软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.006904]

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

基于多标签学习的代码评审意见质量评价*

杨岚心[1,2], 张 贺[1,2], 徐近伟[1,2], 张逸凡[1,2], 王梓宽[1,2], 周 鑫[1,2], 李京悦[3], 荣国平[1,2]

[1](南京大学 软件学院, 江苏 南京 210093)

[2](计算机软件新技术国家重点实验室 (南京大学), 江苏 南京 210093)

[3](Department of Computer Science, Norwegian University of Science and Technology, Trondheim 7030, Norway)

通信作者: 张贺, E-mail: hezhang@nju.edu.cn

摘 要: 代码评审是现代软件开发过程中被广泛应用的最佳实践之一, 其对于软件质量保证和工程能力提升都具有重要意义. 代码评审意见是代码评审最主要和最重要的产出之一, 其不仅是评审者对代码变更的质量感知, 而且是作者修复代码缺陷和提升质量的重要参考. 目前, 全球各大软件组织都相继制定了代码评审指南, 但仍缺少针对代码评审意见质量的有效的评价方式和方法. 为了实现可解释的、自动化的评价, 开展文献综述、案例分析等若干实证研究, 并在此基础上提出一种基于多标签学习的代码评审意见质量评价方法. 实验使用某大型软件企业的34个商业项目的共计17 000条评审意见作为数据集. 结果表明所提出的方法能够有效地评价代码评审意见质量属性和质量等级. 除此以外, 还提供若干建模经验, 如评审意见标注和校验等, 旨在帮助那些受代码评审困扰的软件组织更好地实施所提出的方法.

关键词: 软件质量保证; 代码评审; 代码评审意见; 质量评价; 多标签学习; 实证软件工程

中图法分类号: TP311

中文引用格式: 杨岚心, 张贺, 徐近伟, 张逸凡, 王梓宽, 周鑫, 李京悦, 荣国平. 基于多标签学习的代码评审意见质量评价. 软件学报. http://www.jos.org.cn/1000-9825/6904.htm

英文引用格式: Yang LX, Zhang H, Xu JW, Zhang YF, Wang ZK, Zhou X, Li JY, Rong GP. Multi-label Learning for Evaluating Quality of Code Review Comments. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/6904.htm

Multi-label Learning for Evaluating Quality of Code Review Comments

YANG Lan-Xin[1,2], ZHANG He[1,2], XU Jin-Wei[1,2], ZHANG Yi-Fan[1,2], WANG Zi-Kuan[1,2], ZHOU Xin[1,2], LI Jing-Yue[3], RONG Guo-Ping[1,2]

[1](Software Institute, Nanjing University, Nanjing 210093, China)

[2](State Key Laboratory of Novel Software Technology at Nanjing University, Nanjing 210093, China)

[3](Department of Computer Science, Norwegian University of Science and Technology, Trondheim 7030, Norway)

Abstract: Code review is one of the best practices widely used in modern software development, which is crucial for ensuring software quality and strengthening engineering capability. Code review comments (CRCs) are one of the main and most important outputs of code reviews. CRCs are not only the reviewers' perceptions of code quality but also the references for authors to fix code defects and improve quality. Nowadays, although a number of software organizations have developed guidelines for performing code reviews, there are still few effective methods for evaluating the quality of CRCs. To provide an explainable and automated quality evaluation of CRCs, this study conducts a series of empirical studies such as literature reviews and case analyses. Based on the results of the empirical studies, the study proposes a multi-label learning-based approach for evaluating the quality of CRCs. Experiments are carried out by using a large software enterprise-specific dataset that includes a total of 17 000 CRCs from 34 commercial projects. The results indicate that the proposed

* 基金项目: 国家自然科学基金 (62072227, 62202219); 国家重点研发计划 (2019YFE0105500); 江苏省重点研发计划 (BE2021002-2); 南京大学计算机软件新技术国家重点实验室创新项目 (ZZKT2022A25); 海外开放课题 (KFKT2022A09)
收稿时间: 2022-06-15; 修改时间: 2022-09-19; 采用时间: 2023-01-19; jos 在线出版时间: 2023-08-16

approach can effectively evaluate the quality attributes and grades of CRCs. The study also provides some modeling experiences such as CRC labeling and verification, so as to help software organizations struggling with code reviews better implement the proposed approach.

Key words: software quality assurance; code review; code review comment (CRC); quality evaluation; multi-label learning; empirical software engineering

代码评审 (code reviews) 起源于 20 世纪 70 年代 Fagan 在美国 IBM 公司提出的“代码审查 (code inspections)”[1], 其旨在通过人工检查源代码来减少软件开发错误和缺陷. 在随后的数年中, 代码评审作为一种典型的质量保证措施被世界范围内各大软件组织逐渐推广和使用, 成为现代软件开发的最基本的流程之一[2,3]. 除了保证代码质量以外, 代码评审还有利于提升开发者编程能力、协作能力和质量意识[4,5], 帮助减少静态扫描工具的漏报和误报[6,7]等. 由于这些特性, 代码评审目前已被应用于各类开源软件项目和商业软件项目[8−10], 成为被广泛认可的软件工程“最佳实践”之一[11].

另一方面, 代码评审的负作用日益凸显. 代码评审有时候不能发现缺陷, 变成软件工程中一种典型的浪费[12]. 更糟糕的是, 低效的代码评审容易延缓项目进度[13]. 不友好和不公平的代码评审容易引发团队负面情绪和信任危机, 极端情况下甚至导致员工离职, 进而对团队和项目都产生负面影响[14−18]. 相较于其他质量保证活动 (如测试等), 代码评审缺少系统化理论指导和智能工具支持. 相反地, 代码评审长期以来, 通常由经验性的“最佳实践”指导[19], 即代码评审更多依赖代码作者和评审者的技能、经验、参与和投入程度等.

近年来, 越来越多的研究开始评价和提升代码评审[4,10,18,20]. 其中的大多数都提及“代码评审意见”(后文简称“评审意见”), 并将其视为代码评审的最主要和最重要的产出之一. 评审意见, 即评审者针对开发者提交的代码变更给予的反馈, 是其他评审者和开发者发现代码缺陷的重要提示[21], 同时也是开发者修复缺陷、提升质量的重要参考. 尽管代码评审意见的重要意义获得了学术界和工业界的普通认可, 但目前仍缺少针对其质量的有效的评价方式和方法.

极少数的相关工作[10,20,22]将是否“触发代码变更”作为评价评审意见有用性的标准. 然而, 该方式属于“离线评价”, 即必须依赖于评审后的代码变更状态. 本文认为仅将是否“触发代码变更”作为评价标准, 一方面受人的主观因素影响较大, 难以被评审者广泛接受; 另一方面, 评价并非最终目的, 更重要的是规范评审意见内容, 辅助提升评审质量和代码质量. 因此, 评价需要具备可解释性和指导性. 例如, (1) 如果因为在评审意见中“包含代码元素”可以帮助开发者定位和修复缺陷, 而将其作为评价标准, 那么评审者可能会有意识地复制粘贴代码元素到评审意见中, 而不论其是否必要; (2) 如果因为在评审意见中指出严重程度高的缺陷 (“指出严重缺陷”) 可以降低后期修复成本, 而将其作为评价标准, 那么评审者可能会遗漏其他缺陷; (3) 如果因为在评审意见中“表示赞赏”可以提升代码作者的积极性, 而将其作为评价标准, 那么无疑将背离评审的初衷 (代码质量保证) 和加重评审者的负担. 另一方面, 现有研究[10,20,22]大都基于评审意见文本、评审上下文和评审者相关特征来构建预测模型. 此时这些特征 (样本特征) 也可被视为“评价标准”. 即使模型能够实现相当准确的预测效果, 其在应用时也会遭受质疑. 例如, 很难说服评审者其意见质量低是因为意见中未触发代码变更或未包含代码元素等.

针对上述问题, 本文开展了文献综述、案例分析等若干实证研究来识别和总结代码评审意见的质量评价标准 (属性), 并在此基础上提出了一种基于多标签学习 (multi-label learning)[23]的评价方法. 该方法基于多标签学习范式和预训练语言模型的文本分类模型 (自动化模型), 以及启发式的映射规则 (人工经验) 建立起了评审意见质量属性 (“情绪”“疑问”“评价”和“建议”) 和质量等级 (“优秀”“良好”“一般”和“较差”) 的关联, 属于混合式方法. 与已有的评审意见质量评价方法相比, 本文提出的方法仅需关注评审意见文本, 无需依赖代码变更状态, 能够实现“实时评价”. 此外, 本文提出的方法属于“多元& 多级评价”. 其中, “多元”指考虑了多个质量属性, “多级”指设置了多个质量等级, 从而有利于提升评价方法的可解释性和指导性, 降低实际应用该方法时面临争议的可能性. 在某全球领先的软件企业的 34 个商业项目, 共计 17 000 条代码评审意见上的实验验证表明, 本文提出的方法在多个量化评价指标上表现出色, 能够有效地评价代码评审意见的质量.

本文的主要贡献包括:

(1) 区别于现有相关工作根据是否“触发代码变更”来评价代码评审意见有用性 (有/无), 本文使用了多个质量属性来综合评价其质量等级 (优/良/中/差), 旨在缓解实施评价时可能面临的争议.

(2) 提出了一种基于多标签学习的代码评审意见质量评价方法. 该方法提供了可解释的、自动化的质量评价, 旨在帮助规范和提升代码评审质量.

(3) 提供了若干建模经验, 如针对代码评审意见的标注和校验等, 旨在帮助那些受代码评审困扰的软件组织更好地实施本文提出的方法.

本文第 1 节介绍背景和相关工作. 第 2 节介绍研究目标和科学挑战. 第 3 节介绍本文提出的基于多标签学习的代码评审意见质量评价方法. 第 4 节报告实验设计与结果分析. 第 5 节讨论与代码评审意见标签相关的经验. 第 6 节讨论实验的效度威胁. 最后, 第 7 节总结全文并提出未来工作方向.

1 背景和相关工作

1.1 代码评审

代码评审指通过阅读源代码以检查其是否实现预定义功能和满足质量要求的软件工程实践[1]. 与其他软件质量保证活动相比, 代码评审长期以来缺少系统化理论指导和智能工具支持, 其质量更多依赖于代码作者和评审者的技能、经验、参与和投入程度等[24,25]. 因此, 代码评审一方面发挥着质量保证、知识分享等作用; 另一方面又存在着效率低、质量差等挑战[18,19]. 近年来, 越来越多的软件组织开始重视代码评审, 并逐步建立和完善正式的代码评审指南. 作为 Google 工程实践文档的重要组成部分之一, 其代码评审指南 (https://google.github.io/eng-practices/review/) 由 (1) 代码评审者指南和 (2) 代码作者指南两部分组成. 其中, 前者包括评审标准/规范、关注重点、如何书写评审意见和处理争议等; 后者包括如何准备评审和如何处理评审意见等. 代码托管平台 GitLab 不仅提供了方便的代码评审支持工具, 而且提供了详尽的评审指南 (https://docs.gitlab.com/ee/development/code_review.html), 包括评审流程、检查表、不同角色 (代码作者、评审者和项目管理者) 职责和最佳实践等. 此外, 相当多的研究工作[26−28]旨在通过推荐评审者来提升代码评审质量.

