软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007705] [CSTR: 32375.14.jos.007705]

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

交互式蒸馏驱动的开源项目领域分类方法*

康成希[1], 程 璨[1,3,4], 张 能[5], 游 兰[2], 李 兵[6], 王 伟[7]

[1](湖北大学 人工智能学院, 湖北 武汉 430062)

[2](湖北大学 计算机学院, 湖北 武汉 430062)

[3](智能感知系统与安全教育部重点实验室 (湖北大学), 湖北 武汉 430062)

[4](计算机软件新技术全国重点实验室 (南京大学), 江苏 南京 210023)

[5](华中师范大学 计算机学院, 湖北 武汉 430079)

[6](武汉大学 计算机学院, 湖北 武汉 430072)

[7](华东师范大学 数据科学与工程学院, 上海 200062)

通信作者: 程璨, E-mail: cancheng@hubu.edu.cn

摘　要: 随着开源社区的发展, 开源软件项目的种类也在逐步增多. 但开源社区中已有的数据, 例如标签 (tag)、描述等不能直接推断出项目的细分领域, 造成了开发者和用户需要花费非必要的时间去人为判断开源软件项目的类别、开源研究者研究的项目样本中可能无法有效区分细粒度类别的项目, 进而影响研究结果的针对性. 针对以上问题, 构建交互式蒸馏方法, 可以对开源项目进行自动化分类工作, 其中类别包括代码开发工具或插件、网页应用等 12 个类别. 首先, 标注了 30310 个开源软件项目. 然后, 对已有的大模型方法以及机器学习方法进行项目细粒度领域分类任务测试. 最后, 基于基准方法存在的问题, 利用大模型蒸馏关键词特征, 再通过引入大小模型协同方法优化模型分类能力. 在标注的数据集上的验证结果显示: 基于交互式蒸馏的开源项目细粒度分类方法在十折交叉后取得了 92.1% 的平均精确率、91.8% 的平均准确率和 91.8% 的平均 $F1$ 值, 相较于基准方法提升了 26.3%–32.8% 的精确率, 24.5%–39.9% 的准确率, 26.1%–40.7% 的 $F1$ 值.

关键词: 开源项目细粒度分类; 大小模型协同; 交互式蒸馏; 经验软件工程; 开源社区

中图法分类号: TP311

中文引用格式: 康成希, 程璨, 张能, 游兰, 李兵, 王伟. 交互式蒸馏驱动的开源项目领域分类方法. 软件学报. http://www.jos.org.cn/1000-9825/7705.htm

英文引用格式: Kang CX, Cheng C, Zhang N, You L, Li B, Wang W. Interactive-distillation-driven Method for Open-source Project Domain Classification. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7705.htm

Interactive-distillation-driven Method for Open-source Project Domain Classification

KANG Cheng-Xi[1], CHENG Can[1,3,4], ZHANG Neng[5], YOU Lan[2], LI Bing[6], WANG Wei[7]

[1](School of Artificial Intelligence, Hubei University, Wuhan 430062, China)

[2](School of Computer Science, Hubei University, Wuhan 430062, China)

[3](Key Laboratory of Intelligent Sensing System and Security (Hubei University), Ministry of Education, Wuhan 430062, China)

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

[5](School of Computer Science, Central China Normal University, Wuhan 430079, China)

[6](School of Computer Science, Wuhan University, Wuhan 430072, China)

[7](School of Data Science & Engineering, East China Normal University, Shanghai 200062, China)

* 基金项目: 国家自然科学基金重点项目 (62032016); 湖北省自然科学基金青年项目 (2023AFB374); 南京大学计算机软件新技术全国重点实验室开放课题 (KFKT2025B48)
收稿时间: 2025-09-26; 修改时间: 2026-02-02, 2026-05-07; 采用时间: 2026-05-20; jos 在线出版时间: 2026-08-19

Abstract: With the continuous development of open-source communities, the variety of open-source software projects has also been steadily increasing. However, existing data in these communities, such as tags and descriptions, cannot be used to directly infer the fine-grained domains of projects. As a result, developers and users need to spend unnecessary time manually identifying the categories of open-source software projects, and open-source researchers may be unable to effectively distinguish fine-grained categories in their project samples, thus affecting the specificity of research results. To address these issues, this study proposes an interactive distillation method that can automatically classify open-source projects into 12 categories, including code development tools or plugins, Web applications, and other categories. First, 30310 open-source software projects are annotated in this study. Then, existing large model methods and machine learning methods are tested on the fine-grained domain classification task. Finally, based on the problems identified in the baseline methods, this study distills keyword features from large models and further optimizes the model's classification capability by introducing a collaboration method between large and small models. Validation results on the annotated dataset show that the proposed method achieves an average *Precision* of 92.1%, an average *Accuracy* of 91.8%, and an average *F*1-*measure* of 91.8% after ten-fold cross-validation. Compared with the baseline methods, the proposed method improves *Precision* by 26.3%–32.8%, *Accuracy* by 24.5%–39.9%, and *F*1-*measure* by 26.1%–40.7%.

Key words: fine-grained classification of open-source project; collaboration between large and small model; interactive distillation; empirical software engineering; open-source community

GitHub 平台托管了全球超过 2 亿的开源项目和 12 亿的问题 (issues), 然而从海量的项目中发现与研究课题相关的问题需要花费大量的时间和精力[1]. 对于研究者而言, 如果无法找到大量与研究课题相关的问题和开源软件项目进行分析, 就会产生研究结果置信度较低, 不具有普遍性的问题. 类似地, 对于开源软件项目开发者而言, 难以快速定位某个项目的分类; 对于普通用户而言, 无法高效检索出特定类别的开源软件项目. 为了解决上述问题, 需要对开源软件项目进行细粒度的分类. 然而目前的方法都存在一定的局限: (1) 如果采用自动化的方法, 则需要准确的训练样本进行训练, 否则自动化方法会有较高的错误率. 然而目前这个领域并没有准确的训练样本, 研究表明训练数据量较大时易出现数据中毒 (data poisoning)——当 AI 类项目样本中混入 3% 的边缘数据时, 模型预测误差率将激增 11%[2]; (2) 人工标注虽然准确率较高, 但仅适用于数据量较小的情况, 在这个样本下调查的结论可能不具一般性[3], 这一现象进一步凸显了开源软件项目数据细粒度分类的重要性.

已有研究在开源项目分类方法的构建上取得了一定进展. 在数据可信度层面, Kalliamvakou 等人[4]构建了风险识别方法, 系统识别出 GitHub 数据采集中存在的 13 类风险 (如 GitHub 的 API 无法给出全部数据、大多数项目是个人项目等), 并提出 10 种方法用于筛选项目样本. 他们提出了一些解决方法, 但还有些项目选取的问题没有得到解决. 在实践层面, 自动化项目筛选方法显著改善了样本质量, 例如有研究通过随机森林算法和 J48 决策树等分类公共开发项目 (PDP) 和文档化的公共开发项目 (DPDP)[3], 为解决项目分类问题提供了参考范例. 这些成果为开源软件项目的细粒度分类研究提供了模型和动机.

然而, 尽管取得阶段性进展, 但由于缺乏开源项目细粒度的有效方法, 现有研究不能很好地适应现代软件系统的复杂和多变[5], Borges 等人[6]在研究项目发展规律时, 由于无法获取细粒度项目分类数据, 只能对项目进行人工标注. 不同领域的项目在技术架构、协作模式、用户群体等方面具有显著差异, 这直接导致其更新迭代方向呈现出较大差异[7], 比如开源社区里的嵌入式项目, 参与者数量就明显少于热门领域. 所以对细分领域数据的方法进行研究, 能帮助研究者更好地理解不同技术方向的开发模式, 看清特定技术领域的真实发展状况, 从而促进研究者在更精确项目分类下的研究.

针对上述问题, 本文采取了 3 个步骤. 首先, 为开源软件生态定义 12 项细分领域 (例如, 桌面应用、人工智能和机器学习应用等), 并对所有样本进行标注. 其次, 采用已有方法, 如大模型方法和机器学习方法, 对所有样本进行特征抽取与监督学习算法测试工作. 接着, 基于上述方法的缺陷提出新的方法, 利用机器学习方法和大模型蒸馏方法协同抽取特征, 训练监督学习算法. 最后, 找出小模型易错的样本, 与大模型进行交互式蒸馏. 本文主要贡献如下.

1) 基于 GHTorrent 2021 提供的一个含分类标注的包含 30310 个开源软件仓库的数据集, 供研究者研究. 本文将开源软件项目分为了 12 类, 分别是桌面应用、人工智能和机器学习应用、微信应用开发、企业应用、网页应用、移动应用、代码开发工具或插件、服务器应用、游戏软件、应用插件、其他、未分类.

2) 分别分析可能有效的已有的方法: 大模型方法与机器学习方法, 验证了其缺陷和不足.

3) 基于已有方法提出新的方法, 即利用机器学习方法和大模型蒸馏方法 (利用大模型抽取关键词) 协同抽取特征, 训练监督学习算法. 最后通过大小模型进行交互式蒸馏. 最终结果显示, 加入交互式蒸馏后的模型在十折交叉后取得了 92.1% 的平均精确率、91.8% 的平均准确率和 91.8% 的平均 $F1$ 值. 在 $F1$ 值上相较于大模型直接生成法提升了 40.7%, 相较于大模型蒸馏方法提升了 39.2%, 相较于机器学习法提升了 26.1%.

本文第 1 节介绍相关工作. 第 2 节介绍细粒度分类的定义和标注过程. 第 3 节介绍基准方法的测试. 第 4 节介绍本文的实验设计. 第 5 节展示并解释相关实验结果. 第 6 节对研究结果进行讨论. 第 7 节分析介绍使用场景. 第 8 节分析效度威胁. 最后总结全文.

1 相关工作

开源项目生态系统拥有海量的数据特征[8], 普通用户和开源软件 (open source software, OSS) 的研究者如何高效挖掘出所需分类的开源软件数据是一大难题. 相关研究将从以下 3 个部分展开: 第 1 部分介绍普通用户和开源软件项目研究者在缺乏有效细粒度项目分类方法的环境下遇到的难题以及困境; 第 2 部分介绍针对以上困境和难题, 开源软件项目细粒度分类研究的必要性; 第 3 部分介绍研究者们针对开源软件项目的细粒度分类所做的工作.

对于普通用户而言, 他们大多使用 GitHub 内部基于关键词匹配的搜索引擎, 但 Asyrofi 等人[9]在研究中提及该引擎通常仅依赖于文本匹配, 无法有效区分项目相同与否. 而 Masud 等人[10]也在对 GitHub 项目相似性的研究中进一步指出, GitHub 内部搜索引擎存在关键词依赖、缺乏功能性相似性搜索、多样化相似性定义的忽视以及信息整合不足等局限. 对于开源软件的研究者而言, 研究数据的选择和筛选对研究质量有较大影响[11], 在初步挖掘出数据后研究人员通常还需要更进一步地筛选出与研究主题相关的分类项目数据. 目前主流的数据获取方式有 GitHub API、第三方数据集等, 但 GitHub API 提供的实时数据未经整理[12], 有影响研究可靠性和可迁移性的隐患; 而第三方数据集虽然提供的数据稳定精确, 但通常不具备时效性[12]. 因此研究人员通常需要在时效性和稳定性之间进行权衡. 在早期的研究中, Borges 等人[6]在研究项目演化规律时, 就因数据限制不得不采用人工标注项目细分领域的方法; Franco-Bedoya 等人[13]面对海量数据也不得不采取团队多次审查复核的策略, 这些过程往往耗费巨大的人力物力; 而 Tsay 等人[14]在对 GitHub 贡献评估的研究中提出不采用人工, 而是对星标数等显性特征做初步筛选与分类的机制, 但其准确率仍与人工分类相差较远. 在其后的研究中, 自动化数据筛选与分类的算法逐渐丰富和完善, 例如, Chawla 等人[15]在 OSS 数据研究中, 获取了整个 Sourceforge 数据库, 并采用非负矩阵分解 (non-negative matrix factorization, NMF) 选择独立特征的方式获取数据.

除了上述难题外, 不同分类的开源软件的显著差别也昭示着开源软件项目细粒度分类研究的必要性. 在对 OSS 项目受欢迎程度的预测研究中, Han 等人[16]用星标的数量评估项目的受欢迎程度, 但同时他也意识到仅用星标评估的局限性; Wang 等人[17]对文献 [16] 进行优化与改进, 指出了除星标外, 项目种类对项目受欢迎程度也有较大影响; 最后, Kapur 等人[18]在研究中指出其原因, 不同类型的软件项目由于所需努力不同以及社会关注度不一样, 呈现显著差别. 通过分析以上研究, 本文有以下发现.

1) 不同类别的项目有着显著不同的成长模式[19]. 例如在 Mobile application 中需要深耕垂直场景, 并进行硬件生态整合, 而 Web application 更依赖协议标准与分布式协作. 所以在研究中对于不同领域需要分而治之.

2) 对开源项目进行细粒度分类可以提高搜索和推荐效率[20]. GitHub 中存在的 fighting41love/funNLP 等项目, 对各种大模型项目进行细粒度的分类和整合, 提供了更高效的检索与对照方式.

3) 自动化的细粒度分类方式还可以提高开发效率, 减少人工分类的时间和成本[21]. 在开发与科研的过程中, 对于相同类别项目的研究可以快速了解领域相关技术, 通过代码重用、识别替代实现等[22]对原型进行快速开发, 减少开发成本; 而对不同类别项目的研究, 也可以高效抓取不同类别项目的优势所在, 快速识别出需求所对应的框架、技术、开发流程等, 降低各方面成本.

