软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(3):1321−1340 [doi: 10.13328/j.cnki.jos.006820]

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

基于相关性反馈的开源系统跨层需求追踪方法*

田家豪[1,2], 张 莉[1,2,3], 连小利[1,2], 赵倩慧[1,2]

[1](北京航空航天大学 计算机学院 软件工程研究所, 北京 100191)

[2](软件开发环境国家重点实验室 (北京航空航天大学), 北京 100191)

[3](北京航空航天大学 软件学院, 北京 100191)

通信作者: 连小利, E-mail: lianxiaoli@buaa.edu.cn

摘 要: 大型复杂软件系统的需求分析与生成是一个由上而下逐层分解的过程, 跨层需求间追踪关系的构建对于项目的管理、开发与演化都至关重要. 开源系统的松耦合贡献方式需要每位参与者能便捷地理解需求的来龙去脉及需求状态, 这依赖跨层需求间的追踪. 问题描述日志是开源系统中需求的常见呈现方式, 其无固定模板要求, 内容多样 (含文本、代码、调试信息等), 术语使用自由, 跨层需求间抽象层次大, 给自动追踪带来极大的挑战. 提出一种面向关键特征维度的相关性反馈方法, 通过静态分析项目代码结构, 抽取代码相关术语及其间的关联强度, 构建代码词汇库, 以缓解跨层需求的抽象层次差距及用语不统一的问题; 通过度量词汇对需求描述的重要性并基于此筛选关键特征维度, 以对查询语句进行针对性的优化, 有效减少需求描述长度、内容形式等方面的噪音. 通过在 3 个开源系统需求集上针对两个场景的实验, 表明所提方法相比基线方法在跨层需求追踪方面的优越性, 相比 VSM、Standard Rocchio 和 Trace BERT, *F*2 值的最大提升分别可达 29.01%、7.45% 和 59.21%.

关键词: 需求追踪; 相关性反馈; 开源系统; 问题日志

中图法分类号: TP311

中文引用格式: 田家豪, 张莉, 连小利, 赵倩慧. 基于相关性反馈的开源系统跨层需求追踪方法. 软件学报, 2024, 35(3): 1321–1340. http://www.jos.org.cn/1000-9825/6820.htm

英文引用格式: Tian JH, Zhang L, Lian XL, Zhao QH. Cross-level Requirement Tracing Method of Open-source Systems Based on Correlation Feedback. Ruan Jian Xue Bao/Journal of Software, 2024, 35(3): 1321–1340 (in Chinese). http://www.jos.org.cn/1000-9825/6820.htm

Cross-level Requirement Tracing Method of Open-source Systems Based on Correlation Feedback

TIAN Jia-Hao[1,2], ZHANG Li[1,2,3], LIAN Xiao-Li[1,2], ZHAO Qian-Hui[1,2]

[1](Software Engineering Institute, School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

[2](State Key Laboratory of Software Development Environment (Beihang University), Beijing 100191, China)

[3](School of Software, Beihang University, Beijing 100191, China)

Abstract: In large-scale and complex software systems, requirement analysis and generation are accomplished through a top-down process, and the construction of tracking relationships between cross-level requirements is very important for project management, development, and evolution. The loosely-coupled contribution approach of open-source systems requires each participant to easily understand the context and state of the requirements, which relies on cross-level requirement tracking. The issue description log is a common way of presenting requirements in open-source systems. It has no fixed template, and its content is diverse (including text, code, and debugging information). Furthermore, the terms can be freely used, and the gap in abstraction level between cross-level requirements is large, which brings great challenges to automatic tracking. In this paper, a correlation feedback method for key feature dimensions is proposed. Through static analysis of the project's code structure, code-related terms and their correlation strength are extracted, and a code vocabulary base is

* 基金项目: 国家自然科学基金 (62102014, 62177003); 软件开发环境国家重点实验室基金 (SKLSDE-2021ZX-10)
收稿时间: 2022-03-23; 修改时间: 2022-07-13, 2022-09-15; 采用时间: 2022-10-14; jos 在线出版时间: 2023-05-10
CNKI 网络首发时间: 2023-05-11

constructed to alleviate the gap in abstraction level and the inconsistency of terminology between cross-level requirements. By measuring the importance of terms to requirement description and screening key feature dimensions on this basis, the inquiry statement is optimized to effectively reduce the noise of requirement description length, content form, and other aspects. Experiments with two scenarios on three open-source systems suggest that the proposed method outperforms baseline approaches in cross-level requirement tracking and improves $F2$ value to 29.01%, 7.45%, and 59.21% compared with vector space model (VSM), standard Rocchio, and trace bidirectional encoder representations from transformers (BERT), respectively.

Key words: requirement tracking; correlation feedback; open-source system; issue log

大型复杂软件系统的需求开发是一个多层级、自上而下不断分解的过程. 根据 Gorschek 等人[1]提出的需求抽象模型 (requirement abstraction model), 按抽象程度由高到低可以将需求分为产品级 (product level)、特征级 (feature level)、功能级 (function level) 以及组件级 (component level), 其中高抽象级别的需求会被不断地细化、分解为更低抽象级别的需求, 直至详细到可以交付给组件设计人员的粒度. 在此过程中, 跨层需求追踪关系有助于保证高层需求 (high-level requirement, HLR) 被有效分解为低层需求 (low-level requirement, LLR), 同时也要确认低层需求有源可溯、有据可依. DO-178C[2]、IEEE Std.830[3]以及 CMMI[4]等多个标准和规范都强调了需求追踪能力对软件开发的重要性, 特别地, DO-178C 中明确规定“要保证低层需求可以满足高层需求”, 以及“要保证每一条高层需求被进一步开发为低层需求”.

为了适应快速迭代、持续集成的敏捷开发模式和跨地域、松耦合的合作方式, 大型开源项目通常采用轻量级的需求开发和管理模式——即时需求模式 (just-in-time, JIT)[5]. 在该模式下, 问题跟踪系统 (issue tracking system, ITS) 是许多开源项目用来记录和管理需求的工具. ITS 中, 需求以“问题日志 (issue log)”的形式被开发者或终端用户提出, 而不同类型 (issue type) 的问题日志记录了不同抽象层次的需求. 构建开源项目中跨层需求之间追踪关系非常关键. 首先大型开源项目开发人员流动性强, 缺少跨层追踪关系的支持, 新的参与者难以快速理解需求的来龙去脉. 另外, 开源系统的需求处于频繁变化中, 跨层需求追踪关系能够支持分析师在处理低层需求的变更提交时, 更便捷的检查变更影响范围和有效性, 以确保其不违背上层需求.

大型开源项目需求来源广、规模大, 由于其快速迭代的特点, 需求还会随着时间的推移不断增加, 依靠分析师人工建立完整的需求追踪关系十分费时费力, 且枯燥易错. 由于需求追踪关系的重要性, 开源系统也会鼓励用户在提出问题/需求时, 人工标记该问题与其他问题的追踪链接. 这意味着, 对追踪链接的标注是用户的自主行为, 而非必要活动. 由于项目历史长、需求数量多, 参与者很难掌握全部需求, 他们通常只会为自己了解范围内的部分需求标注追踪链接, 导致开源系统中未标注或漏标注链接情况常有发生[6]. 因此, 研究自动建立跨层需求追踪的方法十分必要.

本文以 JIRA (https://www.atlassian.com/software/jira) 为例, 分析了开源系统中需求的特点. 由 Atlassian 公司出品的 JIRA 是最广为使用的问题跟踪系统, 据 Atlassian 统计, 全球有超过 60 000 家以上的公司在使用它. JIRA 预定义了一些问题类型 (https://support.atlassian.com/jira-cloud-administration/docs/what-are-issue-types/), 如“EPIC”“Wish”“User Story”“Feature Request”以及“Enhancement”等, 用来记录不同层次的需求. 如表 1 所示, 各类型需求问题日志的抽象层次由高到低分别是 Parent, Standard 以及 Child. 同时 JIRA 允许项目管理者根据自身需要或习惯自定义问题类型, 有些开源项目中定义了“New Feature”“Improvement”类型的问题日志, 分别来代替“Feature Request”和“Enhancement”的作用.

表 1 JIRA 预定义的用来记录需求的问题类型

问题类型	解释	层次 (由高到低)
Epic	大型用户故事库, 可被分解	Parent
User story	用户故事	Standard
Feature Request	提出的功能需求	Standard
Enhancement	对某个功能的改进	Standard
Task	需要进行的技术任务	Standard
Subtask	由Task进一步分解得到的子任务	Child

图 1 以 DROOLS 项目 (https://drools.org/) 中的问题日志 DROOLS-4530 (https://issues.redhat.com/browse/DROOLS-4530) 为例, 展示了 JIRA 中以问题日志形式存在的需求. 图中, Issue ID 是问题日志的唯一标识. 问题日志的主要描述信息存在于以下关键字段中: 对问题的一个简短总结 (summary), 问题标签 (labels), 所属组件 (components), 以及一条较为详细的问题描述 (description) 等.

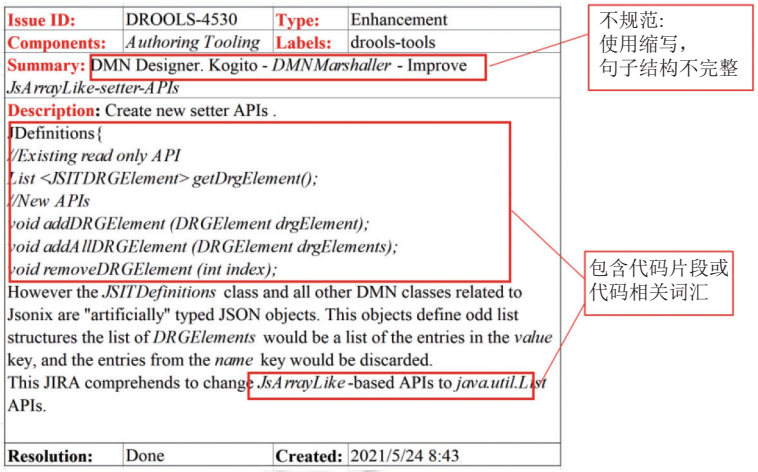


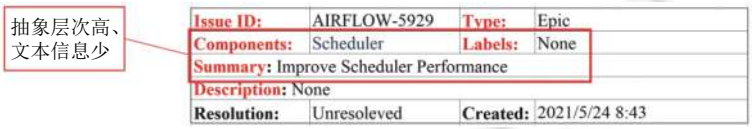
图 1 JIRA 中以问题日志形式存在的需求条目示例: DROOLS-4530

传统前置需求工程 (up-front requirements engineering) 方法, 遵循 IEEE-830[3]、EARS (easy approach to requirement specification)[7]等规范, 主张通过需求抽取、建模、分析等阶段获取形式化/模型化的需求制品, 或规范化/模板化的自然语言 (natural language, NL) 需求制品. 而开源系统采用即时需求模式, 其需求制品更加轻量级和“非正式(informalism)”[8], 对需求的编制过程、描述方式等没有严格的限制, 因此需求制品规范化程度较低. 如图 1 所示, 该需求的文本描述格式自由, summary、description 中有些句子结构不完整 (如 summary 是由几个短语拼凑而成的, 而非完整语句), 使用了缩略语 (如 summary 中的“DMN”). 开源系统的需求中包含的内容形式更多样, 除文本外, 还可能包含代码片段、调试信息等. 由于开源系统快速迭代、持续集成的特点, 许多需求在项目迭代过程中提出, 是针对已发行的软件系统提出的功能改进意见, 开发者或用户为了方便描述, 会直接引用系统中的相关代码. 如图 1 中所示, 其 description 中除了自然语言描述外, 还包含了携带语义信息的代码片段或代码相关的词汇 (图中的斜体字). 一些问题日志的 description 还包含了冗长的调试信息和错误日志, 其中包含大量无辨识度的重复词汇, 如 JBTM 项目 (http://jboss.org/jbosstm) 中的需求问题日志 JBTM-1772 (issues.redhat.com/browse/JBTM-1772). 由于篇幅原因, 文中未展示该问题日志.