1.2 代码评审意见挖掘

代码评审意见挖掘作为代码评审分析、评价和改进的重要证据支持, 近年来受到了来自研究社区和业界越来越多的关注. Li 等人[29]发现开源软件社区中的代码评审意见按照其作用可以归纳为 4 大类, 分别是“更正”“决策”“管理”和“沟通”. 其中, 每一大类包含 2–3 个小类, 如“决策”又可细分为“接受”“拒绝”和“疑问”. 在 Li 等人的模型中, 开源软件社区中的代码评审意见共被细分为 11 小类. Bosu 等人[10]发现评审意见关注广泛的内容, 包括“文档”“组织”“逻辑”“资源”“缺陷”和“验证”等 13 类. Hasan 等人[20]总结了评审意见的 18 类主要关注点. 评审意见在关注缺陷的同时也经常传递出评审者的情绪. Ortu 等人[30]发现 Jira 社区中, 在评审意见中传递负面情绪的评审者通常需要花费更长的时间来修复缺陷. El Asri 等人[15]指出开源软件社区中: (1) 评审者经常表达积极的或消极的情绪; (2) 各类别情绪的评审意见的比例在核心评审者和边缘评审者之间存在明显差异; (3) 边缘评审者在成长为核心评审者的过程中, 其评审意见的情绪通常是中性的; (4) 表达消极情绪的评审者通常会花费更长的时间完成评审; (5) 表达消极情绪的评审者通常会长时间地持续参与到项目维护当中.

关于代码评审意见质量评价, 仅有的少数研究均将其形式化为一个判断“有用性”的二分类问题. Bosu 等人[10]调研了 Microsoft 开发者关于评审意见有用性的看法, 结果表明: (1) “有用”的评审意见通常指出功能性缺陷、提供与验证相关的指导、(对新人而言) 编码规范和可利用的工具等; (2) “部分有用”, 即在受访者中未达成一致认可的评审意见通常指出注释、风格、命名等问题, 或是建议代码作者寻求替代实现方案; (3) “无用”的评审意见通常未指出任何缺陷, 相反地, 而是提出疑问、表示赞赏、要求在未来做出改进等. 然而, Bosu 等人并未基于上述经验构建预测模型. Bosu 等人注意到“有用”的评审意见通常会触发其相邻代码片段的针对性修改 (该类型评审意见被称为“change trigger”). 因此, 在 Bosu 等人构建的决策树模型中, “change trigger”被设置为评价意见有用性的核心标准, 即被设置为决策树的第一层条件判断. 其他层的条件判断 (标准) 还包括“评审意见数量”“评审次数”和“情感倾向”等. Bosu 等人的评价方法属于“离线评价”, 即必须依赖代码经过评审后的变更状态. Rahman 等人[22]同样将“change trigger”作为评价评审意见有用性的核心标准. 为了实现“实时评价”, Rahman 等人构建了若干预测模型. 这

些模型不依赖代码经过评审后的变更状态, 而是基于评审意见的文本特征 (如“可读性”“停用词比例”和“代码元素比例”等) 和评审者相关特征 (如“评审次数”等). 基于上述研究经验, Hasan 等人[20]调研了 Samsung 开发者关于代码评审意见是否有用的看法, 结果表明: (1) “有用”的评审意见通常指出缺陷、逻辑错误、冗余代码、帮助提升设计和可读性等; (2) “部分有用”的评审意见通常提出疑问、表示赞赏、要求改进文档等; (3) “无用”的评审意见通常指出能够被静态分析工具发现的问题、讨论已经被解决的问题、误解代码意图等. 同样地, 为了实现“实时评价”, Hasan 等人基于代码评审意见的文本特征、评审上下文特性和评审者相关特征构建了若干预测模型.

总结来说, 目前尚无被一致认可的代码评审意见质量评价标准. 研究者在构建预测模型时, 通常将是否“触发代码变更”作为评价评审意见有用性的标准. 该评价方式依赖代码作者对评审意见的接受与否, 受人的主观因素影响较大, 因此其有效性仍存在争议. 本文针对上述问题开展了若干实证研究, 在此基础上提出了一种基于多标签学习的代码评审意见质量评价方法, 现将关键研究方法、过程和结果报告如下.

2 准备工作

2.1 实例分析

Git 版本控制系统提供了方便的代码变更比对功能 (通常以不同颜色作为区分), 评审者可以更加专注代码变更, 而无需逐行检视提交的所有代码. 评审者一旦发现问题, 即可选中任意行代码并提出针对性的意见. 图 1 展示了一个典型的代码评审实例 (https://gitee.com/mindspore/mindspore/pulls/30478). 评审者“zhangxx”针对代码作者“徐 xx”提交的代码合入请求中的新增加 (+) 的行 (366) 提出了评审和修改意见“这里建议直接用 auto iter = counter_handlers_.find(name); if (iter == end || !(iter->first_count_handler)){}减少查询次数, iter 可以重复使用”, 代码作者接受了该意见并回复“应该这样做, 这几天 pclint 清理时候带上”. 事实上, 并非每条评审意见都会被代码作者回复和触发代码变更. 表 1 中展示了更多、更复杂的代码评审意见实例.

图 1 代码评审实例

表 1 代码评审意见实例

ID	代码评审意见
C1	同上
C2	逻辑不对
C3	魔鬼数字
C4	info日志打印太多
C5	-1具体含义是什么?
C6	冗余代码, 抽成一个函数
C7	什么鬼啊!!!英文是不是有些太不通顺了
C8	是否有emui标准控件? 有的话用标准控件
C9	MAXBUFLENGTH改为MAX_BUF_LENGTH
C10	这样写不对, 假设第1个关闭时抛异常了, 会导致2, 3不去关闭. 每个需要独立的try-catch; 其中, 文件的输入输出流可以直接使用IoUtils里的方法

2.2 研究目标

针对代码评审意见质量评价任务, 本文旨在实现以下两个研究目标.

● 目标 1: 可解释性评价

本文首先期望提出一个可解释的、具有指导性的代码评审意见质量评价方法. 具体而言, 提出的方法不应该仅局限于给出评价结果 (质量等级), 更重要的是, 告诉评审者什么样的意见是低质量的和什么样的意见是高质量的; 否则, 不透明的质量评价可能会引发信任危机等负面作用. 本文期望提出的方法能起到规范和引导代码评审意见书写的作用.

● 目标 2: 自动化评价

代码评审意见在表现形式和内容上具有广泛的多样性和复杂性. 人工评价一方面效率普遍非常低, 另一方面容易出现评价结果不一致的情况. 不同的人对同一条意见的评价结果可能不一致, 同一人在不同时候对同一条意见的评价结果也可能不一致. 此外, 人工评价还会出现误解和偏见等现象, 容易引发信任危机等负面作用. 因此本文的另一关键研究目标是自动化评价过程.

2.3 科学挑战

实现两个研究目标需要重点解决以下两个科学挑战.

● 挑战 1: 质量评价标准

本文通过分析大量的代码评审意见实例发现, 相当一部分的评审意见, 尤其是那些提出可选方案、提出未来改进建议或者提出疑问的评审意见通常不会立即触发代码变更. 如果将触发代码变更作为评审意见质量唯一的评价标准, 势必会引起广泛的争议. 许多受访的软件工程师表示: “几乎不存在完全没有用的评审意见. 即使是意见仅为一个问号, 至少说明该处代码可读性差, 不利于项目维护; 即使是一个“OK”, 至少说明该处代码变更得到了确认”. 另一方面, 设想将评审意见长度、是否包含代码元素、是否包含关键字词和评审者信息等作为表示原始评审意见的特征, 此时这些特征也可被视为“评价标准”. 即使自动化方法能够实现相当准确的预测效果, 其也很难被广泛接受. 例如, 很难说服评审者其意见质量低是因为意见中未包含代码元素或关键术语. 否则, 评审意见将充斥大量的“无用”信息, 即评审者有意识地在意见中加入代码元素、关键字词等, 而不论其是否必要.

● 挑战 2: 质量评价模型

Efstathiou 等人[31]指出代码评审意见具有丰富的语言学特征, 例如传递“因果”“转折”“示例”和“假设”等. 表 1 中展示的实例更形象地体现了评审意见的多样性和复杂性. 典型地, (1) 词法层面: 自然语言和程序语言/代码元素混用, 如 C9、C10; 软件工程行业术语, 如 C6、C8. (2) 语法层面: 陈述, 如 C3; 疑问, 如 C5; 祈使, 如 C9. (3) 语义/语用层面: 评价, 如 C2; 建议, 如 C6; 解释, 如 C10. 另外, 评审意见还经常包含指代, 如 C1; 和传递情绪, 如 C7. 综合起来, 这些因素给代码评审意见质量评价任务建模带来了巨大挑战: 如何屏蔽评审意见在语言表达形式上的多样性和复杂性, 直接面向更具有规范和指导作用的目标, 构建可解释的、自动化的评价 (预测) 模型?

3 代码评审意见质量评价方法

支持本文提出的代码评审意见质量评价方法的关键研究方法和研究过程如图 2 所示.

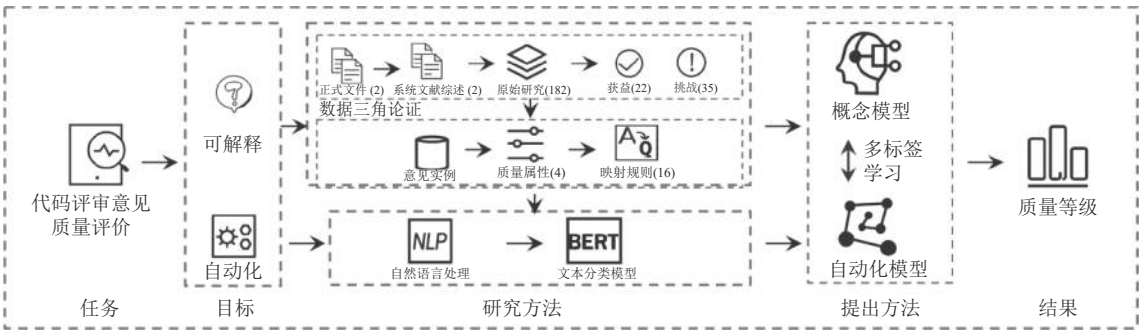


图 2 本文的研究方法和研究过程

3.1 方法形式化

本文旨在提出一个可解释的、自动化的评价方法G. 该方法属于混合方法, 形式化表达如公式 (1) 所示:

$$G = F \circ Z \tag{1}$$

其中, F表示目标函数, 如公式 (2) 所示, 旨在使用自动化模型从评审意见X中学习若干个质量属性Y.

$$F = \Phi(X, Y) \tag{2}$$

Z表示映射, 如公式 (3) 所示, 旨在基于启发式规则/人工经验建立起质量属性与质量等级Q之间的关联.

$$Z: F \to Q \tag{3}$$

3.2 方法具体化: 质量属性和映射

质量属性和映射关系是本文提出的方法实现“可解释的和具有指导性的评价”的目标的支持要素. 质量属性也是构建和优化模型以实现“自动化评价”的目标的指导依据. 关于质量属性和映射关系的详细介绍如下.