由于开源软件的细粒度分类研究的必要性及以上发现, 目前该领域的研究已经有一定的积累和进展. 下文将

从针对特定领域的分类研究以及和领域无关 (即针对所有领域) 的项目关键词预测两方面加以介绍.

在针对特定领域的分类研究中, 杨程等人[23]通过项目流行度、技术相关度和社交关联度在开发者层面对开源软件项目进行分类. Zhang 等人[22]通过 CLAN 方法与 RepoPal 方法相结合的方式计算开源软件项目仓库之间的距离, 实现对不同类别项目的判断, 但无法对各个类别进行明确解释. Coelho 等人[24]改进实验方式, 利用机器学习方法, 对提交次数、分支数量、问题数量和拉取请求数量等特征进行训练, 完成对于是否为积极维护项目的分类. 同时 Munaiah 等人[25]利用架构、社区、持续集成、文档、历史、问题、许可证和单元测试这 8 个维度的特征, 通过随机森林算法构建出工程软件项目分类器. 不满足于单一目标分类, Zhang 等人[26]构建了 HIGITCLASS 框架, 通过构建异构信息网络 (heterogeneous information network, HIN) 来学习节点嵌入, 利用关键词丰富模块扩展单一关键词, 并通过主题建模生成伪文档, 以较高的准确度完成了对生物信息和机器学习开源项目领域的细粒度分类.

和领域无关 (即针对所有领域) 的项目关键词预测, Wang 等人[27]则对 Zhang 等人[22]的项目从另一角度进行优化, 结合了 BERT 和异构图 Transformer (heterogeneous graph Transformer, HGT) 的优势, 实现对问题文本和关键词信息的联合学习, 完成了对开源软件项目的关键词预测工作, 从而更有效地实现项目分类. 随后随着 LLM 的普及, Zhang 等人[28]通过 LLM 更进一步增强了文本关键词预测准确率.

综上所述, 开源项目的细粒度分类研究可以有效帮助普通用户与开发者更加方便快捷地找到所需细分领域的项目, 优化以往寻找及标注项目分类的时间成本, 从而实现对各领域项目的精准定位. 同时该研究也为未来对于各领域开源软件项目发展进程异同的研究奠定了基础. 就目前的研究来看, 在现有开源项目分类领域的研究中, 大多只针对单一主题, 或针对某些领域进行细粒度分类, 存在一定的局限性; 而在全领域的关键词预测领域中又仅采用关键词方式体现部分重点内容, 无法得知其确切分类, 存在着一定的模糊性.

2 细粒度分类的定义和标注过程

本文所有工作主要基于细粒度分类的类别, 下面就细粒度分类的定义和标注过程予以介绍.

2.1 各项分类的定义

本文将所有开源软件项目分为 12 类, 类名和定义见表 1. 本文定义的 12 类来自 Gittee 开源平台, 这是国内著名开源平台, 其分类十分具有参考价值, 因此本文将此分类类推到了 GitHub 平台.

表 1 开源软件项目分类及定义

分类	定义
桌面应用	设计用于在桌面或笔记本操作系统上运行的软件应用程序, 如Windows、macOS或Linux. 主要为桌面环境开发, 包括工具类、生产力工具、独立应用程序等, 不依赖于移动环境或者网页环境
人工智能和机器学习应用	主要集中在实现或利用人工智能 (artificial intelligence, AI) 或机器学习模型 (machine learning, ML) 来提供功能或服务的应用程序. 包括训练、部署或应用AI/ML模型的工具
微信应用开发	专门设计用于开发微信生态系统中的小程序、插件或其他功能的项目. 包括涉及微信API、微信小程序或微信集成的项目
企业应用	为企业设计的软件, 支持大规模组织功能, 如资源规划、客户管理或供应链管理等. 包括ERP系统、CRM软件和业务流程管理工具. 针对企业用户或公司, 而非个人
网页应用	在网页浏览器上运行的应用程序, 通过互联网或内联网访问, 通常涉及客户端-服务器架构. 通常使用HTML、CSS、JavaScript和后端框架等技术构建. 例如电子商务网站、社交媒体平台和在线工具
移动应用	专门设计用于在移动操作系统 (如iOS或Android) 上运行的应用程序. 使用移动应用框架 (如Flutter、ReAct Native) 开发. 包括本地应用、混合应用和渐进式Web应用
代码开发工具或插件	旨在帮助开发人员编写、管理或调试代码的工具、库或插件. 包括IDE插件、代码分析器、调试器和构建工具. 可以独立使用或集成到现有开发环境中
服务器应用	在服务器上运行的应用程序或服务, 用于后台处理或提供API供其他应用程序使用. 包括Web服务器、数据库管理系统和API提供程序. 侧重于服务器端功能, 而非面向用户的界面
游戏软件	涉及游戏或支持游戏软件工作流的工具的项目. 包括游戏引擎、图形渲染工具和设计工具. 可以针对桌面、移动、Web或控制台平台

表 1 开源软件项目分类及定义 (续)

分类	定义
应用插件	增强现有软件应用程序或平台功能的附加组件或扩展. 需要宿主应用程序来运行. 例如桌面软件的浏览器扩展和插件
其他	不清楚归入上述分类的项目, 或用于特殊目的的项目. 仅在项目无法合理归类为其他关键词时使用. 可能是实验性或非常独特的项目
未分类	README文件几乎没有描述, 无法进行分类的项目

2.2 项目类别标注过程

本文采用多人标注的方法以确保数据标注的规范性与可靠性. 标注团队由 3 名计算机科学专业学生构成, 其专业背景保障了领域知识的准确性. 标注流程分为 2 个阶段: 预标注阶段、独立标注阶段.

● 在预标注阶段, ① 3 位标注人员在学习完开源软件项目分类及定义后, 在所有开源软件项目 README 文档中随机抽取 100 个 README 文档作为预标注的样本; ② 3 位标注人员对抽取的 100 个 README 文档进行预标注, 得到各自的预标注结果; ③ 3 位标注人员在对预标注结果的对比中, 发现若干预标注不一致的争议项目; ④ 对于预标注不一致的争议项目, 3 位标注人员通过 3 轮讨论会, 包括标注定义解析、案例辨析与冲突解决机制建立, 一步步深入地对标注细节进行讨论; ⑤ 在讨论结束后, 3 位标注人员对预标注的结果达成标注规范共识.

在标注过程中存在标注不一致问题, 具体而言有两类不一致问题, 下面就这两个问题分别讨论其原因和本文的对策.

(1) 在对开源软件项目进行预标注时, 由于项目功能的多样性和复杂性, 往往会出现模糊不清的标注难点. 以 GitHub 上的开源项目来看, 不同项目在标注时可能会面临诸多挑战. 例如, AI 代码工具, 例如, TabbyML/Tabby 等 AI 编程工具在“人工智能与机器学习应用”和“代码开发工具或插件”两类标签之间存在分类歧义; 移动端多人战争类游戏项目如 retrowars/retrowars, 其标注也可能争议于“移动应用”与“游戏软件”之间. 针对上述标注难题, 本文提出应用领域优先分类标准. 该标准以项目的最终应用领域为核心依据, 通过对项目功能、目标和应用场景的深入分析, 准确判断项目的主要应用领域, 从而做出合理的分类. 具体而言, 在对 tabby 项目的标注问题上, 本文经过对项目文档和用户反馈的深入分析, 确定其最终应用领域为代码插件开发工具, 因此最终将其归类为代码开发工具或插件. 类似地, retrowars/retrowars 项目经过功能分析和应用场景研究后, 其最终应用领域被确定为游戏软件, 因此将其归类为游戏软件类别.

(2) 此外, 在标注过程中还需要对“其他”和“未分类”两类进行细化区分. 具体规则如下: 首先, 判断项目是否具备可以明确分类的明显特征. 如果能够通过项目文档、代码库或用户反馈等信息, 大致识别出其功能或应用领域, 但不属于前 10 大类别, 则应归类为“其他”, 例如 EbookFoundation/free-programming-books 项目作为编程教学项目则归类为其他. 反之, 如果项目文档过于简略, 无法分辨其功能特性和应用领域, 也缺乏可靠的信息来源 (如技术关键词或模块特征等), 则应将其标注为未分类. 例如 Liberate520/homeWork 等没有或几乎没有 README 的项目. 这种分类方式既保证了标注的准确性, 又兼顾了对复杂项目的包容性.

● 在独立标注阶段, ① 3 位标注人员再次学习了开源软件项目及定义和在预标注阶段达成的标注规范共识, 对所有的 30310 个开源软件项目的 README 文档独立完成分类标注, 得到 3 份初次标注结果; ② 3 位标注人员对 3 份标注结果进行对比分析, 验证其标注结果的一致性: 对 3 份初次标注结果计算 Cohen’s Kappa 系数衡量多位评判者之间的信度, 最终结果为 84.8%, 表明 3 位标注人员标注结果具有极强一致性 (almost perfect); ③ 验证完一致性后, 3 位标注人员对 3 份初次标注结果中不一致的部分进行分析, 对于两人标注一致, 一人标注不一致的开源软件项目采用以多数标注为基准的方式统一标注结果; ④ 对于 3 位标注人员标注各不相同的开源软件项目则采用会议讨论的方式进行统一, 最终得到最终标注结果.

标注过程由预标注阶段和独立标注阶段两个阶段共同构成, 得到的最终标注结果的整体标注框架如图 1 所示.

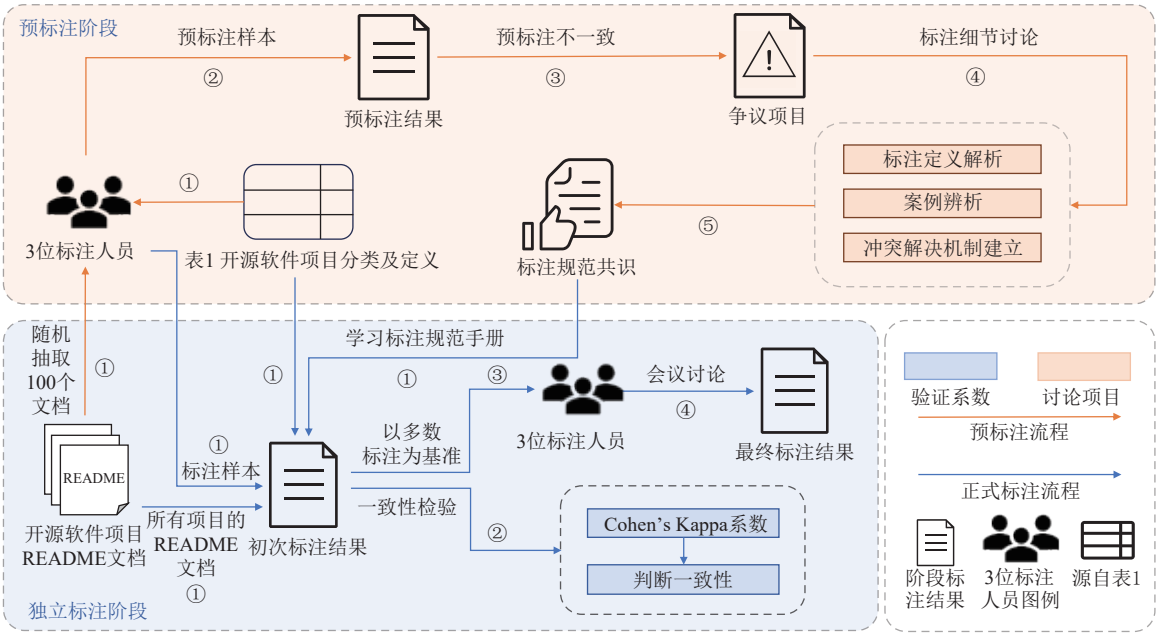

图 1 标注过程框架图

2.3 最终标注结果

最终标注结果分布如表 2 所示.

表 2 数据集中各项目分类的数量

类别	数量 (个)	类别	数量 (个)
代码开发工具或插件	8671	移动应用	759
网页应用	7817	企业应用	732
桌面应用	3476	应用插件	200
人工智能和机器学习应用	1283	微信应用开发	47
服务器应用	926	其他	3235
游戏软件	851	未分类	2313

3 基准方法的测试

开源项目细粒度领域分类是一个典型的分类任务, 因此, 目前有很多可用算法可以直接应用到这个领域, 本节将测试各基准方法的结果.

3.1 基准方法的选择

目前的基准方法中, 本文总结了已有的开源项目筛选方法, 选择 LLM 方法和机器学习方法作为基准方法. 以下是本文认为这两类方法有可能解决细粒度项目分类的理由.

● 大模型方法: 目前的大模型, 如 ChatGPT、DeepSeek 等, 在对文本的理解、概括、判断、总结等方面已有显著成果, 能细致捕捉上下文语境并抓取关键词进行联想判断. 例如, CARP 模型[29]采用 LLM 提示关键词、语气、语义关系等浅层线索, 同时基于 kNN 的示例检索来增强提示信息, 最后通过诊断推理来做最终分类决策. 类似的, 大模型在处理大规模的开源软件项目的分类中也能迅速抽取关键词并进行相关分类. 但生成式大模型在对开源软件进行细粒度分类时容易受到多方面因素的干扰导致误判以及可能出现大模型幻觉.

