f(t_i,h_i),d_i)$$

其中$\mathrm{L}(\cdot,\cdot)$代表损失函数,用于评估当前预测函数的预测值同真实值之间的差距.

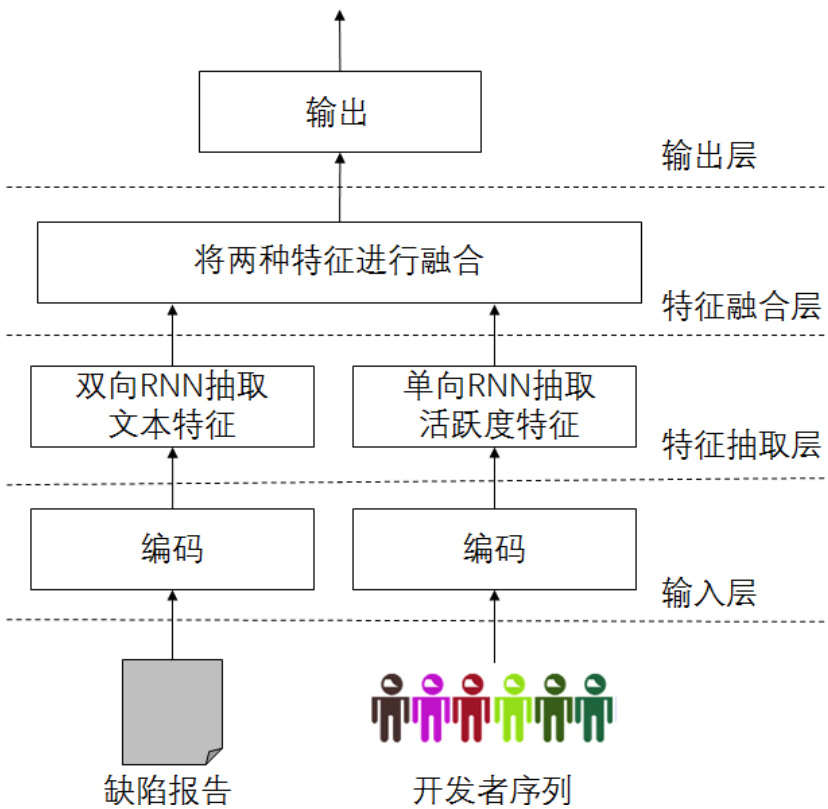


Fig.5 The Overall Workflow of DeepTriage

图 5 DeepTriage 的整体工作流程

本文给出了一个具体的预测函数（即模型）的构造方法,其使用两个循环神经网络,对输入的文本信息、修复记录信息进行高层特征抽取,并将其融合以共同预测合适的开发者.该模型的示意图见图 5.如图所示,模型包括输入层、特征抽取层、特征融合层与输出层.输入层完成输入原始数据的数字化建模,特征抽取层对文本信息和开发者活跃度进行高层特征的抽取,特征融合层将抽取得到的文本特征和活跃度特征进行融合,输出层预测适合于当前缺陷报告的开发者.下面几个小节将对模型的四个层进行分别介绍.

3.3 输入层

本文高层特征的抽取方式,采用了循环神经网络的文本处理方式,其中历史修复记录虽然不是文本,但其每一项（开发者）也是分类值,并且也按序排列,故文本处理的方式对历史修复记录信息同样适用.为了将原始数据输入到神经网络中,可以用嵌入（embedding）的方式学出一个等长的向量[23],也可以用一位有效码（One-Hot）的方式.由于一位有效码在文本处理中的有效性[7,20],且能够节省嵌入方式学习的时间,因此本文采用一位有效码的方式对原始数据进行输入.即使用 K 位非零即一的状态向量来对 K 个状态进行编码,并且在任意时候,其中只有一位有效.比如 eclipse 编号为 496596 的缺陷报告的标题为“intersection lambda type incorrect”,假设词汇表为{“java”,“incorrect”,“intersection”,“lambda”, “type” },那么该句的一位有效码可以表示为:

$$x = [[0\ 0\ 1\ 0\ 0], [0\ 0\ 0\ 1\ 0], [0\ 0\ 0\ 0\ 1], [0\ 1\ 0\ 0\ 0]]$$

在将原始输入进行编码后,令X_i^t表示第 i 份缺陷报告的文本信息,X_i^h表示第 i 条历史记录信息,Y_i^d表示第 i 份缺陷报告的真实修复的开发者.