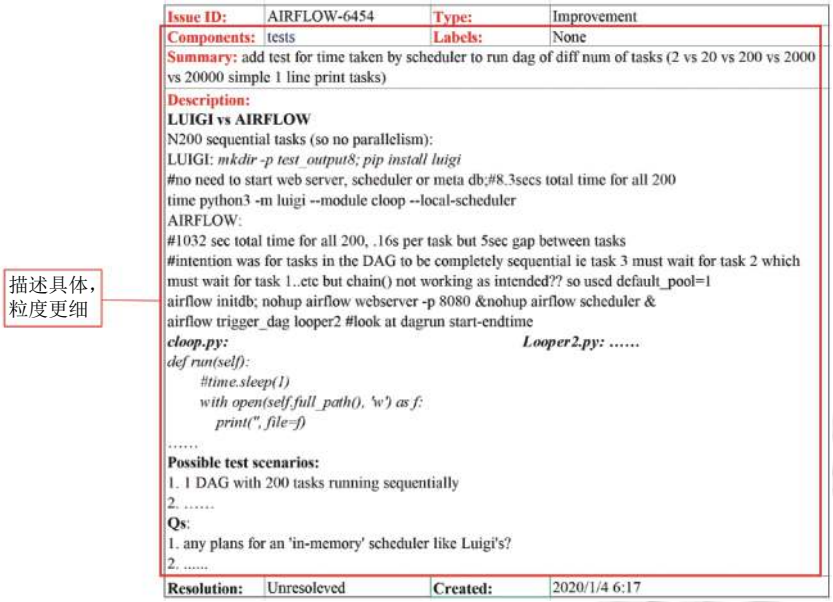
跨层需求间存在天然的抽象层次差异, 通常高抽象层次的需求包含的文本信息更少, 而低层需求的描述则更加详细. 如图 2 中一对来自 Apache AIRFLOW 项目 (https://airflow.apache.org/) 的两条需求所示, 用户将这两条需求标注为存在跨层追踪关系. 如图 2(a) 所示 AIRFLOW-5929 (https://issues.apache.org/jira/browse/AIRFLOW-5929) 的高层需求描述十分简短, summary 字段中只包含一个短语“Improve Scheduler Performance”, 而 description 字段中无任何内容. 而图 2(b) 所示 AIRFLOW-6454 (https://issues.apache.org/jira/browse/AIRFLOW-6454) 的低层需求描述中, 其 summary 和 description 字段中都包含对“Scheduler”的详细分析以及对其性能改进的具体方式. 不同层次需求间描述粒度的差异也增加了构建追踪关系的难度.

目前已有一些关注自动构建需求追踪关系的研究. 由于需求制品的文本特性, 基于信息检索 (information retrieval, IR) 的方法, 如 VSM (vector space model)[9]、LSI (latent semantic index)[10]等是需求追踪领域内研究者广泛应用的方法[11]. 该类方法通过捕捉不同制品所使用的相同词汇, 计算每一对查询语句和待检索文档间的文本相似度, 若相似度高于预设的阈值, 则判断其间存在追踪关系, 否则判定为无追踪关系[12]. 但在实际场景中, 特别是在开源系统中, 基于信息检索的方法建立跨层需求追踪时, 无法解决高层需求和低层需求的抽象程度存在差异、不

同需求制品的术语使用不一致、同义不同形现象普遍等问题. 针对以上问题, 研究者们尝试构建词汇库[12]和领域本体[13], 对文本结构化[14]以及基于相关性反馈技术 (relevance feedback, RF)[15]等改进了基于信息检索的方法在需求追踪中的应用效果[16−19]. 此外, 由于近年来人工智能技术的高速发展, 一些深度学习模型, 如 BERT (bidirectional encoder representations from Transformers)[20]等在文本语义理解方面表现出色, 也使深度学习在需求追踪领域越来越受青睐[21−25]. 研究者们提出通过循环神经网络 (recurrent neural network, RNN)、长短时记忆网络 (long short term memory, LSTM)、BERT 等模型训练需求文本向量以识别需求和代码间的追踪关系[21−23], 或识别需求描述中的一词多义现象[25]以提升需求追踪关系识别效果. 但此类方法的前提是有一定规模的优质标注数据为基础. 在训练集质量和规模不足时, 其效果反而与信息检索类方法有较大差距[21].



(a) 一条 “Parent” 层次的 “Epic” 需求示例: AIRFLOW-5929



(b) 一条 “Standard” 层次的 “Improvement” 需求示例: AIRFLOW-6454

图 2 具有追踪关系的一对跨层需求示例

综上所述, 本文聚焦问题跟踪系统 JIRA 上以 issue 形式描述的需求文本, 针对开源系统需求规范性低、内容杂糅、跨层需求抽象层次差异大, 以及已标注追踪关系数量有限等特点, 提出一种基于关键特征维度筛选的跨层需求追踪关系建立方法 (向量空间中, 每个维度对应一个词汇). 具体来讲, 通过自动从代码中抽取领域术语间关系以缩小跨层需求描述间的抽象层次差异, 并提出基于需求特殊性的关键特征维度选择方法, 以降低 issue 文本描述中弱相关的冗长文本等噪音对追踪的负面影响. 本文在 3 个开源项目中收集需求数据, 分别在追踪关系检索范围确定和不确定的两种场景下进行了实验验证. 实验结果表明, 两种场景下, 本文方法分别能取得 53.75%–77.31% 和 33.90%–53.19% 的 $F2$ 值, 相比基线算法 (包括 VSM[9], 已有的相关性反馈技术 Standard Rocchio[15]及深度学习框架 Trace BERT[23]), 其 $F2$ 提升分别达到了 29.01%、7.45% 和 59.21%. 总体上讲, 本文主要有以下 4 个贡献.

1) 针对开源系统需求规范化程度低、跨层需求间抽象层次差异大的问题, 提出一种面向开源系统的跨层需求追踪方法框架.

2) 针对开源系统需求中可能包含代码片段的特点, 提出一种面向开源系统代码分析的关键词点构建方法, 通

过从代码中挖掘需求中关键词之间的隐含关系, 在一定程度上填补跨层需求间的抽象层次差距.

3) 针对传统相关性反馈技术无法有效处理开源系统需求描述中存在的噪声数据的问题, 提出一种面向自然语言需求的关键特征维度自动筛选方法, 通过识别和过滤反馈过程中的噪音, 提高了信息检索方法在开源系统需求追踪中的准确性和完整性.

4) 基于 3 个不同领域的开源项目, 以准确率、召回率和 *F*2 等需求追踪领域内广泛应用的评价指标, 验证了本文方法相对于基线方法的优越性, 其中 *F*2 值相比 VSM、Standard Rocchio 和 Trace BERT 分别能达到 29.01%、7.45% 和 59.21% 的提升.

本文第 1 节总结需求追踪领域的相关研究现状. 第 2 节详细介绍本文提出的基于相关性反馈的开源系统跨层需求追踪方法. 第 3 节介绍实验验证, 包括实验数据获取过程和实验验证结果等. 第 4 节讨论本文的有效性威胁及局限性. 第 5 节总结全文.

1 相关研究

本节从基于信息检索的方法、基于学习的方法和基于规则的方法这 3 个方面总结了需求追踪领域的相关研究现状.

(1) 基于信息检索的方法

由于需求等软件制品的文本特性, 基于信息检索的方法成为需求追踪领域应用最为广泛的方法之一. 在使用基于信息检索的方法进行需求追踪时, 源制品被作为查询语句, 而目标制品被当作待检索文档[24]. 该类方法的假设是, 当一对文本具有越多相同词汇时, 二者间相似度越高, 存在追踪关系的可能性也就越大[12]. VSM 作为最直观易懂的信息检索方法, 自提出以来就被研究者们广泛使用, 并在许多研究中被选为基线方法[17,21−25]. 实践中不同制品在描述同一概念时所用的词汇可能不同, 即存在关键词误配 (term mismatch)[26]问题, 同时还存在一词多义等问题, 这是传统信息检索类方法难以处理的. 一些研究者针对这些问题对 VSM 做了各方面的改进[27], 如词汇库的支持 (VSM-Thesaurus)[19], 文本语义信息的加入 (part-of-speech-enabled-VSM, VSM-POS)[28]等. 一些研究证明构建词汇库, 标注词汇之间的关联关系, 有助于缓解跨层需求制品间存在的 term mismatch 问题[13,19,29], 借助于词汇关系库, 基于信息检索的方法能够捕捉到那些描述相同或相似概念的不同词汇, 并在一定程度上缩小跨层需求制品间的抽象层次差异. 但大多项目中并不存在现成的词汇库, 而人工标注会带来大量的额外工作. 因此, 鉴于许多开源系统需求条目中存在代码片段, 本文拟从开源系统的代码库中通过分析包结构、类调用关系等静态关系, 自动构建代码词汇库, 提取这些词汇间的相关度.

近年来, 一些研究证实了相关性反馈技术在需求追踪领域起到了积极作用[12,16−18]. 相关性反馈技术利用用户对信息检索结果的反馈信息, 即被标注为与查询语句相关的文档, 调整原始查询语句向量中词汇的权重, 并将改进后的查询语句用于新一轮的追踪关系检索中. 相关性反馈技术的优势在于, 即使已标注的追踪关系的数量较少时, 该技术也可以基于用户反馈不断改进文本向量以提升识别效果. Hayes 等人[12]采用相关性反馈技术 Standard Rocchio 改进传统 VSM, 并将其应用于需求追踪领域, 利用用户对检索结果的反馈信息对查询语句做改进, 验证了该方法在识别高层需求-低层需求间追踪关系时能够提高准确率和召回率, 成功提升了 VSM 算法的效果. 此后, Panichella 等人[16]提出了一种自适应的相关性反馈需求追踪方法. 他们提出在应用相关性反馈策略之前, 先考虑需求制品中文本的长度以及采用基础 VSM 进行预识别的结果. 当作为查询语句的制品中文本长度小于待检索文档的长度, 且查询语句对应的预识别结果中假阳性 (false positive, FP) 数量少于真阳性 (true positive, TP) 时, 再按照 Standard Rocchio 方法对查询语句进行修改, 以使改进后的查询语句在文本向量空间中朝着正确的方向移动. 此方法对 VSM 有较好的提升效果, 且优于 Standard Rocchio 方法. 但传统的相关性反馈技术使用相关文档中的所有词汇来改进查询语句. 理论上讲, 当相关文档中存在较多干扰信息时, 势必会限制该方法的作用. Wang 等人[17,18]认为针对向量空间中的特定维度做相关性反馈要比针对所有维度做相关性反馈更加有效, 因此他们提出了一种基于术语 (term-based) 的相关性反馈机制, 用于对可靠性和安全性需求的追踪. 他们人工标注了每一种可靠性需求对应的术语集合, 且只在这些术语对应的向量空间维度上根据用户反馈对查询语句做调整. 实验表明, 该方法在可靠性需

求追踪方面效果优于文献 [12] 和文献 [16], 但该方法虽然有效, 但需要手工构建可靠性、安全性相关的关键短语/词列表, 增加了额外工作量. 这些研究验证了基于相关性反馈方法, 特别是在关键维度上的反馈调整, 在需求追踪领域的积极效果, 对本文的研究具有启发作用. 但在开源系统中, 人工对每条需求标注关键词的工作量太大, 因此, 本文拟探索关键词汇的自动筛选方法, 从每条需求条目中自动识别最能代表其关键语义信息的词汇, 进而基于这些关键词做有针对性的相关性反馈与改进.