因此, 对于大模型方法, 本文采用大模型直接生成法以及大模型蒸馏方法. 其中大模型直接生成法是指大模型

根据 README 文档直接进行细粒度分类, 生成关键词. 大模型蒸馏法是指大模型根据 README 文档抽取关键词, 并以此作为特征训练监督学习模型. 两种方法的实验将帮助本文了解大模型在开源软件项目细粒度分类任务上的分类准确率及可靠性, 为进一步优化提供依据.

● 机器学习方法: 机器学习方法主要基于传统特征抽取, 然后再使用监督学习的方法训练模型来判断分类. 传统机器学习方法能精准提取出开源软件项目的 star 数量、fork 数量、代码库相关特征等, 为开源软件的细粒度分类提供更多更全面的信息, 在开源软件项目的细粒度分类中可能取得较好效果. 例如有研究通过随机森林算法和 J48 决策树等机器学习算法在公共开发项目 (PDP) 和文档化的公共开发项目 (DPDP) 的分类中取得了良好效果[3]. 因此, 对机器学习方法进行测试能帮助本文评估其在技术层面的优缺点, 理解其在细粒度分类任务上的贡献以及局限性.

总结: 本文将基准方法分为 3 个实验: 1) 大模型直接生成法: 将 README 文档直接输入大模型进行类别标注, 并对比人工标注计算准确率. 2) 大模型蒸馏方法: 将大模型提取的 README 关键词作为特征, 训练监督学习算法, 验证十折交叉的平均评估指标. 3) 机器学习方法: 将除大模型提取的 README 关键词外的其余特征与人工标注结果作为输入, 训练并比较多种监督学习算法, 并通过十折交叉验证计算平均评估指标, 训练多种监督学习算法进行对比, 验证十折交叉的平均评估指标.

3.2 评价指标

在对基准方法进行测试前, 本文将先确立对分类器分类结果的评估指标. 分类器的性能通常通过准确率 (*Accuracy*)、精确率 (*Precision*) 和 *F*1 值 (*F*1-*measure*) 来衡量, 这 3 个指标也反映了研究人员在对开源软件项目进行细粒度分类时的主要研究需求. 下文中, 准确率是指所有项目中正确标记的比例; 精确率是指 12 个类别中正确识别为类别 A 的项目与所有被识别为 A 项目的比例的均值; *F*1 值是两倍精确率和召回率之积与精确率和召回率之和的比值. 本文使用公式 (1)、(2) 和 (4) 分别计算各个类别的准确率、精确率和 *F*1 值. 以桌面应用举例, 分类器正确识别为桌面应用的桌面应用项目被视为真正例 (*TP*), 错误识别为桌面应用的非桌面应用被视为假正例 (*FP*), 错误标记为非桌面应用的桌面应用项目被视为假负例 (*FN*).

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{1}$$

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

$$F1\text{-}measure = \frac{2\times Precision\times Recall}{Precision+Recall} \tag{4}$$

最后将 12 个类别的精确率和 *F*1 值取均值作为评估分类器的精确率和 *F*1 值, 将整体的准确率作为评估分类器的准确率, 将这 3 个结果评价指标用于评估比较.

3.3 大模型方法

本节将针对大模型直接生成法和大模型蒸馏法两种方法进行测试, 两种方法的测试框架图如后文图 2.

3.3.1 大模型直接生成法

大模型直接生成法的数据基于 GHTorrent 2021 版本开源项目数据库, 其拥有 2021 年前持续活跃的 30 310 个仓库, 项目覆盖 GitHub 主流编程语言与高频技术领域. 在该方法中, 本文提取其英文 README 文档作为核心语料: 首先为每个项目生成全局唯一标识符 ID, 将原始 README 内容存储为以 ID 命名的标准化文本文件; 随后通过 ChatGPT API (模型版本为 gpt-4o-mini, 本文尝试过 DeepSeek、gpt-4o-mini、gpt-5 和 gpt-5-nano 等多种大语言模型, 其中分类效果最好的模型为 gpt-5, 但其反应速度较慢且相较于 gpt-4o-mini 提升较小, 所以本文认为 gpt-4o-mini 从准确率以及效率而言是最稳妥的选择) 逐文件注入分类指令, 生成结构化分类关键词及判定依据, 最终提取分类结果, 形成大模型标注与原始数据的精确映射关系. 本文采用的 ChatGPT API 提示词如提示词模板 1 所示.

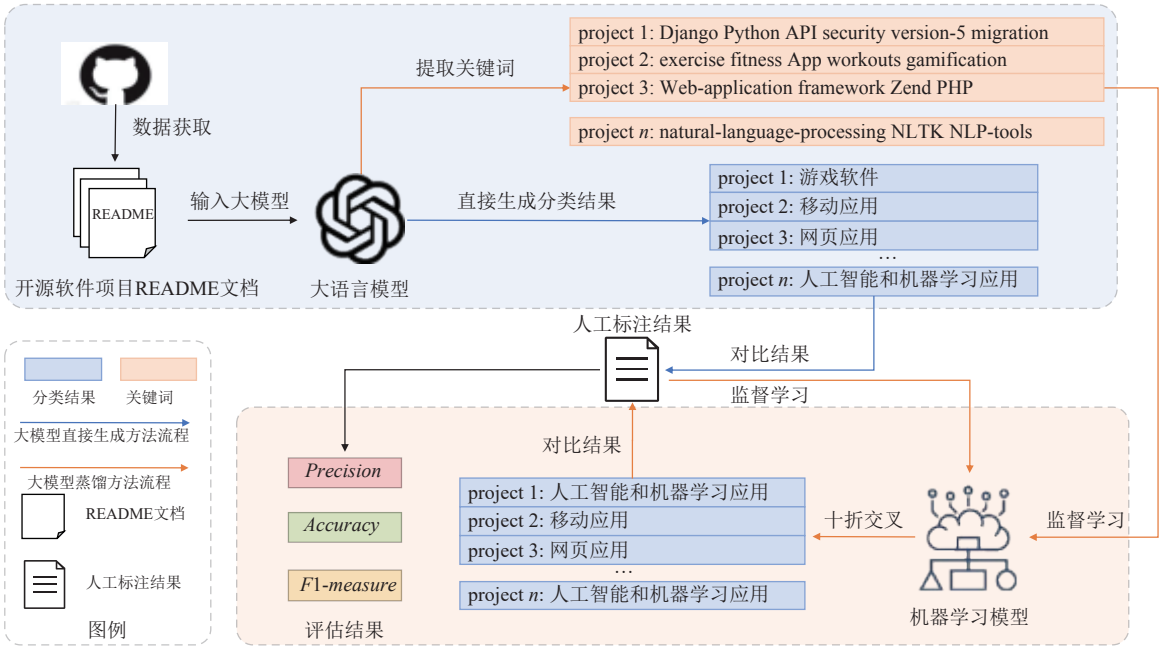

图 2 大模型方法框架图

提示词模板 1. 大模型直接生成分类.

"You are an open source project's development domain classifier. Your task is to classify the domain of software development project according to the following text of the project description and readme files. Categories include: (1) Desktop applications; (2) AI and machine learning applications; (3) WeChat application development; (4) Enterprise applications; (5) Web applications; (6) Mobile applications; (7) Code development tools or plugins; (8) Server application; (9) Game development; (10) Application plugins; (11) Others; (12) Unclassified. We believe there may be issues with the labeling. Please label this project according to its actual application scope. Please provide the result: and reasons:"

本文将 ChatGPT 模型 (gpt-4o-mini 版本) 对 30310 个项目的细粒度分类结果直接以人工标注为标准计算评估指标, 结果显示, 其平均精确率为 59.3%、准确率为 51.9%、*F*1 值为 51.1%, 表明单纯依赖大模型的细粒度分类效能存在显著局限. 从大模型对分类结果的阐释来看, 大模型在需结合技术语境与动态行为的场景中, 分类质量较低. 例如在一些项目描述中泛化的 Optimization、Management 等术语易与开发工具类混淆, 而且一些技术隐喻性表述也会触发语义歧义, 导致模型脱离整体特征进行主观联想, 如数据库工具中的 Realm 被误判为奇幻文学衍生工具. 这些结果表明, 大模型虽能胜任粗精度的术语匹配任务, 却难以支撑细粒度的技术架构与类别判断, 须通过与传统机器学习方法协同优化来弥补其缺陷.

3.3.2 大模型蒸馏方法

在大模型蒸馏方法中, 本文基于大模型对每个 README 文档进行上下文感知的关键词提取, 并据此构建关键词特征. 具体实施流程如下: 通过 ChatGPT API (gpt-4o-mini 版本) 执行关键词生成任务, 动态优化关键词生成效果. 提示词如提示词模板 2 所示.

以仓库 ID 为原子单位, 通过并行化 API 调用逐个处理所有 README 文件, 对基于每个文档生成的关键词进行正则校验, 非法输出触发重试机制 (最多 3 次), 检查完成后将规范化关键词进行储存. 最终 30310 个项目的关键词蒸馏花费了 2341.11 s, 消耗了 28696697 tokens.

提示词模板 2. 大模型抽取关键词.

“You are a keyword annotator for an open source software project, responsible for automatically generating appropriate GitHub keywords based on the project description. Based on the given project description and incorporating GitHub keyword (also known as tag) conventions, assign keywords to the project. Keywords should be concise, relevant, and accurately reflect the project’s key features. The output format should be: tag1, tag2, tag3,..., with multiple keywords separated by commas. If the text is too short to extract keywords, output “none”. Ensure the number and accuracy of keywords to better describe the nature of the project.”

本文将人工标注的 30310 个项目的细粒度分类结果作为标注结果, 将其抽取的关键词特征作为训练特征, 采用多种机器学习方法进行测试, 包括逻辑回归、随机森林、朴素贝叶斯、支持向量机、极端梯度提升和决策树算法, 并以十折交叉方式获取其最终评估指标, 其中支持向量机模型表现最佳, 其平均精确率为 61.9%、准确率为 56.1%、*F*1 值为 52.6%. 结果如表 3 所示, 基于监督学习对大模型抽取的关键词特征进行关键词学习能给开源软件项目的细粒度分类工作带来较显著提升. 但由于大模型标注准确度较低, 特征较少的缘故, 该方法虽然有所提升, 但评估指标仍然较低, 无法完全胜任开源软件项目的细粒度分类工作.

表 3　大模型蒸馏法结果对比 (%)

机器学习方法	精确率	准确率	*F*1值
逻辑回归	61.8	55.9	52.6
随机森林	58.7	42.1	32.4
朴素贝叶斯	32.7	24.1	24.3
支持向量机	61.9	56.1	52.6
极端梯度提升	59.5	53.2	48.5
决策树	41.5	36.6	35.3

注: 下划线标记的是表现最佳的两个机器学习方法

3.4 机器学习方法

3.4.1 机器学习方法的数据集构建

本文利用软件工程数据特点, 并根据细粒度分类任务将结构化关系挖掘方法和语义分析方法有机结合来构建数据集[30], 数据集包含如下特征.

(1) 结构特征

通过自然语言处理技术, 从 README 文档中提取如下两类结构特征.

1) 文本结构特征: 包括文本长度、代码块数量、超链接数量、章节数量等基础属性.

2) 内容规范性特征: 采用正则表达式匹配 5 个关键内容模块的存在性, 包括 API 说明、文档指南 (How to use/Usage)、安装配置 (Setup/Installation/Build)、协作规范 (Contributing)、联系信息 (Contact/Author/Report bug) 等[17].

(2) 语义特征

基于 CodeBERT 模型[31]获取 README 文档的语义嵌入特征, 将文档内容映射为 768 维上下文敏感向量, 捕捉技术文档特有的语义模式.

(3) 基本特征

从仓库基础属性和 README 文档中提取如下 3 类基本特征.

1) 项目流行度指标: 包含 Watcher 数量、Star 数量、Fork 数量[32]、社区成员规模[33]等社区关注度量化参数.

2) 技术生态指标: 涵盖编程语言分布 (按字节占比统计) 以体现不同程序语言使用比例的差异与共性. 不同的程序语言社区在以下 5 个方面显现出明显差异[34]: ① 依赖配置模式; ② 软件库版本的约束规则; ③ 依赖资源冲突下的覆盖规则; ④ 软件中央仓库的更新与维护特性; ⑤ 依赖缺陷的表现形式.

3) 通过 TF-IDF 加权提取的 README 关键词特征[35]. 在抽取出 TF-IDF 属性之前先将 README 中的停止词去掉, 然后对描述中的词进行主干化 (使英文中同一个词的不同形式还原到同一个主干形式) 以降低属性的维度、提高分类的效果[36].

将上述结构特征、语义特征与基本特征拼接, 结合第 2 节所述开源项目标注结果, 构建最终的监督学习数据集. 所有数据集汇总见表 4.

表 4 结构特征、语义特征及基本特征描述

类型	特征	描述
结构特征	README长度	README文档的字符数
	代码块数量	README文档中代码块的数量
	超链接数量	README文档中超链接的数量
	章节数量	README文档中章节的数量
	有无API说明	README文档中是否有API说明
	有无文档指南	README文档中是否有文档指南
	有无安装配置	README文档中是否有安装配置
	有无协作规范	README文档中是否有协作规范
	有无联系信息	README文档中是否有联系信息
语义特征	CodeBERT获取的语义嵌入特征	使用CodeBERT获取README文档的语义嵌入特征, 即768维上下文敏感向量
基本特征	Watcher数量	项目中Watcher的数量
	Star数量	项目中Star的数量
	Fork数量	项目中Fork的数量
	社区成员数量	项目中社区成员的数量
	编程语言及比例	代码库中所使用的所有编程语言的种类以及对应占总代码量的比例
	TF-IDF提取的README关键词	通过TF-IDF模型抽取README当中的关键词

3.4.2 实验设计

本文将该数据集作为输入, 用于训练包括逻辑回归、随机森林等 6 种监督分类算法, 以探究不同算法对开源项目分类预测的能力差异. 所用的监督分类算法如下.