3.4 特征抽取层

高层特征的抽取,使用了循环神经网络的方式,为了抽取出高层特征,使用了两种不同类型的 RNN.如图 6 所示,左侧为文本高层特征的抽取方式,右侧为活跃度高层特征的抽取方式.以下分两部分依次说明.

文本高层特征抽取,使用了文献[7]中提出的方法 oh-2LSTMp.图中,最下层的圆形表示单词,其上方的菱形表示 LSTM 单元（cell）,LSTM 单元对应两个输出,一个是表示当前的结果,即图中的向上箭头;另一个表示 LSTM 的内部状态,该状态会传递给随后的 LSTM 单元,即图中的向右箭头.再上的长形方块表示最大池化（max pooling）层,该层接收 LSTM 单元的输入,并对输入向量的每一维度,输出该维度上最大的值.以一位有效码表示的单词,按照顺序输入到 LSTM 中.对于每一个单词,LSTM 会给出一个输出,这些输出被送入最大池化层,并得到

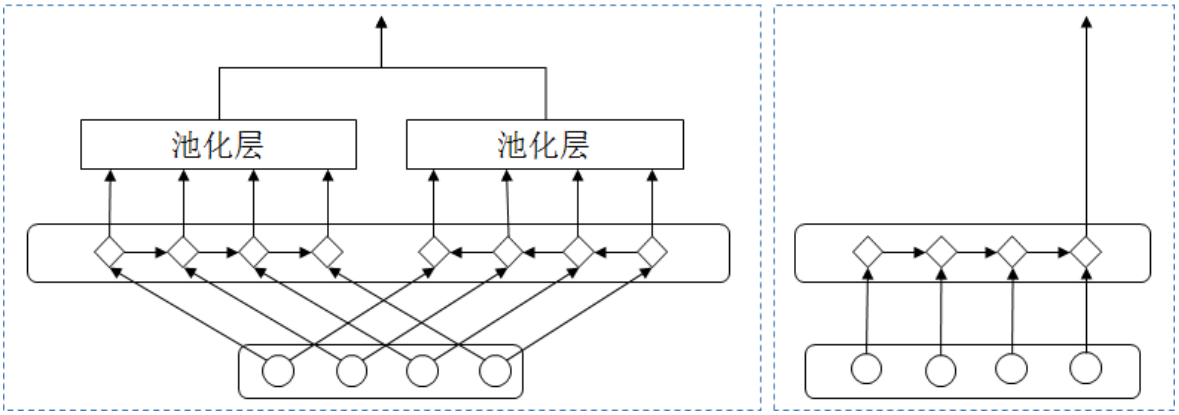


Fig.6 Higher Feature Extraction

图 6 高层特征的抽取

保留了最关键信息的特征向量.单词的输入分为正向和逆向两个过程,更加全面的对文档特征进行了抽取,池化后得到的两个特征向量通过拼接,作为当前文本的高层特征.

抽取活跃度的高层特征,主要包含两个部分.首先,获取开发者的修复记录.其次,将修复记录输入到神经网

络中,以得到高层特征.本节将依次对两部分内容进行说明.

从缺陷报告获取开发者的修复记录,涉及多份缺陷报告.一方面,方法需要获取开发者近期的活跃程度,因此只需获取当前时间节点向前一段时间内的修复记录.另一方面,由上文所述,产品和模块信息对选取开发者有着很好的指导意义,因此抽取同当前缺陷报告相同产品、模块下的修复记录,能够获得更好的特征.因此,需要查找同当前报告产品、模块信息相同,且向前推移一段时间内的修复记录,并将这些修复记录按照时间从远到近的顺序排列.若无法找到相应的修复记录,则将修复记录列表置空.

本文使用 RNN 作为活跃度的高层特征抽取方式.对于活跃度,期望较近期的修复记录影响较大,较早期的修复记录影响较小.对于单向 RNN，越靠前的输入对最终输出结果的影响越小，若取最后节点的输出作为高层特征，刚好符合活跃度直观的需求。其结构如图 6 右侧所示,圆形表示修复的开发者,菱形表示 RNN 单元.与文本的高层特征抽取不同,活跃度更加关注最终的活跃情况,故取最后一个 RNN 单元的输出作为活跃度的高层特征.

3.5 特征融合层

对于多种高层特征,需要将它们融合在一起,才能进行后续的工作.在文献[24]中,给出了三种较为简单的高层特征融合方法,分别为拼接、元素间相加,元素间相乘.由于高层特征皆为向量形式,采用拼接的方式,能够得到更长的高层特征向量,这种方式在联结过程中,高层特征不会相互作用.元素间加、乘的方式,则使得高层特征相互作用,加法的影响程度相对较小,乘法的影响程度则较大.

