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

基于嵌入模型的混合式软件缺陷关联方法(智能化软件新技术)*

张 洋[1,2], 王 涛[1,2], 尹 刚[1,2], 吴逸文[1,2], 王怀民[1,2]

[1](国防科技大学 并行与分布处理重点实验室,湖南 长沙 410073)

[2](国防科技大学 计算机学院,湖南 长沙 410073)

通讯作者: 张洋, E-mail: yangzhang15@nudt.edu.cn

摘 要: 社交化编程使得开源社区中的知识可以快速被传播.其中,缺陷报告作为一类重要的软件开发知识,因为含有特定的语义信息,通常会被人工地与其他相关的缺陷报告关联起来.在一个软件项目中,发现并关联相关的缺陷报告可以为开发者提供更多的资源和信息去解决目标缺陷,从而提高缺陷修复效率.然而,现有人工关联缺陷报告的方法是十分耗费时间的,它很大程度上取决于开发者自身的经验和知识.因此,研究如何及时、高效地关联相关缺陷是对于提高软件开发效率十分有意义的工作.本文将这类关联相关缺陷的问题视为推荐问题,并提出了一种基于嵌入模型的混合式软件缺陷关联方法,将传统的信息检索技术 (TF-IDF) 与深度学习中的嵌入模型 (词嵌入模型和文档嵌入模型) 结合起来,用来自动化地关联相关的缺陷报告.实验结果表明,该方法能有效地提高传统方法的性能,且具有很强的应用扩展性.

关键词: 软件缺陷报告;信息检索;深度学习;嵌入模型;开源软件

中图法分类号: TP311

中文引用格式: 张洋,王涛,尹刚,吴逸文,王怀民.基于嵌入模型的软件缺陷关联方法.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Zhang Y, Wang T, Yin G, Wu YW, Wang HM. An embedding model based approach for linking related issues. Ruan Jian Xue Bao/Journal of Software, 2016 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

An Embedding Model based Hybrid Approach for Linking Related Issues

ZHANG Yang[1,2], WANG Tao[1,2], YIN Gang[1,2], WU Yi-Wen[1,2], WANG Huai-Min[1,2]

[1](Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha, 410073, China)

[2](College of Computer,National University of Defense Technology, Changsha, 410073, China)

Abstract: Social coding facilitates the sharing of knowledge in Open-Source community. In particular, issue reports, as an important knowledge in the software development, usually contain relevant information, and can thus be linked to other related issues manually. In a project, identifying and linking issues to potentially related issues would provide developers more targeted resource and information when they resolve target issues, improving the issue resolution efficiency. However, the current manual linking approach is in general time-consuming and mainly depends on the experience and knowledge of the individual developers. Therefore, investigating how to link related issues timely is a meaningful task which can improve development efficiency of Open-Source projects. In this paper, we formulate the problem of linking related issues as a recommendation problem and propose an embedding model based hybrid approach, combining the traditional information retrieval technique, i.e., TF-IDF, and the embedding models in deep learning techniques, i.e., Word Embedding and

* 基金项目: 国家自然科学基金(61502512, 61432020, 61472430, 61303064);国家重点研发计划(2018YFB1003903, 2016YFB1000805)

Foundation item: National Natural Science Foundation of China (61502512, 61432020, 61472430, 61303064); National Grand R&D Plan (2018YFB1003903, 2016YFB1000805)

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

Document Embedding. Our evaluation results show that, our approach can improve the performance of traditional approaches with a very strong application scalability.

Key words: software issue reports; information retrieval; deep learning; embedding models; open source software

社交化编程(social coding)最早由开源社区 GitHub 提出,旨在提供一个开发者友好的软件开发环境,帮助开发者进行高效的互联、协同、开发[1].社交化编程的出现极大地增强了代码复用[2]以及缺陷解决效率;开发者可以自主地参与报告和讨论缺陷.这样一来,缺陷报告,作为一类重要的软件开发知识,经常会被不同开发者在不同的时间报告;实践中经常会出现两个缺陷含有相关的信息,开发者可以在缺陷讨论过程中通过 URL 链接将相关的缺陷报告关联起来.

链接相关缺陷的过程对于开发者快速、有效地修复缺陷是十分重要的.当一个开发者提交一个新缺陷报告后,其他开发者会参与讨论并对这个缺陷的价值进行评价.在他们的讨论过程中,参与者可能会链接相关的其他缺陷报告来寻找可能有用的资源和信息.最终,受助于这些链接和讨论,缺陷被解决和关闭.然而,现实世界的链接过程需要耗费大量的时间和人力.尤其对于那些大规模的软件项目,开发者可能需要查找大量的历史缺陷数据,才能通过它们的文本描述信息定位到相关的缺陷,因为这样的基于人工的关联方法主要依赖于个体开发者的经验和知识.因此,设计并实现自动化的缺陷关联方法能够帮助开发者更加聚焦他们的缺陷解决,也能够帮助项目管理者获取缺陷间的关联关系,从而提高软件的开发、维护效率.

本文以开源社区 GitHub 中的软件项目为研究对象,主要聚焦 GitHub 中缺陷之间的自动化关联问题.因为 GitHub 是目前全球最大、最著名的开源社区,截止 2018 年 3 月,它已经托管了超过 8 千万的软件版本库.近年来,学术界也产生了大量基于 GitHub 数据开展的研究工作.一方面,在传统的缺陷跟踪系统中,例如 BugZilla,目前学术界和工业界已经出现了很多辅助开发者进行重复缺陷检测[3]、缺陷定位[4]、相似缺陷推荐[5]的方法.然而,针对 GitHub 缺陷跟踪系统的缺陷关联研究目前尚未出现.另一方面,现有相关工作提出的方法主要基于传统的信息检索(IR)技术,通过自动化地抽取给定缺陷报告的文本信息(由单词、句子组成)来搜索相关缺陷报告或文件,例如余弦相似度(cosine similarity)和 TF-IDF.然而传统的 IR 技术主要关注语料库中单词与单词之间的关系,忽略了单词的上下文语义信息.最近,一些工作在自然语言处理(NLP)领域开始利用深度学习技术[6,7,8],例如词嵌入(word embedding)模型,对单词的上下文信息进行刻画.并且,一些研究者也开始尝试利用深度学习技术来解决软件工程问题并取得了不错的结果[9,10,11].

在本文中,我们将关联相关缺陷的问题抽象为推荐(recommendation)任务,为了解决该任务,我们提出了一种基于嵌入模型的混合式软件缺陷关联方法.该方法将传统的 IR 技术(TF-IDF)与深度学习技术(词嵌入模型和文档嵌入模型)结合起来,能够自动化地为新缺陷推荐相关的缺陷报告.对于我们数据库中的每个缺陷报告,我们将其文本信息(标题和描述)抽取出来并保存到文档里.接下来,我们利用三个子方法(TF-IDF、WE 和 DE),分别表征每个缺陷文档的向量并计算他们之间的相似度得分.其中,TF-IDF 主要从词频的角度提取不同单词间的关联关系,WE 和 DE 则分别从单词和文档出现的上下文角度提取单词间以及文档间的关联关系.最后,我们将三个子方法得到的相似度得分相加得到最终得分,进而基于该得分对相关缺陷进行推荐.

为了评价本文方法的有效性,我们选取了两个著名的 GitHub 项目作为研究对象:Request 项目和 Moment 项目.在实验验证过程中,我们将本文方法和四个基准方法(baseline approach)进行比较,包括 NextBug[5],以及我们的三个子方法.具体来说,我们设计了以下四个具体研究问题:

RQ1: 和基准方法相比,本文方法在性能上可以有多少提升?