1) 逻辑回归 (logistic regression, LR): 作为一种经典的广义线性分类模型, 逻辑回归通过引入 Sigmoid 函数将线性组合的特征值映射到 [0, 1] 概率空间, 适用于二元分类问题. 该模型的参数优化过程可视为对特征权重的学习过程, 通过特征加权机制有效区分正向特征与负向特征对分类结果的贡献度.

2) 朴素贝叶斯 (naive Bayes, NB): 该概率分类器基于贝叶斯定理与特征条件独立假设构建. 尽管模型结构简单, 但在实际应用中常展现出与复杂模型相媲美的分类性能. 其参数估计过程仅需计算先验概率与条件概率, 具有训练效率高、实现复杂度低的特性[37].

3) 随机森林 (random forest, RF): 作为集成学习的代表性方法, 该算法通过 Bootstrap 抽样构建多棵决策树, 并引入特征随机选择机制增强基学习器的多样性. 相较于传统决策树, 随机森林通过群体决策机制抑制单棵树的过拟合倾向, 同时提升模型的泛化能力[38]. 其在处理高维特征时也展现出更优的噪声容忍度与分类稳定性, 故将其选为候选方法.

4) 支持向量机 (support vector machine, SVM): 支持向量机是一种有监督的机器学习模型, 通常用于通过找到最大化区分不同类别样本的最优超平面来进行分类和回归分析. 选择这种方法的原因是它相对简单且灵活, 能够有效解决细粒度分类问题[39].

5) 极端梯度提升 (extreme gradient boosting, XGBoost): 该集成算法通过加法模型迭代优化多棵回归树, 其优势体现在: ① 目标函数引入 L1/L2 正则化项约束叶节点权重; ② 采用加权分位数策略加速特征分裂点搜索; ③ 内置缺失值自动处理机制[40]. 故本文采用该算法旨在应对高维稀疏特征下的复杂分类挑战.

6) 决策树 (decision tree, DT): 作为基础分类模型, 其采用贪心策略递归选择最优特征 (通过信息增益或基尼指

数) 进行节点分裂. 虽然单棵决策树易受训练数据扰动而产生过拟合, 但其白盒特性为特征重要性分析提供了直观解释. 本文将其作为候选模型, 通过对比集成方法 (如 RF、XGBoost) 验证性能提升效果.

本文考虑到各个类别数据集的样本数量相差较大, 样本数量不均衡可能对评估结果造成影响, 如表 2 所示. 故本文在对比分析各机器学习模型效能后, 对最佳的机器学习模型进一步进行了以下平衡类别样本的训练策略测试.

1) 过采样方法: 通过加权采样机制提高小样本类别在训练过程中的出现概率, 采样权重按类别样本数的平方根倒数设置, 即对第 c 类赋予 $w = 1/\sqrt{n_c}$ 的采样权重, 并保持每折训练样本总量不变.

2) 损失加权方法: 通过在分类损失函数中引入类别权重, 对样本数量较少类别的误分类施加更高惩罚, 从而平衡不同类别样本差距. 例如类别 c 的惩罚权重 w_c 通过公式 (5) 计算, 其中 n_{samples} 为当前训练折中的总样本数, n_{classes} 为类别总数, n_c 为第 c 类在该训练折中的样本数.

$$w_c = \frac{n_{\text{samples}}}{n_{\text{classes}} \times n_c} \tag{5}$$

3) 无额外加权采样的基线方法: 不采用任何类别样本平衡处理方法的对照组.

实验具体流程图如图 3 所示.

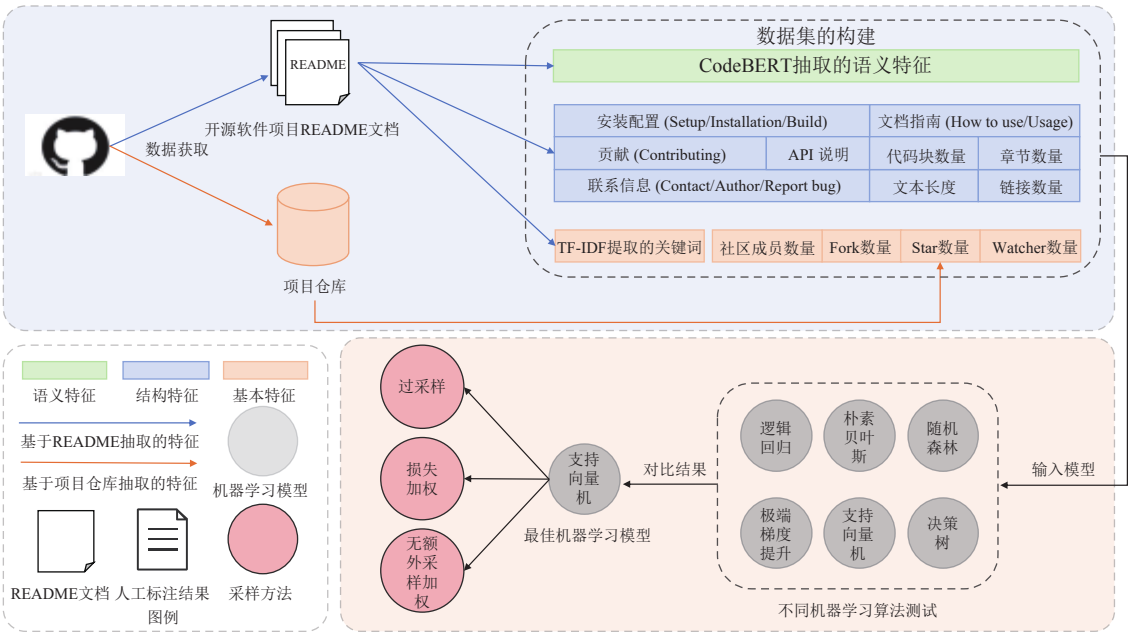

图 3 基于机器学习方法框架图

3.4.3 实验结果

各监督算法在数据集上进行十折交叉实验后的评估结果如图 4 所示.

在各监督学习算法中, 支持向量机和逻辑回归算法的表现效果显著高于其他算法, 而支持向量机又略优于逻辑回归算法. 支持向量机最终平均精确率为 65.8%、准确率为 67.3%、$F1$ 值为 65.7%; 逻辑回归最终平均精确率为 65.6%、准确率为 67.1%、$F1$ 值为 65.9%. 在对特征的分析中, 本文发现使用 TF-IDF 提取的关键词并不能很好地反映项目内容. 所以该方法虽然比单纯基于大模型进行开源软件细粒度分类的效果更优, 但仅基于传统机器学习方法抽取的特征仍存在局限, 不能很好地区分样本, 适配开源软件细粒度分类任务.

对最佳的支持向量机算法进行的关于平衡样本类别数量的测试结果如图 5 所示. 结果显示, 无论采用过采样还是损失加权, 整体评估指标均较无额外加权采样有不同程度下降.

(1) 对于过采样而言, 由于少数类样本在数据集中占比极低, 重采样本质上是对极少量样本的重复抽取, 例如微信应用开发在每折训练集中平均仅有 2 条样本, 重复采样会在训练过程中反复使用高度相似甚至可能含噪声的

样本, 使模型判断向小类别偏移并产生过拟合, 进而压缩多数类的学习与判别, 导致整体评估指标的下降.

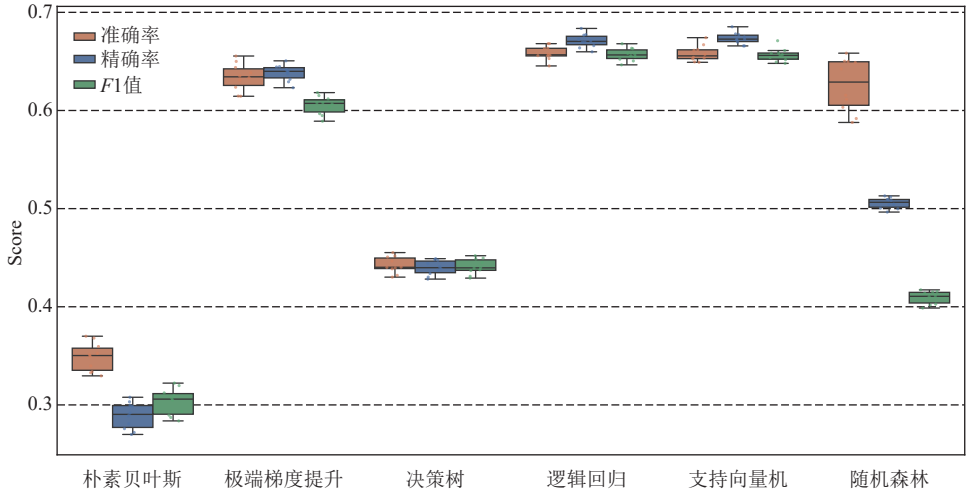

图 4 各监督算法的实验结果

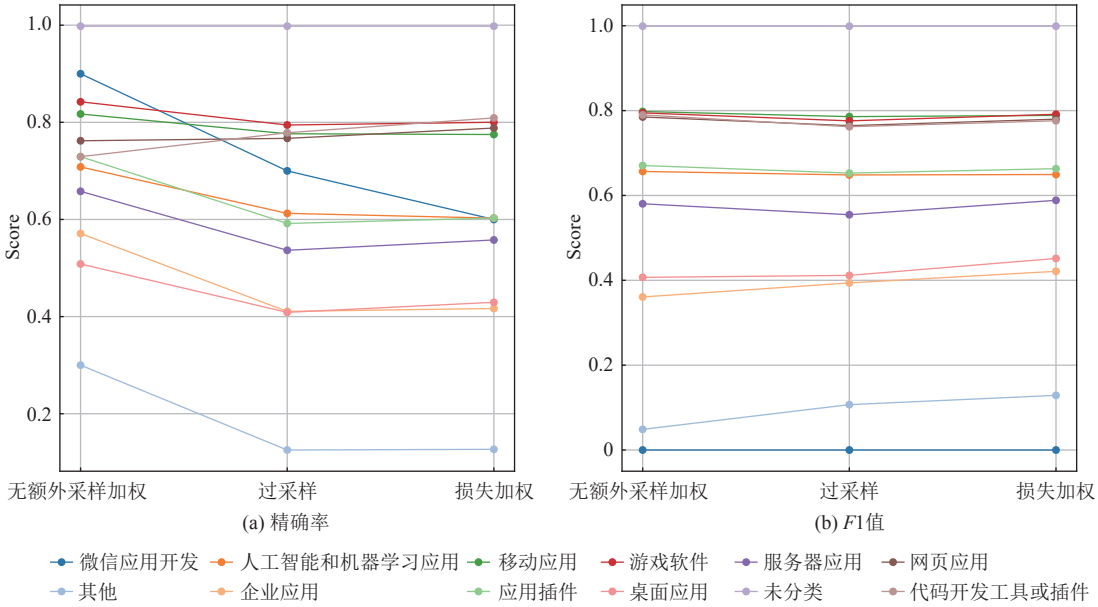

图 5 3 组样本数量平衡策略对比图

(2) 对于损失加权而言, 虽然其不改变数据分布, 但在大小类别不均衡程度较大时, 权重会为样本量极小的类别分配较大的惩罚系数, 使模型在优化过程中对小类别错误高度敏感, 这种放大的梯度信号可能导致支持向量机的决策超平面向少数类方向显著偏移, 而且当少数类样本本身数量有限且类内分布不充分时, 提高其损失权重并不能提供新的判别信息, 反而可能强化噪声特征对模型的影响, 最终表现为整体性能下降.

综合来看, 不采用平衡样本数量策略更优, 因此后续实验将不采用平衡样本数量策略.

3.5 两种方法的缺陷

在对基准方法的测试中, 本文发现了以下问题.

在大模型方法中, 大模型直接生成方法在开源项目细粒度分类中存在语境理解和动态行为推理能力不足的缺

陷. 尽管能够快速提取关键词并进行提取, 但其对技术文档中隐喻性表述和常用术语的深层语义解析能力较弱, 容易因术语的歧义导致系统误判. 这种局限性源于生成式模型对技术文档专业性较高、用词较为小众的表述并不熟悉, 难以准确捕捉领域专有概念的真正含义, 单纯依赖生成式方法会导致分类准确率显著降低. 大模型蒸馏方法在对关键词的提取中表现较为出色, 能有效提取合适恰当的关键词, 但受制于 README 中信息的局限, 无法得到更多有效的项目相关信息进行学习判断. 最终的实验结果也显示其无法单独胜任开源项目的细粒度分类工作.

机器学习方法则受制于技术语义表征深度不足的瓶颈. 基于 TF-IDF 等统计方法提取的关键词特征过度依赖高频通用词汇, 无法有效捕捉低频但具有领域区分性的技术术语, 导致关键特征信息丢失, 例如在 gchef/apt-cookbook 项目中, TF-IDF 抽取了 recipe、cacher、sources 等高频但低信息量的词语, 而忽略了 App、Ubuntu 这种关键性但仅出现过一次的领域区分性技术术语. 这样的特征数据让监督算法中不同分类之间的边界变得模糊不清, 也使传统机器学习方法的推广能力受限.

