软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(11):5082−5101 [doi: 10.13328/j.cnki.jos.007380] [CSTR: 32375.14.jos.007380]

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

基于机器学习的开源软件项目维护状态识别*

罗诗雨[1,2], 李馨蕾[3], 罗俊韬[1], 王 新[1,2], 张国锋[3], 陈 阳[1,2]

[1](复旦大学 计算机科学技术学院, 上海 200438)

[2](上海市智能信息处理重点实验室 (复旦大学), 上海 200438)

[3](上海对外经贸大学 统计与信息学院, 上海 201620)

通信作者: 陈阳, E-mail: chenyang@fudan.edu.cn

摘 要: 随着开源软件的广泛普及和迅速发展, 对开源软件项目的维护工作成为软件开发周期中的一个关键环节. 作为全球范围内代表性的开发者社区, GitHub 往往在同一领域有着大量功能相似的软件项目仓库, 导致用户在选择合适的项目仓库进行使用或进一步开发时面临挑战, 因此协助用户准确识别项目仓库的维护状态具有重要的现实意义. 然而, GitHub 平台并未提供可以直接衡量项目仓库维护状态的信息. 提出一个基于机器学习的项目仓库维护状态自动识别方法, 设计实现一套基于机器学习的分类模型 GitMT, 通过有效整合动态时间序列特征和描述性特征, 可以实现项目仓库“活跃”与“未维护”状态的准确识别. 经过一系列基于大规模真实数据的实验验证, GitMT 在项目仓库维护状态的识别任务中 AUC 值达到了 0.964. 此外, 还构建一个以软件项目仓库维护状态为中心的开源数据集——GitMT Dataset: https://doi.org/10.7910/DVN/OJ2NI3.

关键词: 维护状态识别; 开源软件项目; 机器学习; 动态时间序列特征

中图法分类号: TP311

中文引用格式: 罗诗雨, 李馨蕾, 罗俊韬, 王新, 张国锋, 陈阳. 基于机器学习的开源软件项目维护状态识别. 软件学报, 2025, 36(11): 5082–5101. http://www.jos.org.cn/1000-9825/7380.htm

英文引用格式: Luo SY, Li XL, Luo JT, Wang X, Zhang GF, Chen Y. Identification of Maintenance Status in Open-source Software Projects Based on Machine Learning. Ruan Jian Xue Bao/Journal of Software, 2025, 36(11): 5082–5101 (in Chinese). http://www.jos.org.cn/1000-9825/7380.htm

Identification of Maintenance Status in Open-source Software Projects Based on Machine Learning

LUO Shi-Yu[1,2], LI Xin-Lei[3], LUO Jun-Tao[1], WANG Xin[1,2], ZHANG Guo-Feng[3], CHEN Yang[1,2]

[1](School of Computer Science, Fudan University, Shanghai 200438, China)

[2](Shanghai Key Laboratory of Intelligent Information Processing (Fudan University), Shanghai 200438, China)

[3](School of Statistics and Information, Shanghai University of International Business and Economics, Shanghai 201620, China)

Abstract: With the widespread adoption and rapid advancement of open-source software, the maintenance of open-source software projects has become a critical phase within the software development cycle. As a globally representative developer community, GitHub hosts numerous software project repositories with similar functionalities within the same domain, creating challenges for users when selecting the appropriate project repository for use or further development. Therefore, accurate identification of project repository maintenance status holds substantial practical value. However, the GitHub platform does not provide direct metrics for assessing the maintenance status of repositories. This study proposes an automatic identification method for project repository maintenance status based on machine learning. A classification model, GitMT, has been developed and implemented to achieve this objective. By effectively integrating dynamic time series

* 基金项目: 国家自然科学基金 (62072115, 62472101); 上海市“科技创新行动计划”政府间国际科技合作项目 (22510713600); 上海市“科技创新行动计划”启明星项目 (扬帆专项) (22YF1415000); 上海市“科技创新行动计划”社会发展科技攻关项目 (22dz1204900)
收稿时间: 2024-02-17; 修改时间: 2024-05-28, 2024-07-30; 采用时间: 2024-12-18; jos 在线出版时间: 2025-06-04
CNKI 网络首发时间: 2025-06-05

features and descriptive features, the proposed model enables accurate identification of “active” and “unmaintained” repository status. Through a series of experiments conducted on large-scale real-world data, an AUC value of 0.964 is achieved in maintenance status identification tasks. In addition, this study constructs an open-source dataset centered on the maintenance status of software project repositories—GitMT Dataset: https://doi.org/10.7910/DVN/OJ2NI3.

Key words: maintenance status recognition; open-source software project; machine learning; dynamic time series feature

开源软件 (open-source software, OSS) 正在全世界范围内得到了广泛普及, 并在软件开发的各个层面扮演着越来越重要的角色[1,2]. 当今许多重要的软件产品和解决方案都是在开源软件的基础上开发的[3,4], Synopsys Black Duck 软件成分分析团队在 2023 年的报告 (https://www.synopsys.com/software-integrity.html) 中指出, 高达 96% 的商业代码库包含了开源代码, 诸如 Linux Kernel、Python 和 PyTorch 等著名项目. 这一趋势与开源社区的透明性、协作性和知识共享特性密切相关, 共同促成了一个互利的生态环境[5,6]. 在此背景之下, 开源项目的数量正急剧增加. 根据 2023 年 GitHub Octoverse 报告 (https://octoverse.github.com/), GitHub 平台的公共项目仓库 (repository) 数量已达到 2.84 亿, 相较于前一年增长了 22%.

维护工作[7,8]在开源项目的软件开发生命周期中至关重要, 确保了项目在持续变革的技术环境中保持其竞争力和可靠性. 一个项目仓库的维护活动通常包括代码的优化、文档的更新、社区用户问题的响应以及新功能的迭代添加[9]. “维护状态”是一个描述开源项目在其生命周期中的活跃程度和持续更新、修复以及改进的关键指标[10–12]. 随着开源项目的不断普及和发展, GitHub 平台上出现了大量功能类似且属于同一专业领域的软件项目仓库, 准确且自动化地评估这些项目仓库的维护状态具有重要的现实意义. 对于个人用户而言, 一旦确定了其关注点或技术需求, 他们面临的一个重要挑战是从同一领域的众多相似的项目仓库中, 准确地筛选出那些处于活跃维护状态的项目仓库作为参考; 对于团体用户而言, 在选择开源项目时需要综合考虑当前的技术需求和未来长期开发的风险, 因此偏向于选用活跃且长期维护的项目, 以确保能够对项目进行持续更新、技术支持并维护其安全, 尽力确保项目持续性和可靠性; 对于开源社区而言, 能够识别并推荐活跃的项目对于优化资源配置和增强用户体验至关重要. 然而, 在评估项目仓库的维护状态时, 用户可以查看如星标数量 (star)、关注者数量 (watch) 以及历史操作数据, 包括代码提交 (commit)、分支 (branch)、问题 (issue) 和拉取请求 (pull request) 等指标, 但 GitHub 平台并不直接提供关于项目维护状态的信息. 因此, 用户需要参考这些指标来进行综合判断, 以评估项目是否仍在积极维护以及是否值得采用. 这使得用户不仅需要对各个指标进行整合分析, 还需要深入理解它们各自的含义以及彼此之间的关系, 这为准确判断项目维护状态带来了挑战.

已有的许多研究探讨了开源项目的可维护性[11,13,14]以及各种因素导致项目仓库被放弃的原因[15,16]. 以往的工作中, 有些是基于代码提交活动信息以对项目进行分类的[17,18]. 这些研究从代码提交活动的角度为评估开源项目的可维护性提供了解决方案, 但这种方式存在一些局限性. 首先, 提交活动的频率并不能完全反映项目仓库的维护状况. 例如, 高频率提交可能仅是小的修复或者文档更新, 而非实质性功能改进. 其次, 仅依赖最近的提交活动来评估项目当前的维护状态可能忽视了项目仓库的其他操作记录和发展趋势. 因此, 仅凭提交活动来判断项目仓库的维护状态是不全面的, 且容易导致误判. 此外, 设定一个明确的无活动时间阈值以判断项目仓库的维护状态存在一定主观性, 这种阈值的确定往往没有经过比较严格的论证, 例如基于一年以上的无提交活动来做出推断[17,19]. 为了准确地识别项目仓库的维护状态, 部分工作对项目仓库维护状态进行了研究, 例如, Coelho 等人[10]研究了一种数据驱动的方法来衡量 GitHub 未维护项目仓库的维护活动水平; Alsolai 等人[20]开发了一个用于预测面向对象系统软件可维护性的模型, 并在此过程中探索了多种机器学习方法. 然而, 上述研究都忽视了时间序列数据分析在可维护性评估方面的重要性. 项目仓库的相关特征的时间序列数据对识别项目仓库的维护状态有着重要作用, 这些动态时间序列数据能够充分的反映出项目仓库的当前的维护状况以及发展趋势.

在本研究中, 提出了一种自动化的基于机器学习的分类模型, 旨在通过项目仓库相关特征的时间序列数据和描述性数据以准确识别 GitHub 上指定项目仓库的维护状态. 设计实现了 GitMT (GitHub maintenance tracker) 框架, 这一框架专注于自动判断指定项目仓库的维护状态. 图 1 是本文研究方法的整体概述. 我们从 GitHub 上获取了 10 000 个软件项目仓库的历史数据, 并进行了一系列的数据清洗工作. 本文构建的数据集包含了项目仓库的多种

特征的时间序列数据, 覆盖了长达两年的时间跨度. GitMT 框架结合了动态时间序列特征和描述性特征作为输入, 每个动态时间序列特征进一步细分为多个维度, 并以此为基础, 识别项目仓库的维护状态. 上述数据驱动的方法, 不仅揭示了活跃项目仓库与未维护项目仓库之间的显著差异, 还实现了基于这些特定特征的项目仓库维护状态自动化判别策略. 通过利用 GitHub 的大规模真实数据进行深入的案例分析, 本研究作出了以下 3 个主要贡献.

(1) 提出了一个时间序列分析模块, 该模块能够有效提升对项目仓库历史活动的捕捉和解析能力, 进而极大地提高了项目仓库维护状态识别的准确性.

(2) 设计并实现了 GitMT 框架, 这是一个深度学习框架, 能够综合利用动态时间序列数据和描述性数据来评估项目仓库的维护状态. GitMT 对项目仓库的历史活动以及现有的信息进行了有效的全面建模.

(3) 构建了一个以 GitHub 项目仓库为中心的时间序列数据集, 专门用于识别项目仓库的维护状态. 利用这个真实数据集对 GitMT 框架的性能进行了全面评估, 并将其与多种现有的方法进行了对比. 评估结果显示, GitMT 在识别准确性上具有明显的优势, 其中在识别项目仓库维护状态的任务中 AUC 达到了 0.964.