RQ2: 本文方法对于缺陷语料库是否有很强的依赖性?

RQ3: 本文方法对于训练集规模的依赖如何?

RQ4: 本文方法的训练时间开销如何?

实验结果表明,基于嵌入模型的混合式软件缺陷关联方法充分地考虑了软件缺陷的词频和上下文信息,可以明显地提高基准方法的性能,并且对于不同的缺陷语料库,本文方法的性能仍然具有一定的可靠性,且本文方法的训练时间开销较小,具有很高的应用扩展性.

本文的组织结构如下：本文第 1 节对相关研究背景进行介绍,结合相关工作引出本文研究问题及方法.第 2 节详细介绍本文提出的方法的框架和技术细节.第 3 节对实验验证的方法和评价指标进行介绍.第 4 节介绍具体研究问题下的实验结果.第 5 节介绍案例分析.最后总结全文.

1 研究背景

为了更好地阐述本文工作,我们首先对相关背景知识和相关工作进行介绍.

1.1 语义相似度与嵌入模型

测量两个文本的语义相似度是一类经典的研究问题,现有的解决模型也从向量空间模型(例如 TF-IDF)、n-gram 语言模型、主题建模(例如 LDA[12])向更多的基于人工神经网络的语言模型[7,8,13]发展.在 2013 年,Mikolov 等人[7,8]提出了两个基于人工神经网络的语言模型,分别是 continuous bag-of-words 和 continuous skip-gram,并针对大规模文本数据提出了有效的负采样(negative sampling)方法.

词嵌入模型(word embedding)[7,8]是自然语言处理领域非常著名的深度学习模型,它主要将每个单词映射到 n 维的向量空间.词嵌入模型主要基于这样的假设,即出现在相似上下文的单词间通常会有相似的语义.因为表征单词的每个维度都代表着单词的语义或语法特征,因此越相似的两个单词会在向量空间中具有越近的向量距离.

文档嵌入模型(document embedding)[6]旨在对词嵌入模型进行扩展以实现更高级别(文档级)的向量表示.在其训练过程中,每个文档都映射到一个唯一的向量,由矩阵中的列表示,每个单词也映射到一个唯一的向量.然后对文档向量和单词向量进行平均或连接以预测上下文中的下一个单词.

由于词嵌入模型和文档嵌入模型都是从未标记的数据中学习的,因此可以适用于没有足够标签数据的任务.此外,嵌入模型更侧重于单词和文档的上下文,所以可以很好地补充传统的 IR 技术,例如,TF-IDF.最近,一些研究表明嵌入模型在软件工程任务中也表现良好[9,10].例如,Xu 等人[9]将预测语义可链接的知识单元问题(即 Stack Overflow 中帖子及其回答)抽象为多类别分类问题,并使用词嵌入模型和 CNN 解决该问题.Ye 等人[10]提出,可以通过将自然语言语句和代码片段作为共享表示空间中的含义向量进行投影,来弥合词汇差距.他们在 API 文档,教程和参考文档上训练了词嵌入模型,然后将它们聚合在一起,来计算文档之间的语义相似性.这些研究的提出促使了我们将嵌入模型整合到我们的方法中.

1.2 缺陷修复和重复缺陷检测

在缺陷跟踪系统中,缺陷修复一直是一个复杂、需要不断迭代且耗时的过程,它涉及整个开发人员社区,从而引发特定的协同问题[14].已有对于缺陷解决过程的研究主要关注缺陷分类[15]和推荐相关开发者[16].其中,Guo 等人[17]分析了缺陷修复过程如何受人们声誉、缺陷报告编辑活动以及地理和组织关系的影响.Zimmermann 等人[18]使用混合式方法来表征缺陷 re-open 的过程,并定量评估了各种因素的影响.最近,对缺陷修复过程的研究扩展到去链接其他相关资源,例如,相关代码提交[19]、相关文件[20]和来自问答网站的相关帖子[21].具体来看,Bachmann 等人[19]提出了一个名为 Linkster 的工具,以便于识别缺陷与代码提交之间缺失的链接信息.Ye 等人[20]引入了一种自适应排序方法来关联相关的领域知识,它主要通过功能性地分解源代码文件、代码中使用的库组件的 API 方法描述、缺陷修复历史等.我们之前的研究[21]也探讨了缺陷严重性与链接 Stack Overflow 问答网站帖子的大众属性之间的相关性.

为了帮助修复缺陷,许多研究开始研究重复缺陷检测问题[3,22].例如,Wang 等人[22]通过分析缺陷报告和执行信息中的文本信息,来检测两个缺陷报告是否相互重复.Sun 等人[3]提出了一种信息检索方法 REP 来识别重复的缺陷报告.REP 考虑了缺陷报告中提供的更多信息,并扩展了 BM25F 方法以更准确地测量文本相似性.最近,一些研究人员开始研究相似缺陷报告推荐的问题,例如 Rocha 等人[5]提出了一种名为 NextBug 的推荐方法,依靠传统的信息检索技术来计算缺陷报告之间的余弦相似度.他们的结果显示,用于相似缺陷推荐的 NextBug 在性能

上优于之前的重复缺陷报告检测技术 REP.在本文后续实验中,我们也将 NextBug 作为基准方法之一,用以参照比较本文方法的性能.

1.3 GitHub中的缺陷关联问题

在 GitHub 中,每个托管的项目都与一个缺陷跟踪系统相关联,该系统提供缺陷跟踪过程中的常用功能,例如提交缺陷报告、根据缺陷的性质进行标记、标记缺陷不同时间点的解决状态等.GitHub 中的缺陷解决过程是一个协同且复杂的过程:在开发者提交新的缺陷报告后,其他开发者会参与讨论并对该缺陷进行评估,以确定是否值得解决,如果是,则优先考虑该缺陷的解决过程.关联的链接可以视为缺陷解决过程中的重要一步,因为在讨论过程中,利益相关者可能会在评论中链接其他相关缺陷报告,来提供他们认为有用的资源和信息.因为使用了 Flavored Markdown 技术,在 GitHub 平台上开发者可以轻松链接到相同项目的另一个缺陷报告(例如图 1 示例,在缺陷 *request/request#2397* 中,开发者 *tzachshabtay* 评论并给了一个链接,指向缺陷编号 234).最终,在这些链接和讨论的帮助下,该缺陷将被解决和关闭.

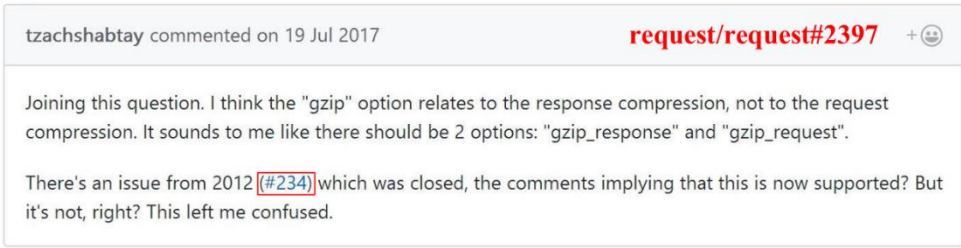

Fig.1 An example of issue linking in the GitHub project.

图 1 GitHub 项目中缺陷链接的示例

为了及时起到作用,关联相关缺陷报告的链接应该被迅速的发现和提出.但实际这些链接出现的时机主要取决于各个开发者的经验和知识,所以会产生很多不同.而现实世界的查找相关缺陷报告并进行链接的过程是非常耗时的,很大程度上依赖开发者人工的搜索和定位.例如,图 2 显示了 Moment 项目中实际链接相关缺陷报告的时间延迟分布(在我们统计中,时间延迟表示从缺陷创建到第一个链接出现的时间,以天为单位).我们可以发现,Moment 项目的开发者平均需要 37.9 天(中位数:2.8 天)才能链接到相关缺陷报告,并且箱图中有许多异常值,这表示在很多缺陷报告中,开发者需要更长的时间才能发现他们相关的缺陷报告.