4 实验设计

本文通过对基准方法的测试发现, 大模型知识库较早但较为全面, 擅长拆分任务, 如解析 README 中的关键词, 而机器学习小模型可以针对性学习新的数据集来提高分类效果, 同时大模型本身也有一定的分类能力. 受敏捷开发中新老程序员协同决策机制启发——类比软件工程领域经典的结对编程 (pair programming)[41], 其中有经验的程序员主要负责拆分任务重点、当新手程序员感到困惑时进行兜底. 而新手程序员负责学习最新领域知识、进行标注和困惑时向有经验的程序员请教. 具体来说, 在此框架下, 1) 模型训练: 现有的生成式大模型作为有经验的程序员, 其经验丰富且善于拆分任务 (关键词总结), 而机器学习小模型有精力学习新知识. 所以有经验的程序员可以帮助将 README 文件分类任务拆分解析成若干关键词 (图 6 中的任务重点), 将其与基础知识 (基本特征、结构特征和语义特征) 和标注结果进行特征融合, 并交给新手程序员 (本文训练的模型) 进行学习; 2) 模型分类: 将新的数据集交给有经验的程序员拆分解析成若干关键词, 将其与基础知识进行特征融合给新手程序员 (本文训练的模型) 进行判断; 3) 不确定样本交互: 基于监督学习模型 (新手程序员) 的可解释性权重分析, 将不确定样本反馈给生成式大模型 (有经验的程序员) 重新评估分类, 并生成分类依据, 最终形成交互式协同决策. 该方案兼具传统机器学习方法的稳定性与大模型的语境理解能力, 为开源软件分类任务提供新的方法论框架. 映射关系见表 5, 示意图见图 6.

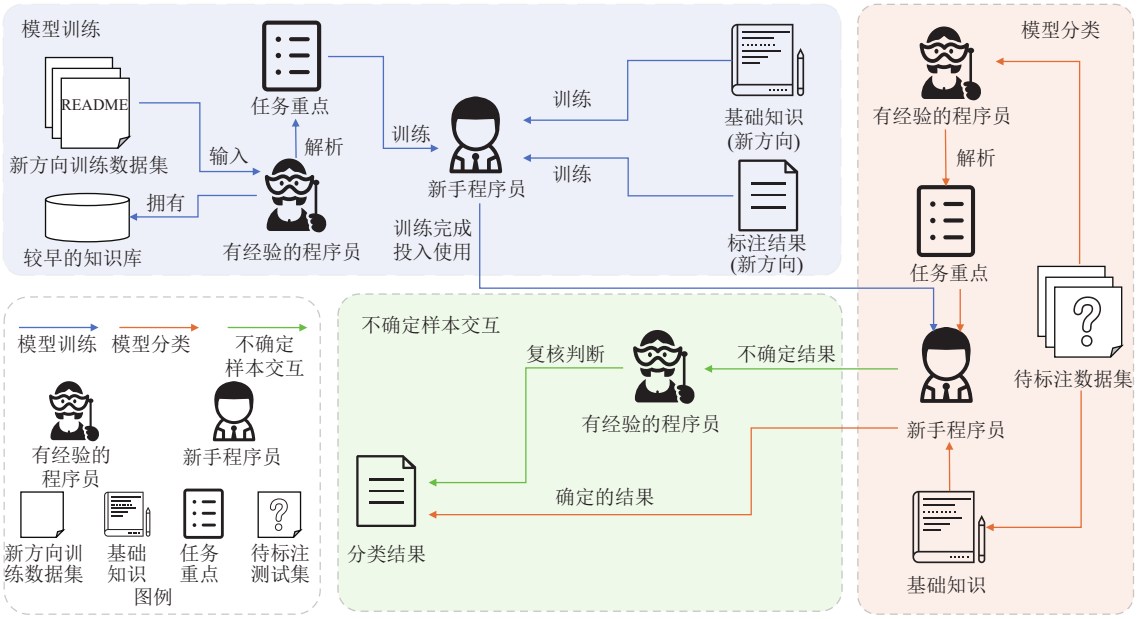

图 6 实验设计类比示意图

表 5 名词映射关系

类比名词	对应名词
新手程序员	机器学习模型
有经验的程序员	大语言模型
任务重点	大语言模型抽取的README关键词
基础知识	基本特征、结构特征和语义特征

4.1 交互式蒸馏方法的总结

基于上述内容, 本文提出的交互式蒸馏方法可总结为以下 4 个步骤.

(1) 选取 1 个 GitHub 项目仓库, 通过其 README 和代码数据计算其结构特征、基本特征和语义特征.

(2) 用带标签数据集训练机器学习模型.

(3) 将待预测项目输入模型, 得到各分类经 Softmax 处理后的概率. 若概率排名前两名的分类的概率之差大于等于阈值, 则直接将机器学习模型的分类结果作为最终结果.

(4) 若概率排名前两名的分类的概率之差小于阈值, 则将该项目仓库的 README 和机器学习模型给出的各分类概率交由大模型进行二次判断, 以大模型的判断作为最终分类.

4.2 研究问题

本文提出了如下两个研究问题.

RQ1: 基于大模型特征蒸馏的机器学习方法在不同特征配置的数据集上的表现如何?

原因: 基于第 3 节中对两种基准方法的测试, 大模型方法在抽取关键词上较为全面准确, 但在分类工作上存在稳定性低, 错误率较高的问题, 导致其无法完全胜任分类工作; 而对于机器学习方法, 其关键词抽取能力存在明显不足, 但在分类工作上较为稳定准确. 故本文希望结合两种方法, 将基本特征、结构特征和语义特征作为基础知识, 将大模型抽取的关键词特征作为任务重点, 与标注结果共同投入小模型学习. 本文通过组合不同的结构特征、语义特征和基本特征, 去探究不使用交互式蒸馏机制情况下, 小模型分类工作的最佳特征配置.

RQ2: 基于交互式蒸馏的方法在全部数据集上表现如何?

原因: 在 RQ1 的基础上, 本文认为该模型能够进一步优化. 由于小模型对同样特征的分类结果输出恒定, 无法自主完成纠错工作, 但小模型具有一定可解释性, 能够输出同一项目判断为各分类的概率. 所以针对小模型并不确定的结果, 本文希望引入纠错机制, 将小模型不确定的结果交由大模型进行二次判断, 并以大模型判断结果为准. 本文问题希望验证在本文提出的交互蒸馏机制的情况下, 该方法在全部数据集上的表现结果.

4.3 数据收集

本文仍选取第 3 节实验中 GHTorrent 2021 数据集中的 30 310 个开源仓库作为样本, 抽取其结构特征、语义特征以及基本特征 (参考表 4) 作为特征向量, 其中关键词特征改为大模型基于 README 上下文感知生成的关键词, 语义特征除 CodeBERT 模型抽取外增设 RoBERTa-Large 模型[42]获取的语义嵌入特征. 最后加入人工标注的分类结果共同组成基于交互式蒸馏方法的特征集. 其中 CodeBERT 模型侧重于代码与自然语言的联合理解, RoBERTa-Large 模型侧重于通用自然语言理解, 不同侧重点能更好地捕捉不同项目 README 的语义差异.

4.4 RQ1 实验具体过程

针对 RQ1 进行实验时, 本文首先挑选了第 3 节验证的传统机器学习体系下最优模型: 支持向量机 (平均 $F1$ 值为 65.7%) 和逻辑回归 (平均 $F1$ 值为 65.9%) 作为基线. 随后对它们进行控制数据集配置变量, 本文基于 CodeBERT 语义特征、RoBERTa-Large 语义特征、基本特征 (大模型抽取的关键词)、基本特征 (传统 TF-IDF 抽取的关键词特征) 和结构特征, 讨论以下 17 种不同的组合以探寻基于交互式蒸馏方法对于大模型生成以及传统机器学习方法的提升, 具体见表 6, 评估过程采用精确率、准确率和 $F1$ 值这 3 种评估指标, 以渐进式增强场景 (从单一特征逐步

叠加), 定量揭示交互式蒸馏机制在关键词语义深度上的优势、对细粒度分类模型的性能增益, 从而为交互式蒸馏机制提供可解释性优化依据. 实验框架见图 7.

表 6　测试组合

分类	特征组合	特征组合含义
单特征集	CB	仅含CodeBERT语义特征 (CB)
	RL	仅含RoBERTa-Large语义特征 (RL)
	BC	仅含基本特征 (大模型抽取的关键词) (BC)
	BT	仅含基本特征 (传统TF-IDF抽取的关键词特征) (BT)
	S	仅含结构特征 (S)
双特征集	BC_CB	含有基本特征 (大模型抽取的关键词) (BC) 和CodeBERT语义特征 (CB)
	BT_CB	含有基本特征 (传统TF-IDF抽取的关键词特征) (BT) 和CodeBERT语义特征 (CB)
	BC_RL	含有基本特征 (大模型抽取的关键词) (BC) 和RoBERTa-Large语义特征 (RL)
	BT_RL	含有基本特征 (传统TF-IDF抽取的关键词特征) (BT) 和RoBERTa-Large语义特征 (RL)
	S_CB	含有结构特征 (S) 和CodeBERT语义特征 (CB)
	S_RL	含有结构特征 (S) 和RoBERTa-Large语义特征 (RL)
	S_BC	含有结构特征 (S) 和基本特征 (大模型抽取的关键词) (BC)
	S_BT	含有结构特征 (S) 和基本特征 (传统TF-IDF抽取的关键词特征) (BT)
三特征集	S_BC_CB	含有结构特征 (S)、基本特征 (大模型抽取的关键词) (BC) 和CodeBERT语义特征 (CB)
	S_BC_RL	含有结构特征 (S)、基本特征 (大模型抽取的关键词) (BC) 和RoBERTa-Large语义特征 (RL)
	S_BT_CB	含有结构特征 (S)、基本特征 (传统TF-IDF抽取的关键词特征) (BT) 和CodeBERT语义特征 (CB)
	S_BT_RL	含有结构特征 (S)、基本特征 (传统TF-IDF抽取的关键词特征) (BT) 和RoBERTa-Large语义特征 (RL)

注: CB代指CodeBERT语义特征; RL代指RoBERTa-Large语义特征; BC代指基本特征 (大模型抽取的关键词); BT代指基本特征 (传统TF-IDF抽取的关键词特征); S代指结构特征

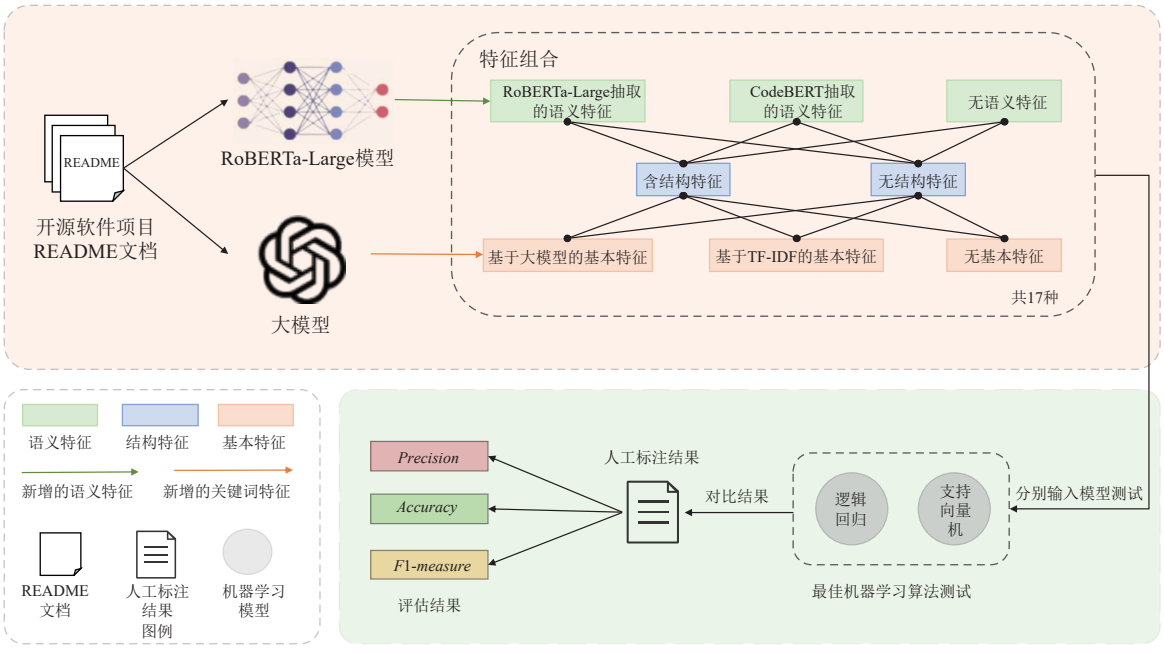

图 7　RQ1 实验框架图

4.5　RQ2 实验具体过程

针对 RQ2 进行实验时, 需要确定相应的配置: (1) 特征集和机器学习算法, 实验选择了 RQ1 中最佳配置的特

征以及两种机器学习模型中的最佳机器学习模型配置. (2) 不确定样本的选取, 本文以 0.05 为步长, 遍历了预测概率前二差值为 0.05–0.3 的 3 种评估指标和消耗的 tokens (最终结果如图 8 所示), 可见 3 种评估指标呈初步快速上升, 而后略有升高保持平稳的态势, 其转折点落在 0.15 上, 而 tokens 消耗呈单调上升趋势. 通过各阈值的尝试并综合考虑效率开销, 本文最终确定使用预测概率前二差值且阈值设置为 15%. 例如, 某样本模型输出为桌面应用的概率为 51%, 输出为游戏应用的概率为 33%, 差值为 18%, 大于 15%, 因此该样本比较稳妥, 不需要与大模型二次交互. 阈值 15% 的设定兼顾了交互样本数量与非交互样本的预测准确. (3) 大模型的选取, 实验沿用了之前的大模型配置 ChatGPT API (gpt-4o-mini).