3.2.1 质量属性

为了识别和总结用于评价代码评审意见质量的属性 (即评价标准), 本文使用了社会科学研究领域中经典的“三角论证 (triangulation)”方法[32]. 三角论证又被称作“三角互证”, 是质性研究领域中一种典型的研究方法, 其旨在使用多种方法、数据源、理论交互证实, 从而保证研究发现的有效性和可信度. 三角论证主要有 3 种实施形式. 其中, 方法三角论证 (method triangulation) 强调使用多种方法 (如访谈、问卷调查等) 来收集与研究问题相关的证据; 数据三角论证 (data source triangulation, source 一词常被省略) 强调从多种数据源收集证据; 理论三角论证 (theory triangulation) 强调使用多种理论来分析和解释数据. 本文使用的是“数据三角论证”, 3 种数据源分别是: (1) 正式文件; (2) 学术文献; (3) 实例分析结果.

● 数据源 1: 正式文件

本文首先回顾了两份重要的文件. 其中一份[1]是 Fagan 在 IBM 公司首次正式提出“代码审查”概念的文章; 另一份[33]是 IEEE 计算机协会发布的关于“软件评审和审计标准”的文件. 总结发现, 尽管软件评审存在若干种变体, 如“technical reviews”“inspections”和“walk-throughs”等, 但都需要评审代码, 并且评审的核心目标是发现和识别编码缺陷.

● 数据源 2: 学术文献

尽管本文尝试在 IEEE Xplore Digital Library、ACM Digital Library 和 Google Scholar 等知名的电子数据库检索研究代码评审意见的文献, 使用的检索字符串为: ((“code review” OR “code inspection”) AND (“comment” OR “feedback”)). 但返回的与代码评审意见直接相关的文献数量极少. 于是本文使用快速文献综述 (rapid reviews)[34]方法寻找研究代码评审的文献. 具体地, 本文选择了两篇最新的、研究主题为代码评审的高质量系统文献综述 (systematic literature reviews)[35,36]. 两篇文献综述均于 2021 年发表在国际知名的软件工程期刊《Journal of Systems and Software》上, 分别包含了 139、112 篇原始研究 (primary studies). 经过汇总和去重后, 还剩下 182 篇.

本文的两位作者独立地阅读每一篇原始研究的至少摘要和引言部分, 从中识别代码评审的获益和挑战. 另一位作者在最后的集体讨论中加入进来解决前两位作者的分歧. 本文期望通过评价和提升代码评审意见质量, 在最大化代码评审的获益的同时, 在一定程度上克服代码评审的挑战. 经过数据抽取和合成, 本文识别出了代码评审的 22 项获益和 35 项挑战. 主要的获益包括: (1) 发现与修复缺陷; (2) 保证与提高软件 (代码) 质量; (3) 传播知识; (4) 提升团队协作意识; (5) 探索替代方法. 主要的挑战包括: (1) 推荐评审者; (2) 时间/精力开销; (3) 理解代码意图和实现; (4) 公平/争执; (5) 未能发现缺陷; (6) 增加工作负担; (7) 缺少工具支持.

本文首先移除了与代码评审意见文本无关的数据项, 如推荐评审者、时间/精力开销、增加工作负担和缺少工具支持等. 然后将提升团队协作意识、公平/争执等合成为“情绪”; 将发现与修复缺陷、保证与提高软件 (代码) 质量、传播知识、探索替代方法和未能发现缺陷等合成为“质量保证”. 此外, 仍有一些数据项未在该阶段的集体讨论中达成一致, 如理解代码意图和实现. 本文期望提出的质量属性不仅是可解释的, 而且是完备的, 即能够覆盖尽可能多的情形. 为此, 本文设计了“数据三角论证”的第 3 部分 (实例分析结果), 即通过分析评审意见实例来进一

步补充和验证质量属性.

● 数据源 3: 实例分析结果

通过分析大量的代码评审意见实例, 本文发现评审者在意见中经常表达“疑问”, 这些疑问有的是 (1) 期望理解代码意图和实现, 如“这里为什么要抛异常?” 有的是 (2) 对指出缺陷的不确定, 如“是否会有越界风险?”. 表达“疑问”的评审意见尽管不能都触发代码变更, 但是可以起到“提醒”(对代码作者) 和“知识传播”(对代码评审者) 的作用, 防止代码评审发生“漏判”和“误判”, 否则将导致代码质量问题和代码作者的负面情绪. 因此, 本文将“疑问”也作为评价评审意见质量的属性之一. 此外, 本文发现评审意见在发挥“质量保证”作用时有两种典型的表达方式: (1) 指出代码缺陷, 如“cleartext 不符合小驼峰”; (2) 提供修改建议, 如“建议 long 后面数字使用大写后缀 L”. 于是, 本文将通过文献综述合成的“质量保证”拆分为“评价”和“建议”两个质量属性, 分别对应上述两种表达方式. 表 2 展示了本文合成的最主要和重要的代码评审意见质量属性、合理性说明和标注时使用的符号. 需要注意的是, 本文在标注“情绪”时仅考虑“消极”和“非消极”. 关于质量属性的标注经验详见第 5.1 节.

表 2 代码评审意见质量属性

属性	合理性说明	标注
情绪	代码评审 (意见)应该就事论事, 不传递负面情绪	消极 (0); 非消极 (1)
疑问	代码评审如有疑问应该及时提出, 不让疑问变成风险和危害; 另一方面, 代码评审 (意见)应该促进团队知识共享, 而非单方(代码作者)受益	未提问 (0); 提问 (1)
评价	代码评审 (意见)的最基本目标是发现和识别代码缺陷	未评价 (0); 评价 (1)
建议	代码评审 (意见)应该帮助修复代码缺陷, 提升质量, 提升开发者质量意识	未建议 (0); 建议 (1)

3.2.2 映射关系

单一质量属性仅从某一方面“描述”了评审意见, 联合若干质量属性才能共同决定质量等级. 本文方法中设置了 4 个质量属性和 4 个质量等级. 前者分别是“情绪”“疑问”“评价”和“建议”; 后者分别是“优秀”“良好”“一般”和“较差”. 如第 3.1 节介绍的, 本文提出的方法需要建立起质量属性和质量等级之间的映射关系. 表 3 展示了一个映射表实例. 该映射表经本文的 3 位作者经调研软件从业者的意见后形成初始版本, 然后经全体作者共同讨论后形成中间版本, 最后经提供本文的实验数据集的软件企业的代码评审专家确认后形成最后版本. 本文提出的方法具有相当的灵活性, 其质量属性和映射关系都可以根据各软件组织的具体需求做裁剪或扩充.

表 3 映射表实例

属性	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
情绪	0	0	1	1	0	0	0	0	0	1	1	1	1	0	1	1
疑问	0	1	0	1	0	1	0	1	1	0	1	0	1	0	0	1
评价	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1
建议	0	0	0	0	0	0	1	1	1	0	0	1	1	1	1	1
等级	较差	较差	一般	一般	一般	一般	一般	一般	一般	良好	良好	良好	优秀	优秀	优秀	优秀

3.3 方法自动化: 基于多标签学习范式和预训练语言模型的文本分类

第 3.1 节和第 3.2 节介绍了本文提出的方法的基本原理 (概念模型), 实际使用时还需要基于算法和模型以实现自动化的质量评价 (自动化模型). 后者由基于多标签学习范式和预训练语言模型的文本分类模型来实现.

3.3.1 基础知识

多标签学习 (multi-label learning)[23]又被称为多标签分类 (multi-label classification), 是一种有监督机器学习范式. 在传统的单标签学习/分类问题中, 每个样本仅有单个标签, 即只能表示一个分类任务; 相反, 在多标签学习中, 每个样本具有互不排斥甚至存在依赖关系的多个标签, 即可以同时表示多个分类任务. 本文基于多标签学习构建评价模型的原因是: (1) 模型简化: 本文提出的方法中设置了 4 个质量属性, 假如使用单标签学习, 需要构建 4 个模型, 分别用于学习 4 个质量属性. 在该情形下, 使用同构模型还是异构模型, 如何对每个模型进行针对性的模型设

置都将成为挑战. 而如果使用多标签学习, 仅需构建一个统一的模型, 从而减少模型设计负担. (2) 模型优化: 近年来自然语言处理和深度学习的理论研究[37]已经证实了深度神经网络在对不同自然语言文本进行嵌入表示 (embedding) 时, 低维向量表示 (权重参数) 的高度相似性. 此外, 自然语言处理在引入深度学习和多任务学习[38]后, 不同任务可以共享词法分析和句法分析模块 (由深度神经网络中的若干组件充当), 模型结构和训练策略进一步简化. 代码评审意见属于一类特殊的自然语言 (可能包含代码元素), 即使本文提出的方法中设置的 4 个质量属性的预测属于不同的分类任务, 它们仍共享对评审意见文本的低维向量表示和中间处理模型. 这种在深度神经网络模型的深层针对不同分类任务作针对性设置, 但在模型浅层共享低维向量表示和中间处理模型, 将有助于增强模型的学习能力, 即模型在具有更强的分类能力的同时, 还具有更强的泛化能力.

自动化模型的最基础部分是对评审意见文本的嵌入表示. 传统的文本嵌入技术, 如词袋模型 (bag of words, BOW)、独热编码 (one-hot encoding) 和词频-逆文件频率 (term frequency-inverse document frequency, TF-IDF) 等都存在维度灾难、无法处理陌生词等问题. 近年, 一种全新的被称为 BERT[37]的文本嵌入技术在多个自然语言任务上取得了突破性进展. BERT 全称为“bidirectional encoder representation from Transformers”, 是由 Google 于 2018 年发布的一个基于 Transformers[39]的预训练语言模型. 预训练模型指在原始任务上预先训练一个初始模型, 然后在目标任务上微调初始模型, 从而提高模型解决目标任务的能力. 预训练语言模型则指的是针对自然语言处理任务而预先训练的初始模型. 传统的预训练语言模型仅能实现对语言 (文本) 的单方向建模 (从左到右或者从右到左), 即只能获取单方向的上下文信息. BERT 使用深层的双向 Transformer 作为模型组件, 并使用“掩码语言模型 (masked language model, MLM)”在超大规模语料库上进行预训练, 从而生成融合了语言 (文本) 左右方向 (上下文信息) 的语言模型. BERT 具有非常强的通用表示能力. 下游任务如文本分类、信息检索等仅需对 BERT 做微调即可取得很好的效果.

3.3.2 模型结构

本文针对代码评审意见质量评价提出的方法的整体架构如后文图 3 所示. 在 3 层混合架构中, 最底层是“表示层”. 该层设置了预训练语言模型 BERT, 用于对评审意见的嵌入表示. 中间层是“属性层”, 设置了 4 个中间输出节点 (合理性说明详见第 5.2 节), 用于表示对评审意见的 4 个质量属性的学习任务, 该层由神经网络中常见的全连接层 (1 层) 充当. 前两层共同实现了“多标签学习”范式. 前两层都属于“学习层”, 即需要通过有监督训练策略来调整各内部层的权重参数. 最后一层是“应用层”, 即通过查找预先定义的映射表 (质量属性到质量等级) 来输出评审意见的质量等级. 通过设置这样的 3 层架构, 本节实现了对第 3.1 节中提出的形式化的方法的具体化; 另一方面, 保持了高度的实现灵活性, 架构中的各层组件随着未来技术的发展均可改造升级.

3.3.3 训练策略

自动化模型的训练使用了两种策略, 现介绍如下.

