软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(11):4027−4045 [doi: 10.13328/j.cnki.jos.006369]

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

基于软件知识图谱的代码语义标签自动生成方法*

邢双双[1,2], 刘名威[1,2], 彭 鑫[1,2]

[1](复旦大学 计算机科学技术学院, 上海 201203)

[2](上海市数据科学重点实验室(复旦大学), 上海 201203)

通信作者: 彭鑫, E-mail: pengxin@fudan.edu.cn

摘 要: 开源及企业软件项目和各类软件开发网站上的代码片段是重要的软件开发资源. 然而, 很多开发者代码搜索需求反映的代码的高层意图和主题难以通过基于代码文本的信息检索技术来实现精准的代码搜索. 因此, 反映代码整体意图和主题的语义标签对于改进代码搜索、辅助代码理解都具有十分重要的作用. 现有的标签生成技术主要面向文本内容或依赖于历史数据, 无法满足大范围代码语义标注和辅助搜索、理解的需要. 针对这一问题, 提出了一种基于知识图谱的代码语义标签自动生成方法 KGCodeTagger. 该方法通过基于 API 文档和软件开发问答文本的概念和关系抽取构造软件知识图谱, 作为代码语义标签生成的基础. 针对给定的代码, 该方法识别并抽取出通用 API 调用或概念提及, 并链接到软件知识图谱中的相关概念上. 在此基础上, 该方法进一步识别与所链接的概念相关的其他概念作为候选, 然后按照多样性和代表性排序, 产生最终的代码语义标签. 通过实验对 KGCodeTagger 软件知识图谱构建的各个步骤进行了评估, 并通过与几个已有的基准方法的比较, 对所生成的代码语义标签质量进行了评估. 实验结果表明, KGCodeTagger 的软件知识图谱构建步骤是合理有效的, 该方法所生成的代码语义标签是高质量、有意义的, 能够帮助开发人员快速理解代码的意图.

关键词: 程序理解; 代码搜索; 知识图谱; 语义标签

中图法分类号: TP311

中文引用格式: 邢双双, 刘名威, 彭鑫. 基于软件知识图谱的代码语义标签自动生成方法. 软件学报, 2022, 33(11): 4027−4045. http://www.jos.org.cn/1000-9825/6369.htm

英文引用格式: Xing SS, Liu MW, Peng X. Automatic Code Semantic Tag Generation Approach Based on Software Knowledge Graph. Ruan Jian Xue Bao/Journal of Software, 2022, 33(11): 4027−4045 (in Chinese). http://www.jos.org.cn/1000-9825/6369.htm

Automatic Code Semantic Tag Generation Approach Based on Software Knowledge Graph

XING Shuang-Shuang[1,2], LIU Ming-Wei[1,2], PENG Xin[1,2]

[1](School of Computer Science, Fudan University, Shanghai 201203, China)

[2](Shanghai Key Laboratory of Data Science (Fudan University), Shanghai 201203, China)

Abstract: Code snippets in open-source and enterprise software projects and posted on various software development websites are important software development resources. However, developer's needs for code search often reflect high-level intentions and topics, which are difficult to be satisfied through code search techniques based on information retrieval. It is thus highly desirable that code snippets can be accompanied with semantic tags reflecting their high-level intentions and topics to facilitate code search and understanding. Existing tag generation technologies are mainly oriented to text content or rely on historical data, and cannot meet the needs of large-scale code semantic annotation and auxiliary code search and understanding. Targeted at the issue, this study proposes an approach based on software knowledge graph (called KGCodeTagger) that automatically generates semantic tags for code snippets. KGCodeTagger constructs a software knowledge graph based on concepts and relations extracted from API documentations and software development Q&A text and uses the knowledge graph as the basis of code semantic tag generation. Given a code snippet, KGCodeTagger

* 基金项目: 国家自然科学基金(61972098)

收稿时间: 2020-12-25; 修改时间: 2021-02-13; 采用时间: 2021-05-06; jos 在线出版时间: 2021-05-20

identifies and extracts API invocations and concept mentions, and then links them to the corresponding concepts in the software knowledge graph. On this basis, the approach further identifies other concepts related to the linked concepts as candidates and selects semantic tags from relevant concepts based on the diversity and representativeness. The software knowledge graph construction steps of KGCodeTagger and the quality of the generated code tags are evaluated. The results show that KGCodeTagger can produce high-quality and meaningful software knowledge graph and code semantic tags, which can help developers quickly understand the intention of the code.