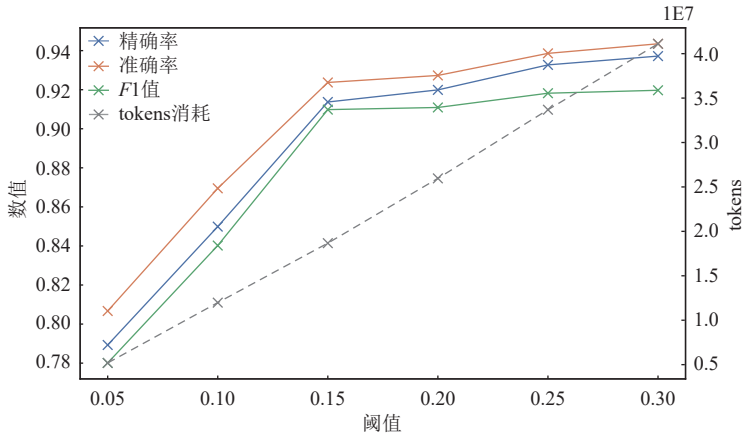

图 8 不同阈值下的评估指标以及 tokens 消耗

本文首先使用最佳机器学习算法对最佳特征集进行十折交叉学习与预测, 并输出每个项目为各个分类的概率. 随后在项目预测结果中找出 Top1 概率–Top2 概率<0.15 的项目, 针对边界样本进行交互式蒸馏, 本文将 README 原始文本、监督模型预测概率 (Top1-xxx, Top2-xxx,…) 输入 ChatGPT API (gpt-4o-mini) 重新标注, 提示词如提示词模板 3 所示.

提示词模板 3. 大模型交互式蒸馏.

“You are an open source project’s development domain classifier. Your task is to classify the domain of software development project according to the following text of the project description and readme files. Categories include: (1) Desktop applications; (2) AI and machine learning applications; (3) WeChat application development; (4) Enterprise applications; (5) Web applications; (6) Mobile applications; (7) Code development tools or plugins; (8) Server application; (9) Game development; (10) Application plugins; (11) Others; (12) Unclassified. The current model predictions for this project are as follows—Top1: xxx, Top2: xxx,… Please relabel this project according to its actual application scope. Please provide the result: and reasons:”

最后将大模型修正过的结果代替机器学习模型分类的结果存入最终分类结果中, 与人工标注结果对比计算各评估指标. RQ2 整体实验流程如后文图 9.

5 实验结果

5.1 RQ1 实验结果

对不同机器学习模型的对比分析表明, 最优模型配置为融合 RoBERTa-Large 语义特征、结构特征与基本特征 (大模型的上下文感知生成关键词), 配合支持向量机算法, 在十折交叉验证中实现 75.3% 的平均精确率、76.3% 的准确率及 75.2% 的 $F1$ 值, 且在各配置相较于逻辑回归算法皆高 1% 左右.

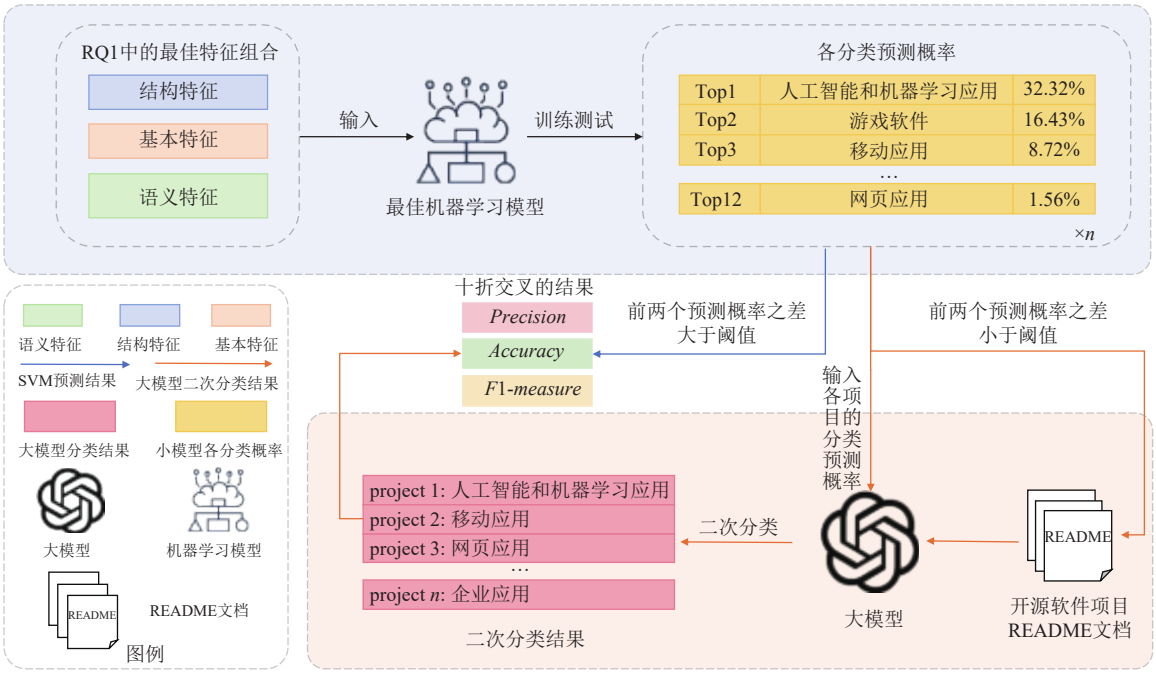

图 9　RQ2 框架图

在单个特征中, 基本特征 (大模型的上下文感知生成关键词) 表现最好, 验证了大模型在总结 README 关键词工作上的优越性; 在双特征融合的情况下, 基本特征 (大模型的上下文感知生成关键词)+RoBERTa-Large 语义特征表现最好, 相较于基本特征 (大模型的上下文感知生成关键词) 提升约 1.7%; 在三特征融合的情况下, 基本特征 (大模型的上下文感知生成关键词)+RoBERTa-Large 语义特征+结构特征表现最好, 相较于基本特征 (大模型的上下文感知生成关键词)+RoBERTa-Large 语义特征提升约 0.4%.

根据实验结果来看, 其中大模型抽取的关键词特征贡献最大, 其余特征如结构特征、RoBERTa-Large 语义特征、CodeBERT 语义特征等特征贡献较小, 但它们仍然对分类判断有一定帮助, 故本文将这些特征全部予以保留, 以实现分类效果的最佳化. 结果如后文图 10 及图 11 所示.

5.2　RQ2 实验结果

RQ2 的最终模型基于 RQ1 确定的最优特征集 (RoBERTa-Large 语义特征+结构特征+基本特征 (大模型的上下文感知生成关键词)) 训练的最优机器学习模型 (支持向量机), 并将 Top1 概率–Top2 概率<0.15 的项目交由 ChatGPT API (gpt-4o-mini) 进行交互式蒸馏, 在测试集上达到 92.1% 的平均精确率、91.8% 的平均准确率和 91.8% 的平均 $F1$ 值, 较 RQ1 中未启用交互式蒸馏的模型提升 15.5–16.8 个百分点.

错误案例分析表明, 交互式蒸馏有效解决了监督模型在跨领域场景 (如含区块链技术的 Web 应用被误判为 Others) 和新兴技术场景中的系统性偏差, 验证了大小模型的互补优势. 具体提升如后文表 7.

5.3　对各 RQ 的回答

回答 RQ1: 本文经过对 17 种不同特征组合以及两种机器学习算法的测试后, 得出结论: 基于大模型特征蒸馏的机器学习方法的最优模型配置为融合 RoBERTa-Large 语义特征、结构特征与基本特征 (大模型的上下文感知生成关键词), 配合支持向量机算法, 达到了 75.3% 的平均精确率、76.3% 的准确率及 75.2% 的 $F1$ 值. 在对 RQ1 的测试过程中, 本文发现在小模型 (机器学习算法) 分类错误的结果中, 其对分类结果并不确定, 即判断为该分类的概率较小. 故本文希望对于小模型不确定的结果交由大模型重新进行判断, 对 RQ1 中的基于大模型特征蒸馏的机器学习方法进一步优化, 从而引出 RQ2 中对交互式蒸馏机制的测试.

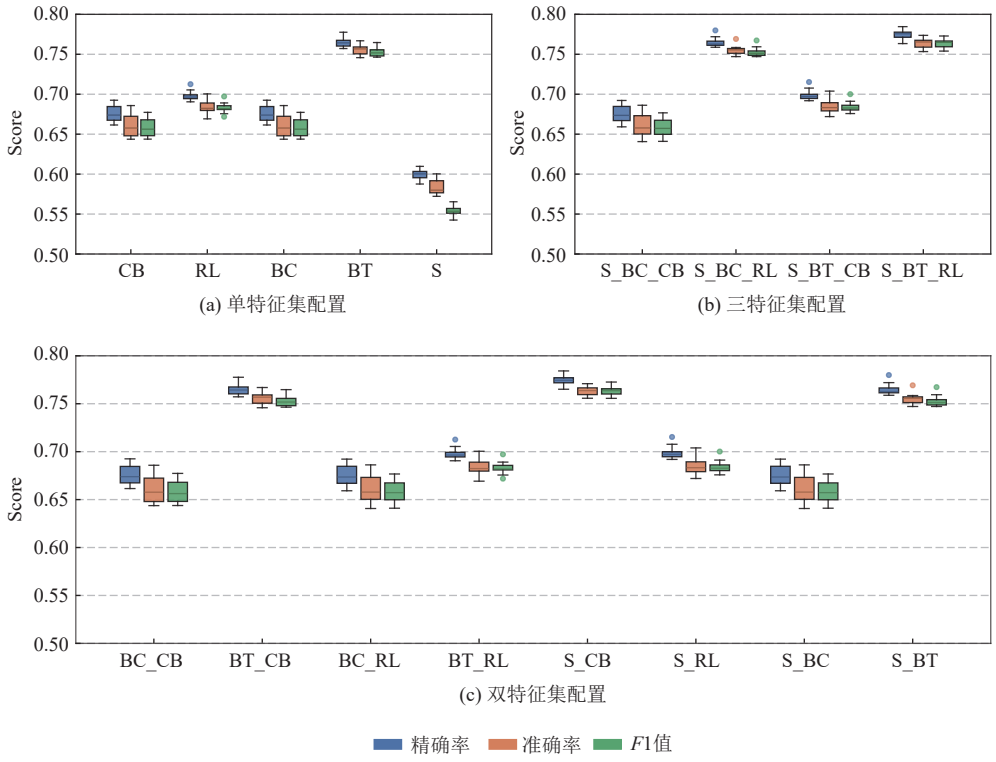

图 10 支持向量机在各数据配置下的评估结果 (特征含义见表 6)

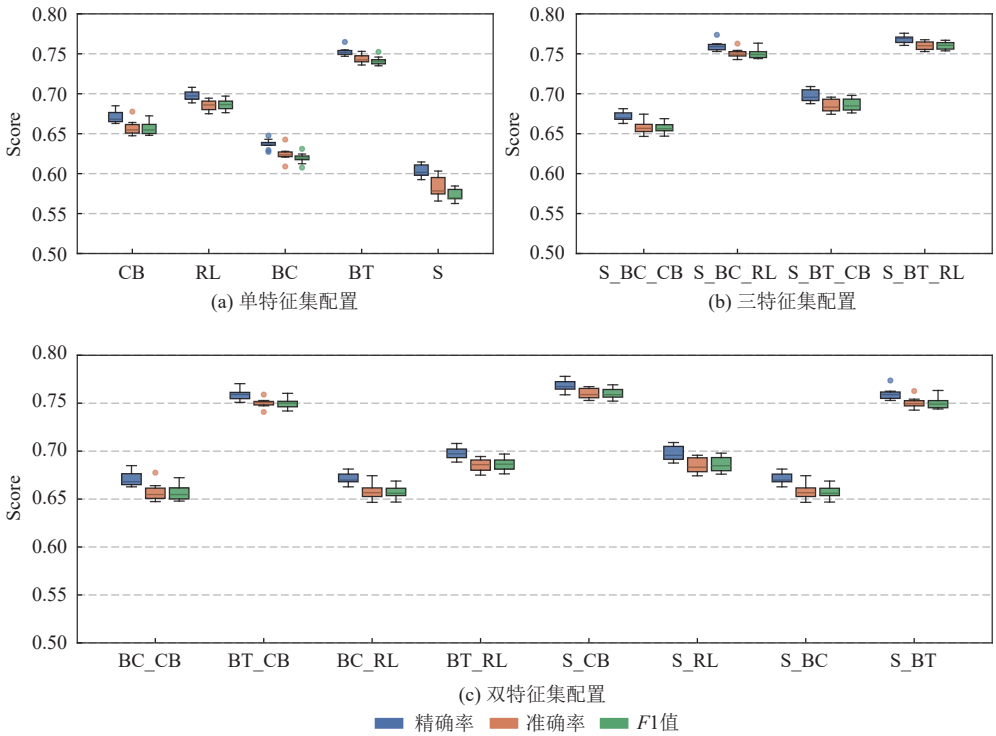

图 11 逻辑回归在各数据配置下的评估结果

表 7　交互式蒸馏模型相较于各方法最佳配置下的提升 (%)

对比方法	精确率	准确率	*F*1值
大模型直接生成法 (LLM)	59.3	51.9	51.1
大模型蒸馏法 (LLM蒸馏特征+机器学习算法)	61.9	56.1	52.6
机器学习法 (常规特征+机器学习算法)	65.8	67.3	65.7
基于大模型特征蒸馏的机器学习方法 (LLM蒸馏特征+常规特征+机器学习算法)	75.3	76.3	75.2
交互式蒸馏方法 (LLM蒸馏特征+常规特征+机器学习算法+LLM二次交互)	92.1	91.8	91.8

注: 括号内为该方法采用的模块

回答 RQ2: 引入交互式蒸馏机制后, 通过将 Top1 概率–Top2 概率<0.15 的项目交由 ChatGPT API (gpt-4o-mini) 来重新判断其分类. 最终在 RQ1 测试下的最优配置中, 模型取得了 92.1% 的平均精确率、91.8% 的平均准确率和 91.8% 的平均 *F*1 值, 相较于未使用交互式蒸馏机制提升了约 16%.