(2) 基于学习的方法

随着机器学习、深度学习等人工智能技术的兴起, 研究者们探索了基于学习的方法在需求追踪领域的实践效果. 他们将需求文本、已构建或标注的需求追踪关系作为训练集, 训练现有的机器学习或深度学习模型以识别更多的需求追踪关系[11]. 一些研究者将其看作二分类问题, 即将一对软件制品间的关系分类为正例 (存在追踪关系) 或负例 (无追踪关系)[30−32], 其中, 贝叶斯算法、概率网络或支持向量机是研究者们较为常用的分类器[11]. 另外一些研究者采用深度学习方法训练出能更好表达需求语义信息的文本向量, 再用这些文本向量进行信息检索, 以更准确地识别追踪关系[21−23]. Wang 等人[25]针对需求文本中常见的一词多义问题, 采用人工神经网络方法识别短语或词汇在不同需求描述上下文中的语义信息, 再将识别结果应用于信息检索方法 (VSM、LSI) 中, 该方法提升了自动化需求追踪的准确率. Lin 等人[23]提出了 Trace BERT 软件制品追踪框架, 用来自动构建需求制品和代码间的追踪关系. 他们首先采用预训练的 BERT 模型来获得文本和代码的向量表示, 在此基础上用代码搜索 (code search) 领域中包含成对“自然语言片段-编程语言 (programming language, PL) 片段”的大规模数据集进行训练, 最后采用从开源项目问题跟踪系统中获取的“需求-代码”数据进行调优. 实验表明 Trace BERT 在识别自然语言需求制品和代码间的追踪关系时, 准确率高于 VSM 和 LSI 等基线方法. 对于此类方法, 训练集至关重要. 要训练出优秀的模型, 必须有规模足够大、质量足够高的训练数据.

(3) 基于规则的方法

基于规则的方法则常用于较为规范的需求制品间的追踪, 如模型化需求、受约束文本的需求描述等[14,33,34]. 此类方法通常包含 3 个步骤[11], 即: 1) 预定义追踪关系构建规则; 2) 将源制品和目标制品分别转化为形式化或结构化表示; 3) 按照预定义规则识别追踪关系. Jirapanthong[33]扩展了 XQuery 语言, 并用其定义了一系列可复用规则. 案例研究表明, 该规则集在构建用例图形式的需求到模型 (如领域模型, 活动图等) 间的追踪关系时取得了较好的效果. Guo 等人[14]提出基于规则的 DoCIT 框架, 专注于交通运输领域内的需求-设计文档追踪关系识别. 他们首先将需求文档转化为包含“动作单元”“主题”“执行者”等语义单元的结构化形式, 然后在此基础上定义规则, 根据两条需求中各语义单元间的关系识别追踪关系. 在其预构建的领域知识库的支持下, 此方法的效果明显超过了 VSM 方法. 但基于规则的方法通常对输入制品有着较为严格的格式要求, 而自然语言需求文本, 特别是开源系统中格式不统一、行文不规范的需求制品, 不满足此类方法的要求.

综上所述, 由于开源系统的贡献者技术背景各异, 编写风格自由, 需求文本信息种类繁杂的低规范性特点, 很难对开源系统的需求文本进行结构化或形式化, 因此, 难以采用基于规则的方法来识别开源系统中的跨层需求追踪关系. 同时, 开源系统的问题日志管理系统中虽然记录着一部分人工标注的追踪关系, 但由于追踪链接标注信息不全等原因, 导致训练集存在如追踪数据量小, 数据中正例和负例的数量差距过大等问题, 使得极大概率上无法训练出较高质量的机器学习或深度学习模型. 因此, 本文拟对信息检索技术和相关性反馈技术进行改进, 以自动构建开源系统中跨层需求间的追踪关系.

2 开源系统中的跨层需求自动追踪——面向关键特征维度的相关性反馈方法

本节首先介绍了基于相关性反馈的开源系统中跨层需求追踪方法的总体框架, 然后依次对关键步骤作详细说明, 包括高低层需求文本向量空间的构建、项目代码词汇库的构建以及面向关键特征维度的相关性反馈技术等.

2.1 方法框架概述

针对开源系统需求低规范性、信息杂糅的特点, 本文提出了一种面向关键特征维度的相关性反馈方法, 该方法包括需求文本特征的抽取、术语关系抽取和面向关键特征维度的相关性反馈 3 个关键步骤. 其总体流程如图 3 所示.

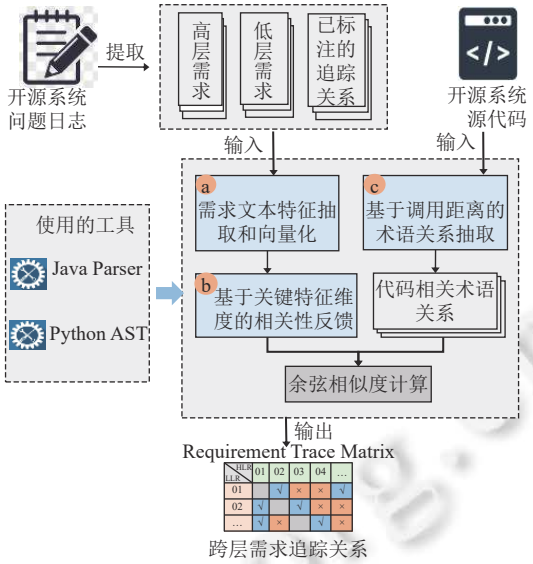


图 3 基于相关性反馈的开源系统跨层需求追踪方法

首先, 面向开源系统需求条目文本信息在各字段中的分布特点, 考虑各字段的信息密度, 改进了传统的 TF-IDF 词汇加权方法, 已在需求文本特征抽取和向量化阶段防止关键词汇的权重被冗长的噪声数据所稀释.

其次, 针对不同层次需求抽象程度差异大及术语不一致等问题, 对项目源码进行静态分析, 提取代码中的关键词列表并量化词汇间的关联程度, 构建项目的代码词汇库, 以充分利用开源项目需求文本中包含的大量代码片段.

最后, 针对开源系统需求文本描述中不同词汇的重要程度, 设计了一种面向关键特征维度的相关性反馈技术, 提出词汇的需求特殊性指标 (requirement document specificity, *ReqS*), 量化各术语为需求提供的辨识度, 以牵引需求描述中关键词的自动筛选. 进而根据已有标注, 面向筛选关键词在语料库文本向量空间中对应的特征维度, 对查询语句做调整. 传统的相关性反馈技术依据用户标注文档向量中的所有词汇对查询语句进而调整, 默认所有词汇对于文档的辨识度一致. 而本文通过 *ReqS* 识别对单条需求描述辨识度高的词汇, 针对性地调整部分维度的权重, 有效避免了噪音对于追踪关系识别的影响.

2.2 面向开源系统需求文本特征抽取和向量化

本文从 Redhat 和 Apache 基金会旗下的开源项目中选择了具有代表性的 3 个项目 AIRFLOW、DROOLS 和 JBTM 做重点分析, 这 3 个项目都曾在先前追踪领域内的研究中作为案例[25,35], 且覆盖了多种需求类型和追踪关系类型. 利用 JIRA 提供的 API, 我们从项目的问题跟踪系统中收集了表 1 中所提到的各类型的问题日志数据. 通过对这些需求集合的人工分析, 我们发现开源系统需求中最关键的信息保存在问题日志中的 summary、components、labels 等字段中, 其次是 description 中. 因此本文将这 4 个字段中的文本信息作为识别需求间追踪关系的依据.

本文统计了 3 个项目中 S、L、C、D 这 4 个字段中所包含词汇的数量, 即长度, 如表 2 所示. 可以看出, 3 个项目中 summary 平均长度为 8, 而 description 字段中平均包含 50 个以上的词汇. Components 的最大长度为 10, labels 最大长度为 8, 但大多数 (95% 以上) 的问题日志中, 这两个字段包含的词汇数为 0–3 个.

表 2 问题日志中各字段所包含的词汇数量

项目	Summary长度		Description长度		Components长度		Labels长度	
	平均	最大	平均	最大	最小	最大	最小	最大
AIRFLOW	7	26	48	1 148	0	5	0	8
DROOLS	8	24	49	1 603	0	10	0	6
JBTM	8	17	58	2 668	0	6	0	4
3个项目	8	26	51	2 668	0	10	0	8

结合以上统计数据以及我们的人工分析, 发现需求问题日志中的 summary (*S*)、labels (*L*)、components (*C*) 字段精炼且信息关键, description (*D*) 中虽然也包含了一些关键信息, 但是大多问题日志的 description 冗长且包含了诸如调试信息, 错误日志等噪声. 即开源系统需求问题日志中, 各字段的文本存在信息浓度差异. 针对开源系统需求的这一特点, 我们对传统的“词频-逆文档频率 (term frequency and inverse document frequency, TF-IDF)”加权方法做了改进.

对于一条包含 n 个词汇的需求条目 $d=\{t_1,\ldots,t_n\}$, 传统的 TF-IDF 加权方法按照如公式 (1) 所示过程来计算其词汇 t_i 的权重. 其中, 第 1 项是 t_i 的 TF 值, 使用 t_i 在文档 d 中出现的次数 $freq(t_i)$ 除以文档 d 包含的总词汇数量计算得来, 即文档中 t_i 的归一化词频; 公式中第 2 项为逆文档频率 IDF. 在计算 IDF 值时要考虑 t_i 是否在其他需求条目中出现. A 是从需求条目集合中建立语料库: $A=\{d_1,\ldots,d_m\}$, 则 t_i 的 IDF 值首先要用语料库 A 中所有文档总数除以包含 t_i 的文档数量, 再对此值取对数. 对 d 中的每个词计算 TF-IDF 权重, 即可得到 d 对应的文本向量 $\vec{d}=(w_1,\ldots,w_n)$.

$$w_{t_i}=\frac{freq(t_i)}{\left|\vec{d}\right|}\cdot\log\frac{|A|}{\left|\{d_j:d_j\in A,t_i\in d_j\}\right|} \tag{1}$$

在绝大多数需求问题日志中, description 字段的长度比其他 3 个字段的长度之和还大. 而由 TF-IDF 文本向量化的公式 (1) 可知, 词汇的 TF-IDF 权重与文档的长度成反比 (词汇的 TF 值计算的是词汇的归一化词频, 即用词汇在文档中出现的次数与文档长度的比值). 这意味着, 当将需求问题日志中这 4 个字段拼接到一起时, 由于 description 文本长, 会降低其他 3 个字段中的词汇的权重, 即被 description 字段中的噪声稀释.