随后的实验表明,三种方法所产生的结果差别较小,但元素间相乘会有略微更好的准确率,故方法最终采用元素间相乘的方式,对高层特征进行融合.

3.6 输出层

本文采用分类的方法来选取合适的开发者,即每一个开发者都是一个标签.为了完成高层特征到标签的转换,本文选用了一个全连接层进行分类,以及一个归一（softmax）层对结果进行归一化.经历了原始信息到高层特征的转换,处于隐层特征空间的高层特征需要进一步映射到样本标记空间,全连接层能够实现这一过程.随后的归一层,则将输入转换为标签的概率,所有概率之和为 1.

分类的损失函数,采用了交叉熵作为损失函数.

$$\mathrm{Cost} = -\frac{1}{n}\sum_{x} y ln y' + (1-y)\ln(1-y')$$

其中,x 表示输入,即本例中的单词序列和开发者活动序列;n 代表数据总数;y 表示期望的输出,即本例中真实修复的开发者;y'表示神经网络实际的输出.

对于训练数据,真实修复的开发者是已知的,采用一位有效码的方式进行表示.对于预测阶段,则会得到一个开发者适合程度的分布,对该分布进行从大到小的排序,便可以作为适合的开发者列表.

3.7 模型的训练与预测

模型主要分为两个阶段,训练阶段通过从已知修复开发者的数据中学习,通过最小化目标函数的方式,不断优化模型参数.预测阶段,模型的参数将保持不变,输入一条新的、未修复的缺陷报告,模型将会输出合适的开发者列表.

本文使用了 AdamOptimizer[19]对模型参数进行优化,该方法可以动态的调节学习率,使模型更好的收敛.此外,本文采用了 Dropout[18]方法,通过每次训练时随机的让网络中的某些节点不工作,以避免过拟合现象的出现,进一步提高预测阶段效果.在训练阶段,dropout 的比率被设置为 0.5,而在预测阶段,则不使用 dropout,让模型发挥全部效果.

4 实验验证

4.1 实验对象

本文选择了四个规模较大、且持续维护的开源项目作为实验对象.项目的名称分别为：Eclipse、Netbeans、Mozilla、Redhat.所有缺陷报告皆从其对应的缺陷追踪系统获取,并且同过往方法[5]一样, 限定收集已经确认修复了的缺陷报告（即解决方案为 FIXED,报告状态为 CLOSED、RESOLVED、VERIFIED 的报告. Redhat 的管理方式不同,其缺陷追踪系统并没有设置 FIXED,取而代之的是 CURRENT_RELEASE）.

为了进行实验,需要从每一份缺陷报告中抽取修复的开发者,文本信息（标题、详细描述）,开发者活动序列信息.文本信息通过分句、分词、词根化,并且过滤掉停词.同时排除了修复次数少于 10 次的开发者以减少噪声[5,9,10,11]. 另外,删除了出现次数过多（超过 50%文档出现）、过少的词（小于 10 次）,以减少噪声并加速模型执行速度.单词会保留文本中的原始顺序,并转换为一位有效码形式（实现中,为了减少内存开销,单词是在被输入模型时,才被转换）.开发者活动序列信息收集了同当前报告的产品和模块信息相同的开发者活动序列,最大的记录报告数为 25 条（如果超过上限,则取最近的 25 条）,时间区间为当前时间节点向前推移 3 个月（观察发现 3 个月中 80%的开发者活动序列能达到 20 条以上）.

参照已有工作[5,12]处理方法,本文将被分派者（assigned to）的标签视为报告的修复者.并且同文献[5]的观察一致, 存在大量的缺陷报告中发现了无效的开发者名（这类名称可能代表了团体名称,或者特殊含义,如 Eclipse 中 AJDT-inbox 、 Platform-UI-Inbox 、 Eclipse Management Organization,Netbeans 中 issues@ide 、 issues@platform,Mozilla 中 MSU Capstone Team,Redhat 中 RHUI Bug List、RHOS Maint 等）,这些特殊含义的名称,只对于项目的从属人员有一定的意义.由于这些名称并非真实的开发者,因此实验剔除了这些缺陷报告.

Table 2 Statistics of Collected Bug Reports

表 2 缺陷报告的统计量

名称	时间区间	缺陷报告数	词汇表大小	开发者数	剩余缺陷报告数
Eclipse	2008.1~2016.12	166,081	12,916	1015	115561
Netbeans	2008.1~2016.9	64,851	8,146	219	58876
Mozilla	2008.1~2014.6	263,285	17,331	1426	208402
Redhat	2008.1~2017.1	108,685	14,148	1245	90012

表 2 列出了每一实验对象的基本数据.表格每一列的含义如下：项目名称,收集缺陷报告的起止时间,收集到的缺陷报告的总数量,参与修复的开发者数量,在删除了停词以及出现次数过少的词汇后不同单词的数量（总词汇量）,无效名称所占的百分比,过滤掉无效名称以及修复次数过少的开发者后剩余的报告数量.