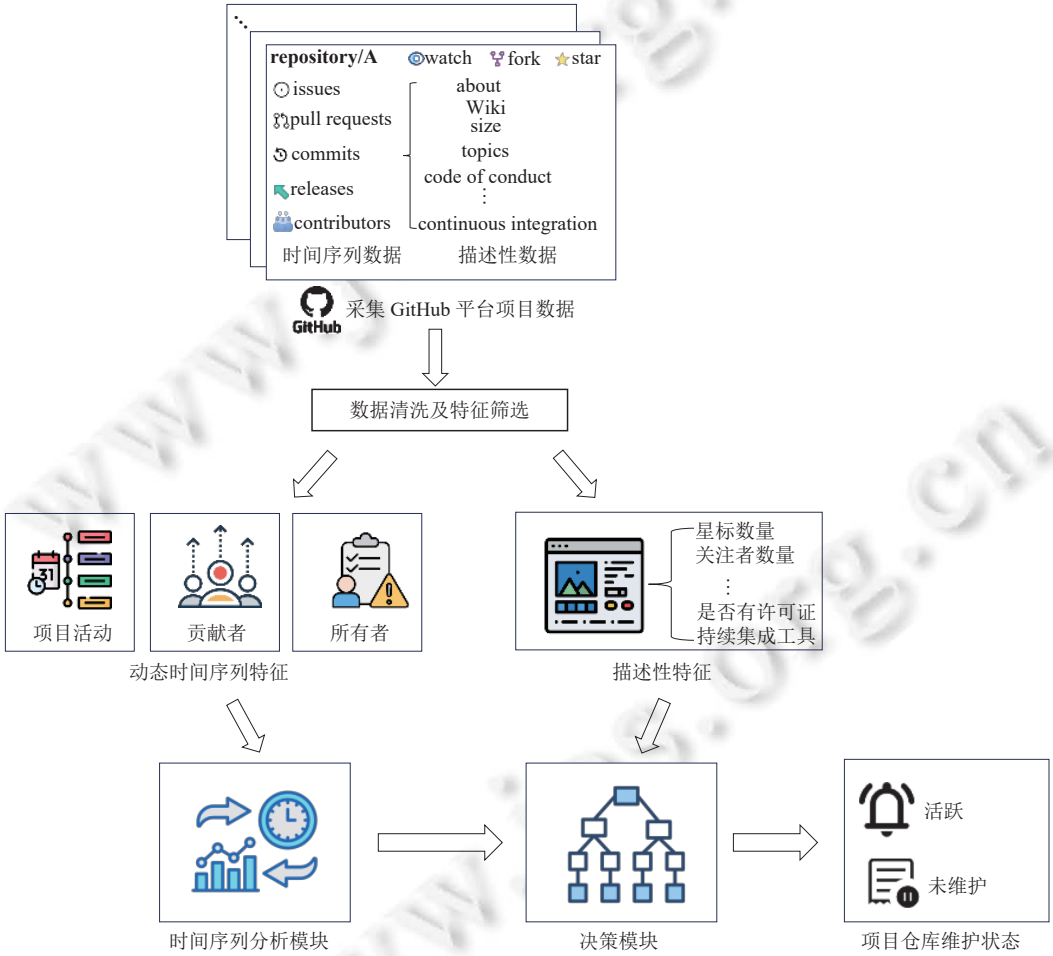


图 1 研究方法概述

1 相关工作

在本节中, 深入探讨与本研究相关的现有文献和研究成果. 第 1.1 节旨在详细回顾和分析 GitHub 平台上开源

软件的发展历程, 同时分析了开源软件研究的多个关键领域. 在第 1.2 节, 将探讨机器学习技术在 GitHub 用户行为研究中的应用. 第 1.3 节将专注于讨论开源项目可维护性方面的相关研究.

1.1 GitHub 中开源软件的发展

近年来, 开源软件研究的重要性日益凸显. GitHub 作为一个标志性的开发者社区, 极大地促进了在线软件开发的进程, 并吸引了全球超过一亿的用户参与其中. 根据过去几年 GitHub Octoverse 的报告数据 (https://octoverse.github.com/), 项目仓库的创建数量呈现出稳定增长的趋势[2]. 这不仅证明了 GitHub 的影响力持续增长, 也体现了其在开源软件领域中的核心地位. GitHub 上的开源软件项目受到了广泛的研究关注[2,5,6,21,22]. 例如, Sun 等人[2] 深入研究了 GitHub 上开源环境中的开发者协作模式, 关注了开发者在不同类型的项目文件上的协作程度. Dabbish 等人[5]在其研究中深入探讨了 GitHub 平台的透明度特性对促进大规模分布式协作和开源社区实践的重要作用. Liang 等人[6] 通过调查 455 名 OSS 贡献者, 开发了一个涵盖技术、管理和人际等多方面技能的 OSS 技能模型, 强调了 OSS 贡献者提升和分享技能的重要性. 吴欣等人[23]针对开源软件许可证选择问题, 提出了一个涵盖 10 个维度的框架, 并通过问卷调查揭示了开发者选择许可证时的主要困难. 杨波等人[22] 的研究深入分析了 GitHub 开源软件开发的关键影响因素, 例如问题解决速度和问题增加速度, 并探讨了这些因素间的相关性, 为理解影响 GitHub 项目仓库成功的要素提供了重要见解.

这些研究反映了 GitHub 上开源软件项目发展的各个方面, 包括开发者协作模式、平台透明度以及影响因素的分析, 为深入理解和优化 GitHub 上的开源软件开发提供了方向.

1.2 机器学习在 GitHub 用户行为研究中的应用

在深入分析 GitHub 这一庞大开源生态系统中用户行为时, 机器学习技术的运用已成为一个显著的趋势[18,24–30]. 通过对大量数据的学习和分析, 机器学习不仅助力于理解开源社区的动态和发展趋势, 而且在问题报告管理、参与者行为预测、企业影响力评估、软件缺陷处理、恶意账户检测等方面均展现出其强大的潜力. 例如, Eluri 等人[24] 利用随机森林等多种机器学习模型, 有效预测了新加入开源项目的贡献者是否会成为长期参与者. 王明宇等人[25] 以 GitHub 为数据源, 构建了企业网络模型, 并采用基于 XGBoost 的机器学习技术来预测企业在未来的影响力水平. 刘宝川等人[28] 在其研究中提出了一种针对软件缺陷问题的处理方法, 该方法首先利用 BERT 预训练模型从开源项目中提取关键特征, 然后应用随机森林算法计算跨开源项目问题之间的相关概率, 最终生成精准的问题推荐列表. Kallis 等人[26] 深入探究了 GitHub 上问题报告的自动化管理问题, 利用 fastText 机器学习算法对问题报告进行分类, 并自动为每个问题分配适当的标签. Gong 等人[27] 提出的 GitSec 框架使用深度学习技术, 可以有效地检测 GitHub 社区中的恶意账户. 因此, 机器学习在提高 GitHub 平台的运作效率和管理方面扮演着至关重要的角色.

这些研究展示了使用机器学习技术对 GitHub 中各方面用户行为进行研究的应用前景. 机器学习的广泛应用不仅提升了对用户行为的深入理解, 还增强了对开源生态系统的有效管理和监控能力. 通过对大量数据的高效分析和训练, 机器学习已成为研究 GitHub 用户行为的关键工具, 对于促进开源社区的健康发展和维持其活力具有重大意义.

1.3 开源项目的可维护性

近年来, 开源软件的可维护性已经成为研究者和开发者日益关注的问题[11,13,14,31]. 例如, Schnappinger 等人[13] 进行了一项大规模研究, 基于专家评估创建了一个关于软件可维护性的数据集. 软件的维护阶段在其整个生命周期中起着至关重要的作用[7,8]. 开源软件的失败不仅可能导致开源项目本身的损失, 还可能对整个社会产生广泛影响. 这主要是因为我们的软件生态系统正在日益变得复杂, 大量相互依赖的开源软件共同构成了这一错综复杂的网络[32]. 尽管开源项目数量持续增长, 但其中也存在许多开源项目缺乏持续维护, 有些甚至在创建后不久就被遗弃[16,33], 对于开源项目的可维护性研究一直是一个多维度且长期探索的研究主题.

一些研究依据提交活动对项目进行分类[17,18]. 例如, Khondhu 等人[17] 研究了如何识别被开发人员放弃的项目, 该研究通过设定一个时间段内无活动的阈值来对 SourceForge 上的休眠项目进行分类. Mens 等人[19] 提出了一个

基于多维度度量的开源项目不活跃预测模型, 如果项目在提取源代码库前的 365 天内无版本提交, 则视为不活跃. Xiao 等人[34]则通过分析若干年内每月的提交数来对项目仓库进行分类. 此外, Avelino 等人[18]通过分析系统的最后一次提交活动来识别那些经历了核心开发者离开后从非活跃状态转变为活跃状态的系统.

同时, 还有部分研究关注于开源项目的维护状态. 例如, Coelho 等人[10,35]运用随机森林算法来识别 GitHub 上的未维护项目仓库, 该研究采用机器学习方法, 利用项目的维护活动特征来识别 GitHub 上不再积极维护的项目. 该研究探讨了开源软件的可持续性问题, 并通过分析提交、分支、问题等数据特征, 构建了基于随机森林算法的模型, 以此来判断项目是否为未维护项目仓库. Alsolai 等人[20]开发了一个用于预测面向对象系统软件可维护性的模型, 并在此过程中探索了多种机器学习方法, 该研究对两类机器学习方法 (单一模型与集成模型) 在软件可维护性预测方面进行了比较. 以 QUES 数据集为基础, 研究发现, 在多种单一模型中, K-最近邻模型的性能最佳. 同时, 相较于单一模型, 集成模型中的 Bagging 方法在提高预测准确性方面表现更加突出. 在本文的比较实验中, 采用了 Bagging 模型对动态时间序列特征和描述性特征进行了训练. Tsakpinis[14]建议在识别 GitHub 上未维护的项目时纳入项目间的依赖关系, 尽管他强调了预测开源项目维护活动在工业界的重要性, 但其研究并未完成数据集准备、模型训练和实验阶段. Xiao 等人[34]探讨了早期参与因素与长期项目可持续性之间的关系, 他们使用基于项目开始 3 个月的活动数据训练 XGBoost 模型进行预测, 该研究探讨了项目早期维护特征与 GitHub 项目长期可持续性之间的联系. 研究结合了 Blumberg 性能模型和机器学习算法, 旨在预测 GitHub 项目的长期可持续性. 其模型是基于项目早期 3 个时间点的数据特征均值进行预测. 尽管该方法提供了对项目可持续性的初步判断, 但仅依赖于给定时间段内每个月的提交活动对项目进行分类, 可能无法准确地对项目仓库的状态进行分类.

虽然这些研究提供了更全面的视角, 但它们通常忽略了动态时间序列数据分析在评估项目仓库维护状态方面的重要性. 相较于上述研究, 本研究覆盖了项目仓库多个特征的两年的时间跨度, 并结合动态时间序列特征和描述性特征, 深入探索了动态的有效性. 我们不仅应用了传统的机器学习还融合了基于注意力机制的深度学习技术, 使识别更为准确和稳健. 经过在两个不同时间段的 GitHub 平台真实数据集上的验证, 本文方法在识别开源项目维护状态上展现出了显著的优越性.

2 数据集收集和特征分析

本节将首先描述我们如何从 GitHub 平台收集本研究所需的数据, 以及数据集划分为不同子集的过程. 接着, 将详细介绍动态时间序列特征和描述性特征的提取方法及相应的特征分析过程.