因此, 为了适应开源系统中需求问题日志的特点, 本文在文本向量化的词汇加权过程中额外考虑了 *S*、*L*、*C*、*D* 这 4 个字段的长度. 当一条需求中 *S*、*L*、*C* 三者的长度之和小于 *D* 的长度时, 本文基于长度比值设计权重调节因子以调整权重来平衡不同字段中信息浓度的差异, 防止 *S*、*L*、*C* 中的关键词权重被过于弱化. 具体来讲, 给定一条需求 $R=\{S, C, L, D\}$, 公式 (2) 中, S_C_L 表示 *S*、*L*、*C* 合并后得到的文本, 其长度为 $|S\_C\_L|$; D 表示 description 中的文本, 其长度为 $|D|$. 对 S_C_L 中包含的每个关键词 k_i, 我们将其 TF-IDF 值——$tf_{ki}\times idf_{ki}\times|D|/|S\_C\_L$ (权重调节因子), 作为 k_i 的权重, 解决由于 $|D|$ 过大导致 k_i 权重被稀释的问题. 由此得到了一个向量 $S\_C\_L=(wk_1,\ldots,wk_n)$; 对于 D, 则使用传统的 TF-IDF 方法进行权重计算. 然后如公式 (3) 所示, 将 S_C_L 和 D 的文本向量相加, 即得到用来表示该需求问题日志的文本向量 $\vec{r}$.

$$w_{ki}=\frac{|D|}{\left|S\_C\_L\right|}tf_{k_i}\times idf_{k_i} \tag{2}$$

$$\vec{r}=\left(\overrightarrow{S\_C\_L}\right)+\vec{D} \tag{3}$$

2.3 基于调用距离的代码中词汇关系抽取

基于信息检索的方法面临的一项主要挑战就是追踪链接的可能性大小取决于源制品与目标制品在描述上的相似程度. 换言之, 当源制品和目标制品存在术语不一致问题时, 即用不同的词汇描述同样的概念时, 该方法可能会漏掉一些文本相似度较低, 但实际上存在追踪关系的制品对. 为了解决此问题, 研究者们提出了基于领域本体的方法[13]和基于辞典的方法[12]来优化传统的信息检索方法. 基于辞典的方法 (thesaurus-based VSM) 考虑到了词汇间的关系 (如近义词关系), 使得信息检索过程中能够捕捉到不同术语间的关联性, 经实验证明, 该类方法能够缓解需求追踪过程中的术语不一致问题, 提升传统 VSM 的表现. 采用该方法时, 向量空间中的各维度不再是正交关系. 辞典 T 是一个形如 (k_i, k_j, a) 三元组的集合, 其中, k_i 和 k_j 是需求制品语料库中的词汇, 而 a 是 k_i, k_j 间的相关系数. 当构建起辞典 $T=\{(k_i, k_j, a)\}$ 以优化 VSM 时, 源制品文本向量 $q=(q_1,\ldots, q_n)$ 和目标制品文本向量 $d=(d_1,\ldots, d_n)$ 的相似度计算方法如公式 (4) 所示.

$$sim(d,q)=\frac{\sum_{i=1}^{n}d_i\cdot q_i+\sum_{(k_i,k_j,\alpha_{i,j})\in T}\alpha_{i,j}(d_i\cdot q_j+d_j\cdot q_i)}{\sqrt{\sum_{i=1}^{N}d_i^2\cdot\sum_{i=1}^{N}q_i^2}} \tag{4}$$

词汇关系的手工标注要消耗额外的人力物力, 目前已有一些从领域文档中抽取术语间关系的方法[36,37], 但研

究表明, 此类方法的效果依赖于文档制品的规范性和复杂性[37], 文档越规范、复杂性越低, 抽取的术语关系的质量越高. 我们在第 1.1 节中提到, 开源系统的需求文本存在信息冗杂, 句子结构不完整多等问题, 规范性较低, 因此从中抽取术语关系也非常困难.

本文统计了 3 个开源系统中包含代码相关文本的问题日志数量, 通过检索需求中的组件名、类名, 以及正则表达式匹配方式, 发现描述中存在与项目代码相关的文本信息 (代码片段、包名、类名、方法名) 的需求问题日志在半数以上. AIRFLOW 项目里共有 3 846 条需求, 其中 2 819 条包含代码片段, 占比 73.3%; 在 DROOLS 的 2 721条中 1 538 条含代码片段, 占比 57%; JBTM 中则是 1 849 条中的 1 298 条含代码片段, 占比 70.2%. 而相比于自然语言, 面向对象语言 (object-oriented programming language) 编写的项目代码有着结构化的组织方式, 便于抽取词汇关系. 基于现有的代码分析工具 (如 JavaParser (www.javaparser.org/) 和 Python AST (https://docs.python.org/3/library/ast.html)), 可以实现自动化获取包结构, 以及类间、类与接口间、类与方法间的关系, 自然地也能够建立起包名、类名、方法名中词汇间关系. 对这些词汇和其间关系的抽取有助于构建更高质量的跨层需求追踪关系. 例如, 一条高层需求的描述中出现了组件名 C 或包名 P, 而与其有追踪关系的低层需求的实现代码中通常会出现组件 C 或包结构 P 下的类名, 或类的成员方法名. 因此, 为了充分利用代码结构信息, 本文提出一种基于调用距离的代码中关键词汇关系抽取方法, 以构建代码词汇库, 以< 词汇, 词汇, 相关度系数> 的三元组形式, 保存词汇间的相关度信息.

本文主要关注代码层级结构信息 (如包与类的包含关系、类与成员方法的包含关系), 以及面向对象编程语言特性中类之间的继承和依赖关系、类与接口间的实现关系、类和方法的调用关系等. 结构上的包含关系是一种整体-部分关系, 类间的继承、类和接口间的实现关系则是由一般到特殊、由抽象到具体的关系, 而抽象的高层需求经过细化和分解, 生成更细粒度的低层需求, 此过程同样是由整体到部分或从抽象到具体的过程. 因此, 我们认为, 当高层需求中的词汇与低层需求中的词汇存在以上关系时, 该跨层需求间存在追踪关系的可能性也更高. 在后续的研究中, 我们将探索更多关系类型对构建跨层追踪关系的作用, 如从 Spring 框架 (https://spring.io) 中的依赖注入等特性.

本文从 GitHub 中收集开源项目的源代码, 通过分析代码库中的包结构抽取包名、类名、方法名等关键词汇, 逐层提取包与类之间的包含关系, 利用面向对象语言中对象间的关系, 基于代码静态分析, 提取类与类间的继承、类与接口的实现关系 (仅针对 Java 语言编写的代码, Python 中无接口概念)、类与方法间的调用关系[38]等, 来获取这些词汇间的相关程度. 对 Java 和 Python 项目, 分别采用 JavaParser 和 Python AST 两种静态分析工具, 首先分析代码的包结构, 保存每个包名、包中的类名之间的层级关系; 然后, 从代码文件中抽取类的父类名、成员方法名以及该类实现的接口名, 类调用的方法名等. 若他们出现在高层需求或低层需求文本描述中, 将它们作为术语词汇保存, 并保存它们在代码结构中的关系.

基于以上信息, 本文定义了一对代码相关词汇<k_i, k_j> 之间的调用距离, 暂时认为各关系所指示的词汇间关联程度大致相同, 故将具有以上关系的词汇间距离统一设置为 1. 如表 3 所示. 特别地, 如果词汇 k_i, k_p 间不存在表 3 中所述的关系, 则自动寻找是否存在与 k_i, k_p 同时存在调用关系的词汇 k_j. 如果有, 则 $dist(k_i, k_p)= dist(k_i, k_j)+dist(k_j, k_p)$. 举例来讲, 如果 $dist(k_i, k_j)$ =1, $dist(k_j, k_p)$=1 时, 则 $dist(k_i, k_p)$=2. 考虑到那些调用距离大于 3 的词汇对之间的实际相关程度已经较低, 本文在构建代码词汇库时, 只记录调用距离小于或等于 3 的词汇对.

表 3 代码相关词汇对间的调用距离

词汇对<k_i, k_j> 类型	关系	$dist(k_i, k_j)$
k_i, k_j都为类名	二者之间有继承关系	1
k_i为类名, k_j为接口名	二者之间有实现关系	1
k_i为包名, k_j为类名	类 k_j 处于包 k_i 的包结构下	1
k_i为类名, k_j为方法名	类 k_i 调用方法 k_j, 或 k_j 是 k_i 的成员	1

直观来讲, 每一对词汇的相关度系数与距离成反比, 即距离越大, 相关度就越小. 因此, 基于词汇间的调用距离, 本文定义每一对词汇的相关度系数 a_{ij} 如公式 (5) 所示.

$$a_{ij}=\frac{1}{1+dist(k_i,k_j)} \tag{5}$$

至此, 可以得到项目代码词汇库. 当得到代码中关键词汇及其间的相关度系数以后, 本文采纳公式 (5) 计算跨层需求条目文本向量之间的相似度.

2.4 面向关键特征维度的相关性反馈: 一种改进的相关性反馈方法

Wang 等人[25]发现, 在标注好的追踪链接中, 极少有 (在其调研的 6 个开源系统中, 仅有 4.3% 的链接) 被修改或删除. 这在一定程度上说明了用户通常会标注他们比较有把握的追踪关系, 因此这些标注好的追踪链接绝大多数是可信的. 而相关性反馈技术恰好可以充分利用这些有限规模的标注数据, 不断改进由高层需求构建的查询语句文本向量.

相关性反馈技术是提高信息检索方法应用效果的常用手段之一, 该技术利用用户对检索结果的反馈意见来对原始查询语句进行改进[14]. 经典的相关性反馈方法 Standard Rocchio 具体实施步骤如下.

1) 使用原始查询语句对待检索文档进行查询, 得到查询结果.

2) 将与查询语句相似度最高的前 k 项返回给用户, 用户标注这 k 个待检索文档是否与查询语句相关.

3) 根据用户的标注结果, 调整原始查询语句文本向量的词汇权重, 并用它进行新的检索任务.

4) 以上过程可重复数次直至分析师对检索结果满意为止.

对于给定的原始查询语句 Q_o, Standard Rocchio 按照公式 (6) 所示的方法调整权重, 得到步骤 3) 中的新查询语句 Q_{new}.

$$\overrightarrow{Q_{\text{new}}}=\left(\alpha\cdot\vec{Q_o}\right)+\left(\beta\cdot\frac{1}{|D_r|}\cdot\sum_{\vec{d_j}\in D_r}\vec{d_j}\right)-\left(\gamma\cdot\frac{1}{|D_{nr}|}\cdot\sum_{\vec{d_k}\in D_{nr}}\vec{d_k}\right) \tag{6}$$

其中, D_r 为被用户标注为“相关”的待检索文档集合, 共包含$|D_r|$个文档, 用于对原始查询向量做正反馈调整; D_{nr} 则为用户标注为与 Q_o 相关的文档集合, 包含$|D_{nr}|$个文档, 用于查询向量的负反馈调整, 一般只用一个文档做负反馈. α, β 和 γ 分别是针对原始查询向量、相关文档向量和不相关文档向量的权重系数. 先前的相关研究发现, 相比负反馈, 正反馈会起到更大的作用, 并且通常将权重系数设置为 α=1.0, β=0.75, γ=0.25 时, 会取得较好的效果[39,40]. 公式 (7) 很直观地展示了相关性反馈技术的思路, 即增加那些出现在正反馈文档中的词汇的权重, 同时削弱出现在负反馈文档集合中的词汇权重. 第 3) 步是相关性反馈方法中最关键的步骤, 研究者通常在这一步骤中对反馈策略做出改进, 以提升算法效果[16,17], 本文同样针对此步骤面向开源系统需求做了改进.

传统的 Standard Rocchio 使用与查询语句相关的反馈文档中包含的所有词汇对原始查询语句进行调整. 而由于开源项目的问题日志描述具有用语不规范、噪音多、冗长等特点, 使得与查询语句相关文档中也包含许多不相关的词汇. 因此, 如果在开源系统的需求追踪过程中直接采用 Standard Rocchio 方法, 会给原始查询语句带来新的噪声, 造成用修改后的查询语句进行新的检索时, 效果不升反降. 因此, 如何自动筛选追踪关键的词汇非常关键. 本文提出了需求特殊性指标, 并基于此指标从相关需求描述文本中筛选出最具代表性的词. 本文仅在这些关键词汇对应的特征维度上对原始查询语句做调整, 使得修改后的查询语句在向量空间中更接近应该建立追踪关系的文档.

需求特殊性指标 $ReqS$ 衡量了特定关键词对一条需求描述的代表性. 给定一条需求描述 req, 其包含的词汇或短语 t 的 $ReqS$ 值可以由公式 (7) 计算:

$$ReqS(t,req)=\left\lceil\frac{f(t,req)}{\sum_{u\in req}f(u,req)}\Big/\frac{f(t,g)}{\sum_{v\in d}f(v,g)}\right\rceil \tag{7}$$

其中, $f(t, req)$ 是 t 在该问题日志条目中的词频, $\sum_{u\in req}f(u,req)$ 是该需求条目描述中所有词汇的词频之和. $f(t, g)$ 是在需求文本语料库中除了 req 以外的其他需求条目中词汇 t 的词频之和. $f(v, g)$ 则是需求本文语料库中除了 req 以外的其他文档中所有词汇的词频之和. 有些词汇 $ReqS$ 较高, 但仅在一条低层需求中出现过, 用这样的词汇做反馈调整并不能使调整后的查询语句在新一轮的检索中找到更多的正确追踪连接, 因此我们还剔除了那些仅在一条低层需求中出现过的词汇.