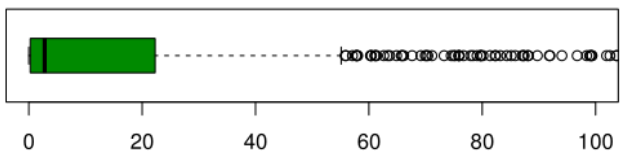

Fig.2 The distribution of link latency (days) in Moment project.

图 2 Moment 项目中链接延迟的分布统计(天)

因此,我们需要能够用于链接 GitHub 项目中相关缺陷的自动化工具.尽管在传统的缺陷跟踪系统中已经提出了许多用于链接相关缺陷资源的自动化方法[3,4,5],但据我们所知,目前还没有研究对 GitHub 项目中的缺陷自动链接问题进行过分析.通过本文,我们试图解决对这一类问题进行研究,并提出缺陷自动关联的初步方法.

2 方法概述

本文将相关缺陷自动关联问题抽象为推荐问题,即对于开发者给出的查询缺陷,我们的任务是自动化地推

荐与该缺陷相关的其它缺陷.下面我们将详细地介绍我们的方法框架和技术细节.

2.1 总体框架

图 3 显示了本文方法的整体框架.首先,我们从所有缺陷数据中提取它们的文本信息.然后,对于每个缺陷报告,我们将其标题和描述组合到一个文档中.接下来,我们使用以下步骤预处理这些缺陷文档:

步骤 1:从每个缺陷文档中提取所有单词;

步骤 2:删除停用词、数字、标点符号和其他非字母字符;

步骤 3:使用 Lancaster stemmer 技术[23]将剩余的单词转换为它们的根形式,以减少特征维度并将相似的单词统一为一个共同的表示.例如,像“interests”这样的复数名词将被替换为单数形式的“interest”,诸如“ate”,“eaten”和“eating”的等表示不同时态的形式将被替换为不定形式“eat”.

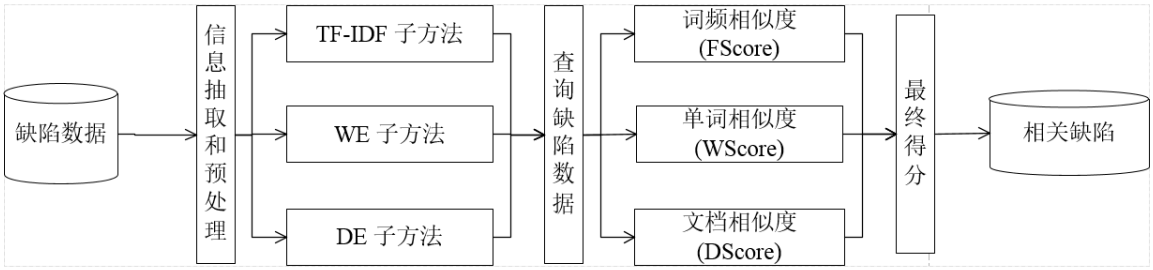

Fig.3 The framework of our approach.

图 3 本文方法的总体框架

基于预处理过的缺陷文档数据,我们分别使用三个子方法(TF-IDF,WE 和 DE)来计算查询缺陷和每个候选缺陷之间的相似性得分(FScore:词频相似度;WScore:单词相似度;DScore:文档相似度).虽然这三个得分都可以代表缺陷间的相似性,但它们之间是互相补充的,因为:

- FScore 是基于 TF-IDF 子方法生成的,它更关注缺陷之间的词频关系,主要考虑整个缺陷语料库中的出现的单词频率信息;
- WScore 是基于 WE 子方法生成的,它更侧重于单词的上下文信息;
- DScore 是基于 DE 子方法生成的,它更侧重于缺陷文档之间的相关性.

最后,我们将这三个分数相加得到每个缺陷对之间的最终相似度得分,并根据这一得分推荐相关缺陷.其中,得分越高说明两个缺陷报告之间的语义相似度越高.

接下来,我们详细介绍三个子方法的技术细节.

2.2 TF-IDF子方法

TF-IDF(Term Frequency-Inverse Document Frequency)是目前最流行的信息检索技术之一.图 4 表示了 TF-IDF 的简要工作流程,其主要思想是,如果一个单词在一个文档中出现多次而在其他文档中只出现很少次数,则该单词具有区分文档的良好能力,因此该单词具有高 TF-IDF 值.特别地,给定一个单词 t 和一个文档集合$D = \{d_1, d_2, \dots, d_N\}$,我们可以利用如下所示公式计算文档$d_i$中单词 t 的 TF-IDF 值:

$$Tf(t, d_i) = \frac{f(t, d_i)}{n(d_i)}$$

$$Idf(t, D) = log \frac{N}{n(d_i \in D; t \in d_i)}$$

$$TfIdf(t,d_i,D)=Tf(t,d_i)\cdot Tdf(t,D)$$

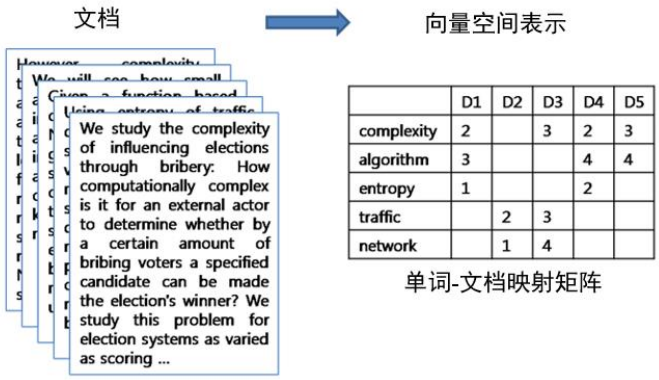

Fig.4 The brief process of TF-IDF sub-method.

图 4 TF-IDF 子方法的简要流程

其中,$f(t,d_i)$表示单词 t 出现在文档d_i中的频率,而$n(d_i)$表示文档d_i中的单词总数.之后,我们通过使用余弦相似度(cosine similarity)来计算两个文档的相似性,因为它是目前一种流行的计算相似度的方法,并且已经被证明非常适用于 TF-IDF 向量.因此,给定两个 TF-IDF 向量$\overrightarrow{V_{f1}}$和$\overrightarrow{V_{f2}}$,它们的词频相似度得分 FScore 计算方法如下:

$$FScore(\overrightarrow{V_{f1}},\overrightarrow{V_{f2}})=\frac{\overrightarrow{V_{f1}}\cdot\overrightarrow{V_{f2}}}{|\overrightarrow{V_{f1}}|*|\overrightarrow{V_{f2}}|}$$

2.3 WE子方法

Mikolov 等人[7,8]提出了一种流行的词嵌入模型,称为 continuous skip-gram,它可以有效地训练并支持各种语言任务的处理.因此在本文方法中,我们使用该 continuous skip-gram 模型来获取词嵌入向量.为了学习目标词的向量表示,该模型将上下文信息定义为围绕目标词的固定数量的词(如图 5 所示).

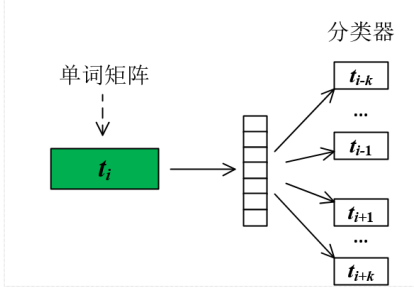

Fig.5 The brief process of WE sub-method.