Key words: program comprehension; code search; knowledge graph; semantic tag

代码片段是软件开发活动中重要的可复用软件资源[1]. 为此, 开发人员经常在开源及企业软件项目以及各类软件开发网站(如 Stack Overflow[2])上搜索所需要的代码片段. 然而, 开发人员的代码搜索需求经常反映代码的高层意图和主题, 与代码文本之间存在着语义鸿沟, 难以通过基于代码的信息检索技术来实现精准搜索[3].

一个可解决方案是给代码打上语义标签. 语义标签能够对标签对象的高层意图和主题进行概括凝练, 是一种常见的辅助检索的手段. 例如, 在开发者问答网站 Stack Overflow 中包含超过 6 万个用户定义的标签, 能够帮助开发者快速检索同一主题的帖子. 反映代码整体意图和主题的语义标签能够在开发者需求和代码之间建立联系, 帮助跨越语义鸿沟, 对于改进代码搜索、辅助代码理解都具有十分重要的作用.

现有的软件领域标签生成技术大都面向文本(如 Stack Overflow 帖子), 通过从文本和历史标签中进行学习构建模型[4−8], 或是计算相似度[9,10], 或是计算其他有意义的得分排序[11]进行标签推荐, 缺乏专门针对代码语义标签生成的工作. 而现有的标签生成技术依赖于文本和历史标签, 存在以下问题, 无法满足大范围代码语义标注和辅助搜索、理解的需要.

- 第一, 标签质量参差不齐[4]. 人工定义的标签存在相似对象被不同标记的现象, 导致标签质量不高. 例如, “sort”, “sorting”, “sorted”是 Stack Overflow 的标签, 但是它们都是排序的意思. 用户生成标签的目的和命名习惯不同, 很难以统一的标准进行约束规范.
- 第二, 用户通常选用高频关键词作为标签[12]. 这种标签不能表示完整的意图, 正如通过统计词频筛选出的单词并不一定具有代表性. 特别是在代码中, 当多个代码概念组合在一起, 往往可以聚合出更高层语义. 例如, 当代码中出现“regex”“pattern”“match”, 可以得出“regular expression”“pattern match”和“string algorithm”等更高层、可读性更强的语义. 而高频 API 或代码概念提及往往表示具象的意思或者某些频繁操作, 甚至是无用的, 例如“*System.out.print*(·)”.

为了解决代码语义标签生成问题, 本文提出了一种基于知识图谱的代码语义标签自动生成方法 KGCodeTagger. 该方法可分为软件知识图谱构建和代码语义标签生成两个部分, 其中, 软件知识图谱构建是离线的, 而代码语义标签生成是在线的. 离线的软件知识图谱的构建主要依赖于对 API 文档和软件开发问答文本进行概念和关系抽取, 以及融合软件领域和通用领域的外部知识. 构建的软件知识图谱是代码语义标签生成的基础.在线的代码语义标签生成环节接受代码作为输入, 首先识别并抽取出代码中的通用 API 调用或概念提及, 并链接到软件知识图谱中的相关概念上. 在此基础上, 借助软件知识图谱丰富的语义关系, 进一步识别与所链接的概念相关的其他概念作为候选, 然后按照多样性和代表性排序产生最终的代码语义标签.

本文对软件知识图谱构建关键步骤的质量进行了评估. 实验结果表明, 知识图谱质量较高, 构建步骤合理. 此外, 本文还选取了 20 个代码片段, 将 KGCodeTagger 生成的标签分别与 Stack Overflow 帖子标签、TextRank[13]和 TagRCNN[8]这 3 个基线方法标签进行对比. 实验结果表明, 本文生成标签符合代码意图, 能够帮助理解代码语义.