对于用户反馈为存在追踪关系的一对高层需求和低层需求< *hlr, llr*>, 我们计算 *llr* 中各关键词的 *ReqS* 值, 并通过以下算法筛选关键特征维度 *k_dims* 并进行相关性反馈, 对查询语句 *hlr* 对应的文本向量 $\overrightarrow{hlr}$ 进行调整, 得到 $\overrightarrow{hlr_m}$.

算法 1. 面向关键特征维度调整高层需求文本向量.

Input: *keywords_code,* $\overrightarrow{hlr}$, $\overrightarrow{llr}$;

Output: modified *hlr* query vector $\overrightarrow{hlr_m}$.

```
//Line 1–8: collect key dimentions in llr
1.  Collect_k_dims(keywords_code, llr)
2.    k_dims ← screen keywords only appeared in one llr and get top 10 keywords in llr ranked by ReqS
3.    FOR each keyword k in llr:
4.    IF k in keywords_code:
5.      Append k to k_dims
6.    ENDIF
7.  ENDFOR
8.  RETRUN k_dims
//Line 9–15: modify weights of key dimentions in hlr with relevance feedback
9.  Modify_query(hlr, llr, k_dims)
10.   For k in llr:
11.     IF k in k_dims:
// Line12: modify the weight of k. W_k: weight of k; α =1.0, β=0.75
12.       W_k_hlr_m ← α*W_k_hlr + β*W_k_llr
13.     ENDIF
14.   ENDFOR
15. RETURN hlr_m
```

该算法的输入为一条高层需求的文本向量 $\overrightarrow{hlr}$, 被用户标记为与之有追踪关系的低层需求的文本向量 $\overrightarrow{llr}$, 和代码相关关键词列表 *keywords_code* (见第 2.3 节), 同时为了计算词汇的 *ReqS* 值, 还要以高层、低层需求集合 (*HLR_set, LLR_set*) 作为输入. 算法的第 1–8 行是关键特征维度筛选的过程. 首先, 排除那些仅在一条 *llr* 中出现的词汇, 并将 $\overrightarrow{llr}$ 中的剩余词汇按照其 *ReqS* 值降序排列, 然后选择前 10 个词汇, 并将它们代表的维度纳入关键特征维度 *k_dims*. 对于 $\overrightarrow{llr}$ 中的任意其他词汇 *k*, 若 *k* 出现在 *keywords_code* 中, 则也将 *k* 在向量空间中代表的维度纳入关键特征维度 *k_dims*.

算法的第 9–15 行是面向关键特征维度对原始查询语句向量 $\overrightarrow{hlr}$ 的修改过程. 对于 $\overrightarrow{llr}$ 中的每一个词汇 *k*, 若 *k* 在关键维度 *k_dims* 中, 则对 $\overrightarrow{hlr}$ 中关键词 *k* 的权重 W_k_hlr 进行调整. 然后得到改进后的查询语句文本向量 $\overrightarrow{hlr_m}$.

这样, 我们针对每条查询语句执行面向关键维度的相关性反馈算法, 得到改进后的查询语句向量集合, 并在第 2.3 节中所述的代码词汇库的支持下, 两两计算高层需求的查询语句文本向量和低层需求的文本向量之间的余弦相似度, 识别开源系统中高层需求和低层需求间的追踪关系.

3 实验验证

本节详细介绍了实验验证过程. 首先, 介绍所选的开源项目、数据集的获取过程, 以及数据集规模等具体信息. 其次, 介绍了选择对照的基线方法和评价指标. 最后, 介绍实验场景与研究问题, 并对实验结果进行展示和分析.

3.1 数据获取

本文选择了 3 个活跃 5 年以上、涉及不同领域、采用不同编程语言实现的开源项目: AIRFLOW、DROOLS 和 JBTM, 项目信息和跨层需求追踪数据规模如表 4 所示. 其中 AIRFLOW 项目来自 Apache 基金会, 是一款分布式任务调度框架, 主要采用 Python 语言编写. DROOLS 和 JBTM 则由 Redhat 基金会赞助, 主要使用 Java 语言实现. 其中 DROOLS 是一款易访问、易调整、高效率的业务规则引擎, 而 JBTM (JBoss transaction manager), 则是一款事务流程管理工具. 从规模上来看, 3 个项目包含的组件数都在 30 个以上, 代码行数分别为 19 万、83 万和 179 万以上, 属于大型复杂软件项目. 我们之所以选这 3 个项目, 是因为它们具备一定的规模, 且都采用 JIRA 问题日志管理系统作为需求管理工具, 都包含了多种跨层次需求类型, 同时需求间追踪关系较为全面, 在以往的需求追踪研究中也多次被其他研究者使用[25,35].

表 4 实验所选择的项目采用的数据集信息

项目	涉及领域	项目起始时间	需求规模		追踪关系数量	所含组件数	代码规模 (line of code)
			高层	低层			
AIRFLOW	工作流执行	2014年10月	313	3 533	428	43	193 101
DROOLS	业务规则管理	2012年11月	1 117	1 604	275	30	830 204
JBTM	事务流程管理	2005年12月	984	865	66	32	1 791 399

我们使用 JIRA 提供的官方 API (https://jira.readthedocs.io/) 采集 3 个案例项目在 2021 年 7 月 14 日之前的问题日志数据. 如前文所述, 本文关注的问题日志类型包括 Epic, Wish 等高层需求, 以及 Feature, Enhancement, Sub-task 等低层需求. 对于每一条需求 (问题日志), 我们抽取其 summary, labels, components 和 description 等关键字段中的文本信息作为识别追踪关系的依据. 同时我们采集了 JIRA 中记录的这些跨层需求之间的追踪关系, 作为后续实验中评估自动追踪方法的标准集.

对于 AIRFLOW 项目, 共抽取出 313 条高层需求, 3 533 条低层需求和 428 条跨层需求追踪链接. DROOLS 项目的需求集合中包含了 1 117 条高层需求, 1 604 条低层需求以及 275 条跨层需求追踪链接. JBTM 需求集则包括了 984 条高层需求, 865 条低层需求, 以及其间的 66 条追踪关系.

3.2 评价标准和基线方法选择

本文采用了需求追踪领域常用的评估指标, 即召回率 (*Recall*)、准确率 (*Precision*) 和 *F*2 值 (*F*2). 召回率评价了自动算法对追踪链接识别的全面性, 即所识别出的正确追踪链接数在所有正确链接数中所占的百分比. 准确率则评价了自动算法对追踪链接识别的准确性, 即在所有识别结果中, 正确链接所占的比例. *F*-measure 是对召回率和准确率的调和平均值. 我们之所以选 *F*2, 是因为在大量需求集中, 过滤一条假阳性链接要比从大量候选需求对中识别一条真阳性链接容易得多, 故召回率比准确率重要.

对于给定的一组高层需求集和低层需求集, 用 T 表示它们之间的真实追踪链接集合, 用 C 表示算法识别出的候选追踪链接集合, C_r、C_w 分别表示候选链接集合中的正确链接集合和错误链接集合, 即 $C=C_r+C_w$. 则有:

$$Recall=\frac{C_r}{T},\ Precision=\frac{C_r}{C} \tag{8}$$

*F*2 值是对召回率和准确率的调和:

$$F2=\frac{5\cdot Precision\cdot Recall}{4\cdot Precision+Recall} \tag{9}$$

我们选择了需求追踪领域中最常用的 VSM 方法[9], 经典的相关性反馈技术 Standard Rocchio[15]以及 Trace BERT 模型[23]作为基线算法. 选择 VSM 的主要原因是其轻便且被证明在计算多个软件相关数据集的相似度时表现出了一致的优越性[41], 具有良好的实践效果. 而 Standard Rocchio 是对 VSM 技术的改进, 已被证实在多个数据集上取得比 VSM 更好的效果[12,16]. 此外, 我们还选择了 Trace BERT[23]作为基线算法, 原因是该方法采用“在线负采样策略”解决了深度学习技术应用于需求追踪领域时容易出现过拟合的问题, 并通过领域知识迁移提升了追踪

关系识别效果. 实验证明, 该方法能够产生比信息检索等技术更优越的效果 (包括查全率和查准率). Trace BERT 以微软发布的针对代码和自然语言的 MS-CodeBERT[42]作为预训练模型, 通过在 CodeSearchNET 数据集 (https://github.com/github/CodeSearchNet) 中提供的大规模“NL-PL pair”数据集进行训练, 以实现领域知识的迁移. 最后采用从开源项目问题跟踪系统中获取的“issue-commit”数据进行调优. 通过对 TWIN、SIAMESE 和 SINGLE 这 3 种 BERT 架构的实验评估, 发现 Trace BERT 的表现优于 VSM、LSTM 等基线方法, 且采用 SIAMESE 架构时 Trace BERT 的性能和训练速度最佳. 因此本文采用了 Trace BERT 中表现最好的 SIAMESE 结构作为一个基线方法.

3.3 实验场景与研究问题

本实验围绕实践中需求追踪关系构建相关的两个典型应用场景展开.

场景 S1. 追踪范围确定而追踪链接不确定

假设有大型复杂软件项目 A, 在项目实践早期阶段曾进行过认真的需求分析与分解, 也即明确将高层需求集 HLR 分解为低层需求集 LLR (HLR 中不含暂不实现的高层需求). 但由于项目时间压力及人手不足等问题, 没有及时将 HLR 与 LLR 间的追踪关系文档化, 导致随着时间的推移及人员的流动, 两者间的追踪关系彻底丢失. 在此场景下, HLR 中每一条高层需求都有对应的低层需求来实现, 同时 LLR 中每一条低层需求必源于 HLR 中某高层需求, 但具体的来龙去脉不确定. 因此, 需要在确定的有限的需求范围间 (HLR, LLR) 建立跨层需求之间的追踪关系.

针对此场景, 本文基于 3 个开源项目需求集中的追踪关系结果, 筛选出存在追踪关系的高层需求和低层需求, 运用本文方法及基线算法来自动构建出 HLR 与 LLR 需求间的追踪关系.

场景 S2. 追踪范围与追踪链接都不确定

假设有大型复杂软件项目 B, 虽然在项目开展早期阶段曾进行过认真的需求分析, 为了项目推进, 将部分高优先级的高层需求分解为低层需求. 但由于项目时间压力, 人手不足等问题, 没能及时将有追踪关系的需求集及其间的追踪关系文档化, 没能及时区分已被分解与尚未被分解的高层需求集. 同样地, 有一部分低层需求可能为其他需求的公共支撑, 无明确的高层需求来源, 也没有及时区分有无高层需求来源的低层需求. 换言之, 对于高层需求集 HLR 与低层需求集 LLR, 可能只有部分 HLR 与部分 LLR 间存在追踪关系, 而有一些需求未涉及任何追踪关系. 而同一项目的需求之间或多或少存在依赖关系 (如语义相似, 用词重叠等), 给跨层需求追踪带来更多的噪音与干扰.

鉴于目前 3 个开源项目的跨层需求集中均存在部分未能分解的高层需求与暂无来源的低层需求, 此场景下, 直接将本文方法及基线算法用于 3 个开源项目的所有需求集以自动构建跨层需求间的追踪关系.

这两个场景中, 对于信息检索系列的 3 个方法 (包括本文方法, VSM 和 Standard Rocchio), 均以高层需求为查询语句集合, 低层需求为待检索文档集合.