2.1 数据集获取

本研究的数据采集工作始于 2023 年 7 月, 目的是构建一个特征丰富且标注准确的 GitHub 项目仓库数据集. 首先, 相较于现有的以项目仓库为中心的开源数据集[35], 我们收集的数据集字段更全面, 它不仅包含了项目仓库多个特征的时间序列数据还涵盖了当前项目仓库的详细描述性数据; 其次, 本研究所构建的数据集规模更大, 能更全面地反映当前开源生态系统的实际情况; 最后, 考虑到项目仓库和用户行为的动态变化特性, 关注了数据的时效性, 确保收集到的数据具有较新的时间标签. 为此, 从 GitHub 平台上获取了星标数量排名前 10 000 的项目仓库. 星标数在 GitHub 中是衡量项目受欢迎程度的一个重要指标[36], 其作用相当于其他社交网络中的“点赞”, 反映了项目仓库的关注度和受欢迎程度. 基于收集的原始数据集, 本文实施了多项筛选措施, 以确保用于构建模型的数据集的高质量和相关性. 首先, 排除了首次提交至最后一次提交不满两年的项目, 共计 831 个. 这一措施旨在确保选定项目拥有充分的历史数据, 以便于我们的模型能够准确分析动态时间序列特征. 接下来, 通过分析项目所用的编程语言和主题标签剔除了与软件开发不直接相关的项目. 比如, 主要使用 HTML、CSS、Markdown 和 Tex 的项目被认定为非软件开发相关项目, 由此, 我们从数据集中移除了 1 382 个此类项目. 最后, 进一步基于项目主题进行了筛选, 例如排除了标记为 books、awesome-lists、awesome 和 pdf 的项目, 共计 411 个. 经过以上的筛选步骤, 最终获得了由 7 376 个项目仓库组成的研究数据集.

在本研究根据 Coelho 等人[35]的定义将数据集划分为“活跃”和“未维护”两个子集. 将距 2023 年 7 月近 3 个月

内有版本发布的项目仓库归类为活跃, 681 个项目仓库被归入此类, 其中包括一些知名项目, 如 docker/kitematic 和 microsoft/TypeScript-Vue-Starter. 关于未维护项目仓库, 本文通过对 7376 个项目仓库的 README 文件进行逐一的人工审查, 标记出了其中 307 个 GitHub 项目仓库, 这些项目仓库的 README 文件中被开发者明确标记为不再维护. 本研究涉及的人工审查工作是由 3 个具有计算机背景且具有良好英文阅读能力的志愿者使用两周时间完成的. 具体的审查方法是通过逐一阅读 7376 个项目仓库的 README 文件来做出判断. 本文在审查过程中将 README 文件中明确声明项目仓库未维护的句子作为了标注的依据, 表 1 列出了部分 README 文件中声明为未维护项目仓库的句子. 3 位志愿者对于绝大多数的标注结果是一致的. 对于占比 2% 的标注结果不一致的情况, 我们进行了丢弃. 此外, 还有一部分项目仓库已被归档 (archived). 在 GitHub 平台, 归档功能允许开发者将项目仓库设置为只读模式, 这通常意味着项目仓库所有者不再积极维护该仓库[37]. 需要说明的是, 本文工作以人工审查时的 README 文件为主, 不考虑后续的变化. 本文对 README 文件进行的人工审查以及对已归档的项目仓库识别视为判定项目仓库维护状态的高优先级标准.

表 1 项目仓库 README 文件中声明仓库不再维护的常见英文表述及其中文释义

英文表述	中文释义
Deprecation Notice	弃用通知
This project is deprecated	本项目已被弃用
This project has been discontinued	本项目已经停止
This project is no longer maintained	本项目不再维护
This project is unmaintained	本项目无人维护
This project is not maintained anymore	本项目不再被维护
The project is currently in maintenance mode, with only bug fixes and minor enhancements being considered	项目当前处于维护模式, 仅考虑进行错误修复和小幅度增强
This project is not supported	本项目不受支持
This project is not more supported	本项目不再受支持
This project is no longer supported or updated	本项目不再支持或更新
This project is no new features should be expected	不应期待本项目有新功能

虽然通过人工阅读 README 文件在某些情况下可以判断项目的维护状态, 但是如果采用 GitMT 方法则可以自动对大量项目仓库的维护状态进行自动化识别. GitMT 能够通过时间序列分析模块和决策模块, 综合动态时间序列特征和描述性特征, 实现对项目维护状态的自动化分类. GitMT 不仅能够处理简单原则难以覆盖的情况, 还能自动化地对大量项目仓库的维护状态进行准确识别.

2.2 动态时间序列特征提取与分析

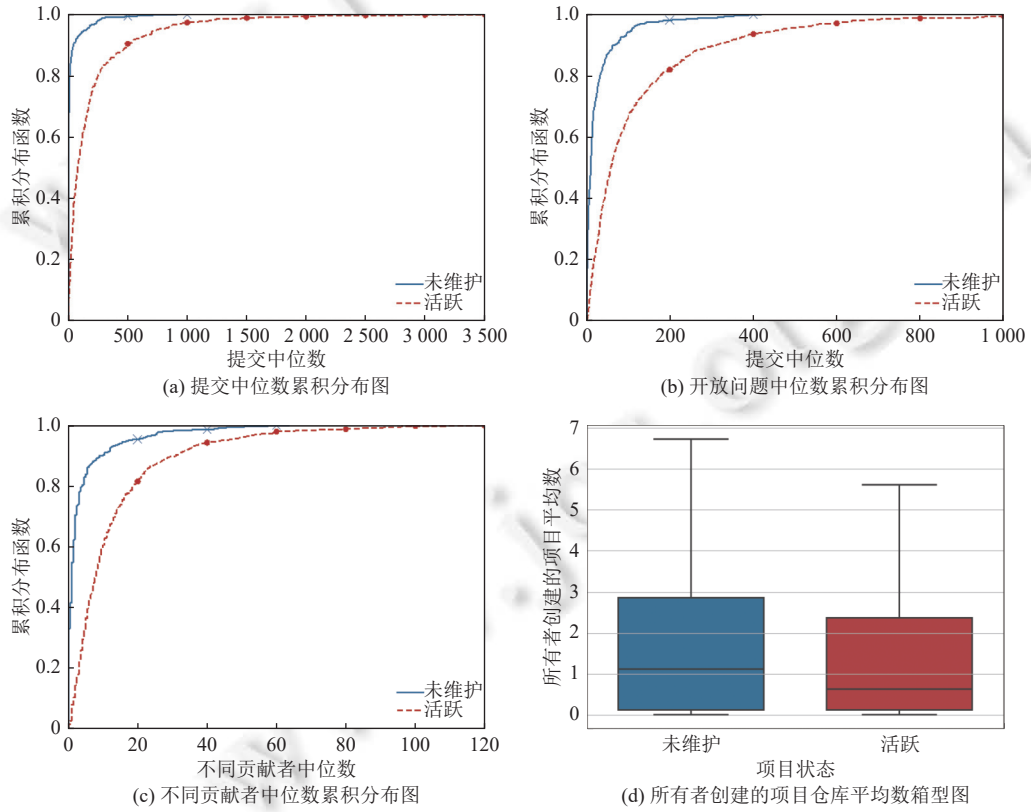
本文的动态时间序列特征细分为 3 大类: (1) 项目仓库活动特征, 涵盖了项目仓库被分叉次数、代码提交数量、拉取请求的数量以及提出和解决问题的数量; (2) 贡献者特征, 包括新加入的贡献者数量和不同贡献者的数量, 该特征设定是由于活跃的维护往往会吸引更多新开发者的参与; (3) 项目仓库所有者特征, 主要关注项目仓库所有者在 GitHub 上管理的项目仓库数量, 因为管理的项目仓库越多, 可能会分散项目仓库所有者的注意力, 进而影响到各个项目仓库的维护质量. 我们总共考量了 12 个特征, 具体内容见表 2. 这些特征通过 GitHub 的官方 API 获得. 需要说明的是, 这些特征反映的不是项目的整个历史, 而是从最后一次提交起的最近 24 个月的数据; 同时, 也以 3 个月为间隔收集每个特征的数据. 目标是构建特征的时间序列, 以便时间序列模型利用这些序列来判断项目仓库的发展趋势.

本研究通过动态时间序列特征分析揭示了关于未维护与活跃开源项目仓库之间的差异. 如图 2(a)–(d), 我们选取了 4 个代表性特征进行比较. 从图 2 中可以明显观察到, 相比于活跃项目仓库, 未维护项目仓库在提交中位数、开放问题中位数以及不同贡献者中位数这 3 个方面表现出明显的劣势, 而在项目仓库所有者所创建的项目仓库平均数上则有所超越. 具体来说, 如图 2(a) 所示, 未维护项目的提交中位数普遍低于活跃项目; 这表明未维

护项目的提交量集中在较低水平. 而活跃项目则提交较多, 分布范围更广. 图 2(b) 揭示了开放问题数量在未维护项目中往往更少, 这可能是因为这些项目仓库缺乏足够的维护, 从而导致问题长时间得不到解决. 图 2(c) 则通过不同贡献者数量的对比, 指出活跃项目仓库倾向于吸引更多的贡献者参与, 而未维护项目则在这一指标上显得较为落后. 最后, 图 2(d) 的箱型图显示, 尽管在项目仓库所有者创建的项目仓库数量上未维护项目仓库表现更好, 但它也意味着项目仓库所有者的精力可能被分散到了多个项目仓库上, 而不是集中在维护现有的项目仓库上.

表 2　动态时间序列特征

维度	特征	描述
项目仓库活动	分叉 (forks)	项目开发者创建的分叉数量
	开放的问题 (open issues)	项目开发者打开的问题数
	已关闭的问题 (closed issues)	项目开发者关闭的问题数
	开放的拉取请求 (open pull requests)	项目开发者打开的拉取请求数
	已关闭的拉取请求 (closed pull requests)	项目开发者关闭的拉取请求数
	合并的拉取请求 (merged pull requests)	项目开发者合并的拉取请求数
	提交 (commits)	开发者执行的提交数
	最大无提交天数 (max days without commits)	未提交的最大连续天数
	开发者最大贡献数 (max contributions by developer)	提交次数最多的开发人员的提交次数
贡献者	新贡献者 (new contributors)	首次进行提交操作的贡献者数
	不同贡献者 (unique contributors)	进行提交操作的不同贡献者数
所有者	所有者创建的项目仓库 (projects created by the owner)	所有者创建的项目仓库数量



(a) 提交中位数累积分布图　(b) 开放问题中位数累积分布图

(c) 不同贡献者中位数累积分布图　(d) 所有者创建的项目仓库平均数箱型图

图 2　GitHub 项目仓库的动态时间序列特征分布

此外, 我们运用了 Welch 的 t 检验方法[38], 对这两类项目仓库在 12 个特征指标上进行了深入研究. 对每个特征指标, 都计算了其 p 值. 当 p 低于 0.05 时, 我们认为在该指标上未维护项目仓库与活跃项目仓库之间存在显著差异. 在评估所选的 12 项特征时, 还计算了它们在 8 个时间点的平均值. 结果显示, 所有特征的 t 检验 p 值均低于 0.005, 这表明两类项目在这些特征上具有显著的差别. 这一结果验证了本文动态特征选择的有效性, 为项目维护状态的评估提供了有力支持.