本文的主要贡献: (1) 提出了软件领域术语抽取和软件知识图谱构建方法, 并构建了节点规模为 288 267、关系规模为 1 018 919 的软件知识图谱; (2) 提出一种自动化代码语义标签生成技术 KGCodeTagger; (3) 进行了两个实验, 分别验证了软件知识图谱构建的合理性以及代码语义标签的可靠性和有效性.

本文第 1 节介绍标签生成技术与软件知识图谱构建的背景知识和相关工作. 第 2 节阐述软件知识图谱构

建和代码语义标签生成的原理. 第 3 节描述软件知识图谱和代码语义标签生成的实现细节. 第 4 节通过实验分析, 验证本文方法的合理性与有效性. 第 5 节是总结.

1 背景及相关工作

本文通过软件知识图谱生成高层代码语义标签, 其核心是软件领域知识图谱构建和标签生成技术. 其中, 软件领域术语列表抽取是软件知识图谱的一个重要环节, 其目的是从相关的领域描述文本中抽取出高质量的软件术语. 近年来已有工作能够做到自动抽取[5,7,14], 但由于噪音的存在以及过分依赖于模型和标注数据[15,16], 仍然有很大的提升空间. 而标签技术则是一项比较成熟的技术, 现有许多关于社交网站和软件信息网站标签推荐的研究工作[4−6,9−11,17−19]. 早在 2010 年, Gupta 等人[20]就对社会标签技术做了一项综述调查. 他们从不同方面研究标签技术, 讨论了人们使用标签的原因、影响标签选择的因素、标签种类和标签生成技术等话题. 但是目前, 仍然缺少代码语义标签生成的工作研究. 接下来将依次介绍软件领域术语抽取、软件知识图谱构建、软件信息网站标签生成和代码语义理解的相关研究.

早期的软件领域术语列表抽取工作使用基于统计的启发式规则方法. Arora 等人[21]通过词性分析, 从自然语言文本中抽取所有名词短语, 并按启发式规则过滤掉常用的名词后作为术语列表. 而 Dwarakanath 等人[14]则结合词性和词频统计, 从需求文档中自动识别出描述性词汇表. 由于传统方法的准确性不高, 一些研究工作提出了使用深度学习模型或者对比通用语料库的方式来提高准确性. Chen 等人[15]通过对比 Stack Overflow 文本语料和 Wikipedia[22]文本来识别特定软件术语, 并且还做了同义词识别, 用以提高该软件术语列表的通用性和实用性. Wang 等人[16]提出了一种基于学习的方法, 该方法使用一组从代码标识符和自然语言概念中识别出的高质量种子术语训练特定领域预测模型, 从源代码和软件文档中自动构建领域术语表. 这些工作的共同点在于都依赖于模型或者标注数据. 为此, 本文使用无监督学习的领域短语挖掘, 在保证准确性的前提下降低抽取成本.

在术语列表抽取技术的基础上, 软件知识图谱构建快速发展. Karthik 等人[23]使用信息提取技术, 从 Q&A 网站发布的非结构化文本中自动识别 3 种不同类型的兼容性关系. Zhao 等人[24]结合规则模板和句子依赖分析, 从软件相关描述文本中抽取三元组, 构造了软件领域知识图谱. 王飞等人[25]针对软件知识代码图谱构建做了综述研究, 探讨了代码知识图谱的建模与描述、构建与精化、存储与演化管理、查询语义理解以及智能化应用这 5 个方面的研究趋势. 软件知识图谱根据其来源、内容以及应用对象的不同, 可以分为不同类型的图谱. Liu 等人[26]从 API 参考文档中自动提取 API 描述句子, 构建了一个包含 API 元素和 API 描述信息的 API 知识图谱, 用于生成特定编程任务的 API 摘要. 在之后的工作中, Liu 等人[27]又从 API 参考文档中自动提取 API 比较知识, 构建了包含 API 元素、功能规范、使用规范、特性说明、特性比较等内容的 API 知识图谱, 以支持对两个 API 类或方法进行功能、特性和分类的比较. Sun 等人[28]使用特定的开放信息提取技术, 从 Stack Overflow 的“How to”开头的帖子中提取任务内容、任务属性和任务关系的候选对象. 最终构建了包含 3 种活动关系和 5 种任务属性的知识图谱 TaskKG, 实现以任务为中心的知识搜索. 而本文构建的软件知识图谱不同于上述图谱, 本文通过从 JDK 1.8 文档[29]、Android 27 文档[30]以及 Stack Overflow 帖子的描述文本抽取软件领域术语, 融合外部知识, 最终构成软件知识图谱.