图 5 WE 子方法的简要流程

因此,给定一个单词集合$T=\{t_1,t_2,\dots,t_n\}$和目标单词t_i,以及上下文窗口大小 k,Continuous skip-gram 模型的目标是最大化以下阈值L_t:

$$L_t = \sum_{i=1}^{n} logPr(t_{i-k}, \dots, t_{i+k}|t_i)$$

其中$t_{i-k}, \dots, t_{i+k}$是目标单词t_i的上下文.概率$Pr(t_{i-k}, \dots, t_{i+k}|t_i)$计算公式如下:

$$Pr(t_{i-k}, \dots, t_{i+k}|t_i) = \prod_{-k\le j\le k, j\neq 0} Pr(t_{i+j}|t_i)$$

其中,我们假设上下文单词和目标单词是相互独立的.此时,$Pr(t_{i+j}|t_i)$定义如下:

$$P\,r(t_{i+j}|t_i) = \frac{exp\left(\vec{t_i}\cdot\vec{t_{i+J}}\right)}{\sum_{t\in T} exp(\vec{t_i}\cdot\vec{t})}$$

其中$\vec{t}$和$\hat{\vec{t}}$是单词 t 的输入和输出向量,T 是所有单词的词汇集合.通过训练整个语料库,语料库词汇表中的所有单词可以表示为 δ维向量,其中 δ是可变参数并且通常被设置为诸如 200 的整数.

之后,我们将每个单词转换为固定长度的向量.这样,每个文档可以表示为一个矩阵,其中每行代表一个单词.考虑到不同的文档具有不同数量的单词,我们通过平均文档包含的所有单词向量将文档矩阵转换为一个向量.特别地,给定具有 n 行的文档矩阵,我们将矩阵的第 i 行表示为$\vec{r_i}$,并且利用如下公式生成变换的文档向量$\overrightarrow{V_t}$:

$$\overrightarrow{V_t} = \frac{\sum_{i=1}^{n}\vec{r_i}}{n}$$

最后,对于给定的两个文档向量$\overrightarrow{V_{t1}}$和$\overrightarrow{V_{t2}}$,我们同样使用余弦相似度来测量它们的单词相似度得分 WScore,计算公式如下:

$$WScore(\overrightarrow{V_{t1}}, \overrightarrow{V_{t2}}) = \frac{\overrightarrow{V_{t1}}\cdot\overrightarrow{V_{t2}}}{|\overrightarrow{V_{t1}}| * |\overrightarrow{V_{t2}}|}$$

2.4 DE子方法

为了构建文档嵌入模型,我们使用向量分布的 bag of words(PV-DBOW)方法[6],它作为 skip-gram 的实例,能够学习任意长度单词序列的向量表示,例如句子、段落,甚至整个大文件.图 6 显示了文档嵌入模型的简要流程.

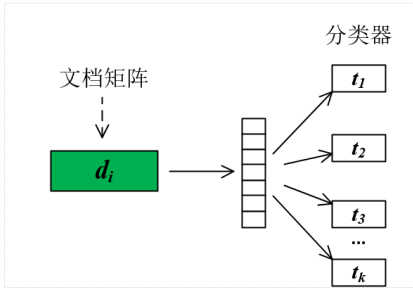

Fig.6 The brief process of DE sub-method.

图 6 DE 子方法的简要流程

具体地,给定一组文档$D = \{d_1, d_2, \dots, d_N\}$和从文档$d_i \in D$中采样的单词集合$T(d_i) = \{t_1, t_2, \dots, t_k\}$,模型可以学习出一个文档$d_i$和从$T(d_i)$采样的每个单词$t_j$的 δ维嵌入向量($\overrightarrow{v_{d_i}} \in R^{\delta}, \vec{t_j} \in R^{\delta}$).该模型通过考虑了在文档$d_i$

中出现单词$t_j \in T(d_i)$的频率并试图使下面的阈值L_d最大化:

$$L_d = \sum_{j=1}^{k} logPr(t_j|d_i)$$

其中,概率$Pr(t_j|d_i)$的计算公式如下:

$$Pr(t_j|d_i) = \frac{exp(\overrightarrow{v_{d_i}} \cdot \overrightarrow{t_j})}{\sum_{t_j \in T} exp(\overrightarrow{v_{d_i}} \cdot \overrightarrow{t_j})}$$

其中 T 是 D 中所有文档中所有单词的词汇集合.最后,给定两个文档向量$\overrightarrow{V_{d1}}$和$\overrightarrow{V_{d2}}$,我们计算其文档相似度得分 DScore 如下:

$$DScore(\overrightarrow{V_{d1}}, \overrightarrow{V_{d2}}) = \frac{\overrightarrow{V_{d1}} \cdot \overrightarrow{V_{d2}}}{|\overrightarrow{V_{d1}}| * |\overrightarrow{V_{d2}}|}$$

3 实验验证

为了评估本文方法的有效性,我们对 GitHub 项目进行了实证研究.下面我们分别从实验数据、Ground truth 构建、评价指标和实验设置四个方面进行阐述.

3.1 实验数据

我们选择了两个著名的 GitHub 项目作为我们的研究对象：Moment(一个轻量级的 JavaScript 日期库)和 Request(一个简化的 HTTP 请求客户端).我们选择它们主要有两个原因:

(1)它们非常著名,并且在 GitHub 上有很多 stars 和 forks;

(2)它们都是具有较长生命周期的活跃项目,拥有众多经验丰富的开发者以及大量历史数据.

表 1 总结他们的基本统计数据(截至 2018 年 4 月).我们可以发现它们都很早就在 GitHub 上托管并且有很多的缺陷报告数据(超过千个).

Table 1 Basic descriptive statistics of studied two projects.

表 1 两个研究项目的基本数据统计

项目	创建时间	Release 数	Star 数	Fork 数	Commit 数	缺陷报告数
Request	2011-01-23	134	18602	2217	2199	2875
Moment	2011-03-01	69	35644	5327	3619	4445

使用 GitHub API,我们收集了 2018 年 4 月之前两个项目的所有缺陷报告数据,包括缺陷 ID、缺陷状态(开放、关闭)、缺陷标题和描述等.

3.2 Ground truth构建

我们的 Ground truth 构建主要基于实际出现的缺陷 URL 链接,即如果两个缺陷报告之间存在 URL 链接,我们认为它们是相互关联的.具体来说,我们定义缺陷 A 和缺陷 B 是一对链接对,(A,B)或 A→B,当且仅当在缺陷 A 的讨论中包含链接到缺陷 B 的 URL 链接.在实际解析过程中,我们使用 GitHub 的“cross-referenced”API 来查找项目所有缺陷的链接信息.

请注意,在本文工作中,我们只考虑项目内部链接,暂不考虑跨项目的缺陷链接.最后,我们在 Request 项目中总共发现了 1110 个缺陷链接对,在 Moment 项目中发现了 2406 个缺陷链接对.