2.3 描述性特征提取与分析

在 GitMT 框架中, 描述性特征的提取和分析对于识别项目的维护状态具有至关重要的作用. 正如表 3 所示, 本文从 GitHub 平台收集了一系列描述性特征数据, 这些数据通过 GitHub 官方 API 获取, 涵盖了项目仓库的基本信息以及一系列与项目仓库活跃度相关的指标. 这些特征为我们评估项目当前的维护状况提供了重要视角. 本文详细分析了包括关注者数量、星标数量、项目仓库大小、主题标签数量等基本特征, 以及是否启用维基、是否包含许可证、是否有拉取请求模板、使用的持续集成工具的类别, 以及项目健康度百分比等指标.

表 3 描述性特征

特征	描述
关注者数量 (follower)	项目仓库拥有者的关注者数量
星标数量 (stargazer)	项目仓库加星标的用户数
项目仓库大小 (repository size)	项目仓库中的文件总量 (以KB、MB或GB计)
主题标签数量 (topic)	项目仓库的主题标签数量
健康度百分比 (health percentage)	GitHub提供的项目仓库“健康状况”的百分比评估
主页 (homepage)	项目仓库是否包含相关联的网页URL
项目板 (project board)	项目仓库是否启用了项目板
维基 (Wiki)	项目仓库是否启用了维基功能
行为守则 (code of conduct)	项目仓库是否包含行为守则文件
问题模板 (issue template)	项目仓库是否包含创建新问题的模板
拉取请求模板 (pull request template)	项目仓库是否包含创建新拉取请求的模板
许可证 (license)	项目仓库是否包含许可证文件
贡献指南 (contributing guideline)	项目仓库是否包含贡献代码的指导原则和步骤
持续集成工具 (continuous integration tool)	项目仓库使用的持续集成工具类型

在本研究中, 本文对描述性特征也进行了深入的分析. 具体而言, 本文针对项目仓库的关注者数量、星标数量、健康度百分比以及项目仓库大小这 4 个指标, 应用了累积分布函数进行了分析. 如图 3 所示, 在星标数量、健康度百分比和项目仓库大小这 3 个特征上, 未维护项目与活跃项目之间存在显著差异. 具体来说, 未维护项目仓库在这 3 个指标上的数值普遍低于活跃项目仓库, 表明它们通常获得较少的星标、健康度评分较低, 且项目仓库的规模相对较小. 这可能反映了未维护项目仓库较低的活跃度和社区关注度. 然而, 值得注意的是, 在关注者数量这一特征上, 两类项目仓库之间并未显示出显著的差异, 这表明关注者数量可能不是判断项目维护状态的关键指标. 这可能是因为关注者数量更多地反映了项目的长期影响力或社区关注度, 而不必然与当前的维护状况直接相关.

此外, 我们还对 8 个布尔型特征为“True”时, 对这两类项目仓库进行了统计分析, 结果如图 4 所示. 该柱状图显示了在活跃与未维护项目仓库中这些特征的值为“True”的数量分布. 从图 4 中可以看到, 活跃项目仓库在所有考察的描述特征上的数量均超过未维护项目仓库. 特别是许可证、主页、持续集成特征这些特征上, 活跃项目的数量是未维护项目的两倍甚至更多. 这一显著差异不仅表明这些特征在评估项目维护状态中的重要性, 也说明着活跃项目仓库更可能积极地维护这些健康和质量指标. 例如, 持续集成的应用可能反映了更高的开发活动水平和对项目仓库质量保证的重视. 此外, 许可证的存在可能提高了项目仓库的可用性和合规性, 而主页的存在则可能提升了项目仓库的可见性和可访问性. 这些结果强调了开发者和项目仓库维护者在项目管理实践中应当关注这些关键特征, 以提高项目的整体活跃度和吸引力.

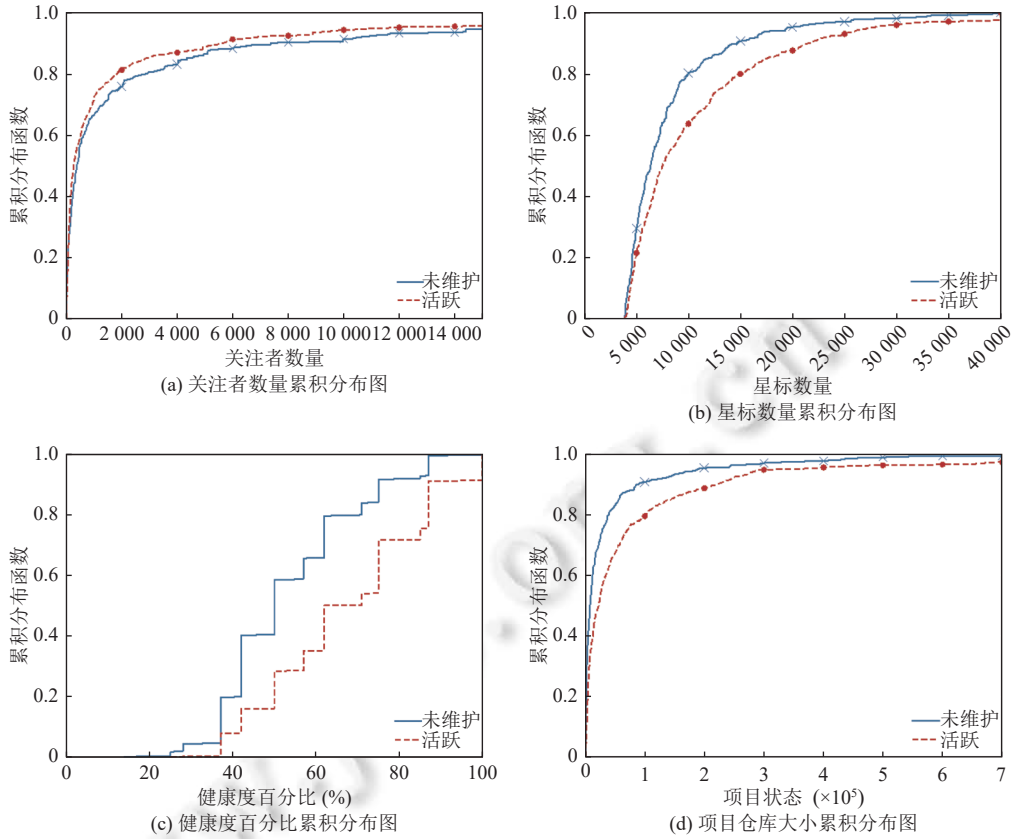


图 3 GitHub 项目仓库的描述性特征分布

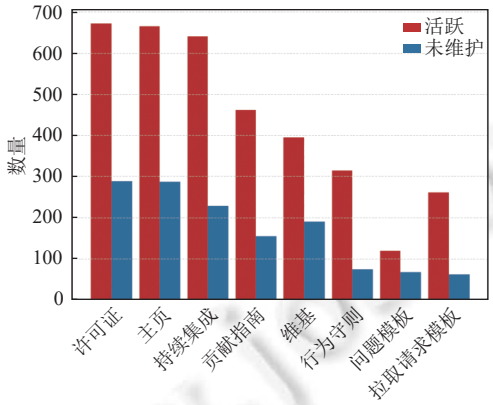


图 4 具备 8 个特征指标的项目仓库数量分布

接着, 对于这两类项目仓库使用持续集成 (continuous integration, CI) 工具的情况进行了统计分析. 由图 5 可以看出, 活跃项目仓库对 CI 工具的使用数量显著多于未维护项目仓库. 特别是 GitHub Actions, 在所有 CI 工具中使用率最高, 这一发现与先前的研究[39]相一致, 即 GitHub Actions 在 GitHub CI 工具中占据主导地位. Travis 和 CircleCI 也在活跃项目仓库中有相对较高的使用率, 而未维护项目仓库则普遍缺乏对于 CI 工具的应用. 总体来看, 使用 CI 工具的活跃项目仓库比例高, 这进一步表明了持续集成实践在现代软件开发中的重要性.

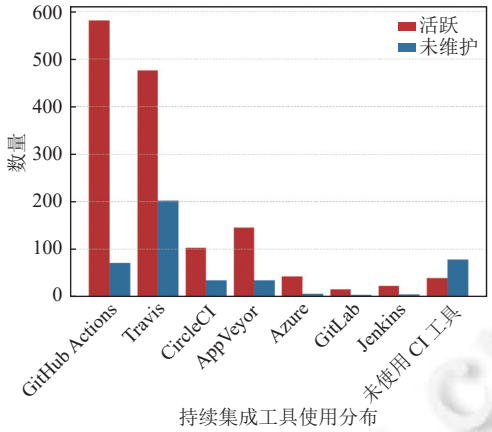


图 5 不同持续集成工具使用分布

最后, 通过 Welch 的 t 检验[38], 我们计算了各特征的 p 值, 结果显示当 p 值低于 0.05 时, 说明在这些指标上未维护项目仓库与活跃项目仓库之间存在显著的统计差异. 在所分析的 14 个特征中, 每个特征的 t 检验 p 值均小于 0.05, 这进一步证实了这些特征在区分两类项目方面的有效性. 这一结果验证了我们描述性特征选择的有效性, 为项目维护状态的评估提供了有力支持.

3 系统设计

在本节中, 将详细介绍 GitMT 框架. GitMT 通过结合动态时间序列特征和描述性特征构建了基于机器学习的分类模型来实现开源项目中的未维护项目仓库和活跃项目仓库的判别, 从而为用户提供一个有效的自动化项目仓库维护状态评估工具. 我们将介绍框架的各部分结构, 包括时间序列模块、描述性特征和决策模块.

3.1 系统概述

GitMT 是一个两阶段的识别系统, 其整体框架如后文图 6 所示. 在第 1 阶段, GitMT 运用时间序列分析模块来处理动态时间序列数据, 其中包括结合了注意力机制的双向长短时记忆网络和全连接层. GitHub 上的项目仓库活动可以被表征为一系列时间序列数据, 即在特定时间跨度内的数据点序列. 这些时间序列数据被输入到 BiLSTM 网络中, 其中每个单元都会生成一个隐藏状态, 随后这些状态通过全连接层以及 *Softmax* 函数计算输出. 第 2 阶段中, GitMT 从项目仓库的页面信息中提取描述性特征. 接着, 将第 1 阶段的输出与描述性特征结合. 最终, GitMT 通过决策模块判断项目仓库的维护状态, 区分为未维护状态或活跃状态.