- 自监督预训练 (self-supervised pretraining)

使用类似 BERT 这样的“大模型” (指模型的网络结构包含数十层, 权重参数达到数百万, 甚至上亿) 需要做预训练. 常见的预训练过程一般分为两步: 第 1 步是在大规模的数据集上训练初始模型, 经此训练后的模型的权重参数不再是随机值, 即学习到“知识”, 此时的模型被称作“预训练模型”; 第 2 步是根据领域和任务, 使用领域和任务相关的数据集对预训练模型的权重参数进行微调 (fine-tuning). 预训练机制可以减少新模型 (基于预训练模型构建的模型) 的训练时间和资源开销, 同时可以有效地提高新模型性能[40]. 尽管 Google 在发布 BERT 的同时也发布了其预训练模型权重参数 (简称“预训练模型”), 但是其预训练模型使用的数据源是通用类型语料, 如维基百科等. 此时的 BERT 还不具备领域和目标“知识”, 因此还需要在领域和目标任务类型语料, 如本文中, 在某一语言 (如中文或英文) 的代码评审意见数据集上做自监督预训练. 自监督预训练是一种不依赖标签的预训练方式, 这主要通过 BERT 的“掩码语言模型”机制来实现. 本文设计的基于 BERT 的自动化模型在训练时首先使用无标签的代码评审意见样本做了自监督预训练.

- 有监督微调 (supervised fine-tuning)

自动化模型在完成自监督预训练后即需要联合样本和标签做有监督训练, 即利用标签信息通过反向传播完成

对模型权重参数做进一步的微调. 在有监督微调训练阶段, 自动化模型使用了复合的损失函数, 如公式 (4) 所示. 其中, 左边部分是“非对称损失 (asymmetric loss, *ASL*)”[41], 右边部分是“虚拟对抗训练损失 (virtual adversarial training loss, *VAL*)”[42].

$$Loss = ASL + VAL \tag{4}$$

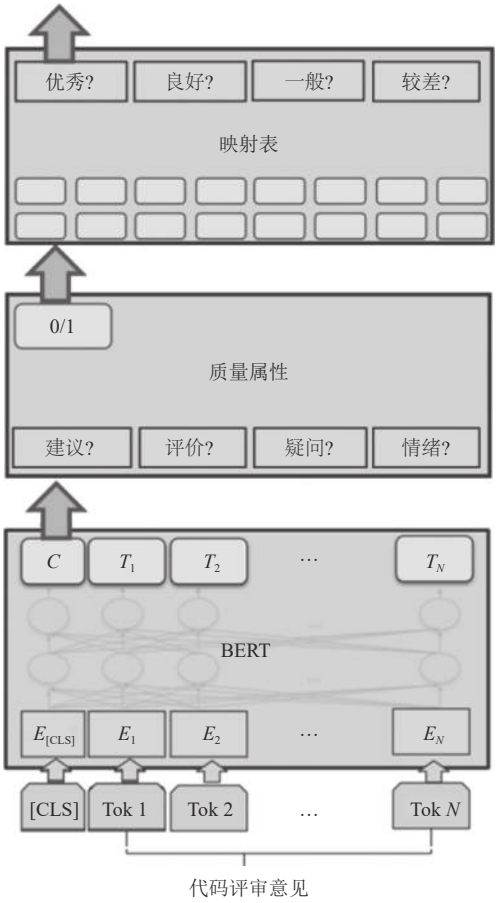


图 3　本文方法的整体架构

具体地, 非对称损失 *ASL* 的定义如公式 (5) 所示:

$$ASL = \begin{cases} L_+ = (1-p)^{\gamma_+}\log(p) \\ L_- = (p_m)^{\gamma_-}\log(1-p_m) \end{cases} \tag{5}$$

其中, L_+ 和 L_- 分别表示预测正、负样本的交叉熵损失. γ 在原始论文中被称为“focusing parameter”, 起平滑作用, γ_+ 和 γ_- 分别表示对预测正、负样本的损失的平滑. $(1-p)^{\gamma_+}$ 和 $(p_m)^{\gamma_-}$ 在原始论文中被称为“modulating factor”, 其作用是减少高置信度样本的损失的权重, 从而使模型在训练时更专注于容易预测出错的样本. p_m 在原始论文中被称为“shifted probability”, 其定义如公式 (6) 所示:

$$p_m = \max(p-m, 0) \tag{6}$$

p 表示模型对目标任务 (如预测输入样本属于某一类) 的预测概率; m 是一个超参数, 用于进一步调整负样本的损失的权重. 以下对设置“非对称损失”的合理性做简要说明.

非对称损失原本针对多标签图像识别任务中正负样本不平衡问题而设计. 本文在分析评审意见实例时发现, 4 个质量属性都存在不同程度的类别不平衡问题. 如仅有相当少比例的意见会传递消极情绪, 提出疑问的评审意

见占总数的比例也较少. 从质量等级视角看, “较差”的评审意见占总数的比例尤其少. 非对称损失旨在自适应地调整多标签分类任务中的多数类和少数类样本在训练时的重要程度来缓解类别不平衡和标签错误问题.

虚拟对抗训练损失 VAL 的定义如公式 (7) 所示:

$$VAL = D_{\mathrm{KL}}(p(\cdot|x;\hat{\theta})||p(\cdot|x+r_{\mathrm{vadv}}(x,\varepsilon);\theta)) = -\sum_{k=1}^{K} p(\cdot|x;\hat{\theta})\log p(k|x+r_{\mathrm{vadv}}(x,\varepsilon);\theta) \tag{7}$$

其中, $D_{\mathrm{KL}}(\cdot)$表示 KL 散度 (Kullback-Leibler divergence) 函数, 用于描述两个概率分布间的差异, K 表示扰动次数, x 表示输入样本, $\hat{\theta}$ 和θ 分别表示经上一次训练迭代已经获得的模型参数和本次训练迭代期望获得的模型参数, r_{vadv} 在原论文中被称为“adversarial perturbation”, 起对输入样本的扰动作用, 其定义如公式 (8) 所示:

$$r_{\mathrm{vadv}}(x,\varepsilon) = \underset{r;||r||_2\leqslant\varepsilon}{\arg\max} D_{\mathrm{KL}}(p(\cdot|x;\hat{\theta})||p(\cdot|x+r;\hat{\theta})) \tag{8}$$

ε 是一个超参数, 用于限制扰动半径. 以下对设置“虚拟对抗训练损失”的合理性做简要说明.

代码评审意见属于混合语言文本, 即在自然语言文本中可能存在若干程序语言文本/代码元素. 这些代码元素具有普遍性、复杂性和不确定性. 普遍性是指代码元素在评审意见中经常出现. 复杂性是指其出现时的表现形式是复杂的, 如程序语言类别 (Java、JavaScript、SQL 等) 和结构特征 (变量名、表达式、语句等), 对于非领域专家而言, 这些代码元素几乎是不可读的. 不确定性是指其是否出现在某一评审意见中, 出现时的位置和表现形式是不确定的. 评审意见除了属于混合语言文本外, 还属于非书面语言文本, 如存在较多的口语、术语和错别字等. 尽管自动化模型通过自监督预训练, 即领域预训练, 可以学习到当前语料库的评审意见的文本特征, 但是其在未来面临其他语料库的评审意见时仍可能失效. 出于上述原因, 本文创新性地将代码元素、术语和错别字等视为对评审意见的“攻击 (attack)/扰动 (perturb)”.

“对抗训练 (adversarial training)”是一种新型的正则化技术, 可以使对输入样本的小扰动 (这样的样本被称为“对抗样本”) 不会导致模型输出产生较大的变化, 从而帮助提升模型的鲁棒性和泛化能力. 目前, 对抗训练已经被广泛用于各类自然语言处理任务, 并取得了良好效果[43]. 需要注意的是, 在本文提出的方法中, 对抗样本并非指那些原始的包含代码元素、术语或错别字等的评审意见, 而是指对评审意见 (无论其是否包含上述元素) 的嵌入表示添加细微扰动后所形成的样本. 本文期望用“对抗样本”来模拟包含代码元素的或是包含术语、错别字等的评审意见, 从而增强模型的鲁棒性和泛化能力. 具体地, 在实施对抗训练时, 本文使用了“虚拟对抗训练”技术. 与一般的对抗训练策略不同, 虚拟对抗训练不依赖样本的标签, 因此虚拟对抗训练可以在自监督预训练和有监督微调阶段都能发挥作用; 更重要的是, 虚拟对抗训练可以在一定程度上缓解因标签不准确和不一致可能造成的效度威胁. 虚拟对抗训练的实施主要通过设置特殊的损失函数来实现, 详见公式 (8),

关于“非对称损失”和“虚拟对抗训练损失”的更多技术细节请参考原始论文[41,42]. 需要注意的是, 本文在正式实验之前, 使用少量评审意见样本设计和实施了若干预实验 (pilot experiments), 特别是设计和实施了验证“非对称损失”和“虚拟对抗训练损失”效果的消融实验 (ablation experiments). 实验结果证实了它们的有效性, 因此被用作本文提出的方法 (自动化模型) 的损失函数. 需要注意的是, 本文发现“虚拟对抗训练损失”在有监督微调阶段对模型的提升效果更显著, 因此该技术仅在有监督微调阶段被使用.

4 实　验

实验旨在验证本文提出的方法, 特别是验证所提方法中的自动化模型. 为了简洁起见, 除非明确说明, 本节后文使用“本文提出的方法”指代“本文提出的方法中的自动化模型”, 即基于多标签学习范式和预训练模型 BERT 的文本分类模型.

4.1 研究问题

本节提出以下两个研究问题 (research questions) 来驱动实验设计和结果分析.

RQ1: 本文提出的方法能否有效地预测代码评审意见质量属性?

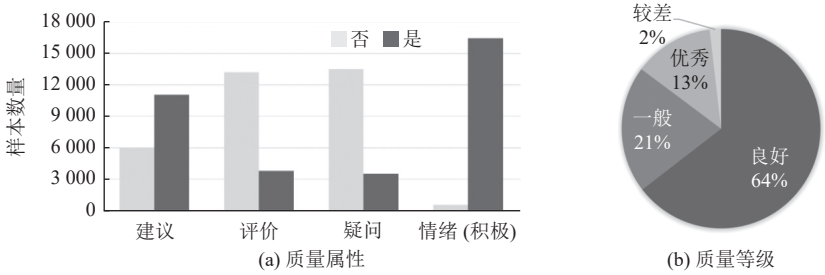
RQ2: 本文提出的方法能否有效地评价代码评审意见质量等级?

本文提出的方法属于混合方法 (概念模型+自动化模型). 基于多标签学习范式和具体的文本分类模型来自动化地预测质量属性是评价质量等级最关键的步骤, 其有效性在回答 RQ1 和 RQ2 中分析和讨论.

4.2 实验设计

4.2.1 数据准备

实验使用的代码评审意见采集自国内某全球领先的软件企业. 为了保证实验数据集的有效性, 尤其是代码评审意见文本的多样性和复杂性, 实验对项目设置了以下准入条件: (1) 在过去 3 年处于开发状态; (2) 至少积累了 1 000 条评审意见; (3) 至少有 20 名开发者以评审者身份参与评审. 最终 34 个项目满足准入条件. 实验对 34 个项目的评审意见进行了随机采样, 以保证每个项目都贡献了相同数量 (500) 的实验样本. 此外, 尽管评审意见中的特殊标签 (如“缺陷类型”和“严重等级”等) 有助于评价其质量, 本文在数据准备阶段仍将其完全移除的原因是: 仅有部分项目的部分评审意见包含该类特殊标签, 考虑特殊标签将会导致实验的内部效度威胁, 如影响人工标注和模型训练等. 全体作者首先集中分析评审意见实例, 制定标注检查表; 然后通过标注少量评审意见, 计算不一致程度, 反复调整、验证, 直至确定标注检查表; 随后 3 位作者根据检查表 (详见第 5.2 节) 独立地标注每一条评审意见, 最后通过召开集中会议讨论和解决不一致的意见. 图 4 展示了数据集中的评审意见的质量属性分布和质量等级分布.