3.3 评价指标

为了评估我们方法的有效性,我们使用了三个评价指标:Top-k 召回率(*R@k*)、平均精度(*MAP*)、平均排序等级(*MRR*),它们通常被用于评估软件工程任务中的推荐系统性能[4,5,11].特别地,给定一个待查询缺陷 i,我们将其实际的缺陷链接集合定义为$Lk(i)$,并将推荐系统产生的 top-k 推荐集合定义为$R(i)$.下面我们介绍 *R@k*、*MAP* 和 *MRR* 的具体计算过程:

- *R@k*：其目的是检查 top-k 推荐结果是否正确.对于待查询缺陷 i,其 *R@k* 可以计算为:

$$R@k(i) = \begin{cases} 1, if\ Lk(i) \cap R(i) \neq \emptyset \\ 0, otherwise \end{cases}$$

- *MAP*: 它被定义为所有评估查询结果获得的平均精度值(*AvgPrec*)的平均值.单个待查询缺陷 i 的 *AvgPrec* 计算方法如下:

$$AvgPrec(i) = \sum_{n=1}^{N} \frac{Prec@k(i)}{N}$$

其中 *Prec@k* 是排名列表中 top-k 问题的检索精度,即 top-k 推荐中存在待查询缺陷 i 的实际缺陷链接数目的百分比:

$$Prec@k(i) = \begin{cases} \frac{|Lk(i) \cap R(i)|}{k}, if\ Lk(i) \cap R(i) \neq \emptyset \\ 0, otherwise \end{cases}$$

- *MRR*: 它被定义为所有推荐结果对应的倒数排序(Reciprocal Rank, RR)值的平均值,其中单个查询缺陷 i 的 RR 值是第一个正确答案的排序$first_i$的倒数:

$$RR(i) = \begin{cases} \frac{1}{first_i}, if\ Lk(i) \cap R(i) \neq \emptyset \\ 0, otherwise \end{cases}$$

3.4 实验设置

在我们的实验中,我们将数据集内拥有实际链接的所有已关闭缺陷都视为待查询缺陷.给定一个待查询缺陷 i,我们按照与 i 的相似性得分大小将所有待判断缺陷进行排序,从而推荐 top-k 个最相似的缺陷.注意的是,我们只推荐在待查询缺陷 i 产生之前出现的缺陷,即他们的缺陷 ID 必须小于 i 的 ID.在具体代码中,我们主要使用 Python 的 Gensim 包来实现我们的嵌入模型方法.特别地,对于 WE 和 DE 子方法,我们将上下文窗口大小 s 设置为 5,将初始学习速率 α 设置为 0.025,并且将表征向量 d 的维度设置为 200.

4 实验结果

在本节,我们介绍具体的实验结果.根据设计的三个研究问题,我们分别对每个研究问题的研究动机、分析方法、分析结果进行详细阐述.

4.1 RQ1: 和基准方法相比,本文方法在性能上可以有多少提升?

4.1.1 研究动机

在第一个研究问题中,我们试图调查本文方法在缺陷自动化关联任务中的有效性.此外,由于本文方法结合

了 TF-IDF 和深度学习技术来计算两个 GitHub 缺陷文本之间的语义相似性,我们希望分析这种混合式方法是否以及在多大程度上可以改善现有的方法和我们的三个子方法的性能.

4.1.2 分析方法

因为目前并没有专门针对 GitHub 缺陷自动关联问题提出的方法,所以我们将本文方法与传统缺陷跟踪系统中最先进的方法(NextBug[5])进行比较.NextBug 主要基于传统的 IR 技术,依靠余弦相似度得分来推荐相似的缺陷报告.此外,为了验证我们的混合式方法的优越性,我们将本文方法与我们的三个子方法(TF-IDF、WE 和 DE)进行比较.因此,我们在实验中设置了四个作为参考比照的基准方法:NextBug、TF-IDF、WE 和 DE.通过分别使用本文方法和基准方法对 Request 项目和 Moment 项目缺陷数据进行训练、测试,我们比较了不同方法在 *R@1*、*R@5*、*R@10*、*MAP* 和 *MRR* 评价指标值上的差异.

4.1.3 分析结果

(1) *Request 项目*:图 7 给出了在 Request 项目缺陷数据中,进行相关缺陷推荐的性能比较结果.

我们发现本文方法可以实现较高的推荐性能,其中 *R@1* 值为 27.7%,*R@5* 值为 51.2%,*R@10* 值更是达到 60.6%,即当推荐 10 个候选缺陷时,有 60.6%的概率候选缺陷即为真正的实际关联缺陷.

其次,我们发现本文方法在所有评价指标上都优于 NextBug.相对于 NextBug 的性能值,本文方法在 *R@1*、*R@5*、*R@10*、*MAP* 和 *MRR* 上分别提升了 51.4%、56.1%、42.9%、53.0%和 52.0%.

此外我们发现,与我们的三个子方法相比,本文方法在所有指标方面也实现了更好的性能.相对于三个子方法性能值,本文方法分别在 *R@1*、*R@5*、*R@10*、*MAP* 和 *MRR* 上提升了 9.1-28.8%、15.6-40.3%、9.4-19.0%、10.9-34.2%和 10.8-33.5%.

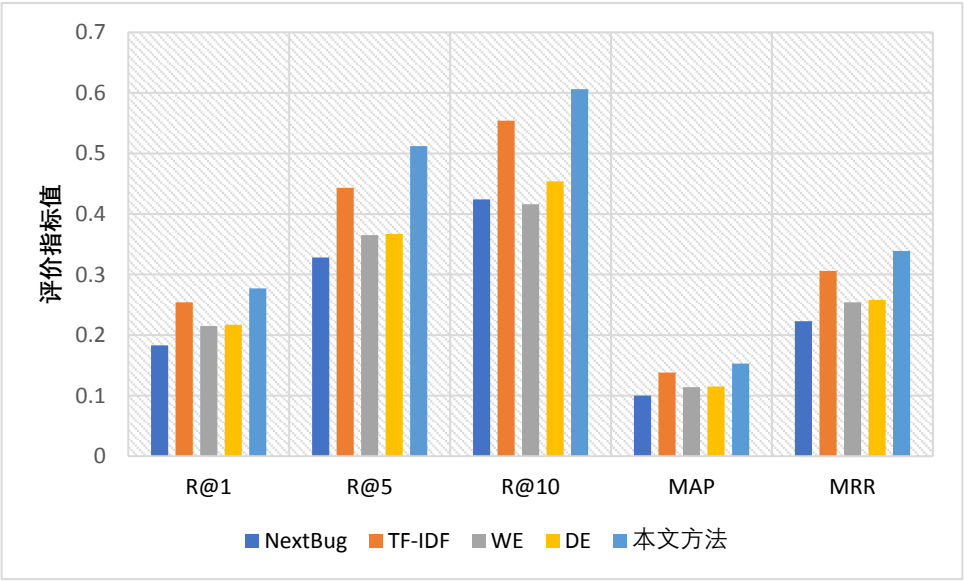

Fig.7 The performance comparison results in Request project.

图 7 Request 项目内各方法性能比较结果

(2) *Moment 项目*:图 8 给出了在 Moment 项目缺陷数据中,进行相关缺陷推荐的性能比较结果.

我们同样发现本文方法可以实现较高的推荐性能.其中,*R@1* 值为 22.2%,*R@5* 值为 40.0%,而 *R@10* 值达到 49.5%,即当推荐 10 个候选缺陷时,有 49.5%的概率候选缺陷即为真正的实际关联缺陷.

我们同样发现本文方法在所有评价指标上都优于 NextBug.相对于 NextBug 的性能值,本文方法分别在 *R@1*、*R@5*、*R@10*、*MAP* 和 *MRR* 上提升了 41.4%、35.1%、36.0%、40.7%和 40.4%.

此外我们同样发现,与我们的三个子方法相比,本文方法在所有指标方面也实现了更好的性能.相对于三个子方法性能值,本文方法分别在 *R@1*、*R@5*、*R@10*、*MAP* 和 *MRR* 上提升了 12.7-47.0%、2.0%-42.3%、2.9-42.7%、9.0-45.8%、10.9-34.2%和 9.3-45.7%.