4.2 实验设置

为了模拟现实中的场景,本文采用时间顺序的方式将数据集分割[5,11,12].首先,将每个项目的缺陷报告按照提交时间进行排列,然后将它们分割为没有重叠且尺寸相同的 11 个窗口,如此得到 10 叠符合真实应用的训练、测试数据.即对于第一叠数据,使用第一个窗口的数据进行训练,第二个窗口的数据进行测试;对于第二叠,使用前两个窗口的数据进行训练,第三个窗口的数据进行测试.以此类推,每个项目含有 10 叠数据以供验证.

实验采用准确率作为评估指标,即预测命中数占所有测试数据的比例.参照文献[5,12],实验同时考察了 Top-1 和 Top-5 的准确率,即真实修复的开发者是否在推荐的第 1 或者 5 条之内.对于 10 叠数据,每一叠数据的实验结果会存在差异,最终采用平均的准确率作为评估指标,即 10 叠执行的结果的平均值.

本文代码采用 Tensorflow 实现,使用了 LSTM 单元的一种变种,GRU 单元.在实践中,GRU 单元效果同 LSTM 单元不相上下,但由于模型参数更少,在计算性能方面的优势较为明显.实验采用显卡进行加速,显卡配置为 GTX1080.对比方法中的 SVM 使用 libsvm[25]实现,使用了其提供的 python 接口.主题模型 LDA、MTM 使用 python 实现,基于 cvb0 方法[26],相较于原文中[5,6]使用的 Gibbs 采样[27],cvb0 方法采用空间换时间的方式,提高了运行速度,并且能够更快收敛.主题模型的参数设置,由于采用了 cvb0 方法,迭代次数降低为 100 次,其余参数设

置参考原始论文.对比方法皆通过 CPU 执行,运行的 CPU 为 intel i7.

4.3 研究问题

本文试图回答如下四个研究问题.

RQ1：对比基准方法,本文方法在准确率上是否提高?

首先,本文关注 DeepTriage 能否在四个实验对象上胜过一些基准方法.本文考虑如下三个基准对比方法.SVM[9]是目前在自动缺陷分派上取得较好效果的方法.Yang[6]的方法,通过元数据对候选的开发者进行了过滤. TopicMinerMTM[5]首先使用结合元数据（产品、模块）的主题模型,以监督的方式获得更好的主题分布,其次通过统计开发者在某产品、模块内修复过的各个主题的占比,刻画了在不同产品、模块下开发者对主题的熟悉程度,进而为新的缺陷报告推荐开发者.

RQ2：随着训练数据的增加,本文的方法会受到怎样的影响?

其次,本文关注训练数据的增加是否会影响方法的有效性.特别地,本文借用基于时间的 10 叠划分,查看 DeepTriage 在每一叠上的性能.

RQ3：高层特征的不同组合方式,对本文的方法有何影响?

如前文所述,高层特征存在着不同的组合方式,不同的组合方式对最终的结果有影响吗？为了观察这一问题,本文在其余条件不变的情况下,分别以拼接、元素间相加和元素间相乘的组合方式进行实验,并以平均的准确率作为评估的指标.

RQ4：开发者的活跃程度,对自动缺陷分派有帮助吗?

除了文本信息,本文方法使用的另外一项重要信息为开发者的活跃度.活跃度对于自动缺陷定位真的有帮助吗？为了验证这一问题,本文分别以加入活跃度、不加入活跃度的情况进行了对比实验,观察平均的准确率有如何的变化.

4.4 实验结果

RQ1：对比基准方法,本文方法在准确率上是否提高?

图 7 展示了本文方法同基准方法比较的结果.左、右两图分别展示了 top-1 和 top-5 的平均准确率.可以看到本文的方法相较于基准方法有着一定的提高.本文方法在 SVM 方法原有准确率上分别提升了 60%、70%（提升的相对百分比,如原有方法的准确率为 x,新方法的准确率为 y,提升幅度为 a,则有$x*(1+a)=y$.以下表述方式相同）;相较于 Yang 的方法,提升了 23%、17%;相较于 TopicMinerMTM 方法提升了 12%、4%.