实验中要解决的研究问题 (research question, RQ) 如下.

RQ1: 在场景 S1 与 S2 中, 本文方法对跨层需求追踪的识别效果是否优于基线算法?

RQ2: 本文的方法在哪种场景下表现得更好?

RQ3: 本文方法中的代码词汇库和基于关键特征维度的相关性反馈方法, 是否均对跨层需求的追踪有正向作用?

本文提出的方法和基线方法都是基于需求文本对间的相似度来识别追踪关系, 随着阈值的变化, 其返回的候选追踪关系也不同. 我们通过对阈值的调整, 分别计算了 3 个方法所能达到的最优 $F2$ 值, 以及此 $F2$ 值对应的准确率和召回率.

3.4 实验设置

对于每一个项目, 其高层需求集合、低层需求集合分别用 HLR 和 LLR 表示, JIRA 中记录的追踪关系答案集用 $Trace_{answer}$ 表示. 针对场景 S1, 我们筛选出那些有需求链接的高层和低层需求条目集合 HLR_linked 和 LLR_linked, 分别作为源制品和目标制品集; 针对场景 S2, 则直接使用 HLR 作为源制品集合, LLR 作为目标制品集合.

(1) 基于信息检索的方法

对于 VSM, 基于 TF-IDF 构建向量空间, 将需求文本向量化. 计算 HLR (HLR_linked) 中每条查询语句向量与 LLR (LLR_linked) 中待检索文本向量间的余弦相似度, 得到候选追踪关系集合 $Trace_{vsm}$, 通过与 $Trace_{answer}$ 作比

对, 采用 *Precision*, *Recall* 和 *F*2 等指标进行评价.

为了回答 RQ3, 即验证本文方法的两个设计 (代码词汇库和关键特征维度的选择) 的有效性, 针对已有相关性反馈技术 Standard Rocchio, 设置了 3 套实现. 首先, 实现传统 Standard Rocchio, 即对于每条高层需求文本向量, 选取被标注为与其有追踪关系的且与其相似度最高的 top-2 低层需求文本向量用于改进高层需求文本查询语句, 然后再次计算改进后的高层需求与低层需求间的文本余弦相似度, 得到候选追踪关系 *Trace_rf*. 通过对比 *Trace_rf* 与 $Trace_{answer}$, 评估 Standard Rocchio 的表现. 其次, 为了验证代码词汇库的有效性, 在传统 Standard Rocchio 经过查询向量改进后, 借助代码词汇库, 在识别与各低层需求文本向量的追踪关系时, 利用词汇库 (以<$term_i$, $term_j$, $dist_{(ij)}$> 形式存储), 采用公式 (4) 计算余弦相似度, 得到候选追踪关系 $Trace\_rf_{thesaurus}$, 与 Standard Rocchio 得到的候选追踪关系 *Trace_rf* 对比, 观察代码词汇库对追踪关系识别的作用. 最后, 为了验证关键特征维度选择的有效性, 结合了面向关键维度的相关性反馈技术和基于代码词汇库的余弦相似度计算: 在相关性反馈阶段, 按照算法 1 筛选低层需求文档中具有代表性的关键词汇对相关高层需求文本向量做调整; 在文本向量调整后的文本相似度计算阶段引入代码词汇库, 得到候选追踪关系集 *Trace_kdim*, 将其评估结果与 $Trace_{answer}$, $Trace\_rf_{thesaurus}$ 对比, 以验证面向关键维度的相关性反馈技术的效果. 最后一个实现也是本文方法的完整实现.

(2) Trace BERT

对于 HLR (HLR_linked) 中任意一条高层需求 R_i 和 LLR (LLR_linked) 中的低层需求 r_j, 我们将这一对需求的文本 (R_i, r_j) 作为一条数据. 根据 $Trace_{answer}$, 若 R_i 和 r_j 间存在追踪关系, 则将其标签标注为 1, 否则标注为 0. 按照这样的方式, 我们生成了 3 个项目的需求追踪集. 在 Trace BERT 的追踪关系识别实验中, 本文按照文献 [23] 中推荐配置设置了模型参数. 将模型 batch size 设置为 8, 学习率 (learning rate) 为 1E–5, 对模型进行了 400 轮次 (epoch) 的训练. 我们在 S1 和 S2 场景中均采用了五折交叉验证, 即将数据集拆成 5 份, 每次选择一份做测试集, 按照 3 (训练集):1 (验证集):1 (测试集) 的比例划分, 共得到 5 份数据集, 其中训练集、验证集和测试集中的正负例的比例保持一致, 在试验结果评估时则取了 5 份数据集的平均值.

3.5 实验结果与分析

(1) 针对场景 S1

针对实验场景 S1 的实验结果如表 5 所示. 为了便于比较, 我们对每个数据集上表现最好的指标结果进行了加粗标识. 同时我们计算了本文方法相对于 VSM、Standard Rocchio 以及 Trace BERT 在 3 个指标上取得的改进 (improvements) 和 gains 值 (*a* 相对于 *b* 的 improvement 计算方式为 $a - b$, gain 计算方式为 $(a - b)/b$), 见表 6, 其中括号内为 gains.

表 5 S1 场景下的跨层需求追踪关系识别结果 (%)

数据集	评价指标	VSM	Standard Rocchio	Standard Rocchio+词汇库	Trace BERT	本文的方法
AIRFLOW	*Precision*	15.46	23.85	19.56	5.32	26.72
	Recall	65.61	63.41	72.20	59.31	71.95
	*F*2	39.79	47.62	46.94	8.56	53.75
DROOLS	*Precision*	32.93	37.03	47.27	1.06	41.05
	Recall	50.70	63.72	60.47	51.66	72.56
	*F*2	45.76	55.69	57.27	3.69	62.90
JBTM	*Precision*	38.64	50.50	61.51	7.46	45.26
	Recall	51.52	77.27	74.24	62.56	93.94
	*F*2	48.30	69.86	71.29	23.51	77.31

结合表 5 和表 6 可以看出:

1) 整体来看, 本文的方法在 3 个项目中基于 3 个指标的对比上几乎全部胜出: *F*2 值相比于 VSM 提升了 13.96%–29.01%, 取得 gains 范围在 35.08%–60.06% (平均 44.2%); 相比于 Standard Rocchio, *F*2 提升了 6.13%–7.45%, gains 约 10.66%–12.87% (平均 11.62%). 虽然在 JBTM 数据集上, *Precision* 值小幅度低于 Standard Rocchio

(5.24%), 但综合指标 *F*2 值仍在 3 个方法里最优 (比 Standard Rocchio 高 7.45%). 受限于追踪关系规模和需求文本质量, Trace BERT 在 3 个项目中 *F*2 分别为 8.56%, 3.69% 和 23.51%, 相比于 Trace BERT, 本文的方法的 *F*2 提升高达 45.19%, 59.21% 和 53.80%. 这意味着方法中的代码词汇库+关键维度的选择与应用对于跨层需求追踪的有效性.

表 6　S1 场景下本文方法相对于基线算法在各指标上的提升 (%)

数据集	评价指标	VSM-TF-IDF	Standard Rocchio	Trace BERT
AIRFLOW	*Precision*	11.26 (72.83)	2.87 (12.03)	21.40 (402)
	Recall	6.34 (9.66)	8.54 (13.47)	12.64 (21.31)
	*F*2	13.96 (35.08)	6.13 (12.87)	45.19 (528)
DROOLS	*Precision*	8.12 (24.66)	4.02 (10.86)	41.05 (3 773)
	Recall	21.86 (43.12)	8.84 (13.87)	20.90 (40.45)
	*F*2	17.14 (37.46)	7.21 (11.32)	59.21 (1 605)
JBTM	*Precision*	6.62 (17.13)	−5.24 (−10.38)	37.80 (507)
	Recall	42.42 (82.34)	16.67 (21.57)	31.38 (50.16)
	*F*2	29.01 (60.06)	7.45 (10.66)	53.80 (229)

2) 与基线算法的纵向对比来看:

● 相对于 VSM, 在 AIRFLOW 和 DROOLS 数据集上, 本文方法的 *F*2 值分别提升了 13.96% 和 17.14%, gains 分别为 35.08% 和 37.46%. 在 JBTM 项目中, 本文方法在 94.94% 的召回率水平下, 达到了 45.26% 的准确率, *F*2 值为 77.31%, 比 VSM 提升了 29.01%. 可以说, 本文方法相对于 VSM 有明显的改进, 意味着通过代码词汇库的运用和基于关键特征维度相关性反馈, 修改后的查询语句与其有追踪关系的需求描述在向量空间里更为接近.

● 相对于 Standard Rocchio, 在 AIRFLOW、DROOLS 和 JBTM 这 3 个项目中, 本文方法对跨层追踪关系的识别效果均更好, *F*2 值分别高出了 6.13%, 7.21% 和 7.45%.

● 对比 VSM 与传统 RF 方法 Standard Rocchio, 发现 Standard Rocchio 在 3 个项目中都取得了更好的效果 (*F*2 分别提升: 7.83%, 9.93%, 和 21.65%), 这体现了相关性反馈技术对查询语句的调整有效提升了追踪效果.

● 对比基于深度学习的方法 Trace BERT 在 3 个项目中的表现 (*F*2 分别为 8.56%, 3.69% 和 23.51%), 本文的方法在 *F*2 值上的提升最高可达 53.08%. 另外, Trace BERT 的表现与其他基于信息检索的基线方法也有明显差距, 我们认为可能原因是开源系统跨层需求场景下, 可获得的训练数据规模有限 (3 个项目中追踪关系分别有 428、275 和 65 条), 且需求规范性低造成的. Lin 等人[21,43]指出, 当追踪关系数据不足时, 基于深度学习方法表现会与 VSM 等信息检索方法有较大的差距. 基于深度学习的方法面临一个共同的问题, 即其表现十分依赖于训练集的规模和质量, Dekhtyar 等人[44]认为, 需求追踪问题通常是“低资源、小数据”问题, 跨层需求追踪问题更是如此. 另外, 在开源系统中, 需求制品文本的规范性低, issues 中存在的大量无区分度的、大量使用的词语, 以及错误日志、调试信息等噪声项, 给模型的训练带来了干扰.

3) 横向对比本文方法在 3 个项目上的效果, 可以看出, 4 个算法均在 JBTM 中取得最佳的识别效果, 特别是本文方法的查全率能达到 93.94%, 在 DROOLS 上效果其次. 可能的原因有两个: 首先, JBTM 中存在追踪关系的高层需求和低层需求对中重叠的词汇较多, 文本相似度也较高, 因此, 它们更容易被以 VSM 为基础的方法识别出来; 其次, 相比于 DROOLS 和 AIRFLOW, JBTM 中高层需求查询语句所对应的相关待检索文档 (低层需求) 向量在向量空间中较为密集, 而经过相关性反馈对查询语句进行改进后, 使查询语句向量与这些相关文档更为接近. 这样的应用场景对于基于相关性反馈的方法更加理想, 很好地发挥了相关性反馈方法的优势.

4) 观察 Standard Rocchio 方法在结合代码词汇库时的表现, 可以发现, 在 DROOLS 和 JBTM 项目中, 对词汇库的使用起到了积极作用, 使得 Standard Rocchio 的最优 *F*2 值有所提升. 然而在 AIRFLOW 项目中, 使用了词汇库之后的表现反而略低于 Standard Rocchio. 通过分析 3 个项目中的数据集, 发现导致此现象的原因可能有两个.