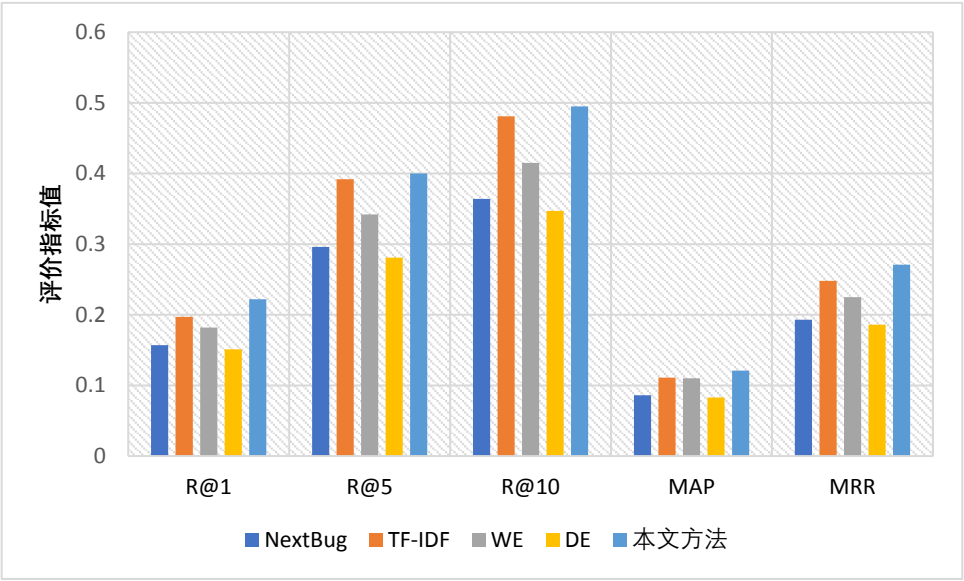

Fig.8 The performance comparison results in Moment project.

图 8 Moment 项目内各方法性能比较结果

小结:总的来看,本文方法可以很好地抽取缺陷文档内单词和文档的上下文信息,在推荐性能上要优于现有的 NextBug 方法.此外,混合式的推荐方法在性能上也优于单独的三个子方法.

4.2 RQ2: 本文方法对于缺陷语料库是否有很强的依赖性?

4.2.1 研究动机

在本文方法中,我们从每个项目的缺陷文本中学习语料库和嵌入模型,这些缺陷语料库反应了开发者报告缺陷的方式以及他们使用的词汇特点.因此,在第二个研究问题中,我们想要研究从不同项目的缺陷语料库中学习嵌入模型是否以及在何种程度上会影响本文方法的性能.对该问题的研究将有助于我们理解合适的语料库对于特定项目相关缺陷推荐任务的重要性,以及探讨本文方法是否对于跨项目缺陷推荐同样具有很好的性能.

4.2.2 分析方法

我们通过交叉使用 Request 项目和 Moment 项目的缺陷语料库来验证本文方法的推荐性能,即对于 Request 项目和 Moment 项目,我们分别使用它们自身的缺陷语料库和另外一个项目的缺陷语料库来进行测试.

4.2.3 分析结果

(1) *Request 项目*:图 9 给出了在 Request 项目中使用不同项目缺陷语料库时的性能比较结果.

我们发现对于 Request 项目,与从它自己的缺陷语料库学习相比,从其他项目的缺陷语料库中学习会降低本文方法的性能.具体而言,在 *R@1*、*R@5*、*R@10*、*MAP* 和 *MRR* 上,本文方法的性能分别下降了 6.6%、12.6%、14.3%、4.3%和 8.2%.

(2) *Moment 项目*:图 10 给出了在 Moment 项目中使用不同项目缺陷语料库时的性能比较结果.

我们发现对于 Moment 项目,与从它自己的缺陷语料库学习相比,从其他项目的缺陷语料库中学习同样会降低本文方法的性能.具体而言,在 *R@1*、*R@5*、*R@10*、*MAP* 和 *MRR* 上,本文方法的性能分别下降了 5.7%、7.1%、8.7%、2.8%和 6.1%.

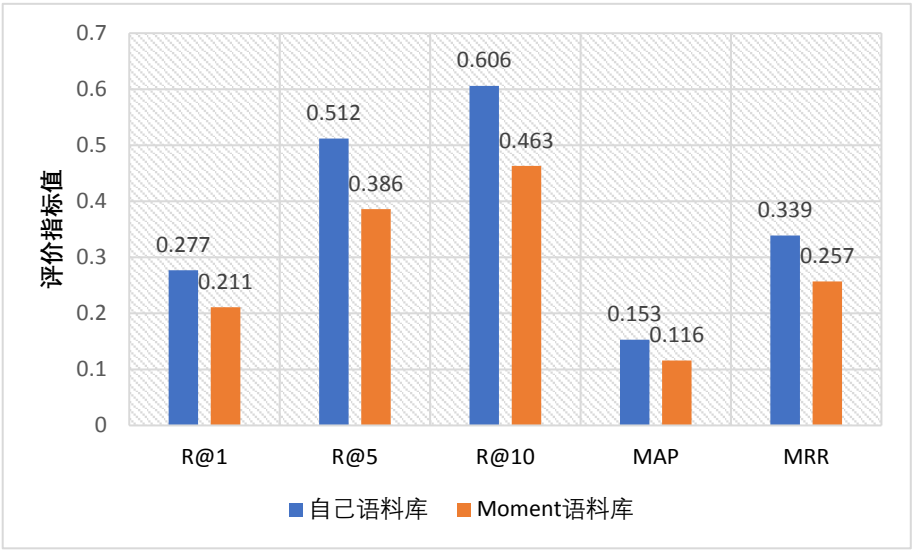

Fig.9 The performance comparison results when using different issue corpus in Request project.

图 9 Request 项目内使用不同缺陷语料库的性能比较结果

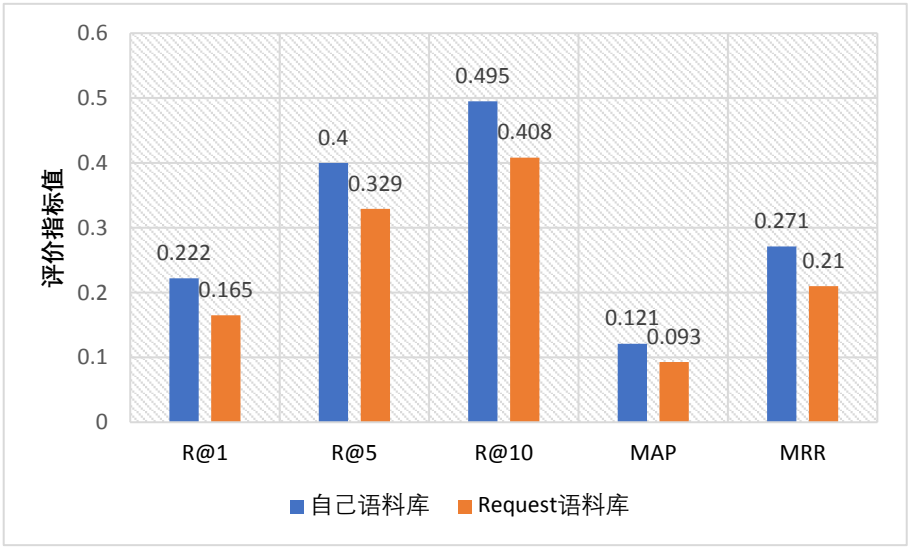

Fig.10 The performance comparison results when using different issue corpus in Moment project.

图 10 Moment 项目内使用不同缺陷语料库的性能比较结果

然而我们发现,虽然使用其他项目的缺陷语料库时本文方法的性能会下降,但本文方法仍然要优于基准方法 NextBug.在 Request 项目中,本文方法在各个评价指标上比 NextBug 有至少 9.2%的提升.在 Moment 项目中,本文方法在各个评价指标上比 NextBug 有至少 5.1%的提升.