6 讨　论

经过测试大模型方法和机器学习方法效果较差, 本文提出的交互式蒸馏方法在 *F*1 值上相较于大模型直接生成法提升了 40.7%, 相较于大模型蒸馏方法提升了 39.2%, 相较于机器学习法提升了 26.1%, 接下来本文将讨论各个方法会产生该结果的原因.

6.1 基准方法的讨论

(1) 大模型直接生成方法

基于 gpt-4o-mini 的纯生成式方法在细粒度分类任务中表现出显著局限性 (平均 *F*1 值=51.1%), 主要体现在技术语境理解与动态行为推理的不足. 其优势在于无需人工特征工程的自动化处理能力, 能够快速完成粗粒度术语匹配 (如识别 API、Database 等显性关键词). 然而, 该方法对技术文档的隐喻性表述 (如数据库中的 Realm 被误理解为奇幻游戏) 和泛化术语缺乏深层解析能力. 根本原因在于大模型的训练语料与技术文档分布存在差异, 且缺乏领域知识的理解, 导致其难以区分有多种语义的专业名词. 这一缺陷表明, 大模型生成方法需与传统机器学习方法协同才能突破当前瓶颈.

(2) 大模型蒸馏式方法

尽管蒸馏式方法通过监督学习机制部分缓解了生成式模型的随机噪声, 但其性能天花板仍受限于大模型生成关键词的固有误差. 实验表明, 生成关键词的误标率高达 48.1%, 主要源于技术文档的语境敏感性缺失, 例如 edward/Shopify-Developer-Book 项目为 Shopify 开发的技巧电子书, 但因描述中出现 Web、Ruby 等被错误归类至网页应用, 而非实际所属的其他类别. 这种关键词偏差在监督学习中被固化, 导致模型建立错误的特征-类别关联, 进而引发模型误判.

(3) 机器学习方法

传统机器学习方法的核心局限源于 TF-IDF 关键词提取对技术语义的表征不足. 以 tanmaster/EVM-Simulator 项目为例, TF-IDF 提取的高权重词 (如 testing、validation 等) 未能捕获 EVM, debugger 等低频核心术语, 导致模型误判为人工智能和机器学习应用. 这种缺陷本质上反映了 TF-IDF 的统计特性与细粒度分类需求的矛盾: 技术领域的关键区分性术语往往呈现低频、高信息熵的特征, 而 TF-IDF 的权重分配机制过度偏向高频通用词汇, 造成一些多场景通用的词被过多提取, 造成项目类别之间的混淆, 从而造成误判.

6.2 交互式蒸馏方法的优点

交互式蒸馏机制通过异构模型的能力互补, 在开源软件项目的细粒度分类任务中展现出多种优势. 首先在处理技术融合型项目时, 该机制展现出独特的适应性. 以自动回复评论机器人项目为例, 监督算法 (如 SVM) 基于 Deep-Learning、Robot-API 等关键词特征将其误判为人工智能和机器学习应用, 同时因为关键词特征中含有 Plugin, 应用插件类别位居 Top2. 而生成式大模型通过解析 README, 识别出其应用场景为某视频平台, 触发交互式蒸馏结

合技术生态知识库将其准确归类至应用插件类别, 使细粒度分类的准确率相较普通机器学习方法提升 24.5%. 这种技术术语的深度解构能力, 有效弥补了机器学习方法对领域专有概念的表征局限.

同时生成式大模型强大的知识库使得其在交互式蒸馏中领域术语的抓取理解能力增强. 交互式蒸馏机制有效提升了技术专有术语的语义解析准确率. 以 lynaghk/c2 项目为例, 监督算法 (如 SVM) 可能因 hiccup、singult 等 Clojure 生态专有术语将其误判为其他类别. 生成式大模型通过解析 README 中声明式 DOM 映射等技术文档核心概念, 结合 GitHub Topic 知识库中关联规则, 触发交互式蒸馏将其准确归类至代码开发工具或插件类别, 显著提升了其垂直领域工具的分类准确率.

交互式蒸馏机制通过构建“小模型概率分布-低可信样本”的二次决策架构, 解决了大模型方法和机器学习方法的核心缺陷. 该机制的核心优势在于: 监督学习模型输出的分类概率分布为不确定性量化提供了可解释的客观指标——当 Top1 概率–Top2 概率<0.15 时, 表明模型在技术概念边界处存在决策模糊性. 这种基于概率差异的触发机制, 将大模型的语义理解能力精准聚焦于关键争议样本, 而非全量数据集的粗粒度处理. 在交互式蒸馏过程中, 监督模型提供的概率排序 (各类别及其置信度) 作为结构化知识输入大模型, 相较于 RQ1 中的基于大模型特征蒸馏的机器学习方法精确率提升 16.8%、准确率提升 15.5%、*F*1 值提升 16.6%.

6.3 交互式蒸馏模型的不足

经过对分类错误样本的分析, 本文发现以下问题.

(1) README 文档信息过少导致的特征稀疏性

部分 README 文档存在内容简略化问题, 仅用 1、2 句话泛化描述项目功能, 例如 FIN-Vorkurs/finvorkurs 项目 README 文档中仅有“New FIN Vorkurs Homepage”, 缺乏核心技术栈、应用场景、架构设计等关键细节. 这种低信息密度导致监督学习模型提取的特征向量 (如大模型抽取的关键词、CodeBERT 语义编码) 难以有效区分, 迫使模型过度依赖统计噪声进行推断, 显著增加误判风险.

(2) 小众及多领域专有术语的语义歧义性

部分 README 文档中高频出现的领域特定缩略词、小众技术术语及对小众项目的过多引用, 与通用语言模型预训练语料存在显著分布差异. 例如, mirage/ocaml-dns 项目 README 文档中 DNS/协议领域缩略词极密集, 出现 DNSSEC、TSIG、AXFR、IXFR、EDNS、HINFO、ISDN 和大量 RFC 引用等. 这种小众专业术语导致交互式蒸馏模型在语义的特征抽取上产生理解偏差, 将专业概念映射到不恰当的分类类别; 或者基于已有知识库进行猜测, 进而引发类别混淆.

(3) 技术细节与应用背景的表述失衡

部分 README 文档存在实现细节的过度技术化描述与应用价值的语境缺失. 例如 jiem/start-stop-daemon 项目的 README 文档将 90% 以上的篇幅聚焦于代码模块的接口定义、性能调优参数及底层算法实现, 却未系统阐述项目的核心应用场景定位、目标用户群体的功能性需求或典型部署案例. 这种信息架构的失衡导致监督模型在特征提取过程中, 难以捕捉技术指标与功能属性之间的语义关联, 使分类决策过度依赖间接特征 (如章节标题数量、代码块数量) 进行推断, 从而引发细粒度分类错误.

7 使用场景

7.1 追求最高准确率场景下的策略

在需要追求分类准确率最大化的场景中, 推荐采用机器学习方法与大模型生成特征的融合方案. 完整保留基本特征数据 (章节数量、代码块数量等)、语义特征, 通过监督学习算法 (SVM) 的特征加权机制实现准确率较高的细粒度分类. 该策略充分利用传统机器学习方法对技术文档组织规范性的捕捉能力与大模型对领域术语的深度捕捉能力, 在开源项目的细粒度分类中可达 92.1% 的平均精确率、91.8% 的平均准确率和 91.8% 的平均 *F*1 值, 但需付出更高的计算成本和时间成本.

7.2 效率优先场景下的轻量化部署方案

面向实时性要求高或资源受限的环境, 建议仅保留大模型动态生成的关键词特征作为输入. 该方案通过舍弃语义特征与基本特征数据, 将特征空间压缩至原有规模的 7.4%, 让推理速度提升且占用空间减少. 尽管分类准确率较完整特征方案下降 2.2 个百分点 ($F1$ 值从 91.8 降至 89.6%), 但其仍通过大模型的关键词提炼能力维持可用性能. 此方案特别适配于对开源软件细粒度分类效率较高且不追求极限准确率的场景.

8 效度威胁

本节将讨论若干可能影响实验结果有效性的威胁, 主要分为构造效度、内部效度和外部效度, 最后讨论本文的可靠性.

8.1 构造效度

本文将所有开源软件项目归为 12 类, 构造效度的核心问题在于被标记为某类的项目是否真实属于该类.

第 1 个威胁是研究人员可能会误判项目类型. 由于一些项目的描述侧重于技术架构、使用方法等而非对项目的具体介绍. 例如 pism/regional-tools 项目, 在 README 中对项目的描述仅限于流域划分工具, 其余为技术架构与使用教程, 难以确定其到底为网页应用、人工智能和机器学习应用还是桌面应用. 若研究人员未仔细从 README 中获取用户安装、鼠标交互等关键词, 则难以将其正确归类为桌面应用. 且研究人员仅从 README 文件中获取信息, 可能会遗漏代码库中的关键信息, 从而导致错误分类.

第 2 个威胁是标注过程可能存在主观偏差. 尽管在标注开始的初期, 通过 100 个项目的示例明确了 12 个分类的定义与分类标准, 且 3 位研究人员的最终平均 Cohen's Kappa 系数为 84.8%, 但最终仍存在意见并不统一的项目. 但最终意见不统一的项目仅有 173 个, 所以该威胁可控.

8.2 内部效度

内部效度涉及实验中的因果关系是否真实存在. 本文内部效度威胁主要在于研究中使用的模型、特征和参数选择. 具体可能存在的几点威胁如下.

(1) 特征选择不全面. 数据集中现有的基本特征、结构特征和语义特征可能无法充分捕捉 12 种细粒度分类下的细微差异和对应关系, 这种情况可能导致模型的准确率下降, 但模型十折交叉后结果表现良好, 故该威胁基本可控.

(2) 过拟合或欠拟合. 30 310 个开源软件项目, 25 项特征可能导致模型过度拟合训练数据, 影响模型的实际效果. 但模型十折交叉结果稳定, 所以该威胁可控.

(3) 模型优化不足. 实验中未考虑对不同监督学习模型或者参数的调优, 可能导致了模型选择不当或未充分挖掘模型的性能.

(4) 分类定义单一. 实验中采用应用领域优先分类标准, 但在实际场景中可能存在一个项目从属于多个分类的情况. 所以本文针对项目的应用领域进行分类, 可能会遗漏掉部分项目存在的其他分类可能.

8.3 外部效度

本文的外部有效性取决于是否能够将所获得的结果推广到 GitHub 平台或其他开源软件开发平台. 本文用于对开源软件项目进行细粒度分类的信息包含基本特征、结构特征和 README 中的语义特征, 这些信息在其他开源软件开发平台 (如 SourceForge) 上的每个项目中也可以找到. 因此, 该模型也可以应用于其他开源软件开发平台.

同时该模型十折交叉结果稳定且良好, 可知该模型有较好的泛化能力, 在未见过的数据集中也拥有较强的判断能力, 故对于 GitHub 的其他开源软件项目和其他开源软件开发平台上的开源软件项目也可以取得近似的效果.

8.4 可靠性

本文具有高度可复现性, 原因包括如下.

(1) 使用的 GHTorrent 2021 版本数据集是研究开源开发的公开且广泛使用的数据集.

(2) 本文在 GitHub (https://github.com/BINGOik/jos2025) 发布了相关工具, 可供使用测试.

(3) 模型构建使用了广泛应用的机器学习算法.

基于以上 3 点, 本文确信其他研究者可复现我们的成果.

9 总 结

本文提出了一种基于交互式蒸馏的细粒度分类框架, 旨在解决开源软件项目分类任务中传统机器学习方法特征关键词抽取能力不足与大模型细粒度分类能力不足的双重挑战. 核心方法包含以下贡献.