3.2 时间序列分析模块

在时间序列数据分析的传统方法中, 常见的模型包括自回归模型 (auto regressive model, AR)、移动平均模型 (moving average model, MA) 及其综合扩展——自回归综合移动平均模型 (auto regressive integrated moving average model, ARIMA)[40]. 这些模型基于线性回归构建, 主要适用于短期预测. 然而, 它们在处理长期依赖关系, 特别是在长期预测的场景中表现通常有限. 为了增强预测的准确性, 研究者们提出了 ARIMA 模型的多种变体, 如季节性 ARIMA 模型[41]和引入外部解释变量的 ARIMA 模型[42]. 这些变体模型旨在更好地解释季节性波动和外部因素对时间序列的影响. 然而, 当处理开源项目中包含的复杂非线性模式动态时间序列特征时, 传统统计模型由于受限于其固有的线性假设而无法适用. 与此相反, 基于深度学习的方法在处理数据驱动和非线性建模方面的高效性能, 为时间序列数据的预测提供了新的解决思路. 例如长短期记忆网络 (long short-term memory network, LSTM)[43]和双向长短期记忆网络 (bidirectional long short-term memory network, BiLSTM)[44]. LSTM 在处理长期时间依赖问题方面表现出色, 而 BiLSTM 则通过同时分析时间序列的过去和未来信息, 为数据分析提供了更加全面和深入的视角. 根据现有研究[45], 在处理长期预测任务方面, 这些深度学习模型通常表现出优于 ARIMA 模型的能力. 尤其

是在时间序列预测问题上, 有研究[46,47]对比了各种方法, 并特别指出了 BiLSTM 在时间序列预测方面的有效性. 为了更准确地识别开源项目的维护状态, 本研究选用了 LSTM 和 BiLSTM 模型. 在接下来的部分中, 将详细讨论这些模型的内部结构和优势, 并探究它们在分析 GitHub 上的开源项目动态时间序列特征时的具体应用场景和效果.

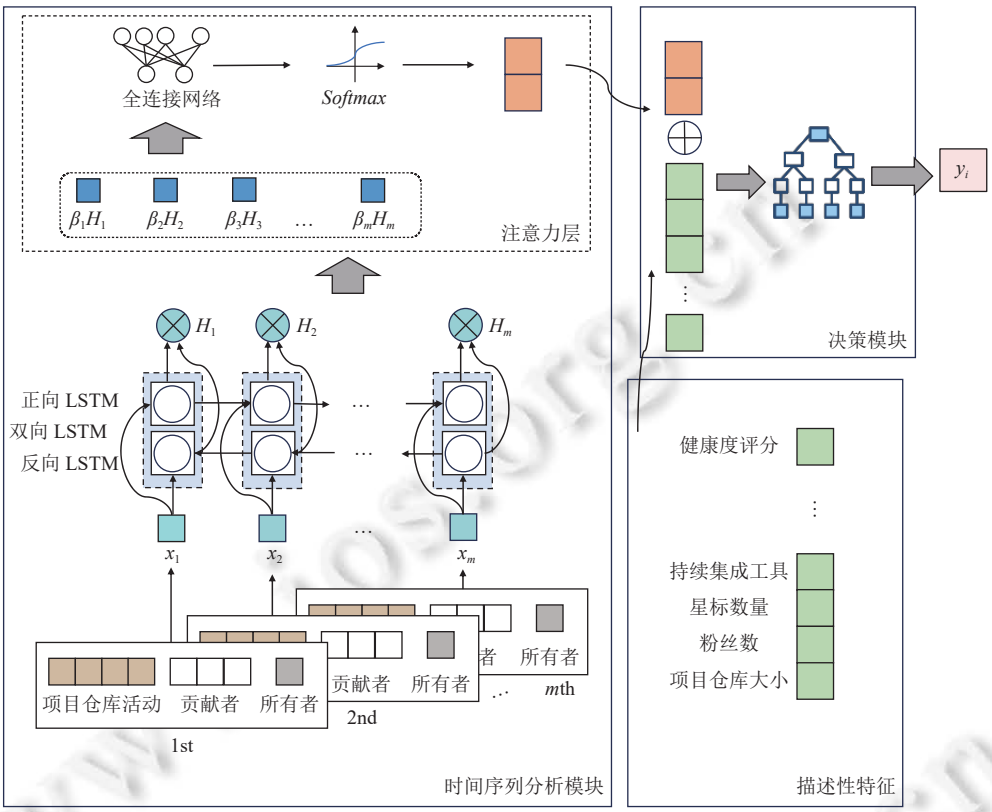


图 6 GitMT 总体框架

LSTM 是一种处理序列数据长期依赖问题的特殊循环神经网络. 它由若干循环单元组成, 每个单元在时间间隔 t 时接收输入元素 x_t 并更新长期状态 c_t. 同时, 这些单元输出隐藏状态 h_t 包含当前信息且传递到下一时间点, 保持信息流连续. 每个 LSTM 单元通过遗忘门 f_t、输入门 i_t 和输出门 o_t 来调节信息流. 这些门结构利用 Sigmoid 激活函数 σ 来控制信息传递, 输出范围为 0 (完全阻断) 到 1 (完全通过). 公式如下:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{1}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3}$$

其中, W_f、W_i、W_o 以及 b_f、b_i、b_o 分别代表权重矩阵和偏置项. 单元状态 c_t 的更新分两步进行: 首先, 生成介于 −1 和 1 之间的中间状态 $\tilde{c}_t$. 然后结合输入门和遗忘门进行状态更新, 更新的具体过程如下:

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{4}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{5}$$

每个单元输出的隐藏状态 h_t 融合了单元状态和输出门信息, 表达式如下:

$$h_t = o_t \cdot \tanh(c_t) \tag{6}$$

GitMT 模型利用交叉熵损失函数和反向传播算法进行训练, 其中损失函数结合了 *Softmax* 运算, 允许直接使

用模型输出作为输入. 损失函数公式如下:

$$L_{\text{neural}} = -\sum_{i \in U} y_i \log(\hat{y}_i) \tag{7}$$

其中, U 表示训练样本集, $\hat{y}_i$ 和 y_i 分别代表预测输出和真实标签.

在本研究中, 采用了双向长短期记忆网络模型. BiLSTM 由两个独立的 LSTM 单元组成, 分别处理输入序列的正向 (即从过去到未来) 和反向 (即从未来到过去) 信息, 以确保每个时间间隔的输出既包含了过去的信息, 又融入了未来的上下文信息. 通过实验发现 BiLSTM 相比于标准 LSTM 在处理我们的时间序列数据时展现出了更为优异的性能. 这一结果表明了在分析过程中同时考虑历史和未来的上下文信息对于提高模型预测能力的重要性. 在 BiLSTM 模型中, 每个时间点的最终输出是由正向 LSTM 和反向 LSTM 的隐藏状态结合而成, 从而为模型提供了对当前时间点前后信息的充分理解. 其中, h_t^f 和 h_t^b 分别代表在时间点 t 的正向和反向 LSTM 的隐藏状态, 表达式如下:

$$h_t^f = o_t^f \tanh\left(c_t^f\right) \tag{8}$$

$$h_t^b = o_t^b \tanh\left(c_t^b\right) \tag{9}$$

GitMT 系统中支持不同的时间序列分析算法, 通过实验的对比, 我们选择 BiLSTM 网络来分析项目的维护状态. 如图 6 所示, $x_1 - x_m$ 代表序列中的数据点, 其中 m 是时间间隔的总数. 每个时间间隔的数据包含了多个维度的特征, 涵盖了项目仓库活动、贡献者特征和项目仓库所有者这 3 个维度. BiLSTM 模型结合了正向 LSTM 和反向 LSTM, 能够有效捕捉时间序列数据的前后依赖关系. 在每一个时间间隔 t 上, 模型处理一个多维的特征向量 x_t, 同时计算出正向 h_t^f 和反向 h_t^b 的隐藏状态. 这两个状态的集成, 进而通过一个全连接层和一个 *Softmax* 层来识别项目的维护状态, 模型的输出 H_t 公式如下:

$$H_t = \left[h_t^f, h_t^b\right] \tag{10}$$

紧接着, 全连接层使用 BiLSTM 输出的 H_t 来计算其自身的输出 Z_t, 公式如下:

$$Z_t = W_h \cdot H_t + b_h \tag{11}$$

其中, W_h 为全连接层的权重矩阵, b_h 为偏置向量. 对于二分类问题, *Softmax* 层将全连接层的输出 Z_t 转换为两个类别的预测概率. 所计算出的概率值反映了项目处于未维护状态或活跃状态的可能性. 这两个概率值作为时间序列分析模块的输出, 被用作项目动态时间序列特征, 进一步输入到决策支持模块中, 输出的概率值计算公式如下:

$$y_t = \frac{e^{Z_t}}{e^{Z_{t0}} + e^{Z_{t1}}} \tag{12}$$

为了进一步提升对项目仓库维护状态预测的准确性, 我们选择了 Ma 等人[48] 提出的基于位置的注意力机制 (location-based attention, AttentionLoc) 和基于拼接的注意力机制 (concatenation-based attention, Attention-Concat). 这些机制能够自动学习不同时间间隔的特征的注意力评分, 使模型更加集中于对预测结果有重要影响的输入. 注意力机制的关键问题是衍生出一个能捕捉相关信息以辅助预测上下文. 当前的隐藏状态表示为 H_t, 先前的隐藏状态为 H_i $(1 \leqslant i < t)$, 这里 t 是输入序列的长度. 这些状态从 BiLSTM 网络中获得. 我们采用了两种方法来计算上下文向 S_t.

在基于位置的注意力机制中, 每个时间间隔都独立地决定自己的注意力分布, 而不是基于整个序列的上下文. 该机制下注意力权重的计算过程如下:

$$\alpha_{ti} = W_\alpha^{\mathrm{T}} H_i + b_\alpha \tag{13}$$

其中, $W_\alpha \in R^{2p}$ 是权重矩阵, T 代表矩阵的转置操作, $b_\alpha \in R$ 是模型参数, p 是隐藏状态的维度.

在基于拼接的注意力机制中, 模型首先将当前隐藏状态和历史状态进行拼接, 然后通过一个多层感知机将拼接后的向量映射为一个潜在向量. 对于每个时间间隔 t 和历史时间间隔 i, 首先将当前隐藏状态 H_t 和历史状态 H_i 进行拼接, 得到一个 $2p$ 维的向量 $[H_t; H_i]$. 接下来, 注意力权重 α_{ti} 的计算公式如下:

$$\alpha_{ti} = v_{\alpha}^{\mathrm{T}} \tanh(W_{\alpha} \cdot [H_t; H_i]) \tag{14}$$

其中, $W_{\alpha} \in R^{q \times 4p}$ 和 $v_{\alpha} \in R^{q}$ 是待学习的参数. q 是潜在的维度, tanh 是激活函数.

随后, 根据上述两种方式中的任意一个注意力权重计算机制, 利用 *Softmax* 函数计算得到注意力权重向量 α_t, 公式如下:

$$\alpha_t = Softmax([\alpha_{t1}, \alpha_{t2}, \ldots, \alpha_{t(t-1)}]) \tag{15}$$

最后, 上下文向量 $S_t \in R^{2p}$ 可以基于注意力权重向量和从 H_1 到 $H_{(t-1)}$ 的隐藏状态计算得出, 公式如下:

$$S_t = \sum_{i=1}^{t-1} \alpha_{ti} H_i \tag{16}$$

时间序列分析模块的输出是二维概率向量. 此向量体现出, 项目仓库分别属于两个类别的概率, 即 “项目是未维护的概率”和“项目是活跃的概率”.

3.3 描述性特征

在 GitMT 获得了第 1 阶段的动态时间序列特征后, GitMT 进一步从项目仓库的页面中提取一系列项目仓库的描述性特征. 本文利用 GitHub API 采集了一系列目标项目仓库的描述性特征, 涵盖了从项目仓库的星标数量、关注者数量到项目仓库大小、主题标签数量, 以及所采用的持续集成工具种类等多维度信息, 如表 3 所示.