小结:实验结果显示,特定、合适的项目缺陷语料库可以给本文方法带来更好的性能.但是,即便利用其他项目的缺陷语料库来学习嵌入模型,本文方法的推荐结果仍然具有一定的可靠性,下一步,可进一步分析跨项目缺陷关联问题.

4.3 RQ3: 本文方法对于训练集规模的依赖如何?

4.3.1 研究动机

因为本文方法所有的子方法(IF-IDF,WE 和 DE)都需要在一定规模的数据集上进行训练,所以我们需要考虑训练集规模对于本文方法性能的影响.因此,在第三个研究问题中,我们希望分析本文方法在不同规模训练集下性能的变化.

4.3.2 分析方法

我们分别根据 Request 项目和 Moment 项目的训练集规模,按照 10%、20%、…、100%的比例各分为 10 组训练集,并对基于不同规模训练集训练得到的模型进行性能测试和比较.

4.3.3 分析结果

图 11 和图 12 分别显示了在 Request 项目和 Moment 项目中,使用不同规模的训练集,本文方法在 R@1、R@5、R@10、MAP 和 MRR 上的指标值变化情况.我们发现,同其他机器学习算法相似,本文方法在不同规模训练集下的性能存在差异.例如在 Request 项目中,使用 10%缺陷数据用于训练,本文方法的 R@10 值仅为 0.25,而当使用 90%缺陷数据用于训练时,R@10 值可达到 0.57,非常接近我们前面的测试结果 0.61 (100%训练集).同样,在 Moment 项目中,使用 10%缺陷数据用于训练时,本文方法的 R@10 值仅为 0.25,而使用 90%缺陷数据用于训练时,R@10 值可达到 0.46,接近我们前面测试结果 0.50 (100%训练集).

另外,我们发现,随着训练集规模的增加,在 Request 项目和 Moment 项目中,各个指标值均有增长趋势.这表示大量的模型训练样本将有助于进一步提高本文方法的性能.

小结:本文方法的性能随着训练集规模不同会产生差异,训练集规模越大,本文方法的性能越好.下一步可以设计实现增量式的模型训练方法,即,每次只针对新加入的缺陷数据进行训练,从而降低模型的训练成本.

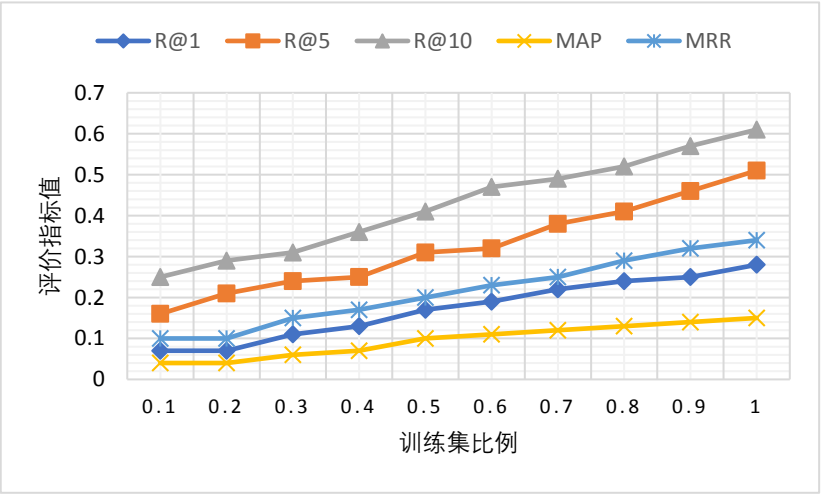

Fig.11 The performance comparison results when using different scale of training dataset in Request project.

图 11 Request 项目内使用不同训练数据时的性能比较结果

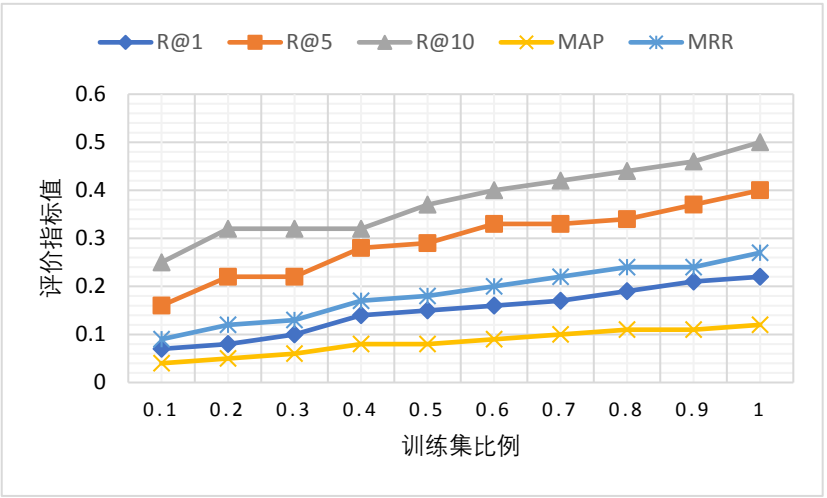

Fig.12 The performance comparison results when using different scale of training dataset in Moment project.

图 12 Moment 项目内使用不同规模训练数据时的性能比较结果

4.4 RQ4: 本文方法的时间开销如何？

4.4.1 研究动机

因为本文方法所有的子方法(IF-IDF,WE 和 DE)都需要进行预处理和训练才能用于实际的推荐任务,因此我们需要考虑模型训练的时间开销问题.因此,在第四个研究问题中,我们希望分析本文方法的训练时间开销,以了解本文方法的实用性.

4.4.2 分析方法

我们通过记录程序执行的开始时间和结束时间,来获得本文方法的训练时间开销.实验运行在 Windows8 OS(64 位)、8GB RAM、Intel(R)Core(TM)i5 2.6GHz 的 PC 上.

4.4.3 分析结果

在我们的训练过程中,Request 项目包含了 2896 个缺陷报告和 223595 个单词(预处理后),其词汇量(单个单词数)大小为 129750.而 Moment 项目包含了 4509 个缺陷报告和 287757 个单词,其词汇量为 153673.图 13 给出了本文方法的训练时间开销统计结果.

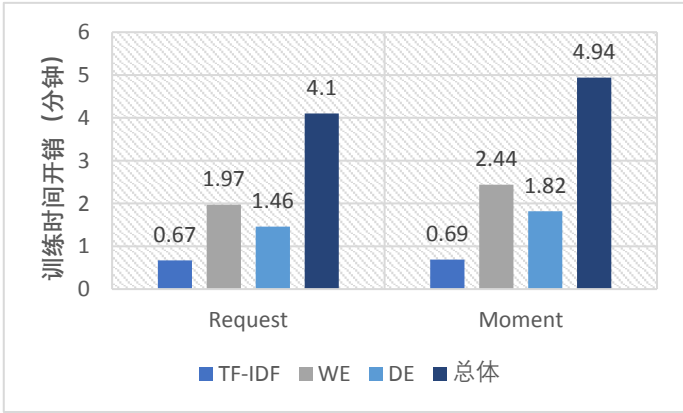

Fig.13 Training time cost in Minutes.

图 13 训练时间开销(分钟)

我们发现,在两个项目中,TF-IDF 子方法比其他两个子方法都需要更少的训练时间,因为它只训练所有缺陷文本的单词语料库,而 WE 和 DE 子方法都需要训练单词和文档的神经网络,所以训练时间较长.从总体结果来看,对于 Request 项目缺陷的训练时间开销仅为 4 分钟左右,而对于 Moment 项目缺陷的训练时间开销只需要大约 5 分钟.