可以看到本文方法在 Mozilla 数据集上效果较差,通过人工观察,发现这是由于 Mozilla 项目的人员流动较大,新加入的开发者数量较多.而本文采用的分类方法,在模型固定之后,无法预测新加入的开发者.对于其他对

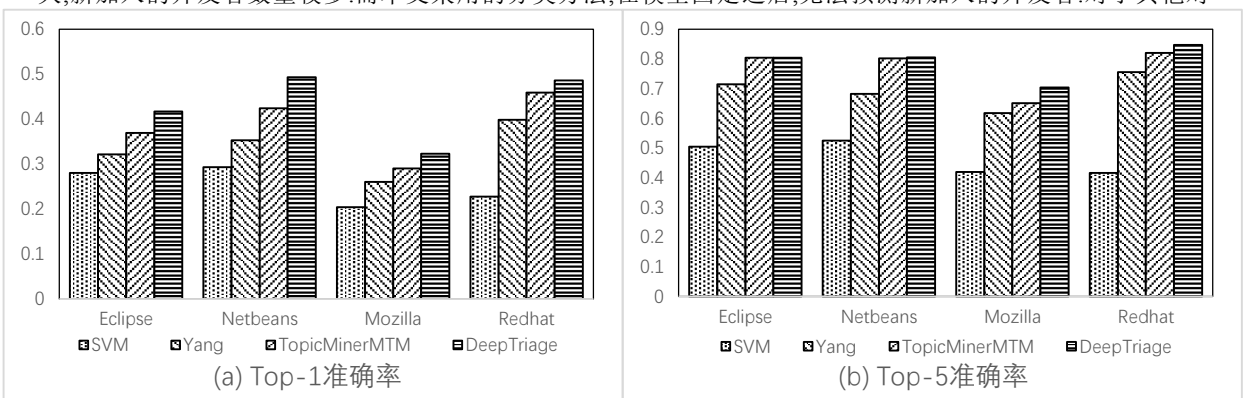


Fig.7 Top-1 and top-5 accuracies for DeepTriage with baseline approaches

图 7 同基准方法对比的 Top-1、Top-5 准确率

比方法,也存在着类似的问题.

另外,在 top-5 的准确率上,本文的方法同 TopicMinerMTM 差距很小,甚至某些情况下 TopicMinerMTM 的效果会略胜一筹.一方面原因在于此时准确率已经较高,由于无法预测新加入的开发者,故理论最大准确率要低于 100%,如 Eclipse 项目的平均最大命中率只有 91%;另一方面,TopicMinerMTM 能够在测试时,不断对开发者的状态进行更新,更容易预测到新加入的开发者.

在所有方法中,SVM 方法的结果较差,原因在于其只使用了文本信息.而很多开发者在缺陷的修复经验都很相似,只使用文本信息难以加以区分.Yang 同 TopicMinerMTM 的方法,由于考虑了元数据的信息,利用元数据对开发者进行了多一次的筛选,效果有所提升.本文的方法则在此基础上,建模了活跃度,更倾向将缺陷分派给当前更活跃的开发者,进一步提高了预测的准确率.

RQ2:随着训练数据的增加,本文的方法会受到怎样的影响?

为了模拟真实的使用环境,本文将实验数据按照时间顺序划分为 11 份,共进行按照时间顺序推进的 10 叠实验.图 8 采用折线图的方式,展示了每一个数据集上,每一叠数据在 top-1、top-5 的准确率上的变化.横坐标对应了当前的叠数,纵坐标对应了方法的准确率.从总体来看,本文方法整体上是优于基准方法的.

图中可以看出,在 10 叠数据中,各方法的整体准确率是较为平稳的,没有出现明显的随着数据增多准确率提升的情况.通过观察,发现随着数据的增多,开发者的数量也在不断的增加,在进行推荐时面临的干扰也增加了;同时,新的数据中还伴随着新的词汇,如新开发的组件、新引入的程序名称等,数据本身的复杂程度也在不断的增加.其中,对于开发者数量的增加,采用信息检索(IR)方式,能够更快的将新加入的开发者纳入候选列表,但新加入的开发者往往修复的缺陷较少,故难以将新缺陷报告分派给此类开发者;对于新词汇等内容的加入,当前的文本处理方法很难解决这一问题,因而在测试数据中出现这样的词汇,也只能将其过滤.综合以上两点,随着训练数据的增加,准确率也难以有明显的提高.

另一方面,在各个数据集中都会发现某一叠实验中准确率下降的情况,如 Eclipse 的第 3 叠(从 0 开始计数,故叠数编号为 0~9),Mozilla 的第 1 叠、第 6 叠.通过观察,发现其中主要原因在于,在该叠时间段内,开发者人员发生了较大的变动,而本文的方法难以预测新加入的开发者,故准确率有所下滑.