为确保数据的高质量和一致性, 我们对这些描述性特征进行了一系列预处理步骤, 包括数据清洗、标准化处理及缺失值填补. 对于类别型的特征如持续集成工具使用类型, 采用了独热编码 (one-hot encoding) 技术以转换为模型可处理的数值格式, 描述性特征的集成不仅为模型提供了更加丰富的输入特征, 还让模型去学习了项目仓库的当前状态, 这对于项目仓库维护状态的预测也是至关重要的.

3.4 决策模块

在 GitMT 中, 我们通过决策模块实现最终判别, 模块输出为“0”或“1”, 分别代表项目仓库处于活跃状态或未维护状态. 该模块包含一个基于监督学习的分类器, 例如随机森林[49]、决策树[50]、XGBoost[51] 和 CatBoost[52].

我们利用对数损失函数 (log loss) 作为性能度量标准, 并据此优化分类器参数. 对数损失函数定义如下:

$$L = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) \tag{17}$$

其中, N 表示评估的项目实例的总数, y_i 为实例 i 的真实标签, 而 p_i 是模型预测实例 i 为正类的概率. 参数调优通过格点搜索 (grid search) 方法实施, 目的是寻找能最小化 L 值的参数组合. 在确定了参数后, 可以计算出对数损失 L 的值. 我们选定的参数组合旨在帮助决策模块实现最小的 L 值. 在此参数设置下, 对于特定的项目仓库 u_i, 经过训练的决策模块能够准确识别项目仓库 u_i 的维护状态.

4 实验与评估

在本节中, 将详细介绍本研究的实验设置和评估方法. 这些实验旨在验证本文提出的模型和方法在各种场景下的有效性和适用性. 第 4.1 节重点讨论实验设计和模型性能度量标准, 包括数据集的选择、实验流程和评价指标的详细说明. 接着, 在第 4.2 节, 将呈现实验结果, 并深入分析不同模型和方法在各个评估指标上的表现.

4.1 实验与度量

本研究所用的数据通过 GitHub 官方 API 获取. 按照 3:2 的比例划分数据集, 构建了训练集和测试集. 对于训练集, 我们采用了 5 折交叉验证方法.

为了构建时序分析模块, 选用了 PyTorch, 一种广泛应用的开源机器学习框架. 在神经网络参数的设置方面, 将隐藏大小设置为 128, 即每个神经网络单元中含有 128 个神经元. 学习率设置为 0.001. 对于决策层的实现, 使用了基于 Python 的机器学习库 scikit-learn[53].

我们采用 4 个经典的指标来评估分类器的性能, 包括精确率 (precision)、召回率 (recall)、$F1$ 值 ($F1$-score) 以及受试者工作特征曲线 (receiver operating characteristic curve, ROC) 下面积 (area under curve, AUC)[54]. 这些指标

在机器学习研究中广泛使用[32,36,55]. 这些性能指标的计算基于标准统计概念, 如真阳性 (true positive, TP), 假阳性 (false positive, FP), 真阴性 (true negative, TN), 假阴性 (false negative, FN).

● 精确率: 在所有被判断为未维护的项目仓库中, 实际上确实未维护的比例. 这个指标反映了模型在所有判定为未维护的项目仓库中, 准确识别出未维护项目仓库的能力.

● 召回率: 在所有实际未维护的项目仓库中, 被模型准确识别出的比例. 召回率衡量的是模型识别出所有未维护项目仓库的能力.

● *F*1 值: 精确率和召回率的调和平均值. *F*1 值是一个综合性指标, 当模型在精确率和召回率方面都有良好表现时, *F*1 值也会较高.

● AUC: AUC 衡量了 ROC 曲线下的面积, 这是一个常用的评价二分类模型性能的指标. AUC 值的范围在 0–1 之间, 值越大表示模型区分两个类别的能力越强.

4.2 实验结果

GitMT 的设计涵盖了几个关键部分: 其一为采用深度学习神经网络的时间序列分析模块; 其二为生成隐藏状态的注意力层; 其三为负责最终决策的分类器. 在评估过程中, 本研究探讨了 GitMT 的不同架构变体, 包括多种神经网络模型、注意力机制以及决策方法的应用. 此外, 研究还考察了不同特征对最终决策的影响, 并将 GitMT 与多个已有方案进行了对比分析. 通过一系列实验, 本研究旨在全面展现 GitMT 在多种情境下的适用性与鲁棒性.

4.2.1 不同神经网络模型比较分析

Ma 等人[56]提出了一种方法, 旨在通过应用 4 种不同类型的循环神经网络来检测微博中的谣言. 这些网络包括标准 RNN、LSTM、单层门控循环单元 (gated recurrent unit, GRU) 以及双层 GRU. 此外, 我们还考虑了两种 RNN 变体: Clockwork RNN (CWRNN)[57] 和 Dilated RNN (D-RNN)[58]. 为了进行性能比较, 在相同的数据集上测试了这 6 种神经网络模型.

在动态时间序列特征数据上应用这些神经网络模型后, 在图 7 中展示了各模型的性能比较结果. 从这些结果中可以观察到, 不论对于 *F*1 值还是 AUC, BiLSTM 在所有模型中均展现出了最佳性能. 具体而言, BiLSTM 相较于其他模型的主要优势在于其能够同时捕捉序列数据的前向和后向依赖关系. 普通的 RNN 只能处理单向的时间序列信息, 而 LSTM 虽然解决了长短期依赖的问题, 但仍然是单向的. BiLSTM 通过双向处理, 提高了对复杂时间序列数据的建模能力, 因此在本文实验中展现了最佳性能. RNN 和 LSTM 在 AUC 值相等, 可能是因为在我们的数据集中, 两者捕捉到主要的模式相似, 因此二者的性能表现相近. GRU 的评估结果优于其他多数模型, 可能是因为其结构简单但有效地处理了长短期依赖问题, 参数更少, 计算效率更高. 至于 CWRNN 和 D-RNN, 虽然它们在某些应用场景下表现优异, 但在本文实验中表现不如 GRU. 这可能是因为这些模型在特定数据类型上更有优势, 而我们的数据集具有随时间变化的特性. GRU 能够通过其门控机制动态调整对不同时间步的关注程度, 从而更好地适应这种变化.

4.2.2 不同注意力机制性能比较

为了进一步评估不同注意力机制在 BiLSTM 中的效果, 我们实施了 Ma 等人[48]提出的两种注意力方法: 基于位置的注意力 (AttentionLoc) 和基于串联的注意力 (AttentionConcat). 基于位置的注意力机制通过计算每个隐藏状态与当前解码状态的关联度来分配注意力. 这种方式较为直接, 仅考虑每个单独的隐藏状态信息, 因此它不会捕获当前隐藏状态与所有先前隐藏状态之间的关系. 基于拼接的注意力机制计算上下文向量时使用了多层感知机 (multilayer perceptron, MLP), 将当前解码状态与每个隐藏状态拼接起来, 并计算出一个更为复杂的注意力分布. 这种方法能够更全面地捕捉隐藏状态之间的交互信息, 从而得出更加细致的上下文向量.

根据表 4 的性能评估结果, BiLSTM 网络在处理动态时间序列特征时, 加入 AttentionConcat 方法使 GitMT 的 AUC 提升了 0.001. 实验结果显示, BiLSTM+AttentionConcat 表现更好, 主要原因是 AttentionConcat 通过 MLP 拼接和综合分析隐藏状态和解码状态, 更充分利用了时间序列数据中的复杂模式和特征, 从而在 AUC 指标上略有提升. 因此, 我们选择了 AttentionConcat 的注意力机制进行 GitMT 方法构建.

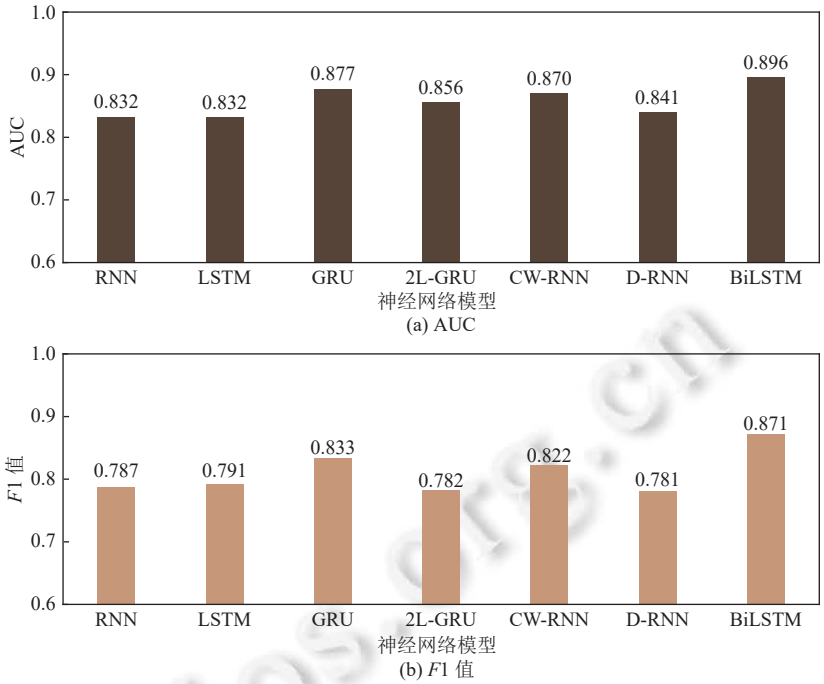


图 7 不同神经网络模型效果对比

表 4 不同注意力机制的评估

模型	精确率	召回率	*F*1值	AUC
BiLSTM	0.927	0.822	0.871	0.896
BiLSTM+AttentionLoc	0.915	**0.823**	0.866	0.895
BiLSTM+AttentionConcat	**0.942**	0.816	**0.874**	**0.897**

4.2.3 不同决策方法效果对比

在 GitMT 中, 其决策模块支持多种分类算法. 为了全面评估其分类能力, 我们在实验中采用了多种算法, 包括基于树的增强方法 (例如 CatBoost 和 XGBoost) 以及传统的机器学习算法 (例如随机森林 (random forest, RF)、支持向量机 (support vector machine, SVM) 和线性回归 (linear regression, LR)). 通过对比这些算法在各种分类指标下的性能来进行评估, 并将实验结果汇总于表 5. CatBoost 在我们的实验中的 *F*1 值和 AUC 表现最佳, 这表明其在平衡精确率和召回率方面具有显著优势. CatBoost 算法相对于其他算法表现好的原因如下: 首先, 其能够有效地处理类别特征, 无需进行额外的预处理步骤, 从而简化了数据处理过程; 其次, CatBoost 使用了独特的对称树结构和基于随机排列的增量式训练方法, 这一方法有效地防止了过拟合, 提高了模型的泛化能力; 最后, CatBoost 的训练速度较快, 尤其在处理大规模数据时表现优异. 综上所述, 我们选择 CatBoost 算法作为 GitMT 的决策模块的默认算法, 以用于后续的实验.