小结:训练时间开销结果表明本文方法可以被很快地离线训练,且只需完成一次模型训练,即可用于相关缺陷的关联推荐.较少的训练时间开销也说明本文方法具有很强的应用扩展性,我们可以进一步基于本文方法研发相关工具和插件.

5 案例分析

为了进一步分析本文方法,我们对实际的缺陷链接对进行了案例分析.案例 1(见图 14)给出了 *request/request #880* 和 *request/request #1005* 这一缺陷链接对.我们发现,这两个缺陷之间存在许多相同单词,例如“optional”,“dependencies”和“cookie”.推荐结果表明,NextBug、TF-IDF 和 DE 方法都无法捕获这两个缺陷之间的关联关系,而 WE 和本文方法都正确判断了它们之间的关联关系.所以,当两个相关缺陷存在许多相同单词时,*词嵌入模型*可以很好地通过分析单词之间的上下文语义信息来帮助本文方法捕获缺陷之间的关联关系.

request/request#880
*Title: Disabling features based on the status of **optional Dependencies** causes confusion*
*Description: /.../ I couldn't get **cookie** jars working with request, /.../ but since it's part of **optional Dependencies**, /.../ I realized that it was marked as **optional**, /.../ I think it's okay to disable functionality based on missing **libraries**, but the absolute last thing the **library** should do is provide "stub" functionality /.../ if you try to use **cookies** without tough-**cookie** installed, or is it better to make the **dependency** mandatory /.../*
- -
request/request#1005
Title: Title*: 3.0 – Cut **optional dependencies***
*Description: **Optional dependencies** have been tricky and painful since we moved to them. /.../ That doesn't mean I think that we should just take all those deps back as regular **dependencies**. The deps I think should just return to being regular **dependencies** are: form-data and tunnel-agent. /.../ Then signing **libraries** can produce those objects and we can just accept them, /.../ We can handle **cookies** /.../*

NextBug: ✗; TF-IDF: ✗; WE: ✓; DE: ✗; 本文方法 : ✓

Fig.14 Case study of *request/request#880* and *request/request#1005*.

图 14 *request/request#880* 和 *request/request#1005* 的案例分析

案例 2(见图 15)给出了缺陷链接对:*request/request #699* 和 *request/request #803*.我们发现,这两个缺陷并没有很多共同的单词,只有“Error”、“getaddrinfo”和“ENOTFOUND”几个单词相同.推荐结果显示,NextBug、TF-IDF 和 WE 方法都无法捕获它们两的链接关系,而 DE 和本文方法都正确判断了它们之间的关联关系.这表明,当两个相关缺陷没有太多文本信息或没有多个共同单词时,*文档嵌入模型*可以很好地通过分析缺陷文档之间的上下文语义信息来帮助本文方法捕获缺陷之间的关联关系.

request/request#699
Title: ***Error: getaddrinfo ENOTFOUND***
*Description: The app uses request to GET >10K files. After 50% downloaded, it starts to fail catastrophically with the remaining requests in the queue. The error is {[**Error: getaddrinfo ENOTFOUND**] code: '**ENOTFOUND**', errno: '**ENOTFOUND**', syscall: '**getaddrinfo**' }. Any idea?*
- -
request/request#803
Title: Unhandled stream ***error*** *in pipe. "**Error: getaddrinfo ENOTFOUND**"*
Description: Hi, I am using mikeal/request in order to proxy API requests. It works fine like 99% of the time, but every once in a while it throws a ***getaddrinfo ENOTFOUND error****. /.../ Two points of interest: /.../*

NextBug: ✗; TF-IDF: ✗; WE: ✗; DE: ✓; 本文方法: ✓

Fig.15 Case study of *request/request#699* and *request/request#803*.

图 15 *request/request#699* 和 *request/request#803* 的案例分析

因此,通过案例分析,我们发现融合*词嵌入模型*和*文档嵌入模型*后,本文方法可以更好地捕获两个相关缺陷之间的关联关系.

6 结束语

本文提出了一种基于嵌入模型的混合式软件缺陷关联方法,用于在 GitHub 项目中自动链接相关缺陷.本文方法结合了传统的信息检索技术(TF-IDF)和深度学习技术(词嵌入模型和文档嵌入模型).在我们的实证验证中,我们将本文方法与 NextBug 方法(传统缺陷跟踪系统中的最先进方法),以及其他三种基准方法(即我们的三个子方法)进行比较.实验结果表明,本文提出的方法可以很好地融合缺陷语料库的上下文信息,在性能上要优于基准方法.此外,我们发现本文方法在项目特定的训练语料库下有助于提高嵌入模型的性能,但对于不同项目的缺陷语料库,本文方法的推荐结果仍能表现一定的可靠性,下一步可进一步分析跨项目的缺陷关联问题.并且,较短的时间开销表明本文方法可以有效地进行离线训练,可进一步研发相关工具和插件,具有很强的应用扩展性.

References:

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In: Proceedings of the ACM conference on Computer Supported Cooperative Work. ACM, 2012, pp. 1277–1286.

[2] M. Gharehyazie, B. Ray, and V. Filkov. Some from here, some from there: cross-project code reuse in github. In: Proceedings of the International Conference on Mining Software Repositories. IEEE Press, 2017, pp. 291–301.

[3] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011, pp. 253–262.

[4] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: Proceedings of the International Conference on Software Engineering. IEEE Press, 2012, pp. 14–24.

[5] H. Rocha, M. T. Valente, H. Marques-Neto, and G. C. Murphy. An empirical study on recommendations of similar bugs. In: IEEE International Conference onSoftware Analysis, Evolution, and Reengineering (SANER). vol. 1. IEEE, 2016, pp. 46–56.

[6] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In: International Conference on Machine Learning. 2014, pp. 1188–1196.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. 2013, pp. 3111–3119.

[9] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In: ASE. ACM, 2016, pp. 51–62.

[10] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In: ICSE. ACM, 2016, pp. 404–415.

[11] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun. Combining word embedding with information retrieval to recommend similar bug reports. In: IEEE International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2016, pp. 127–137.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. Journal of machine Learning research, vol. 3, no. Jan, pp. 993–1022, 2003.

[13] A. M. Dai, C. Olah, and Q. V. Le. Document embedding with paragraph vectors. arXiv preprint arXiv:1507.07998, 2015.

[14] K. Crowston and B. Scozzi. Bug fixing practices within free/libre open source software development teams. 2008.

[15] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In: Proceedings of the joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2009, pp. 111–120.

[16] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In: Proceedings of the international conference on Software engineering. ACM, 2006, pp. 361–370.

[17] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In: ICSE. IEEE, 2010, pp. 495–504.

[18] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In: ICSE. IEEE, 2012, pp. 1074–1083.

[19] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: bugs and bug-fix commits. In: Proceedings of the ACM international symposium on Foundations of software engineering. ACM, 2010, pp. 97–106.

[20] X. Ye, R. Bunescu, and C. Liu. Learning to rank relevant files for bug reports using domain knowledge. In: Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014, pp. 689–699.

[21] Y. Zhang, G. Yin, T. Wang, Y. Yu, and H. Wang. Evaluating bug severity using crowd-based knowledge: An exploratory study. In: Proceedings of the Asia-Pacific Symposium on Internetware. ACM, 2015, pp. 70–73.

[22] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In: ACM/IEEE International Conference on Software Engineering. IEEE, 2008, pp. 461–470.

[23] C. Paice. A word stemmer based on the lancaster stemming algorithm. In: ACM SIGIR, 1990, pp. 56–61.