(a) 质量属性 (b) 质量等级

图 4 数据集分布

4.2.2 基准模型

本文提出的方法将预测/学习代码评审意见质量属性形式化为文本分类任务, 本文的评价对象, 即“行内评审意见”属于短文本类型 (行内评审意见 (inline comments): 针对 pull request 中的某一行或几行代码的评审意见, 大部分少于 100 字, 属于短文本类型; 行外评审意见 (non-inline comments): 针对 pull request 全体内容的评审意见, 大部分有数百字之多, 属于长文本类型). 因此实验的基准模型选择了近年来被广泛使用的用于短文本分类任务的自动化模型, 包括 TextCNN[44]、TextRCNN[45]、FastText[46]、DPCNN[47]和 Transformer[39].

4.2.3 评价指标

本文提出的方法旨在基于多标签学习范式构建自动化模型, 因此实验使用多标签学习任务中最常见的若干指标[48]来量化评价各模型表现.

- 汉明损失 (Hamming loss, *HL*):

$$HL = \frac{1}{NL}\sum_{i=1}^{N}\sum_{j=1}^{L} XOR(y_{ij}, \hat{y}_{ij}) \tag{9}$$

- 0-1 损失 (0-1 loss, 01*L*):

$$01L = \frac{1}{N}\sum_{i=1}^{N} I(Y \neq \hat{Y}) \tag{10}$$

- 宏 *F*1 分数 (macro-*F*1 score, *mF*):

$$mF = \frac{1}{L}\sum_{j=1}^{L}\frac{2\sum_{i=1}^{N} y_{ij}\hat{y}_{ij}}{\sum_{i=1}^{N} y_{ij} + \sum_{i=1}^{N} \hat{y}_{ij}} \tag{11}$$

其中, N 表示样本数, L 表示标签数, Y 表示标签的真实值, $\hat{Y}$ 表示标签的预测值, y_{ij} 表示第 i 个样本的第 j 个标签的真实值, $\hat{y}_{ij}$ 表示第 i 个样本的第 j 个标签的预测值. 汉明损失是基于标签的评价指标, 即不考虑在样本尺度上的正确性. 相反, 0-1 损失是基于样本的评价指标, 即只要对样本中的任一标签预测出错, 该样本即被算作预测错误. 宏 $F1$ 分数同样是基于标签的评价指标, 其用于评价模型在不平衡数据集上的表现. 此外, 在评价自动化模型对单一质量属性的学习能力时, 实验还使用了查准率 (precision, P)、召回率 (recall, R) 和 $F1$ 分数 (F). 各评价指标中, 汉明损失和 0-1 损失值越低, 表示模型表现越好; 其余值越高, 表示模型表现越好. 对各模型的评估使用 5 折交叉验证方式, 在每一折验证中, 从每个项目中选择 400 条评审意见组成训练集, 剩余 100 条评审意见作为测试集. 本文使用非参数检验方法“Wilcoxon 符号秩检验 (Wilcoxon signed rank test)”来保证实验结论的正确性. 具体地, 针对实验提出的空假设为: 本文提出的方法的表现相较于基准模型在各评价指标上没有差异.

4.2.4 超参数设置

本文提出的方法涉及若干超参数. 现将主要超参数的设置说明如下: 代码评审意见的最大输入长度 (截断长度) 为 256, 批大小为 32, 学习器为 Adam, 学习率为 3E–5. 非对称损失函数 ASL 中, γ_+ 为 1, γ_- 为 4, m 为 0.05. 虚拟对抗损失函数 VAL 中, ε 和 K 均为 1. 本文遵循机器学习模型训练的一般流程, 即在训练集中划分出验证集, 在验证集上观察超参数对模型性能的影响, 最终选定在验证集上使得模型表现最优的参数作为超参数. 例如, “学习率”影响模型拟合数据集的效果, 是模型最重要的超参数之一. 通过多次实验和调整, 本文发现学习率设置为 3E–5 可以使得本文提出的方法在验证集上最优. 对于不需要通过验证集来确定的“最大输入长度”, 本文统计了数据集中所有评审意见的字符数, 发现当前设置 (256) 可以覆盖处理 95% 的评审意见, 因此将 256 作为超参数.

4.3 结果与分析

表 4–表 6 分别从项目、质量属性和质量等级视角展示了本文提出的方法及其他 5 种基准模型的表现, 加粗的数字表示某一视角下评价指标的最优值.

4.3.1 基于质量属性视角的分析

各模型对质量属性的预测表现如表 5 所示. 首先关注对“情绪”的预测, 本文提出的方法的查准率和召回率分别是 0.76 和 0.42, 分别排在所有模型的第 2 位和第 1 位. TextCNN 在预测“情绪”时, 拥有最高的查准率 (0.85), 但是其召回率仅为 0.22, 远远低于本文提出的方法表现. 综合表现最接近本文提出的方法的是 FastText, 其查准率和召回率分别为 0.75、0.31. 需要注意的是, 尽管 Transformer 的查准率为 0.63, 但是其召回率仅为 0.06, 意味着其几乎不能准确预测“不积极” (本文提出的方法在预测质量属性“情绪”时, 将“不积极”作为正类, 将“积极”作为负类) 的代码评审意见. 在对“疑问”的预测中, 本文提出的方法的查准率、召回率和 $F1$ 分数分别是 0.93、0.96、0.94, 分别排在所有模型的第 2 位、第 1 位、第 1 位. 综合表现最接近本文提出的方法的模型是 TextCNN 和 TextRCNN, 其 $F1$ 分数均为 0.92. 对“评价”的预测, 本文提出的方法的查准率和召回率分别是 0.80、0.81. 对“建议”的预测, 本文提出的方法的查准率和召回率分别是 0.92、0.94. 本文提出的方法在上述两个预测任务中的表现均排在所有模型的第 1 位. Transformer 在上述两个预测任务中的查准率表现和本文提出的方法持平 (0.80、0.92), 但是其召回率远远落后本文提出的方法 (0.51、0.82). 由于拥有较高的查准率, 同时在召回率指标上取得优势, 本文提出的方法在关于 4 个质量属性的预测任务的 $F1$ 指数均排在了所有模型的首位.

表 4 详细展示了各模型在 34 个项目上的预测表现. 此时不再区分每一质量属性, 而是使用多标签学习任务常见的评价指标 (汉明损失、0-1 损失和宏 $F1$ 分数) 来分析各模型表现. 首先关注本文提出的方法, 其在 34 个项目上的平均汉明损失为 0.06, 最接近的模型是 TextCNN 和 TextRCNN, 其平均汉明损失均为 0.08. 本文提出的方法的平均 0-1 损失为 0.19, 最接近的模型仍是 TextCNN 和 TextRCNN, 但其平均 0-1 损失高了 7 个百分点, 为 0.26. 其余模型的平均 0-1 损失接近或超过 0.30. 最后在宏 $F1$ 分数这一项评价指标中, 本文提出的方法同样处于领先的位置, 超过排在第 2 名的模型 (TextRCNN) 6 个百分点 (0.74 vs. 0.68).

表 4　模型表现 (项目视角)

项目	TextCNN			TextRCNN			FastText			DPCNN			Transformer			本文方法		
	HL	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*
P1	0.06	0.19	0.63	0.06	0.18	0.67	0.06	0.20	0.61	0.06	0.20	0.61	0.07	0.24	0.59	**0.04**	**0.12**	**0.73**
P2	0.06	0.20	0.67	0.06	0.21	0.67	0.07	0.21	0.65	0.06	0.19	0.68	0.08	0.26	0.58	**0.04**	**0.13**	**0.77**
P3	0.12	0.37	0.63	0.12	0.35	0.69	0.13	0.37	0.67	0.14	0.39	0.66	0.16	0.44	0.52	**0.08**	**0.25**	**0.79**
P4	0.08	0.28	0.68	0.09	0.28	0.69	0.10	0.31	0.65	0.10	0.31	0.71	0.10	0.33	0.58	**0.05**	**0.17**	**0.84**
P5	0.10	0.33	0.66	0.09	0.30	0.74	0.10	0.31	0.65	0.10	0.32	0.70	0.12	0.39	0.53	**0.06**	**0.20**	**0.79**
P6	0.06	0.22	0.62	0.06	0.19	0.63	0.07	0.23	0.56	0.07	0.23	0.59	0.09	0.27	0.51	**0.04**	**0.14**	**0.66**
P7	0.05	0.18	0.72	0.05	0.18	0.75	0.06	0.20	0.75	0.05	0.19	0.78	0.07	0.25	0.64	**0.03**	**0.12**	**0.83**
P8	0.07	0.26	0.63	0.07	0.22	0.64	0.08	0.26	0.62	0.08	0.26	0.63	0.08	0.28	0.60	**0.04**	**0.15**	**0.72**
P9	0.07	0.24	0.62	0.07	0.22	0.72	0.08	0.26	0.65	0.08	0.27	0.69	0.08	0.27	0.60	**0.05**	**0.16**	**0.73**
P10	0.06	0.21	0.64	0.06	0.19	**0.69**	0.07	0.22	0.65	0.06	0.20	0.69	0.07	0.23	0.63	**0.04**	**0.12**	**0.69**
P11	0.09	0.28	0.76	0.09	0.26	0.78	0.10	0.31	0.73	0.10	0.30	0.73	0.12	0.38	0.58	**0.06**	**0.21**	**0.85**
P12	0.06	0.22	0.71	0.06	0.21	0.78	0.07	0.23	0.64	0.06	0.21	0.67	0.07	0.24	0.64	**0.04**	**0.16**	**0.80**
P13	0.08	0.26	0.62	0.08	0.25	0.63	0.08	0.25	0.60	0.08	0.25	0.62	0.09	0.30	0.59	**0.05**	**0.16**	**0.68**
P14	0.06	0.22	0.64	0.06	0.21	0.63	0.07	0.24	0.62	0.06	0.23	0.66	0.09	0.28	0.57	**0.05**	**0.17**	**0.67**
P15	0.13	0.40	0.63	0.13	0.39	0.66	0.14	0.41	0.67	0.15	0.43	0.63	0.15	0.45	0.53	**0.10**	**0.32**	**0.73**
P16	0.05	0.17	0.63	0.04	0.15	0.67	0.05	0.17	0.67	0.05	0.17	0.63	0.07	0.23	0.58	**0.03**	**0.11**	**0.70**
P17	0.06	0.21	0.70	0.07	0.22	0.72	0.08	0.25	0.71	0.08	0.24	0.68	0.09	0.29	0.58	**0.05**	**0.19**	**0.72**
P18	0.09	0.29	0.74	0.10	0.31	0.74	0.10	0.32	0.74	0.10	0.30	0.77	0.12	0.37	0.59	**0.06**	**0.20**	**0.86**
P19	0.09	0.27	0.58	0.09	0.28	0.57	0.11	0.32	0.53	0.11	0.33	0.56	0.11	0.34	0.51	**0.06**	**0.20**	**0.70**
P20	0.09	0.30	0.65	0.08	0.28	**0.68**	0.09	0.30	0.65	0.10	0.31	0.62	0.10	0.32	0.60	**0.06**	**0.20**	**0.68**
P21	0.06	0.22	0.65	0.06	0.23	0.65	0.07	0.23	0.64	0.07	0.25	0.63	0.08	0.26	0.61	**0.04**	**0.16**	**0.69**
P22	0.09	0.31	0.66	0.09	0.31	0.66	0.13	0.36	0.60	0.12	0.37	0.63	0.13	0.39	0.53	**0.07**	**0.24**	**0.74**
P23	0.12	0.38	0.63	0.12	0.38	0.68	0.12	0.39	0.67	0.12	0.36	0.69	0.13	0.40	0.55	**0.10**	**0.31**	**0.74**
P24	0.09	0.30	0.61	0.09	0.29	0.67	0.10	0.32	0.59	0.11	0.33	0.65	0.12	0.37	0.56	**0.07**	**0.24**	**0.75**
P25	0.08	0.25	0.59	0.08	0.26	0.59	0.10	0.32	0.54	0.11	0.33	0.54	0.12	0.37	0.50	**0.07**	**0.24**	**0.67**
P26	0.06	0.19	0.68	0.06	0.19	0.67	0.05	0.18	0.69	0.05	0.18	0.65	0.06	0.20	0.65	**0.03**	**0.11**	**0.71**
P27	0.06	0.20	0.63	0.05	0.19	0.63	0.06	0.20	0.62	0.06	0.19	0.62	0.07	0.23	0.59	**0.04**	**0.15**	**0.65**
P28	0.10	0.33	0.68	0.10	0.31	0.68	0.12	0.36	0.66	0.12	0.35	0.67	0.13	0.39	0.58	**0.07**	**0.25**	**0.78**
P29	0.12	0.38	0.65	0.13	0.37	0.64	0.12	0.37	0.67	0.12	0.35	0.72	0.14	0.41	0.55	**0.08**	**0.26**	**0.82**
P30	0.06	0.21	0.66	0.06	0.19	0.68	0.07	0.23	0.62	0.07	0.24	0.62	0.08	0.25	0.56	**0.05**	**0.16**	**0.70**
P31	0.08	0.27	0.75	0.08	0.24	0.76	0.08	0.25	0.73	0.08	0.24	0.75	0.08	0.28	0.66	**0.06**	**0.18**	**0.79**
P32	0.07	0.24	0.66	0.08	0.26	0.66	0.07	0.22	0.65	0.07	0.23	0.72	0.09	0.29	0.61	**0.05**	**0.17**	**0.75**
P33	0.09	0.29	0.74	0.09	0.31	0.74	0.09	0.31	0.78	0.10	0.33	0.73	0.11	0.35	0.63	**0.08**	**0.25**	**0.81**
P34	0.10	0.34	0.55	0.10	0.34	0.57	0.12	0.38	0.51	0.13	0.42	0.52	0.13	0.40	0.51	**0.07**	**0.25**	**0.63**
Avg	0.08	0.26	0.66	0.08	0.26	0.68	0.09	0.28	0.65	0.09	0.28	0.66	0.10	0.32	0.58	**0.06**	**0.19**	**0.74**