RQ3:高层特征的不同组合方式,对本文的方法有何影响?

表 3 展示了不同拼接方式对方法的影响.表格中 concat 代表拼接的方式,add 代表元素间相加的方式,mul 代表元素间相乘的方式,其后的数字代表了是 top-N 的准确率,如 1 代表了 top-1 的准确率.

由实验结果可见,元素间相乘的方式效果最好,因此方法最终选用了元素间相乘的拼接方式.另外也可以看出,组合方式对方法的影响较小.推测原因在于,神经网络的学习能力较强,能够适应不同的拼接方式,学习出符合该拼接方式的参数.

除此之外,采用另外两种高层特征的组合方式,效果仍然好于基准方法,只是提升力度有所下降.因而,本文高层特征的抽取是有意义的.

Table 3 Influence of Different Combing Methods

表 3 不同组合方式的影响

名称	concat-1	add-1	mul-1	concat-5	add-5	mul-5
Eclipse	0.4024	0.4095	**0.4165**	0.785	0.7918	**0.8038**
Netbeans	0.4817	0.49	**0.493**	0.7936	0.7914	**0.8047**
Mozilla	0.32	0.317	**0.3231**	0.7014	0.6994	**0.7112**
Redhat	0.465	0.462	**0.4721**	0.8365	0.8407	**0.8514**