首先, AIRFLOW 项目中, 高层需求里包含的代码片段较少, 因此词汇库的作用也有限; 其次, 我们观察了从 AIRFLOW 代码库中提取的代码词汇关系列表, 发现其项目的代码中包含一些使用通用词汇命名的类或方法名, 如<“action”, “state”> 等词汇关系, 而这些词汇在需求描述中也存在, 并且实际上不存在语义关联, 因此, 词汇库的使用反而导致假阳性结果数量增加, 效果有所下降. 我们需要在后续的改进中对代码词汇库进行过滤和筛选, 排除那些在自然语言描述中也经常出现的词, 提高词汇库的质量.

(2) 针对场景 S2

针对实验场景 S2 的实现结果如表 7 所示. 如同针对 S1 的实验, 我们同样计算了本文方法相对于基线算法在 3 个指标上的 improvement 和 gains 见表 8 (括号内为 gains).

表 7　S2 场景下对所有需求条目识别追踪关系的结果 (%)

数据集	评价指标	VSM	Standard Rocchio	Standard Rocchio+词汇库	Trace BERT	本文的方法
AIRFLOW	*Precision*	7.08	12.66	**14.76**	1.36	14.52
	Recall	37.82	44.56	36.08	**62.58**	49.74
	*F*2	20.24	29.62	27.99	5.95	**33.50**
DROOLS	*Precision*	4.69	5.97	6.21	1.80	**6.44**
	Recall	55.74	**70.49**	68.21	18.76	67.21
	*F*2	17.54	22.30	22.76	6.13	**23.27**
JBTM	*Precision*	10.07	51.11	34.90	3.88	**53.19**
	Recall	21.88	35.94	42.31	**57.74**	39.06
	*F*2	17.72	38.21	40.59	6.18	**41.25**

表 8　S2 场景下本文方法相对于基线算法在各指标上的提升 (%)

数据集	评价指标	VSM-TF-IDF	Standard Rocchio	Trace BERT
AIRFLOW	*Precision*	7.44 (105.1)	1.86 (14.69)	13.16 (967)
	Recall	11.92 (31.52)	5.18 (11.62)	−12.84 (−20.51)
	*F*2	13.26 (65.51)	3.88 (13.10)	27.55 (463)
DROOLS	*Precision*	1.75 (37.31)	0.47 (7.87)	4.64 (258)
	Recall	11.47 (20.58)	−3.28 (−4.65)	48.45 (258)
	*F*2	5.73 (32.69)	0.97 (4.35)	17.14 (280)
JBTM	*Precision*	43.12 (428.2)	2.08 (4.07)	49.31 (1 271)
	Recall	17.18 (78.52)	3.12 (8.68)	−18.68 (−32.35)
	*F*2	23.53 (132.8)	3.04 (8.46)	35.07 (567)

在此场景下, Trace BERT 的效果甚至低于 S1 场景, 在 3 个项目中 *F*2 值分别只有 5.95%、6.13%, 和 6.18%, 对于导致此结果的原因在上文中已做过详细分析, 这里不再赘述.

结合表 7 与表 8, 我们可以观察到:

1) 整体上来看, 本文方法在 3 个项目数据集中的表现仍优于基线方法 VSM 和 Standard Rocchio. 相比于 VSM 方法, 在 3 个项目中 *F*2 值提升分别为 13.26%, 5.73%, 和 23.53%, 平均 gains 为 77%; 相比于 Standard Rocchio, *F*2 提升为 3.88%, 0.97% 和 3.04%, 平均 gains 为 8.64%; 相比于 Trace BERT, *F*2 值则提升了 27.55%, 17.14% 和 35.07%. 结合表 5 和表 7, 可以看出, 在 S2 场景下, 无论是本文的方法还是基线方法, 其识别效果都分别要弱于它们在 S1 中的表现, 最主要原因在于无追踪关系的高/低层需求的数量大大增加, 而同一系统的需求描述必然存在着或多或少的重叠, 给信息检索类方法带来难以消除的噪音干扰. Trace BERT 在所有基线方法中仍表现最差, 上文中已经分析了其中原因, 即训练数据规模和质量的不足.

2) 对比方法在 3 个项目的表现, 发现 Standard Rocchio 和本文方法依然在 JBTM 上效果最佳, 而 VSM 在 AIRFLOW 上结果最好, Trace BERT 在 3 个项目中 *F*2 值差距不明显 (均在 6% 左右). 此外, 4 个技术在 AIRFLOW 和 DROOLS 上的准确率都低于 20%. 主要原因是, 相比于场景 S1, 那些无追踪关系但包含相同词汇的高层需求-

低层需求对大量增加. 在这种情况下, 虽然一些文本对间实际不存在追踪关系, 但它们的文本相似度较高, 导致方法识别到的假阳性 (false positive) 追踪数量增加, 准确率大幅度下降.

3) 本场景中代码词汇库对 Standard Rocchio 的作用与 S1 中基本一致, 都是在 DROOLS 和 JBTM 中相对 Standard Rocchio, 提升了性能, 而在 AIRFLOW 中的表现却有所下降. 具体原因在场景 S1 中已经做过分析, 这里不再赘述.

至此, 通过对 S1、S2 场景中实验结果的分析, 并对比各算法在两种场景中的表现, 我们可以回答 3 个研究问题.

● RQ1. 无论是场景 S1 中仅筛选那些涉及到追踪关系的需求条目进行跨层需求追踪关系识别, 还是面向场景 S2 对数据集中所有需求条目进行追踪关系识别, 本文的方法的表现都优于 3 个基线算法. 相比 VSM 方法, 本文方法在 S1 场景下, 最优 $F2$ 值提升可达 29.01%, 在场景 S2 最多提升 23.53%. 这是因为, 开源系统的许多需求中包含了代码相关词汇, 而这些词汇间的关系也可以为需求追踪关系的识别提供线索, 本文通过项目源代码静态分析得到的代码词汇库很好地利用了这一点. 本文的方法表现同样优于 Standard Rocchio 相关性反馈方法 (在场景 S1 和 S2 下, $F2$ 值最多提升分别可达 7.45% 和 3.88%), 主要是由于本文提出的面向关键特征维度的相关性反馈方法只筛选待检索文档中具有代表性的词汇来对查询语句做调整, 有效缓解了传统方法使用待检索文档向量中所有词汇 (包括在多个文档中都出现, 区分度较低的词汇) 进行调整时给查询语句引入噪音的问题. 受限于开源系统中跨层需求数据的规模小和质量差的问题, Trace BERT 表现最弱, 与之相比, 本文方法, 在 S1 场景下, $F2$ 提升最高可达 59.21 (DROOLS 项目中), 3 个项目的平均 $F2$ 提升为 50.27%, 在 S2 场景下, 最高提升为 35.07% (JBTM 项目中), 平均提升为 26.59%.

● RQ2. 相比于场景 S2, 在实验场景 S1 中, 本文的方法相对于 VSM 和 Standard Rocchio 优越性更明显, 特别是 $F2$ 值指标有更大的提升. 场景 S2 对应的是, 需求集合中有追踪关系的需求范围不确定, 追踪链接也不确定, 即有些高层需求在提出后未能在后续的过程中被分解与实现, 而低层需求提出时也没能依据现有的高层需求. 在此场景下, 那些不涉及任何追踪关系的需求, 其文本信息中仍包含着与其他需求相同的词汇或表述, 对自动追踪造成干扰, 降低了识别效果. 并且, 随着需求条目数量的逐步累积, 需求的管理也会越发困难. 为了避免此场景, 我们应该在需求生成与管理过程中制定更规范和严格的流程规范, 及时构建与审查追踪关系, 做好记录, 以支持持续演化.

● RQ3. 本文主要从两个角度对基于信息检索的跨层需求追踪关系识别进行改进, 首先是基于关键特征维度的相关性反馈方法, 本文通过从低层需求中筛选具有代表性的词汇来对高层需求文本的关键维度向量进行调整和优化. 从 S1、S2 的实验结果来看, 该方法在 3 个项目中取得的 $F2$ 值都超过了基线方法. 其次, 本文提出通过静态分析项目对应的源代码中包、类、方法等节点的关系, 自动化提取和量化包名、类名、方法名等词汇间的语义关联, 并用于信息检索技术. 实验结果表明, 该方法在两个项目 (DROOLS, JBTM) 中的表现均优于 Standard Rocchio, 而仅在一个项目 (AIRFLOW) 中使识别效果有微小的降幅. 我们认为这主要是因为没有排除 AIRFLOW 代码词汇库中的通用词汇, 这也是后续工作的重点.

4 讨 论

4.1 有效性威胁

为了减轻内部和结构有效性威胁, 本文选择了在需求追踪领域被多次使用的 3 个开源项目的需求数据, 并且采用了领域内被广泛应用的 3 个评价指标, 即准确率、召回率和 $F2$ 值, 从不同角度综合评价了本文方法和 3 个基线算法的识别效果. 此外, 为了保证测试的充分性, 本文设计了两种实验场景, 分别测试自动化方法在追踪范围确定的需求集和追踪范围不确定的需求集间建立追踪链接的效果.

为了减弱外部有效性威胁, 我们选择了具有代表性的 3 个开源项目 (AIRFLOW, DROOLS, JBTM), 抽取其问题日志管理系统中的需求条目和跨层追踪关系作为验证数据集. 这 3 个项目都存在了 5 年以上, 并活跃至今, 且面向不同的领域, 规模各异. 尽管如此, 在今后的工作中我们仍需要筛选更多具有代表性的开源项目, 构建更大规模的数据集以提高结论的可推广性.

4.2 局限性

有限的项目术语库. 此前的研究[13,14]证实了知识库的支持可以提升追踪关系识别效果. 本文在构建项目词汇库时, 仅从项目源码库中抽取了代码的词汇关系, 这些词汇只是需求文本库中的一小部分. 在后续的研究中, 我们将尝试从项目相关的领域文档中抽取词汇, 构建更为全面的词汇关系库.

自动化的数据清理. 开源系统的问题日志文本书写风格随意, 内容复杂, 包含大量噪声 (如调试信息, 错误日志, 图片等), 我们需要结合每条需求的上下文来判断这些信息是否有用, 而需求集的规模较大, 难以一一人工确认. 因此, 我们仅采用自动化的数据清洗和预处理手段, 但这难以彻底清除这些噪声, 可能会影响到追踪关系的识别效果.

相关性反馈技术的局限性. 本文基于相关性反馈的信息检索技术展开, 也就面临着所有相关性反馈方法都存在的问题. 相关性反馈技术的假设是, 通过对查询语句的改进, 使它在向量空间中一步步更接近相关文档. 但当查询语句所对应的待检索文档在向量空间中较为分散时, 无法保证对查询语句文本向量的修改一定能使查询语句向量越来越靠近待检索文档向量, 在此情况下本方法的性能可能会有所下降. 实验说明即使有此潜在局限性, 本文方法在 3 个开源项目的实验数据上 (含 8416 条需求和 769 条追踪关系) 上的表现均优于基线算法.

5 结 论

跨层需求追踪关系的构建为软件系统的需求验证、需求确认等活动提供了关键支持, 可以辅助确认高层需求被分解和实现, 低层需求有据可依. 本文针对开源系统中需求编制风格随意, 文本形式多样, 包含大量噪声等特点, 提出了一种面向开源系统的跨层需求追踪方法. 针对开源系统需求中包含代码片段的特点, 设计从源代码中挖掘关键词汇的实现距离以弥补跨层需求抽象层次差异; 针对开源系统需求中包含调试、出错等噪声信息的问题, 设计基于关键特征维度筛选的相关性反馈方法过滤噪音提升跨层需求追踪关系识别的准确性和完整性. 本文针对需求追踪关系构建的典型应用场景, 设置了“追踪范围确定而追踪链接不确定 (场景 S1)”和“追踪范围和追踪链接都不确定 (场景 S2)”两种实验场景. 通过在 3 个开源项目数据集上的验证, 表明本文的方法在两种场景下的表现都优于基线方法. 相对于 VSM (传统信息检索方法)、Standard Rocchio (传统相关性反馈方法) 和 Trace BERT (深度学习方法) 等基线, 本文的方法在准确率、召回率和 $F2$ 值等指标上有明显提升. 其中, 在场景 S1 下, 相对 3 种基线方法的 $F2$ 值最高提升分别达到 29.01%、7.45% 和 59.21%; 在场景 S2 下，$F2$ 值的最高提升分别可达 23.53%、3.88% 和 35.07%. 此外, 消融实验证明了代码词汇库构建和基于关键特征维度的相关性反馈方法这两个设计对追踪构建的正向作用. 另外, 本文的实验结果说明, 在数据“低资源、小规模”的跨层需求追踪场景下, 项目中需求文本的规范性差、需求追踪关系数据规模不足时, 基于信息检索的方法表现优于基于深度学习的方法.