表 5　模型表现 (质量属性视角)

属性	TextCNN			TextRCNN			FastText			DPCNN			Transformer			本文方法		
	P	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*	*P*	*R*	*F*
情绪	**0.85**	0.22	0.35	0.57	0.32	0.40	0.75	0.31	0.44	0.70	0.35	0.43	0.63	0.06	0.10	0.76	**0.42**	**0.54**
疑问	**0.95**	0.89	0.92	0.92	0.93	0.92	0.90	0.82	0.85	0.89	0.86	0.87	0.87	0.84	0.85	0.93	**0.96**	**0.94**
评价	0.79	0.60	0.68	0.74	0.68	0.71	0.78	0.62	0.69	0.75	0.68	0.71	**0.80**	0.51	0.62	**0.80**	**0.81**	**0.81**
建议	0.89	0.90	0.90	0.89	0.91	0.90	0.89	0.89	0.89	0.89	0.89	0.88	**0.92**	0.82	0.86	**0.92**	**0.94**	**0.93**

表 7 展示了本文提出的方法与基准模型在 34 个项目上关于汉明损失、0-1 损失和宏 *F*1 分数的 Wilcoxon 符号秩检验结果, 实证了本文提出的方法在预测评审意见质量属性任务上领先基准模型 (当 p 值小于 0.001 时, 拒绝空假设).

表 6 模型表现 (质量等级视角)

等级	TextCNN			TextRCNN			FastText			DPCNN			Transformer			本文方法		
	HL	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*	*HL*	01*L*	*mF*
优秀	0.16	0.57	0.66	0.14	0.49	0.70	0.18	0.62	0.68	0.16	0.53	0.67	0.25	0.81	0.50	**0.10**	**0.36**	**0.77**
良好	0.05	0.16	0.56	0.05	0.16	0.54	0.06	0.17	0.51	0.06	0.18	0.52	0.06	0.19	0.50	**0.03**	**0.11**	**0.59**
一般	0.10	0.34	0.42	0.11	0.36	0.42	0.12	0.36	0.43	0.12	0.40	0.44	0.11	0.33	0.34	**0.09**	**0.30**	**0.48**
较差	0.25	0.79	0.33	0.23	0.73	0.36	0.26	0.76	0.35	0.26	0.78	0.35	0.31	0.95	0.26	**0.20**	**0.65**	**0.39**

表 7 Wilcoxon 符号秩检验 (p 值)

模型	本文方法		
	HL	01*L*	*mF*
TextCNN	<2E–7	<2E–7	<2E–7
TextRCNN	<2E–7	<2E–7	<3E–7
FastText	<2E–7	<2E–7	<2E–7
DPCNN	<2E–7	<2E–7	<3E–7
Transformer	<2E–7	<2E–7	<2E–7

4.3.2 基于质量等级视角的分析

需要首先说明的是, 根据本文提出的方法的设计原理 (详见第 3.1 节), 质量等级即评审意见质量评价最终的产出是根据预先定义的映射表来确定的, 该部分不参与“预测/学习”. 换言之, 质量等级由预测的质量属性间接确定. 评价自动化模型对不同质量等级的预测表现, 尽管视角不同 (聚合评审意见样本的方式不同, 本文回答 RQ1 按“项目”和“质量属性”聚合; 回答 RQ2 按“质量等级”聚合), 实际仍是评价其对各质量属性的预测表现. 因此, 本文回答 RQ2 仍沿用在多标签学习任务中常见的评价指标 (汉明损失、0-1 损失和宏 $F1$ 分数).

各模型对质量等级的预测表现如表 6 所示. 首先关注对“优秀”的预测, 本文提出的方法的汉明损失、0-1 损失和宏 $F1$ 分数分别为 0.10、0.36、0.77, 均排在所有模型的第 1 位. 排在第 2 位的模型是 TextRCNN, 3 项指标分别为 0.14、0.49、0.70. 在对“良好”的预测中, 本文提出的方法的汉明损失、0-1 损失和宏 $F1$ 分数分别为 0.03、0.11、0.59, 均领先于其他所有模型. 排在第 2 位的模型是 TextCNN, 3 项指标分别为 0.05、0.16、0.56. 在对“一般”的预测中, 本文提出的方法的汉明损失、0-1 损失和宏 $F1$ 分数分别为 0.09、0.30、0.48, 均排在所有模型的第 1 位. 在该预测任务中, 不再有 3 项评价指标均排在第 2 位的模型. 汉明损失排在第 2 位的模型是 TextCNN (0.10); 0-1 损失排在第 2 位的模型是 Transformer (0.33), 宏 $F1$ 分数排在第 2 位的模型是 FastText (0.43). 在对“较差”的预测中, 本文提出的方法的汉明损失、0-1 损失和宏 $F1$ 分数分别为 0.20、0.65、0.39, 同样均排在所有模型的第 1 位. 排在第 2 位的模型是 TextRCNN, 3 项指标分别为 0.23、0.73、0.36.

值得注意的是, 本文提出的方法对“少数类”的预测表现相较于其他模型具有领先优势, 实证了设置非对称损失函数的正确性和必要性 (详见第 3.3.2 节). 如图 4 所示, 质量等级为“优秀”评审意见占总数的 13%, 属于少数类; 质量等级为“较差”评审意见仅占总数的 2%, 属于极少数类. 以分析各模型的 0-1 损失为例, 本文提出的方法预测“优秀”和“较差”的 0-1 损失分别为 0.36、0.65, 表现最接近的模型是 TextRCNN (“优秀”—0.49, “较差”—0.73). 表现最糟糕的模型是 Transformer, 其预测“优秀”的 0-1 损失为 0.81; 对“较差”的预测几乎完全失败, 其 0-1 损失高达 0.95.

总结来说, 本文提出的方法 (自动化模型) 能够有效地预测代码评审意见的质量属性 (RQ1) 和质量等级 (RQ2), 在多项评价指标上领先于其他模型.

5 讨 论

本节讨论应用本文提出的方法时应该注意的若干事项. 具体地, 第 5.1 节和第 5.2 节分别讨论其数据标注和模型选择; 第 5.3 节讨论其对代码评审的影响和启示.

5.1 标签标注经验

本节讨论针对代码评审意见的标注经验和一致性保证措施.

5.1.1 标注检查表

针对 4 个质量属性的标注检查表总结如下.

消极情绪: (1) 出现连续的标点, 如 “, , , ”“. . . ”“! ! !”等; (2) 出现带有反问、讽刺语气的字词, 如“啊”“难道”“难不成”“谁能”“神经”“干啥”“都啥”“毛病”等, 该评审意见通常传递了消极情绪.

提出疑问: (1) 出现问号; (2) 出现疑问词, 如“吗”“呢”“啥”“什么”“为什么”“为何”“为啥”等, 该评审意见通常提出了疑问.

给予评价: (1) 出现形容词, 如“无用”“冗余”“错误”“重复”“啰嗦”“完整”“清晰”等; (2) 出现程度副词, 如“太”“过”“十分”“非常”等; (3) 出现关键词“未”“不”“缺少”“没有”等, 该评审意见通常给予了评价/指出了缺陷.

提供建议: 出现动词, 尤其是实义动词和表指示或建议的抽象动词, 如“增”“删”“改”“关闭”“补充”“完善”“需要”“不需要”“建议”“不建议”等, 该评审意见通常提供了针对代码的修改或提升建议.

尽管可以总结出若干标注经验和使用 SentiCR[49]等自动化模型/工具来帮助判断代码评审意见的质量属性, 但是人工经验 (“正则表达式”) 不能覆盖所有的情形; 自动化模型/工具经常存在“漏报”和“误报”问题[50]. 因此在构建 ground-truth 数据集时, 人工标注和检查是重要的和必须的. “人在回路 (human-in-the-loop)”的数据标注[51]结合了人工经验和自动化模型的优点, 是缓解数据标注工作挑战的有效措施之一.

对于第 2.1 节表 1 中的评审意见实例, 其质量属性 (由人工标注/模型学习) 和质量等级 (根据映射表生成) 如下.

如表 8 所示, 本文提出的方法一方面能够精细地区分不同评审意见的质量属性和质量等级; 另一方面, 尽管评审意见 C2—“逻辑不对”、C3—“魔鬼数字”、C4—“info 日志打印太多”在表达形式上有细微差异 (C2 和 C4 出现关键字: “逻辑”和“日志”, 并且直接评价缺陷; C3 出现术语: “魔鬼数字”, 并且间接评价缺陷; C4 出现代码元素), 但对它们的标注结果却是一致的, 反映了本文提出的方法的泛化能力.