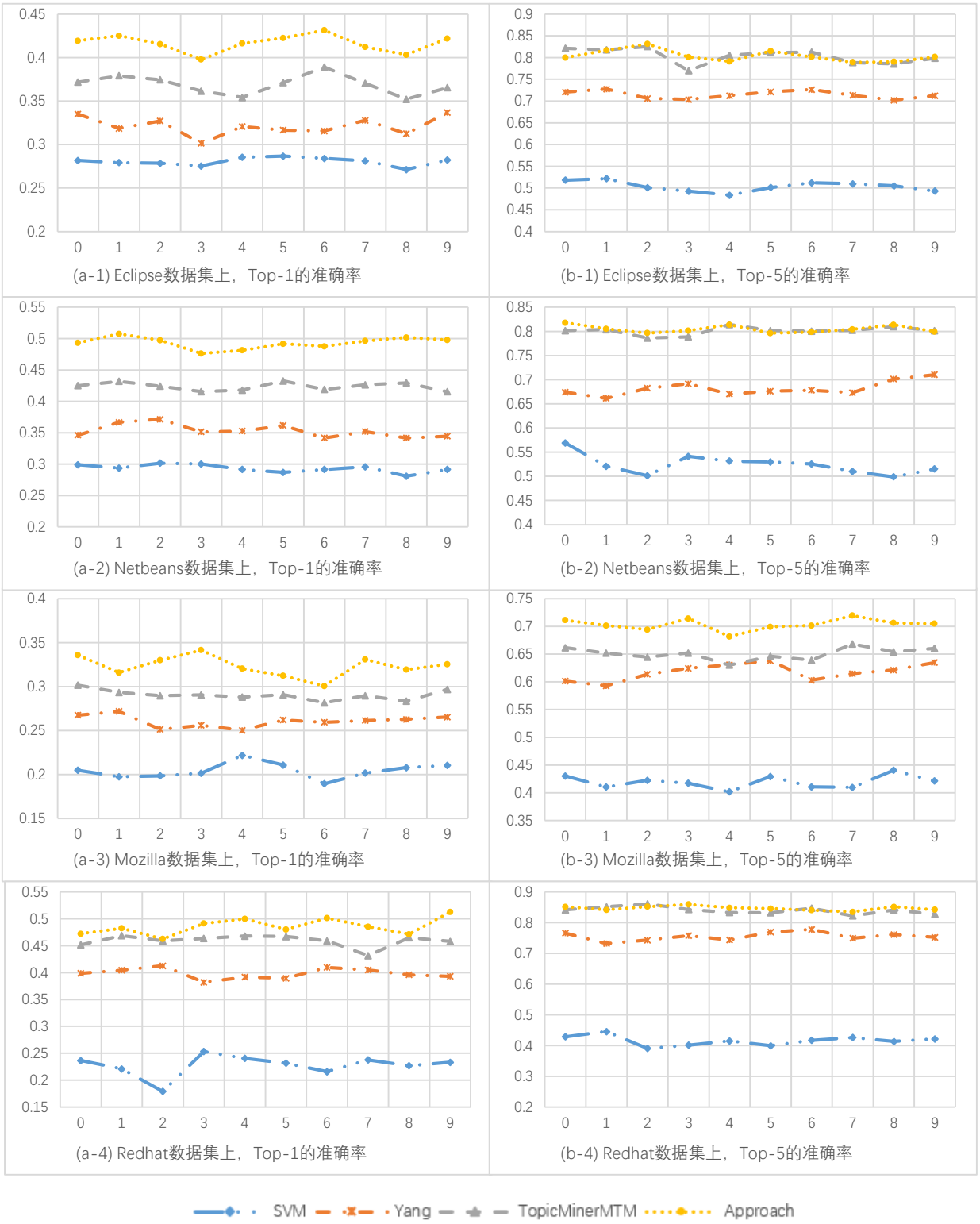


Fig.8 Top-1 and top-5 accuracies for Different Folds

图 8 实验数据集上 10 叠实验的 Top-1、Top-5 准确率

RQ4：开发者的活跃程度,对自动缺陷分派有帮助吗?

表 4 展示了在添加和去除活跃度时,方法的表现.表格中的 ws 代表了文本信息,active 代表了活跃度信息,表格的首行出现该字段说明使用了该项信息.其后的数字含义同表 3 相同,代表了 top-N 的准确率.因此,表格列从左到右依次为添加活跃度信息的 top-1 准确率,去除活跃度信息的 top-1 准确率,添加活跃度信息的 top-5 准确率,去除活跃度信息的 top-5 准确率.

可以观察到,添加了活跃度信息,不论在 top-1 准确率上、还是在 top-5 准确率上,都相较只使用文本信息有了较大的提升.因此,活跃度信息对自动缺陷分派很有帮助.

另外,通过对比只使用文本的结果,和基准方法中的 SVM 方法,可以发现其效果仍然优于 SVM 方法.

Table 4 Influence of Active Feature

表 4 活跃度的影响

名称	ws+active-1	ws -1	ws+active-5	ws -5
Eclipse	**0.41655**	0.31	**0.80385**	0.5013
Netbeans	**0.49301**	0.3195	**0.80469**	0.5204
Mozilla	**0.26069**	0.2204	**0.70337**	0.4256
Redhat	**0.39825**	0.2316	**0.84656**	0.4294

4.5 评估的有效性威胁

4.5.1 内部的有效性威胁

在实验设计方面.实验中,缺陷报告是按照时间排序的等数量划分.这种划分方式,同软件项目的开发进程是不同的.对于某一阶段的软件项目,其开发者和软件的功能是较为固定的.而随着版本的更新和演变,项目组的开发人员会发生变化,项目包含的专有名词也会发生变化（由程序或功能本身的演化导致）.若在测试集中将这部分信息纳入,一方面开发者和词汇的变化会严重影响方法的效果;另一方面,这种划分也不符合现实的应用场景,现实中会在项目发生重大变化之后,重新对模型进行训练.

另外,如果缺陷报告被重新打开（REOPENED）,它会重新经历缺陷报告的修复流程,因而相当于一份新的缺陷报告,此时用缺陷报告的创建时间进行排序,显然不够正确.

在基准方法方面.实验选取了三个基于缺陷报告本身,且较为有效的方法进行对比.在实现上,本文尽可能的将对比方法的效果做好,在主题模型的实现上,采用了 Gibbs 采样、CVB0 两种方式实现,并得到了相似的结果.在参数选择方面,本文一方面参考了原始论文中涉及的参数,另一方面也在参数设置上进行了一系列实验,以求达到其应有效果.

另外,本文使用了神经网络的方式,完成自动缺陷分派任务.然而,并没有同其他神经网络方法进行对比.一方面,据我们所知,目前还没有将神经网络应用到自动缺陷分派的方法.另一方面,本文使用了我们所知的,最先进的基于神经网络的文本分类方法,因此相信本方法在神经网络方法中也能取得很好的结果.

4.5.2 外部的有效性威胁

实验只收集了 4 个基于 Bugzilla 的大规模且持续维护的开源项目的缺陷报告,并且收集的缺陷报告也主要集中在近 10 年间.在未来的研究中,我们期望在更多的软件项目中进行实验,并且考查其他缺陷追踪系统下的缺陷报告,以更好的验证方法的有效性,并且试图寻找更好的自动缺陷分派方法.