References:

[1] Gorschek T, Wohlin C. Requirements abstraction model. Requirements Engineering, 2006, 11(1): 79–101. [doi: 10.1007/s00766-005-0020-7]

[2] RTCA. DO-178C, Software considerations in airborne systems and equipment certification. Washington: RTCA, 2011. 32–42.

[3] IEEE Software Engineering Standards Committee. IEEE Std 830-1998 IEEE recommended practice for software requirements specifications. New York: IEEE, 1998. [doi: 10.1109/IEEESTD.1998.88286]

[4] McClure J. CMMI for development v 1. 3 by Mary Beth Chrissis, Mike Konrad and Sandy Shrum. ACM SIGSOFT Software Engineering Notes, 2011, 36(4): 34–35. [doi: 10.1145/1988997.1989007]

[5] Ernst NA, Murphy GC. Case studies in just-in-time requirements analysis. In: Proc. of the 2nd IEEE Int'l Workshop on Empirical Requirements Engineering. Chicago: IEEE, 2012. 25–32. [doi: 10.1109/EmpiRE.2012.6347678]

[6] Rath M, Rendall J, Guo JLC, Cleland-Huang J, Mäder P. Traceability in the wild: Automatically augmenting incomplete trace links. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: ACM, 2018. 834–845. [doi: 10.1145/3180155.3180207]

[7] Mavin A, Wilkinson P, Harwood A, Novak M. Easy approach to requirements syntax (EARS). In: Proc. of the 17th IEEE Int'l Requirements Engineering Conf. Atlanta: IEEE, 2009. 317–322. [doi: 10.1109/RE.2009.9]

[8] Do AQ, Bhowmik T. Refinement and resolution of just-in-time requirements in open source software: A case study. In: Proc. of the 25th

IEEE Int'l Requirements Engineering Conf. Workshops. Lisbon: IEEE, 2017. 407–410. [doi: 10.1109/REW.2017.42]

[9] Salton G, Wong A, Yang CS. A vector space model for automatic indexing. Communications of the ACM, 1975, 18(11): 613–620. [doi: 10.1145/361219.361220]

[10] De Lucia A, Oliveto R, Tortora G. Assessing IR-based traceability recovery tools through controlled experiments. Empirical Software Engineering, 2009, 14(1): 57–92. [doi: 10.1007/s10664-008-9090-8]

[11] Wang BC, Peng R, Li YB, Lai H, Wang Z. Requirements traceability technologies and technology transfer decision support: A systematic review. Journal of Systems and Software, 2018, 146: 59–79. [doi: 10.1016/j.jss.2018.09.001]

[12] Hayes JH, Dekhtyar A, Osborne J. Improving requirements tracing via information retrieval. In: Proc. of the 11th IEEE Int'l Requirements Engineering Conf. Monterey Bay: IEEE, 2003. 138–147. [doi: 10.1109/icre.2003.1232745]

[13] Murtazina MS, Avdeenko TV. An ontology-based approach to support for requirements traceability in agile development. Procedia Computer Science, 2019, 150: 628–635. [doi: 10.1016/j.procs.2019.02.044]

[14] Guo J, Monaikul N, Plepel C, Cleland-Huang J. Towards an intelligent domain-specific traceability solution. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 755–766. [doi: 10.1145/2642937.2642970]

[15] Rocchio J. Relevance feedback in information retrieval. In: Salton G, ed. The SMART Retrieval System: Experiments in Automatic Document Processing. Englewood Cliffs: Prentice-Hall, 1971. 17.

[16] Panichella A, De Lucia A, Zaidman A. Adaptive user feedback for IR-based traceability recovery. In: Proc. of the 8th IEEE/ACM Int'l Symp. on Software and Systems Traceability. Florence: IEEE, 2015. 15–21. [doi: 10.1109/sst.2015.10]

[17] Wang WT, Gupta A, Niu N, Xu LD, Cheng JRC, Niu ZD. Automatically tracing dependability requirements via term-based relevance feedback. IEEE Trans. on Industrial Informatics, 2018, 14(1): 342–349. [doi: 10.1109/tii.2016.2637166]

[18] Wang WT, Dumont F, Niu N, Horton G. Detecting software security vulnerabilities via requirements dependency analysis. IEEE Trans. on Software Engineering, 2022, 48(5): 1665–1675. [doi: 10.1109/tse.2020.3030745]

[19] Hayes JH, Dekhtyar A, Sundaram SK. Advancing candidate link generation for requirements tracing: The study of methods. IEEE Trans. on Software Engineering, 2006, 32(1): 4–19. [doi: 10.1109/tse.2006.3]

[20] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: ACL, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[21] Lin JF, Poudel A, Yu WH, Zeng QK, Jiang M, Cleland-Huang J. Enhancing automated software traceability by transfer learning from open-world data. arXiv:2207.01084, 2022.

[22] Guo J, Cheng JH, Cleland-Huang J. Semantically enhanced software traceability using deep learning techniques. In: Proc. of the IEEE/ACM 39th Int'l Conf. on Software Engineering. Buenos Aires: IEEE, 2017. 3–14. [doi: 10.1109/ICSE.2017.9]

[23] Lin JF, Liu YL, Zeng QK, Jiang M, Cleland-Huang J. Traceability transformed: Generating more accurate links with pre-trained BERT models. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 324–335. [doi: 10.1109/icse43902.2021.00040]

[24] Li Y, Li J, Li MS. Research on dynamic requirement traceability method and traces precision. Ruan Jian Xue Bao/Journal of Software, 2009, 20(2): 177–192 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3365.htm [doi: 10.3724/SP.J.1001.2009.03365]

[25] Wang WT, Niu N, Liu H, Niu ZD. Enhancing automated requirements traceability by resolving polysemy. In: Proc. of the 26th IEEE Int'l Requirements Engineering Conf. Banff: IEEE, 2018. 40–51. [doi: 10.1109/re.2018.00-53]

[26] Guo J, Gibiec M, Cleland-Huang J. Tackling the term-mismatch problem in automated trace retrieval. Empirical Software Engineering, 2017, 22(3): 1103–1142. [doi: 10.1007/s10664-016-9479-8]

[27] Mahmoud A, Niu N. On the role of semantics in automated requirements tracing. Requirements Engineering, 2015, 20(3): 281–300. [doi: 10.1007/s00766-013-0199-y]

[28] Capobianco G, De Lucia A, Oliveto R, Panichella A, Panichella S. Improving IR-based traceability recovery via noun-based indexing of software artifacts. Journal of Software: Evolution and Process, 2013, 25(7): 743–762. [doi: 10.1002/smr.1564]

[29] Zou XC, Settimi R, Cleland-Huang J. Improving automated requirements trace retrieval: A study of term-based enhancement methods. Empirical Software Engineering, 2010, 15(2): 119–146. [doi: 10.1007/s10664-009-9114-z]

[30] Li ZH, Chen MR, Huang LG, Ng V. Recovering traceability links in requirements documents. In: Proc. of the 19th Conf. on Computational Natural Language Learning. Beijing: ACL, 2015. 237–246. [doi: 10.18653/v1/k15-1024]

[31] Cleland-Huang J, Czauderna A, Gibiec M, Emenecker J. A machine learning approach for tracing regulatory codes to product specific requirements. In Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering (ICSE). Cape Town: ACM, 2010. 155–164. [doi: 10.

1145/1806799.1806825]
[32] Gervasi V, Zowghi D. Supporting traceability through affinity Mining. In: Proc. of the 22nd IEEE Int'l Requirement Engineering Conf. Karlskrona: IEEE, 2014. 143–152. [doi: 10.1109/re.2014.6912256]
[33] Jirapanthong W. Requirements traceability on Web applications. In: Procs of the 7th Int'l Conf. on Information Technology and Electrical Engineering. Chiang Mai: IEEE, 2016. 18–23. [doi: 10.1109/iciteed.2015.7408905]
[34] Grechanik M, McKinley KS, Perry DE. Recovering and using use-case-diagram-to-source-code traceability links. In: Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM Sigsoft Symp. on the Foundations of Software Engineering. Dubrovnik: ACM, 2007. 95–104. [doi: 10.1145/1287624.1287640]
[35] Rath M, Rempe P, Mäder P. The IlmSeven dataset. In: Proc. of the 25th IEEE Int'l Requirements Engineering Conf. Lisbon: IEEE, 2017. 516–519. [doi: 10.1109/re.2017.18]
[36] Dalpiaz F, Gieske P, Sturm A. On deriving conceptual models from user requirements: An empirical study. Information and Software Technology, 2021, 131: 106484. [doi: 10.1016/j.infsof.2020.106484]
[37] Hazem A, Daille B. Word embedding approach for synonym extraction of multi-word terms. In: Proc. of the 11th Int'l Conf. on Language Resources and Evaluation (LREC). Miyazaki: European Language Resources Association, 2018. 297–303.
[38] Pengő E, Ságodi Z. A preparation guide for Java call graph comparison: Finding a match for your methods. Acta Cybernetica, 2019, 24(1): 131–155. [doi: 10.14232/actacyb.24.1.2019.10]
[39] De Lucia A, Oliveto R, Sgueglia P. Incremental approach and user feedbacks: A silver bullet for traceability recovery. In Proc. of the 22nd IEEE Int'l Conf. on Software Maintenance (ICSM). Philadelphia: IEEE, 2006. 299–309. [doi: 10.1109/icsm.2006.32]
[40] Ruthven I, Lalmas M. A survey on the use of relevance feedback for information access systems. The Knowledge Engineering Review, 2003, 18(2): 95–145. [doi: 10.1017/s0269888903000638]
[41] Lohar S, Amornborvornwong S, Zisman A, Cleland-Huang J. Improving trace accuracy through data-driven configuration and composition of tracing features. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg: ACM, 2013. 378–388. [doi: 10.1145/2491411.2491432]
[42] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Findings of the Association for Computational Linguistics. ACL, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]
[43] Lin JF, Liu YL, Cleland-Huang J. Information retrieval versus deep learning approaches for generating traceability links in bilingual projects. Empirical Software Engineering, 2022, 27(1): 5. [doi: 10.1007/s10664-021-10050-0]
[44] Dekhtyar A, Hayes JH. Automating requirements traceability: Two decades of learning from KDD. In: Proc. of the 1st Int'l Workshop on Learning from other Disciplines for Requirements Engineering. Banff: IEEE, 2018. 12–15. [doi: 10.1109/d4re.2018.00009]

附中文参考文献:

[24] 李引, 李娟, 李明树. 动态需求跟踪方法及跟踪精度问题研究. 软件学报, 2009, 20(2): 177–192. http://www.jos.org.cn/1000-9825/3365.htm [doi: 10.3724/SP.J.1001.2009.03365]

田家豪(1991－), 男, 博士生, 主要研究领域为需求工程, 软件体系结构.

连小利(1985－), 女, 博士, 助理研究员, CCF 专业会员, 主要研究领域为需求工程, 软件体系结构追踪, 自然语言处理.

张莉(1968－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为经验软件工程, 软件体系结构.

赵倩慧(2001－), 女, 博士生, 主要研究领域为智能化软件工程, 自然语言处理.