标签技术领域是成熟的研究领域, 现有多种针对不同类型网站(如社交网站[17−19]和软件信息网站[4−6,9−11])推荐或预测针标签的方法. Wang 等人[17]和 Zhao 等人[18]利用主题模型和协作过滤技术研究微博标签推荐. 而 Nguyen 等人[19]考虑用户的偏好以及视觉信息, 采用了卷积神经网络(CNN), 通过有监督的方式从图像中获得视觉特征, 并结合文本信息进行标签推荐. 这种方式比仅仅基于标签历史信息的技术提高了准确性. 在软件信息领域, 也有基于历史标签信息以及文本信息进行标签推荐的研究工作. Zhou 等人[9]提出了 TagMulRe, 通过使用信息检索的方式返回相似帖子的标签作为当前帖子的标签. Wang 等人[4]提出了 EnTagRec, 使用有监督的学习方式, 结合贝叶斯技术和频率学思想, 为新的软件文本推荐标签. Xia 等人[10]从历史软件对象文本和标签构建学习模型, 他们构造了多标签分类学习组件、相似度排序组件和标签术语组件, 并综合这 3 个组件的

评分给软件对象生成标签. Liu 等人[5]提出了基于神经网络分类的自动可扩展标签推荐方法 FastTagRec. Wang 等人[6]则结合卷积神经网络模型和协同过滤方法，通过学习 Stack Overflow 帖子的文本信息和标签，为新的帖子推荐标签. 由于标注数据的匮乏以及人工标注成本的高昂，Chen 等人[7]提出了半监督的标签推荐方法. 该方法使用已标记数据集和扩展标记数据不断迭代训练模型，给 Docker 存储库文本推荐标签. 由于仅从文本中学习存在局限性，Yang 等人[11]在推荐过程中则结合了代码片段、文本内容和户对标签的偏好. Zhou 等人[8]对前面所描述的方式进行了性能对比，他们比较了 3 种传统方法 EnTagRec[4], FastTagRec[5]和 TagMulRec[9]，以及 4 种深度学习方法 TagCNN, TagRNN, TagHAN 和 TagRCNN，证明了在标签推荐任务中，TagCNN 和 TagRCNN 方法的性能优于传统方法. 本文第 4.2 节将 TagRCNN 作为实验的对比方法之一.

在代码语义理解领域中，代码摘要(又称代码注释生成)和本文的代码语义标签生成目的最相近，都是为了帮助用户理解代码语义. 早期的代码摘要使用基于模板的方法[31−34]依赖预定义的规则或模板. 而基于信息检索的方法[35,36]依赖于相似的代码片段选择来合成注释. 随着大型源代码库的出现，基于深度学习的代码摘要[37−39]在很大程度上超过了基于模板和基于信息检索的方法. Sutskever 等人[40]使用 Seq2Seq 模型，用带有编码器-解码器架构和注意机制的网络模型来处理基于序列的源代码表示，并逐字生成自然语言总结. LeClair 等人[41]提出了一种代码摘要方法，是目前最为先进的方法之一. 该方法在 Seq2Seq 模型中结合了来自代码的单词和来自抽象语法树(AST)的代码结构的扁平表示. 而本文提出的代码语义标签自动生成方法采用无监督、非学习的方式，可以用更小的成本帮助用户理解代码语义.

2 方 法

针对代码片段，本文提出了基于软件知识图谱的代码语义标签自动生成方法 KGCodeTagger. 通过构造软件知识图谱，为代码生成具有更高层语义的标签. 方法的输入是一段具有实际功能的代码片段，输出是前 k 个代码语义标签. 图 1 展示了方法的流程，可分为软件知识图谱构建和代码语义标签生成两个部分. 其中，软件知识图谱构建是离线的，而代码语义标签生成是在线的.