5 总结与未来工作

本文提出了一种新的自动缺陷分派方法.在文本信息上,利用了单词间的次序信息,通过神经网络的方式抽取了文本的高层特征.对于开发者能力相似,难以区分的情况,本文观测到开发者不同时间在领域的活跃程度有所差异,故采用时间序列预测的方法,以同一产品、模块下开发者的修复记录为基础,抽取活跃度的高层特征.通过将两类高层特征进行组合,以分类的方式,共同对合适的开发者进行预测.实验表明,本文方法在准确率上要优于已有方法.

当前方法仍然存在着一些问题：首先,本文方法对新增开发者不够敏感.由于采用分类的方法,开发者被视为类别,若有新的开发者加入项目,必须重新训练模型.并且需要确保新加入的开发者已经解决了一定数量的缺陷报告,否则模型对新开发者无法得到很好的结果.其次,当前模型只考虑将缺陷报告分派给当下最合适的开发者,往往会倾向于将过多的缺陷报告分派给同一开发者.在这种没有考虑开发者负载的条件下,容易导致某一开发者积压大量需要修复的缺陷报告,进而拖慢整体的修复流程.在未来的工作中,我们会着眼于解决上述问题.

References:

[1] https://bugs.eclipse.org/bugs/

[2] https://netbeans.org/bugzilla/

[3] https://bugzilla.mozilla.org/

[4] https://partner-bugzilla.redhat.com/

[5] Xia X, Lo D, Ding Y, et al. Improving Automated Bug Triaging with Specialized Topic Model[J]. IEEE Transactions on Software Engineering, 2017, 43(3):272-297.

[6] 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]// Computer Software and Applications Conference. IEEE, 2014:97-106.

[7] Johnson R, Zhang T. Supervised and semi-supervised text categorization using LSTM for region embeddings[J]. 2016:526-534.

[8] D. Cubranic. Automatic bug triage using text categorization. In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering, 2004.

[9] Anvik J, Hiew L, Murphy G C. Who should fix this bug?[C]// International Conference on Software Engineering. DBLP, 2006:361-370.

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

[11] Bhattacharya P, Neamtiu I. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging[C]// IEEE International Conference on Software Maintenance. IEEE, 2010:1-10.

[12] Tamrawi A, Nguyen T T, Al-Kofahi J M, et al. Fuzzy set and cache-based approach for bug triaging[C]// Sigsoft/fse'11, ACM Sigsoft Symposium on the Foundations of Software Engineering. DBLP, 2011:365-375.

[13] Somasundaram K, Murphy G C. Automatic categorization of bug reports using latent Dirichlet allocation[C]// India Software Engineering Conference. ACM, 2012:125-130.

[14] Naguib H, Narayan N, Brügge B, et al. Bug report assignee recommendation using activity profiles[C]// Mining Software Repositories. IEEE, 2013:22-30.

[15] Huzefa K, Malcom G, Denys P, et al. Assigning change requests to software developers[J]. Journal of Software Maintenance & Evolution Research & Practice, 2012, 24(1):3-33.

[16] Linares-Vasquez M, Hossen K, Dang H, et al. Triaging incoming change requests: Bug or commit history, or code authorship?[C]// IEEE International Conference on Software Maintenance. IEEE, 2013:451-460.

[17] Shokripour R, Anvik J, Kasirun Z M, et al. Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation[C]// Mining Software Repositories. IEEE, 2013:2-11.

[18] Hinton G E, Srivastava N, Krizhevsky A, et al. Improving neural networks by preventing co-adaptation of feature detectors[J]. Computer Science, 2012, 3(4):pág. 212-223.

[19] Kingma D P, Ba J. Adam: A Method for Stochastic Optimization[J]. Computer Science, 2014.

[20] Kim Y. Convolutional Neural Networks for Sentence Classification[J]. Eprint Arxiv, 2014.

[21] Gu X, Zhang H, Zhang D, et al. Deep API Learning[J]. 2016.

[22] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[23] Dai A M, Le Q V. Semi-supervised Sequence Learning[J]. 2015:3079-3087.

[24] Allamanis M, Tarlow D, Gordon A, et al. Bimodal Modelling of Source Code and Natural Language[J]. 2015:2123-2132.

[25] Chang C C, Lin C J. LIBSVM: A library for support vector machines[J]. Acm Transactions on Intelligent Systems & Technology, 2011, 2(3):1-27.

[26] Hoffman M D, Blei D M, Bach F. Online learning for Latent Dirichlet Allocation[C]// International Conference on Neural Information Processing Systems. Curran Associates Inc. 2010:856-864.

[27] Asuncion A, Welling M, Smyth P, et al. On smoothing and inference for topic models[C]// Conference on Uncertainty in Artificial Intelligence. AUAI Press, 2009:27-34.