表 8 代码评审意见质量属性和等级实例

属性	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
情绪	1	1	1	1	1	1	0	1	1	1
疑问	0	0	0	0	1	0	0	1	0	0
评价	0	1	1	1	0	1	1	0	0	1
建议	0	0	0	0	0	1	0	1	1	1
等级	较差	良好	良好	良好	一般	优秀	一般	优秀	良好	优秀

5.1.2 一致性保证

除了制定标注检查表, 本文还采取了若干措施来尽可能地保证和提升评审意见质量属性标注的一致性, 现总结如下.

预标注: 预标注的好处在于: (1) 使标注人员更快熟悉评审意见实例/样本和标注检查表, 从而更快进入工作状态. (2) 能够及时发现“异常”和“边界”实例, 如有必要, 需要重新制定质量属性和标注检查表等.

锁定检查表: 临时锁定检查表的好处在于: (1) 保证每次标注前都有一份可参考的完整指南. (2) 保证每次标注后都有一份可使用的完整数据集. (3) 帮助发现“异常”和“边界”实例等.

结对标注: 结对标注的好处同“结对编程”和“结对评审”类似: (1) 帮助理解检查表. (2) 帮助发现和处理“异常”和“边界”实例. (3) 互相监督以保证工作效率和质量等.

一致性检验: 实施一致性检验的好处在于量化地评估同一标注者在不同时候, 或多个标注者, 对同一份数据集的标注结果是否一致, 同时间接评估其对标注检查表的理解是否一致. 针对出现不一致标注的实例, 需要重点分析; 如有必要, 需要重新制定质量属性和标注检查表等. 一致性检验除了可以使用经典的 Kappa 系数外, 还可以使用本文实验所用的评价指标 (详见第 4.2.3 节). 此时, 两份标注结果分别被视作“真实值”和“预测值”.

5.2 标签相关分析

相关分析指的是分析两个或多个可能具备相关性的变量元素, 从而衡量两个变量因素的相关密切程度. 分析评审意见质量属性标签 (变量) 相关性的重要意义在于: (1) 辅助标签设计、标注与验证; (2) 辅助模型设计、构建与验证. 如果标签存在相关/依赖关系, 则在标注时可以先标注某一标签, 利用该标签信息指导下一标签的标注; 否则可以独立标注. 在对多标签学习任务建模时, 同样需要考虑多个标签之间是否存在相关性: 如果不存在, 可使用“binary relevance”类算法; 如果存在, 可使用“classifier chain”类算法[23].

图 5 展示了针对本文实验使用的 17 000 条代码评审意见, 使用“Kendall 秩相关系数 (Kendall-Tau rank correlation coefficient)”相关性分析方法的结果. 图中每个方块内的数字即两两质量属性的 Kendall 秩相关系数τ. 如果τ小于 0.2, 表示几乎不存在相关性; 如果τ大于 0.2 但是小于 0.4, 表示存在弱相关性; 如果τ大于 0.4, 表示至少存在中等程度的相关性. 从图 5 可以看出, 大多数质量属性的相关性都非常低, 实证了本文提出的方法 (自动化模型) 使用“binary relevance”一类算法的正确性. 注: 自动化模型中设置了 4 个同步的中间输出节点 (详见第 3.2.2 节), 用于表示 4 个质量属性的学习任务.

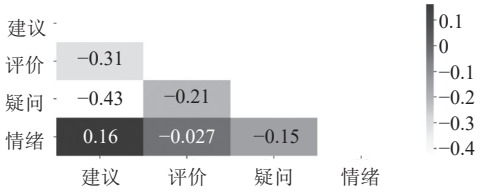


图 5 质量属性相关分析

5.3 影响和启示

5.3.1 对管理者的影响和启示

当实施评价, 尤其是在实施本文提出的方法时, 管理者应该首先需要考虑评价结果的呈现形式. 例如, 系统仅实时地向评审者展示对评审意见的 4 个质量属性的预测结果 (起规范评审意见书写的作用), 评审者需要手动查看才能知晓质量等级 (减轻对评审过程的干扰). 另一方面, 呈现形式应尽可能友好. 例如, 以卡通动画的表情包的形式呈现质量属性和质量等级. 其次, 管理者在团队及以上层面公开代码评审的评价结果时, 应该公开代码变更和评审意见, 而不是项目或个人. 此外, 尽管评审意见是评审的最主要和最重要产出, 管理者需要考虑结合更多方面的因素 (例如参与评审的频次, 评审的详细程度, 项目背景和成熟度等) 来综合评价代码评审. 总结来说, 评价是方法, 提升才是目标.

5.3.2 对评审者的影响和启示

评审者应该首先关注于提升代码质量, 而不是刻意地追求“高质量”的评审意见. 例如, 对于命名和风格一类的缺陷, 无需向有经验的开发者提供解释和修改建议. 另一方面, 当评审重要的和紧急的代码变更时, 评审者应该就功能性和安全性等高优先级的、复杂的缺陷提供解释和修改建议, 从而帮助开发者更好地提升代码质量和编程水平.

6 效度威胁

内部效度 (internal validity): 实验的内部效度威胁可能来自数据准备环节. 代码评审意见因其可能包含程序语言文本/代码元素而属于一类特殊的自然语言文本, 具有多样性和复杂性. 本文使用的“三角论证”方法仅能建立起评审意见质量评价的理论/概念模型, 为了训练和验证可用的自动化模型, 实验需要构建 ground-truth 数据集. 为此, 本文作者首先在超过 2 000 条的意见上进行了“预标注”, 所得经验经过多次整理和验证后形成了正式的“标注检查表” (详见第 5.1.1 节). 在正式的标注阶段, 3 位作者首先独立地标注数据集中的每一条意见, 然后通过面对面的集中会议讨论解决所有不一致的意见, 从而能够保证标签标注的正确性和一致性.

结论效度 (conclusion validity): 为了避免可能存在的结论效度威胁, 本文实验首先构建了一个远超过其他同类型研究工作的实验规模的数据集 (34 个项目, 共计 17 000 条代码评审意见). 同时, 实验使用了严格的交叉验证和 Wilcoxon 符号秩检验方法来检验提出的自动化模型的有效性、先进性, 从而能够保证结论的正确性.

外部效度 (external validity): 本文提出的方法中的概念模型, 其质量属性经过了严格的“三角论证”. 使用的数据源是“正式文件”“学术文献”和“实例分析结果”. 前两个数据源汇聚了世界范围不同背景下的多个软件组织、学术团队等的经验与知识, “实例分析结果”包含了对开源项目和商业项目、中文和外文 (英文) 的评审意见的分析. 由此产生的评价标准具备普适性, 即对开源项目和商业项目, 中文和外文代码评审意见均适用. 但是质量属性与质量等级的映射关系仅为一个实例; 同时实验样本, 即数据集仅来自一家软件企业, 将可能导致本文提出的方法的自动化模型的外部效度威胁. 对此, 本文在实验的数据准备阶段设定了严格的项目准入条件 (详见第 4.2.1 节) 来保证实验样本的多样性和代表性. 另一方面, 本文第 4 节实验和第 5.1 节的标签标注经验对象均为中文代码评审意见, 考虑到语言特征的差异, 本文报告的相关发现和经验不能直接迁移至其他语言环境. 总结地, 尽管本文仅报告了提出的方法应用于某软件企业的商业项目的中文评审意见的有效性, 其仍有潜力被应用于更广泛的项目和语言的评审意见的质量评价. 其他软件组织在使用本文方法时, 可以根据自身条件和需求, 灵活地调整质量属性、映射关系和自动化模型. 同时参考本文总结的数据标注经验, 以更好地实施本文方法.

7 总结与展望

评审意见是代码评审最主要和最重要的产出之一, 其文本挖掘对代码评审分析、评价和改进具有重要支持作用. 针对当前代码评审意见评价方式依赖代码变更状态、可解释性差、实践指导性不足等问题, 本文开展了若干实证研究, 并在此基础上提出了一种基于多标签学习的代码评审意见质量评价方法. 该方法使用 4 个质量属性 (“情绪”“疑问”“评价”和“建议”) 来综合评价评审意见质量 (“优秀”“良好”“一般”和“较差”). 其中, 质量属性通过数据三角论证识别和合成, 并由基于多标签学习范式和预训练语言模型 BERT 以“端到端”的方式自动学习; 质量属性到质量等级映射由若干软件工程学者和业界专家共同修订和确定. 实验使用国内某大型软件企业的 34 个商业项目, 共计 17 000 条代码评审意见作为数据集. 结果表明本文提出的方法能够有效地评价代码评审意见质量.

未来工作一方面将持续验证本文提出的方法在其他软件企业、语言环境 (如英文) 和社区环境 (如 GitHub、Gerrit 等开源软件社区) 的可行性和先进性. 另一方面, 未来工作将持续深入挖掘代码评审意见, 如缺陷类型和严重程度等, 并结合项目背景和评审过程等数据, 持续调整和优化本文提出的方法. 最后, 未来工作将持续研究代码评审, 开发支持方法、技术和工具等, 助力实现系统化的代码评审质量评价和过程改进.

References:

[1] Fagan ME. Design and code inspections to reduce errors in program development. IBM Systems Journal, 1976, 15(3): 182–211. [doi: 10.1147/sj.153.0182]

[2] Rigby PC, Bird C. Convergent contemporary software peer review practices. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg: ACM, 2013. 202–212. [doi: 10.1145/2491411.2491444]

[3] Sadowski C, Söderberg E, Church L, Sipko M, Bacchelli A. Modern code review: A case study at Google. In: Proc. of the 40th Int’l Conf. on Software Engineering: Software Engineering in Practice. Gothenburg: ACM, 2018. 181–190. [doi: 10.1145/3183519.3183525]

[4] Bacchelli A, Bird C. Expectations, outcomes, and challenges of modern code review. In: Proc. of the 35th Int’l Conf. on Software Engineering. San Francisco: IEEE, 2013. 712–721. [doi: 10.1109/ICSE.2013.6606617]

[5] Caulo M, Lin B, Bavota G, Scanniello G, Lanza M. Knowledge transfer in modern code review. In: Proc. of the 28th Int’l Conf. on Program Comprehension. Seoul: ACM, 2020. 230–240. [doi: 10.1145/3387904.3389270]

[6] Zampetti F, Mudbhari S, Arnaoudova V, Di Penta M, Panichella S, Antoniol G. Using code reviews to automatically configure static analysis tools. Empirical Software Engineering, 2022, 27(1): 28. [doi: 10.1007/s10664-021-10076-4]

[7] Huq F, Hasan M, Haque MA, Mahbub S, Iqbal A, Ahmed T. Review4Repair: Code review aided automatic program repairing. Information and Software Technology, 2022, 143: 106765. [doi: 10.1016/j.infsof.2021.106765]

[8] Alami A, Cohn ML, Wąsowski A. Why does code review work for open source software communities? In: Proc. of the 41st Int’l Conf. on

Software Engineering. Montreal: IEEE, 2019. 1073−1083. [doi: 10.1109/ICSE.2019.00111]

[9] Bosu A, Carver JC, Bird C, Orbeck J, Chockley C. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at Microsoft. IEEE Trans. on Software Engineering, 2017, 43(1): 56−75. [doi: 10.1109/TSE.2016.2576451]