表 5 不同决策方法评估

模型	精确率	召回率	*F*1值	AUC
CatBoost	**0.951**	**0.959**	**0.955**	**0.964**
XGBoost	0.900	**0.959**	0.929	0.955
RF	0.942	0.934	0.938	0.954
SVM	0.934	0.901	0.917	0.943
LR	0.892	0.950	0.920	0.949

4.2.4 不同特征影响评估

在本项研究中, 采用 CatBoost 分类器处理了两种主要特征类型. 本节的实验目的是评估不同特征对 GitMT 性能的影响. 具体的实验方法包括: 对于每一组实验, 我们分别去除一个特征子集, 并观察此举对 GitMT 性能的影响. 通过分析表 6 所展示的数据, 我们得出结论, 动态时间序列特征对于提升分类准确率极为关键. 在忽略动态时间序列特征的情况下, 分类器的 *F*1 值和 AUC 均出现显著下降. 另外, 本研究还采用了逐步增加特征的实验方法. 该方法从一个基于随机猜测的基线分类器出发, 逐步加入不同类型的特征, 以此观察性能的变化. 实验结果显示, 动态时间序列特征在提升性能方面具有最显著的作用. 例如, 仅向 CatBoost 输入动态时间序列特征时, 分类器的 *F*1 值达到了 0.922, 而仅依赖描述性特征时, *F*1 值则只达到 0.579.

表 6 不同特征子集的评估

模型	精确率	召回率	*F*1值	AUC
GitMT	**0.951**	**0.959**	**0.955**	**0.964**
–动态时间序列特征	0.690	0.475	0.563	0.690
–描述性特征	0.943	0.923	0.933	0.947
随机猜测	0.273	0.319	0.294	0.306
+动态时间序列特征	0.918	0.845	0.879	0.919
+描述性特征	0.681	0.626	0.652	0.713

4.2.5 单一特征影响分析

在本研究中, 采用 SHAP (Shapley additive explanations) 方法[59]深入分析了模型的分类决策过程, 并对各特征在所有特征子集中的相对重要性进行了评估. 如表 7 所示, 我们详细阐述了各特征对分类任务的贡献程度. 分析结果表明, 在动态时间序列特征中, 项目维护状态的判定对于分类准确性的影响最大. 在描述性特征方面, 持续集成工具的使用类型和主题标签数量表现出最高的贡献度.

表 7 特征重要性

特征	重要性
项目是未维护的概率	1.023
项目是活跃的概率	0.972
持续集成工具为GitHub Actions	0.476
主题标签数量	0.180
健康度百分比	0.118
关注者数量	0.096
项目仓库大小	0.062
行为守则	0.039
持续集成工具为Travis	0.029
星标数量	0.019

4.2.6 与已有方案的对比研究

在本研究中, 对 GitMT 的性能与已有的开源项目可维护性评估分类方法进行了比较. 具体而言, 我们基于从 GitHub 收集的项目仓库活动数据集和 Coelho 等人[10]、Alsolai 等人[20]和 Xiao 等人[34]的研究进行了对比.

如表 8 所示, GitMT 模型在判定 GitHub 上未维护的项目仓库上具有最佳性能. 其精确率、召回率、*F*1 值和 AUC 均优于其他方法. 这可能归因于其对多维度的项目仓库的动态时间序列特征以及描述性特征的综合利用, 提供了对历史及现有数据的深入洞察, 从而准确判断项目仓库的维护状态. 而 Coelho 等人[10]和 Xiao 等人[34]的研究在所有评价指标上的相对较低表现可能源于其特征选择的有限性及模型处理的简化. Alsolai 等人[20]的研究采用集成模型的性能位居中等, 展现了集成方法在提高分类准确性方面的潜力. 这些对比表明, 全面的特征选取与选取恰当的模型对于提高预测性能的重要性.

表 8 与已有方案的对比

模型	精确率	召回率	*F*1值	AUC
GitMT	**0.951**	**0.959**	**0.955**	**0.964**
Coelho等人[10]	0.870	0.770	0.817	0.859
Alsolai等人[20]	0.902	0.804	0.850	0.882
Xiao等人[34]	0.867	0.796	0.830	0.870

4.2.7 模型鲁棒性的实验验证

为了验证本文模型的鲁棒性, 本研究在 Coelho 等人[35]所提供的数据集上进行了实验. 该数据集涵盖了开源项目的多维度特征, 为评估模型的稳定性和可靠性提供了测试基础.

如表 9 所示, 我们将 GitMT 模型的性能与 Coelho 等人[35]、Alsolai 等人[20]和 Xiao 等人[34]的方法进行了比较. 实验结果表明, 即使在去除描述性特征的情况下, GitMT 在 3 项关键指标上均显示出较好的竞争力. 在 *F*1 值和 AUC 方面, GitMT 均超越了另外 3 个对比模型, 这一结果表明本文模型在平衡真阳性率和假阳性率方面具有更优表现. 这些发现凸显了 GitMT 在识别项目未维护状态方面的高效性和鲁棒性. 根据表 9 的数据, GitMT 在精确率方面略低于其他模型, 可能是由于数据集中难以区分的样本或噪声数据以及模型设计和参数设置的影响. 尽管如此, GitMT 在召回率、*F*1 值和 AUC 上表现优秀, 尤其在 *F*1 值和 AUC 方面均超越了其他模型, 表明 GitMT 在综合检测效果上的优势.

表 9 在 Coelho 等人[35]数据集上的模型性能比较

模型	精确率	召回率	*F*1值	AUC
GitMT	0.817	**0.770**	**0.792**	**0.861**
Coelho等人[35]	0.807	0.741	0.773	0.847
Alsolai等人[20]	**0.830**	0.701	0.760	0.826
Xiao等人[34]	0.822	0.722	0.769	0.838

5 总结与未来工作

本研究深入探讨了在线开发者社区中的开源项目维护状况, 实现了自动化地准确识别软件项目仓库处于活跃状态或未维护状态. 本研究基于 GitHub 平台, 设计并实现了 GitMT 系统——一种基于机器学习的软件项目仓库自动化分类工具. GitMT 通过使用机器学习技术, 对项目仓库在时间序列上的细粒度活动特征以及描述性特征进行了分析, 从而实现对项目维护状态的自动化分类. 在本文构造的 GitMT 模型的数据集上大量的实验结果显示, GitMT 的 AUC 高达 0.964, 超越了已有的解决方案. 本研究已将数据集开源, 访问链接为: https://doi.org/10.7910/DVN/OJ2NI3.

未来工作将采取以下措施应对项目未及时更新 README 文件的问题: 定期更新数据集, 增加多源数据验证, 标准化人工审查流程, 并探索自动化和智能化标注方法以提高标注准确性和效率. 我们计划关注开源项目的可持续发展问题. 具体而言, 研究将专注于利用项目早期时间序列特征来预测开源项目未来的维护状况. 我们将对 GitMT 系统进行进一步的发展和改进, 使其不仅能自动识别项目仓库的维护状态, 还能基于历史及当前数据预测未来趋势. 此种扩展将使 GitMT 成为一款更全面的工具, 不仅助于理解项目仓库当前的维护状况, 还能提前预测潜在的维护状态变化情况, 为开源社区提供更深入的洞见和支持.

References:

[1] Eghbal N. Roads and bridges: The unseen labor behind our digital infrastructure. 2016. https://www.fordfoundation.org/wp-content/uploads/2016/07/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure.pdf

[2] Sun WJ, Iwuchukwu S, Bangash AA, Hindle A. An empirical study to investigate collaboration among developers in open source software (OSS). In: Proc. of the 20th Int'l Conf. on Mining Software Repositories. Melbourne: IEEE, 2023. 352–356. [doi: 10.1109/

MSR59073.2023.00054]
[3] Ebert C. Open source software in industry. IEEE Software, 2008, 25(3): 52–53. [doi: 10.1109/MS.2008.67]
[4] Godfrey MW, Tu Q. Evolution in open source software: A case study. In: Proc. of the 2000 Int'l Conf. on Software Maintenance. San Jose: IEEE, 2000. 131–142. [doi: 10.1109/ICSM.2000.883030]
[5] Dabbish LA, Stuart HC, Tsay J, Herbsleb JD. Social coding in GitHub: Transparency and collaboration in an open software repository. In: Proc. of the 2012 Computer Supported Cooperative Work. Seattle: ACM, 2012. 1277–1286. [doi: 10.1145/2145204.2145396]
[6] Liang JT, Zimmermann T, Ford D. Understanding skills for OSS communities on GitHub. In: Proc. of the 30th Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 170–182. [doi: 10.1145/3540250.3549082]
[7] Arthur LJ. Software Evolution: The Software Maintenance Challenge. Hoboken: Wiley-Interscience, 1988.
[8] Bennett KH, Rajlich VT. Software maintenance and evolution: A roadmap. In: Proc. of the 22nd Int'l Conf. on Software Engineering, Future of Software Engineering Track. Limerick: ACM, 2000. 73–87. [doi: 10.1145/336512.336534]
[9] Ghaleb M, Taghipour S. Assessing the impact of maintenance practices on asset's sustainability. Reliability Engineering & System Safety, 2022, 228: 108810. [doi: 10.1016/j.ress.2022.108810]
[10] Coelho J, Valente MT, Milen L, Silva LL. Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects. Information and Software Technology, 2020, 122: 106274. [doi: 10.1016/j.infsof.2020.106274]
[11] Nguyen E. Do all software projects die when not maintained? Analyzing developer maintenance to predict OSS usage. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 2195–2197. [doi: 10.1145/3611643.3617849]
[12] Hata H, Kula RG, Ishio T, Treude C. Same file, different changes: The potential of meta-maintenance on GitHub. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 773–784. [doi: 10.1109/ICSE43902.2021.00076]
[13] Schnappinger M, Fietzke A, Pretschner A. Defining a software maintainability dataset: Collecting, aggregating and analysing expert evaluations of software maintainability. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution. Adelaide: IEEE, 2020. 278–289. [doi: 10.1109/ICSME46990.2020.00035]
[14] Tsakpinis A. Analyzing maintenance activities of software libraries. In: Proc. of the 27th Int'l Conf. on Evaluation and Assessment in Software Engineering. Oulu: ACM, 2023. 313–318. [doi: 10.1145/3593434.3593474]
[15] Coelho J, Valente MT. Why modern open source projects fail. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 186–196. [doi: 10.1145/3106237.3106246]
[16] Ait A, Izquierdo JLC, Cabot J. An empirical study on the survival rate of GitHub projects. In: Proc. of the 19th IEEE/ACM Int'l Conf. on Mining Software Repositories. Pittsburgh: IEEE, 2022. 365–375. [doi: 10.1145/3524842.3527941]
[17] Khondhu J, Capiluppi A, Stol KJ. Is it all lost? A study of inactive open source projects. In: Proc. of the 9th IFIP WG 2.13 Int'l Conf. on Open Source Software: Quality Verification. Koper-Capodistria: Springer, 2013. 61–79. [doi: 10.1007/978-3-642-38928-3_5]
[18] Avelino G, Constantinou E, Valente MT, Serebrenik A. On the abandonment and survival of open source projects: An empirical investigation. In: Proc. of the 2019 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Porto de Galinhas: IEEE, 2019. 1–12. [doi: 10.1109/ESEM.2019.8870181]
[19] Mens T, Goeminne M, Raja U, Serebrenik A. Survivability of software projects in GNOME—A replication study. In: Proc. of the 7th Int'l Seminar Series on Advanced Techniques & Tools for Software Evolution. L'Aquila: SATToSE, 2014. 79–82.
[20] Alsolai H, Roper M, Nassar D. Predicting software maintainability in object-oriented systems using ensemble techniques. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution. Madrid: IEEE, 2018. 716–721. [doi: 10.1109/ICSME.2018.00088]
[21] Wang HB, Ji HR. Evolution model of open-source software projects in GitHub. In: Proc. of the 2nd IEEE Int'l Conf. on Software Engineering and Artificial Intelligence. Xiamen: IEEE, 2022. 135–145. [doi: 10.1109/SEAI55746.2022.9832099]
[22] Yang B, Yu Q, Zhang W, Wu J, Liu C. Influence factors correlation analysis in GitHub open source software development process. Ruan Jian Xue Bao/Journal of Software, 2017, 28(6): 1330–1342 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5222.htm [doi: 10.13328/j.cnki.jos.005222]
[23] Wu X, Wu JY, Zhou MH, Wang ZQ, Yang LY. Selection of open source license: Challenges and influencing factors. Ruan Jian Xue Bao/Journal of Software, 2022, 33(1): 1–25 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6279.htm [doi: 10.13328/j.cnki.jos.006279]
[24] Eluri VK, Mazzuchi TA, Sarkani S. Predicting long-time contributors for GitHub projects using machine learning. Information and Software Technology, 2021, 138: 106616. [doi: 10.1016/j.infsof.2021.106616]
[25] Wang MY, Gong QY, Qu JJ, Wang X. Analysis and prediction of GitHub company influence based on machine learning. Chinese Journal of Intelligent Science and Technology, 2023, 5(3): 330–342 (in Chinese with English abstract). [doi: 10.11959/j.issn.2096-6652.202327]
[26] Kallis R, Di Sorbo A, Canfora G, Panichella S. Predicting issue types on GitHub. Science of Computer Programming, 2021, 205: 102598.