(1) 基于交互式蒸馏的混合特征工程

构建多模态特征空间, 融合大模型生成的上下文感知关键词、CodeBERT/RoBERTa-Large 语义向量及基本数据 (章节数量、代码块数量等), 形成混合特征集. 该方法突破传统 TF-IDF 的局限, 使监督模型 (如 SVM) 的基准 $F1$ 值从 65.7% 提升至 75.2%.

(2) 监督模型可解释性驱动的交互式蒸馏机制

基于监督模型输出的概率分布, 设计自适应阈值 (Top1 概率–Top2 概率<0.15) 触发大模型二次分类. 将监督模型预测类别及其概率作为已有知识输入大模型, 引导其聚焦关键特征, 生成新的分类判断以及原因.

(3) 性能提升

实验表明, 交互式蒸馏机制使分类性能显著优化. 对于全样本而言, 精确率从单轮预测的 75.3% 提升至 92.1%, $F1$ 值从 75.2% 提升至 91.8%. 且在 $F1$ 值上相较于大模型直接生成法提升了 40.7%, 相较于大模型蒸馏方法提升了 39.2%, 相较于机器学习法提升了 26.1%.

基于上述讨论, 我们未来计划: (1) 提升交互式蒸馏的次数, 由单次交互式蒸馏改为构建概率驱动的循环交互蒸馏机制. (2) 本文要做的是在初次交互式蒸馏后, 若大模型输出的类别与监督模型的原始预测类别概率存在明显误差 (如交互蒸馏结果在监督模型概率中为 Top12), 则触发二次交互式蒸馏, 将初次交互蒸馏结果作为先验知识输入大模型进行再次交互式蒸馏. 依照上述步骤进行多轮动态交互蒸馏以提升结果的可靠性. (3) 在实际应用场景中, 开源项目可能属于多个分类, 让算法的判断结果不再局限于单一分类, 而是全面判断出项目所属分类.

本文通过交互式蒸馏机制, 实现了开源项目的细粒度分类任务中大模型与监督模型的协同优化. 未来工作需在迭代深度、计算效率及领域适配性等维度持续突破, 以应对开源生态快速演化的复杂需求.

References

[1] 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]

[2] Steinhardt J, Koh PW, Liang P. Certified defenses for data poisoning attacks. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 3520–3532.

[3] Cheng C, Li B, Li ZY, Liang P, Yang X. An in-depth study of the effects of methods on the dataset selection of public development projects. IET Software, 2022, 16(2): 146–166. [doi: 10.1049/sfw2.12050]

[4] Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D. An in-depth study of the promises and perils of mining GitHub. Empirical Software Engineering, 2016, 21(5): 2035–2071. [doi: 10.1007/s10664-015-9393-5]

[5] Deng WT, Cheng C, He P, Chen MY, Li B. Interaction prediction of multi-granularity software system based on graph neural network. Ruan Jian Xue Bao/Journal of Software, 2025, 36(5): 2043–2063. http://www.jos.org.cn/1000-9825/7207.htm [doi: 10.13328/j.cnki.jos.007207]

[6] 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 (ICSME 2016). Raleigh: IEEE, 2016. 334–344. [doi: 10.1109/ICSME.2016.31]

[7] Pascarella L, Palomba F, Di Penta M, Bacchelli A. How is video game development different from software development in open source? In: Proc. of the 15th Int'l Conf. on Mining Software Repositories. Gothenburg: ACM, 2018. 392–402. [doi: 10.1145/3196398.3196418]

[8] Lei J, Ye HJ, Wu ZS, Zhang P, Xie L, He YX. Big-data platform based on open source ecosystem. Journal of Computer Research and Development, 2017, 54(1): 80–93.

[9] Asyrofi MH, Thung F, Lo D, Jiang LX. AUSearch: Accurate API usage search in GitHub repositories with type resolution. In: Proc. of

the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER 2020). London: IEEE, 2020. 637–641. [doi: 10.1109/SANER48275.2020.9054809]

[10] Masud MR, Rokon MOF, Zhang Q, Faloutsos M. MetaSim: A search engine for finding similar GitHub repositories. In: Proc. of the 2024 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME 2024). Flagstaff: IEEE, 2024. 868–872. [doi: 10.1109/ICSME58944.2024.00093]

[11] Gousios G, Spinellis D. Mining software engineering data from GitHub. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering Companion (ICSE-C 2017). Buenos Aires: IEEE, 2017. 501–502. [doi: 10.1109/ICSE-C.2017.164]

[12] Cosentino V, Luis Cánovas Izquierdo J, Cabot J. Findings from GitHub: Methods, datasets and limitations. In: Proc. of the 13th IEEE/ACM Working Conf. on Mining Software Repositories. Austin: IEEE, 2016. 137–141.

[13] Franco-Bedoya O, Ameller D, Costal D, Franch X. Open source software ecosystems: A Systematic mapping. Information and Software Technology, 2017, 91: 160–185. [doi: 10.1016/j.infsof.2017.07.007]

[14] Tsay J, Dabbish L, Herbsleb J. Influence of social and technical factors for evaluating contribution in GitHub. In: Proc. of the 36th Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 356–366. [doi: 10.1145/2568225.2568315]

[15] Chawla S, Arunasalam B, Davis J. Mining open source software (OSS) data using association rules network. In: Proc. of the 7th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining. Seoul: Springer, 2003. 461–466. [doi: 10.1007/3-540-36175-8_46]

[16] Han JX, Deng SG, Xia X, Wang DJ, Yin JW. Characterization and prediction of popular projects on GitHub. In: Proc. of the 43rd IEEE annual computer software and applications conference (COMPSAC 2019). Milwaukee: IEEE, 2019. 21–26. [doi: 10.1109/COMPSAC.2019.00013]

[17] Wang TL, Wang SW, Chen TH. Study the correlation between the readme file of GitHub projects and their popularity. Journal of Systems and Software, 2023, 205: 111806. [doi: 10.1016/j.jss.2023.111806]

[18] Kapur R, Sodhi B. OSS effort estimation using software features similarity and developer activity-based metrics. ACM Trans. on Software Engineering and Methodology, 2022, 31(2): 33. [doi: 10.1145/3485819]

[19] Rokon MO F, Yan P, Islam R, Faloutsos M. Repo2Vec: A comprehensive embedding approach for determining repository similarity. In: Proc. of the 2021 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Luxembourg: IEEE, 2021. 355–365. [doi: 10.1109/ICSME52107.2021.00038]

[20] Auch M, Weber M, Mandl P, Wolff C. Similarity-based analyses on software applications: A systematic literature review. Journal of Systems and Software, 2020, 168: 110669. [doi: 10.1016/j.jss.2020.110669]

[21] Yang YM, Xia XX, Lo D, Bi TT, Grundy J, Yang XH. Predictive models in software engineering: Challenges and opportunities. ACM Trans. on Software Engineering and Methodology (TOSEM), 2022, 31(3): 56. [doi: 10.1145/3503509]

[22] Zhang Y, Lo D, Kochhar PS, Xia X, Li QL, Sun JL. Detecting similar repositories on GitHub. In: Proc. of the 24th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Klagenfurt: IEEE, 2017. 13–23. [doi: 10.1109/SANER.2017.7884605]

[23] 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]

[24] 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: ACM, 2018. 15. [doi: 10.1145/3239235.3240501]

[25] Munaiah N, Kroh S, Cabrey C, Nagappan M. 2017. Curating GitHub for engineered software projects. Empirical Software Engineering, 2017, 22(6): 3219–3253. [doi: 10.1007/s10664-017-9512-6]

[26] Zhang Y, Xu FF, Li S, Meng Y, Wang X, Li Q. HiGitClass: Keyword-driven hierarchical classification of GitHub repositories. In: Proc. of the 2019 IEEE Int'l Conf. on Data Mining (ICDM). Beijing: IEEE, 2019. 876–885. [doi: 10.1109/ICDM.2019.00098]

[27] Wang XJ, Huang JW, Ye CY, Zhou H. GlobalTagNet: A graph-based framework for multi-label classification in GitHub issues. In: Proc. of the 32nd IEEE Int'l Requirements Engineering Conf. (RE). Reykjavik: IEEE, 2024. 67–78. [doi: 10.1109/RE59067.2024.00017]

[28] Zhang YY, Yang RZ, Xu XQ, Li R, Xiao JF, Shen JM, Han JW. TELEClass: Taxonomy enrichment and LLM-enhanced hierarchical text classification with minimal supervision. In: Proc. of the 2025 ACM on Web Conf. Sydney: ACM, 2025. 2032–2042. [doi: 10.1145/3696410.3714940]

[29] Sun XF, Li XY, Li JW, Wu F, Guo SW, Zhang TW, Wang GY. Text classification via large language models. In: Findings of the Association for Computational Linguistics: EMNLP 2023. Singapore: ACL, 2023. 9801–9813. [doi: 10.18653/v1/2023.findings-emnlp.603]

[30] Yin G, Wang T, Liu BX, Zhou MH, Yu Y, Li ZX, Ouyang JQ, Wang HM. Survey of software data mining for open source ecosystem. Ruan Jian Xue Bao/Journal of Software, 2018, 29(8): 2258–2271 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5524.htm [doi: 10.13328/j.cnki.jos.005524]

[31] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020. ACL, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[32] Crowston K, Howison J, Annabi H. Information systems success in free and open source software development: Theory and measures. Software Process: Improvement and Practice, 2006, 11(2): 123–148. [doi: 10.1002/spip.259]

[33] Yu Y, Wang HM, Yin G, Wang T. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? Information and Software Technology, 2016, 74: 204–218. [doi: 10.1016/j.infsof.2016.01.004]

[34] Wang Y, Wu YX, Gao T, Chen ZY, Xu C, Yu H, Zhang CZ. Survey on governance technology of open-source software library ecosystem: Twenty years of progress. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 629–674 (in Chinese with English abstract). 2024, 35(2): 629–674. http://www.jos.org.cn/1000-9825/6983.htm [doi: 10.13328/j.cnki.jos.006983]

[35] Aizawa A. An information-theoretic perspective of TF-IDF measures. Information Processing & Management, 2003, 39(1): 45–65. [doi: 10.1016/S0306-4573(02)00021-3]

[36] Han L, Li M. Open source software classification using cost-sensitive multi-label learning. Ruan Jian Xue Bao/Journal of Software, 2014, 25(9): 1982–1991 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4639.htm [doi: 10.13328/j.cnki.jos.004639]

[37] Fu W, Menzies T. Easy over hard: A case study on deep learning. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 49–60. [doi: 10.1145/3106237.3106256]

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

[39] Loosli G, Canu S, Ong CS. Learning SVM in Kreĭn spaces. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2016, 38(6): 1204–1216. [doi: 10.1109/TPAMI.2015.2477830]

[40] 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]

[41] Imai S. Is GitHub Copilot a substitute for human pair-programming? An empirical study. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. (ICSE-Companion). Pittsburgh: IEEE, 2022. 319–321. [doi: 10.1145/3510454.3522684]

[42] Kim B, Seo J, Koo MW. Randomly wired network based on RoBERTa and dialog history attention for response selection. IEEE/ACM Trans. on Audio, Speech, and Language Processing, 2021, 29: 2437–2442. [doi: 10.1109/TASLP.2021.3077119]

附中文参考文献

[1] 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]

[5] 邓文涛, 程璨, 何鹏, 陈孟瑶, 李兵. 基于图神经网络的多粒度软件系统交互关系预测. 软件学报, 2025, 36(5): 2043–2063. http://www.jos.org.cn/1000-9825/7207.htm [doi: 10.13328/j.cnki.jos.007207]

[8] 雷军, 叶航军, 武泽胜, 张鹏, 谢龙, 何炎祥. 基于开源生态系统的大数据平台研究. 计算机研究与发展, 2017, 54(1): 80–93.

[23] 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]

[30] 尹刚, 王涛, 刘冰珣, 周明辉, 余跃, 李志星, 欧阳建权, 王怀民. 面向开源生态的软件数据挖掘技术研究综述. 软件学报, 2018, 29(8): 2258–2271. http://www.jos.org.cn/1000-9825/5524.htm [doi: 10.13328/j.cnki.jos.005524]

[34] 王莹, 伍盈欣, 高天, 陈子莺, 许畅, 于海, 张成志. 开源软件库生态治理技术研究综述: 二十年进展. 软件学报, 2024, 35(2): 629–674. http://www.jos.org.cn/1000-9825/6983.htm [doi: 10.13328/j.cnki.jos.006983]

[36] 韩乐, 黎铭. 基于代价敏感多标记学习的开源软件分类. 软件学报, 2014, 25(9): 1982–1991. http://www.jos.org.cn/1000-9825/4639.htm [doi: 10.13328/j.cnki.jos.004639]

作者简介

康成希, 本科生, 主要研究领域为机器学习, 软件生态系统, 数据挖掘.

程璨, 博士, 讲师, CCF 专业会员, 主要研究领域为经验性软件工程, 软件生态系统, 软件库挖掘.

张能, 博士, 副教授, CCF 高级会员, 主要研究领域为软件工程, 服务计算, 知识图谱, 推荐系统, 智能问答系统.

游兰, 博士, 教授, CCF 高级会员, 主要研究领域为时空大数据, 自然语言方法, 社会计算, 知识图谱, 空间信息云计算, 虚拟地理环境.

李兵, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为服务计算, 智能化软件工程, 软件服务工程, 人工智能, 复杂系统与复杂网络, 云计算, 大数据安全.

王伟, 博士, 研究员, CCF 杰出会员, 主要研究领域为开源生态.