[10] Bosu A, Greiler M, Bird C. Characteristics of useful code reviews: An empirical study at Microsoft. In: Proc. of the 12th IEEE/ACM Working Conf. on Mining Software Repositories. Florence: IEEE, 2015. 146−156. [doi: 10.1109/MSR.2015.21]

[11] McIntosh S, Kamei Y, Adams B, Hassan AE. An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering, 2016, 21(5): 2146−2189. [doi: 10.1007/s10664-015-9381-9]

[12] Fatima N, Nazir S, Chuprat S. Software engineering wastes—A perspective of modern code review. In: Proc. of the 3rd Int'l Conf. on Software Engineering and Information Management. Sydney: ACM, 2020. 93−99. [doi: 10.1145/3378936.3378953]

[13] Czerwonka J, Greiler M, Tilford J. Code reviews do not find bugs. How the current code review best practice slows us down. In: Proc. of the 37th IEEE/ACM IEEE Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 27−28. [doi: 10.1109/ICSE.2015.131]

[14] German DM, Robles G, Poo-Caamaño G, Yang X, Iida H, Inoue K. "Was my contribution fairly reviewed?": A framework to study the perception of fairness in modern code reviews. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: ACM, 2018. 523−534. [doi: 10.1145/3180155.3180217]

[15] El Asri I, Kerzazi N, Uddin G, Khomh F, Idrissi MAJ. An empirical study of sentiments in code reviews. Information and Software Technology, 2019, 114: 37−54. [doi: 10.1016/j.infsof.2019.06.005]

[16] Egelman CD, Murphy-Hill E, Kammer E, Hodges MM, Green C, Jaspan C, Lin J. Predicting developers' negative feelings about code review. In: Proc. of the 42nd Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 174−185. [doi: 10.1145/3377811.3380414]

[17] Bosu A, Carver JC. Impact of peer code review on peer impression formation: A survey. In: Proc. of the 2013 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Baltimore: IEEE, 2013. 133−142. [doi: 10.1109/ESEM.2013.23]

[18] Doğan E, Tüzün E. Towards a taxonomy of code review smells. Information and Software Technology, 2022, 142: 106737. [doi: 10.1016/j.infsof.2021.106737]

[19] Dong LM, Zhang H, Yang LX, Weng ZL, Yang X, Zhou X, Pan ZF. Survey on pains and best practices of code review. In: Proc. of the 28th Asia-Pacific Software Engineering Conf. Taipei: IEEE, 2021. 482−491. [doi: 10.1109/APSEC53868.2021.00055]

[20] Hasan M, Iqbal A, Islam MRU, Rahman AJMI, Bosu A. Using a balanced scorecard to identify opportunities to improve code review effectiveness: An industrial experience report. Empirical Software Engineering, 2021, 26(6): 129. [doi: 10.1007/s10664-021-10038-w]

[21] Spadini D, Çalikli G, Bacchelli A. Primers or reminders?: The effects of existing review comments on code review. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 1171−1182. [doi: 10.1145/3377811.3380385]

[22] Rahman MM, Roy CK, Kula RG. Predicting usefulness of code review comments using textual features and developer experience. In: Proc. of the 14th IEEE/ACM Int'l Conf. on Mining Software Repositories. Buenos Aires: IEEE, 2017. 215−226. [doi: 10.1109/MSR.2017.17]

[23] Zhang ML, Zhou ZH. A review on multi-label learning algorithms. IEEE Trans. on Knowledge and Data Engineering, 2014, 26(8): 1819−1837. [doi: 10.1109/TKDE.2013.39]

[24] Jiang J, Wu QD, Zhang L. Open source community review process measurement system and its empirical research. Ruan Jian Xue Bao/Journal of Software, 2021, 32(12): 3698−3709 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6127.htm [doi: 10.13328/j.cnki.jos.006127]

[25] Baysal O, Kononenko O, Holmes R, Godfrey MW. Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering, 2016, 21(3): 932−959. [doi: 10.1007/s10664-015-9366-8]

[26] Mirsaeedi E, Rigby PC. Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 1183−1195. [doi: 10.1145/3377811.3380335]

[27] Hu YZ, Wang JJ, Li SB, Hu J, Wang Q. Response time constrained code reviewer recommendation. Ruan Jian Xue Bao/Journal of Software, 2021, 32(11): 3372−3387 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6079.htm [doi: 10.13328/j.cnki.jos.006079]

[28] Rong GP, Zhang YF, Yang LX, Zhang FL, Kuang HY, Zhang H. Modeling review history for reviewer recommendation: A hypergraph approach. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1381−1392. [doi: 10.1145/3510003.3510213]

[29] Li ZX, Yu Y, Yin G, Wang T, Fan Q, Wang HM. Automatic classification of review comments in pull-based development model. In: Proc. of the 9th Int'l Conf. on Software Engineering and Knowledge Engineering. Pittsburgh: SEKE, 2017. 572−577. [doi: 10.18293/

SEKE2017-039]

[30] Ortu M, Adams B, Destefanis G, Tourani P, Marchesi M, Tonelli R. Are bullies more productive? Empirical study of affectiveness vs. issue fixing time. In: Proc. of the 12th IEEE/ACM Working Conf. on Mining Software Repositories. Florence: IEEE, 2015. 303–313. [doi: 10.1109/MSR.2015.35]

[31] Efstathiou V, Spinellis D. Code review comments: Language matters. In: Proc. of the 40th Int'l Conf. on Software Engineering: New Ideas and Emerging Results. Gothenburg: ACM, 2018. 69–72. [doi: 10.1145/3183399.3183411]

[32] Denzin NK. Triangulation 2.0. Journal of Mixed Methods Research, 2012, 6(2): 80–88. [doi: 10.1177/1558689812437186]

[33] IEEE Computer Society. IEEE Std 1028™—2008 IEEE standard for software reviews and audits. New York: IEEE, 2008. 1–53. [doi: 10.1109/IEEESTD.2008.4601584]

[34] Tricco AC, Antony J, Zarin W, Strifler L, Ghassemi M, Ivory J, Perrier L, Hutton B, Moher D, Straus SE. A scoping review of rapid review methods. BMC Medicine, 2015, 13(1): 224. [doi: 10.1186/s12916-015-0465-6]

[35] Davila N, Nunes I. A systematic literature review and taxonomy of modern code review. Journal of Systems and Software, 2021, 177: 110951. [doi: 10.1016/j.jss.2021.110951]

[36] Wang D, Ueda Y, Kula RG, Ishio T, Matsumoto K. Can we benchmark code review studies? A systematic mapping study of methodology, dataset, and metric. Journal of Systems and Software, 2021, 180: 111009. [doi: 10.1016/j.jss.2021.111009]

[37] 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]

[38] Liu PF, Qiu XP, Huang XJ. Recurrent neural network for text classification with multi-task learning. In: Proc. of the 25th Int'l Joint Conf. on Artificial Intelligence. New York: AAAI, 2016. 2873–2879.

[39] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[40] Gururangan S, Marasović A, Swayamdipta S, Lo K, Beltagy I, Downey D, Smith NA. Don't stop pretraining: Adapt language models to domains and tasks. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, 2020. 8342–8360. [doi: 10.18653/v1/2020.acl-main.740]

[41] Ridnik T, Ben-Baruch E, Zamir N, Noy A, Friedman I, Protter M, Zelnik-Manor L. Asymmetric loss for multi-label classification. In: Proc. of the 2021 IEEE/CVF Int'l Conf. on Computer Vision. Montreal: IEEE, 2021. 82–91. [doi: 10.1109/ICCV48922.2021.00015]

[42] Miyato T, Maeda SI, Koyama M, Ishii S. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2019, 41(8): 1979–1993. [doi: 10.1109/TPAMI.2018.2858821]

[43] Zhang WE, Sheng QZ, Alhazmi A, Li CL. Adversarial attacks on deep-learning models in natural language processing: A survey. ACM Trans. on Intelligent Systems and Technology, 2020, 11(3): 24. [doi: 10.1145/3374217]

[44] Kim Y. Convolutional neural networks for sentence classification. In: Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing. Doha: ACL, 2014. 1746–1751. [doi: 10.3115/v1/d14-1181]

[45] Lai SW, Xu LH, Liu K, Zhao J. Recurrent convolutional neural networks for text classification. In: Proc. of the 29th AAAI Conf. on Artificial Intelligence. Austin: AAAI, 2015. 2267–2273. [doi: 10.1609/aaai.v29i1.9513]

[46] Joulin A, Grave E, Bojanowski P, Mikolov T. Bag of tricks for efficient text classification. In: Proc. of the 15th Conf. of the European Chapter of the Association for Computational Linguistics. Valencia: ACL, 2017. 427–431. [doi: 10.18653/v1/e17-2068]

[47] Johnson R, Zhang T. Deep pyramid convolutional neural networks for text categorization. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics. Vancouver: ACL, 2017. 562–570. [doi: 10.18653/v1/P17-1052]

[48] Wu XZ, Zhou ZH. A unified view of multi-label performance measures. In: Proc. of the 34th Int'l Conf. on Machine Learning. Sydney: JMLR. org, 2017. 3780–3788.

[49] Ahmed T, Bosu A, Iqbal A, Rahimi S. SentiCR: A customized sentiment analysis tool for code review interactions. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana: IEEE, 2017. 106–111. [doi: 10.1109/ASE.2017.8115623]

[50] Jongeling R, Sarkar P, Datta S, Serebrenik A. On negative results when using sentiment analysis tools for software engineering research. Empirical Software Engineering, 2017, 22(5): 2543–2584. [doi: 10.1007/s10664-016-9493-x]

[51] Pandey R, Purohit H, Castillo C, Shalin VL. Modeling and mitigating human annotation errors to design efficient stream processing systems with human-in-the-loop machine learning. Int'l Journal of Human-computer Studies, 2022, 160: 102772. [doi: 10.1016/j.ijhcs.2022.102772]

附中文参考文献:

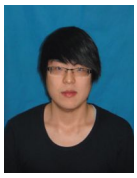
[24] 蒋竞, 吴秋迪, 张莉. 开源社区评审过程度量体系及其实证研究. 软件学报, 2021, 32(12): 3698−3709. http://www.jos.org.cn/1000-9825/6127.htm [doi: 10.13328/j.cnki.jos.006127]

[27] 胡渊喆, 王俊杰, 李守斌, 胡军, 王青. 响应时间约束的代码评审人推荐. 软件学报, 2021, 32(11): 3372-3387. http://www.jos.org.cn/1000-9825/6079.htm [doi: 10.13328/j.cnki.jos.006079]

杨岚心(1993－), 男, 博士生, 主要研究领域为软件工程, 软件质量, 软件代码评审.

王梓宽(1999－), 男, 硕士生, 主要研究领域为持续集成相关技术, 软件代码评审.

张贺(1971－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件工程, 开发运维一体化, 软件研发效能, 软件安全, 经验及循证软件工程, 区块链.



周鑫(1991－), 男, 博士生, 主要研究领域为经验软件工程, 灰色文献, 自然语言处理.

徐近伟(1994－), 男, 博士生, CCF 学生会员, 主要研究领域为软件代码评审, 软件供应链.

李京悦(1974－), 男, 博士, 副教授, 主要研究领域为经验软件工程, 软件安全和数据隐私, 区块链技术.

张逸凡(1998－), 男, 硕士, 主要研究领域为软件代码评审, 推荐系统.

荣国平(1977－), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为软件过程, 实证软件工程.