[doi: 10.1016/j.scico.2020.102598]

[27] Gong QY, Liu YS, Zhang JY, Chen Y, Li Q, Xiao Y, Wang X, Hui P. Detecting malicious accounts in online developer communities using deep learning. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(10): 10633–10649. [doi: 10.1109/TKDE.2023.3237838]

[28] Liu BC, Zhang L, Liu ZW, Jiang J. Cross-project issue recommendation method for open-source software defects. Ruan Jian Xue Bao/Journal of Software, 2024, 35(5): 2340–2358 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6992.htm [doi: 10.13328/j.cnki.jos.006992]

[29] Yang C, Fan Q, Wang T, Yin G, Wang HM. Multi-feature based personal recommendation approach for open source project. Ruan Jian Xue Bao/Journal of Software, 2017, 28(6): 1357–1372 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5230.htm [doi: 10.13328/j.cnki.jos.005230]

[30] Chidambaram N, Mazrae PR. Bot detection in GitHub repositories. In: Proc. of the 19th IEEE/ACM Int'l Conf. on Mining Software Repositories. Pittsburgh: IEEE, 2022. 726–728. [doi: 10.1145/3524842.3528520]

[31] Boyalakuntla K, Nagappan M, Chimalakonda S, Munaiah N. RepoQuester: A tool towards evaluating GitHub projects. In: Proc. of the 2022 IEEE Int'l Conf. on Software Maintenance and Evolution. Limassol: IEEE, 2022. 509–513. [doi: 10.1109/ICSME55016.2022.00069]

[32] Decan A, Mens T, Grosjean P. An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Software Engineering, 2019, 24(1): 381–416. [doi: 10.1007/s10664-017-9589-y]

[33] Coelho J, Valente MT, Silva LL, Hora A. Why we engage in FLOSS: Answers from core developers. In: Proc. of the 11th Int'l Workshop on Cooperative and Human Aspects of Software Engineering. Gothenburg: ACM, 2018. 114–121. [doi: 10.1145/3195836.3195848]

[34] Xiao WX, He H, Xu WW, Zhang YX, Zhou MH. How early participation determines long-term sustained activity in GitHub projects? In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 29–41. [doi: 10.1145/3611643.3616349]

[35] Coelho J, Valente MT, Silva LL, Shihab E. Identifying unmaintained projects in GitHub. In: Proc. of the 12th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Oulu: IEEE, 2018. 15. [doi: 10.1145/3239235.3240501]

[36] Borges H, Hora A, Valente MT. Understanding the factors that impact the popularity of GitHub repositories. In: Proc. of the 2016 IEEE Int'l Conf. on Software Maintenance and Evolution. Raleigh: IEEE, 2016. 334–344. [doi: 10.1109/ICSME.2016.31]

[37] Xia XY, Zhao SY, Zhang XR, Lou ZH, Wang W, Bi FL. Understanding the archived projects on GitHub. In: Proc. of the 2023 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Taipei: IEEE, 2023. 13–24. [doi: 10.1109/SANER56733.2023.00012]

[38] Welch BL. On the comparison of several mean values: An alternative approach. Biometrika, 1951, 38(3–4): 330–336. [doi: 10.2307/2332579]

[39] Golzadeh M, Decan A, Mens T. On the rise and fall of CI services in GitHub. In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2022. 662–672. [doi: 10.1109/SANER53432.2022.00084]

[40] Newbold P. ARIMA model building and the time series analysis approach to forecasting. Journal of Forecasting, 1983, 2(1): 23–35. [doi: 10.1002/for.3980020104]

[41] Tseng FM, Tzeng GH. A fuzzy seasonal ARIMA model for forecasting. Fuzzy Sets and Systems, 2002, 126(3): 367–376. [doi: 10.1016/S0165-0114(01)00047-1]

[42] Juberias G, Yunta R, Moreno JG, Mendivil C. A new ARIMA model for hourly load forecasting. In: Proc. of the 1999 IEEE Transmission and Distribution Conf. New Orleans: IEEE, 1999. 314–319. [doi: 10.1109/TDC.1999.755371]

[43] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780. [doi: 10.1162/neco.1997.9.8.1735]

[44] Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 2005, 18(5–6): 602–610. [doi: 10.1016/j.neunet.2005.06.042]

[45] Siami-Namini S, Tavakoli N, Namin AS. A comparison of ARIMA and LSTM in forecasting time series. In: Proc. of the 17th IEEE Int'l Conf. on Machine Learning and Applications. Orlando: IEEE, 2018. 1394–1401. [doi: 10.1109/ICMLA.2018.00227]

[46] Siami-Namini S, Tavakoli N, Namin AS. The performance of LSTM and BiLSTM in forecasting time series. In: Proc. of the 2019 IEEE Int'l Conf. on Big Data. Los Angeles: IEEE, 2019. 3285–3292. [doi: 10.1109/BigData47090.2019.9005997]

[47] Woźniak M, Wieczorek M, Siłka J. BiLSTM deep neural network model for imbalanced medical data of IoT systems. Future Generation Computer Systems, 2023, 141: 489–499. [doi: 10.1016/j.future.2022.12.004]

[48] Ma FL, Chitta R, Zhou J, You QZ, Sun T, Gao J. Dipole: Diagnosis prediction in healthcare via attention-based bidirectional recurrent neural networks. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Halifax: ACM, 2017. 1903–1911. [doi: 10.1145/3097983.3098088]

[49] Breiman L. Random forests. Machine Learning, 2001, 45(1): 5–32. [doi: 10.1023/A:1010933404324]

[50] Quinlan JR. C4.5: Programs for Machine Learning. Amsterdam: Elsevier, 2014.

[51] Chen TQ, Guestrin C. XGBoost: A scalable tree boosting system. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge

Discovery and Data Mining. San Francisco: ACM, 2016. 785–794. [doi: 10.1145/2939672.2939785]
[52] Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. CatBoost: Unbiased boosting with categorical features. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems. Montreal: Curran Associates Inc., 2018. 6639–6649.
[53] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É. Scikit-learn: Machine learning in Python. The Journal of Machine Learning Research, 2011, 12: 2825–2830.
[54] Fawcett T. An introduction to ROC analysis. Pattern Recognition Letters, 2006, 27(8): 861–874. [doi: 10.1016/j.patrec.2005.10.010]
[55] Ni C, Yin X, Yang KW, Zhao DH, Xing ZC, Xia X. Distinguishing look-alike innocent and vulnerable code by subtle semantic representation learning and explanation. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 1611–1622. [doi: 10.1145/3611643.3616358]
[56] Ma J, Gao W, Mitra P, Kwon S, Jansen BJ, Wong KF, Cha M. Detecting rumors from microblogs with recurrent neural networks. In: Proc. of the 25th Int'l Joint Conf. on Artificial Intelligence. New York: AAAI Press, 2016. 3818–3824.
[57] Koutnik J, Greff K, Gomez FJ, Schmidhuber J. A clockwork RNN. In: Proc. of the 31st Int'l Conf. on Machine Learning. Beijing: JMLR.org, 2014. 1863–1871.
[58] Chang SY, Zhang Y, Han W, Yu M, Guo XX, Tan W, Cui XD, Witbrock M, Hasegawa-Johnson MA, Huang TS. Dilated recurrent neural networks. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 76–86.
[59] Lundberg SM, Lee SI. A unified approach to interpreting model predictions. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 4765–4774.

附中文参考文献:

[22] 杨波, 于茜, 张伟, 吴际, 刘超. GitHub 开源软件开发过程中影响因素的相关性分析. 软件学报, 2017, 28(6): 1330–1342. http://www.jos.org.cn/1000-9825/5222.htm [doi: 10.13328/j.cnki.jos.005222]
[23] 吴欣, 武健宇, 周明辉, 王志强, 杨丽蕴. 开源许可证的选择: 挑战和影响因素. 软件学报, 2022, 33(1): 1–25. http://www.jos.org.cn/1000-9825/6279.htm [doi: 10.13328/j.cnki.jos.006279]
[25] 王明宇, 宫庆媛, 瞿晶晶, 王新. 基于机器学习的 GitHub 企业影响力分析与预测. 智能科学与技术学报, 2023, 5(3): 330–342. [doi: 10.11959/j.issn.2096-6652.202327]
[28] 刘宝川, 张莉, 刘桢炜, 蒋竞. 开源软件缺陷的跨项目相关问题推荐方法. 软件学报, 2024, 35(5): 2340–2358. http://www.jos.org.cn/1000-9825/6992.htm [doi: 10.13328/j.cnki.jos.006992]
[29] 杨程, 范强, 王涛, 尹刚, 王怀民. 基于多维特征的开源项目个性化推荐方法. 软件学报, 2017, 28(6): 1357–1372. http://www.jos.org.cn/1000-9825/5230.htm [doi: 10.13328/j.cnki.jos.005230]

罗诗雨(1999－), 女, 硕士生, 主要研究领域为社会计算, 数据挖掘.

王新(1973－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为新一代互联网体系结构, 无线与移动网络, 数据分析, 大数据挖掘.

李馨蕾(1992－), 女, 博士, 讲师, CCF 专业会员, 主要研究领域为深度学习, 大数据分析.

张国锋(1964－), 男, 副教授, 主要研究领域为开源, 区块链, 数字"一带一路".

罗俊韬(2000－), 男, 本科生, 主要研究领域为大数据分析, 社会计算.

陈阳(1981－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为社会计算, 数据挖掘, 计算机网络.