- 第 1 部分软件知识图谱构建主要由术语列表抽取、关系抽取和外部知识融合这 3 个步骤组成. 具体实现为：输入领域文本语料，使用无监督的方式进行软件术语列表抽取；然后，通过 Hearst Pattern[42]和启发式规则给软件术语添加节点关系；最后，融合软件领域知识和通用知识.
- 第 2 部分代码标签生成主要由代码概念提及抽取、候选标签生成和候选标签排序这 3 个步骤组成.具体实现为：从输入的代码片段中抽取代码概念提及；然后，将代码概念提及映射到软件知识图谱,寻找高层语义软件概念作为候选标签；最后，通过排序算法，为每个候选标签进行打分，按照得分从候选标签中筛选出最能概括代码语义的前 k 个概念作为代码的语义标签.

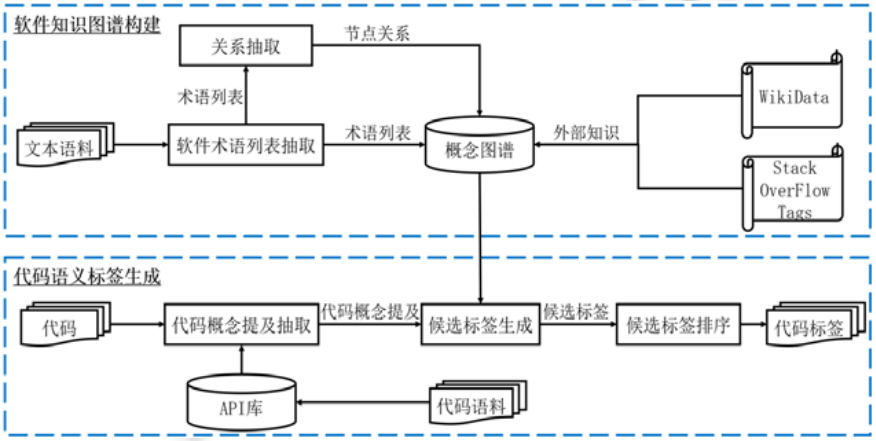

图 1 代码语义标签生成流程

2.1 代码语义标签生成示例

借助软件知识图谱丰富的外部信息, KGCodeTagger 生成的语义标签并非停留在代码提及表层, 而是具有高层语义理解能力的标签. 图 2 展示了一段用正则表达式解析字符串的代码. 图 3 展示了 KGCodeTagger 为这段代码生成语义表标签的过程.

- 首先, 使用第 2.3.1 节中的代码概念提及抽取方法, 从代码元素中(图 2 加粗部分)中过滤掉不重到要的“API RegexDriver”和“System.out.println”, 得到代码概念提及列表[“regex”,“matcher, match”,“pattern”,“string”,“compile at pattern”,“matcher for pattern”,“match for matcher”,“format for string”].
- 然后, 按照第 2.3.2 节的要求进行概念映射, 得到映射后的 9 个软件概念节点(底色为灰色的节点). 以这 9 个节点为起点进行概念扩展, 得到图 3 的 17 个软件概念节点. 其中, “regular expression”“pattern matching”“string algorithm”“text”“series”是软件概念图谱通过融合外部知识引入的, 它们没有直接出现在代码中, 但是与代码密切相关. 本文综合考虑了标签的多样性和主题概括能力, 从这 17 个概念筛选出候选标签(第 2.3.2 节), 然后对候选标签进行排序(第 2.3.3 节). 最终得到前 5 个概念节点作为代码语义标签.

图 3 中虚线边框节点表示最终产生的代码语义标签, 包括“regular expression”“pattern matching”“string algorithm”“format”“compile”.

```
package regex;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegexDriver {
    public static final String PATTERN = "^(\\w+[+])?\\w+[*]\\w+[|]\\w+$";
    public static void main(String[] args) {
        Pattern pattern = Pattern.compile(PATTERN);
        for (String arg: args) {
            Matcher matcher = pattern.matcher(arg);
            boolean isMatch = matcher.matches();
            System.out.println(String.format("'%s' match? %s", arg, isMatch));
        }
    }
}
```

Tags: regular expression, pattern matching, string algorithm, format, compile

图 2 高层代码语义标签举例

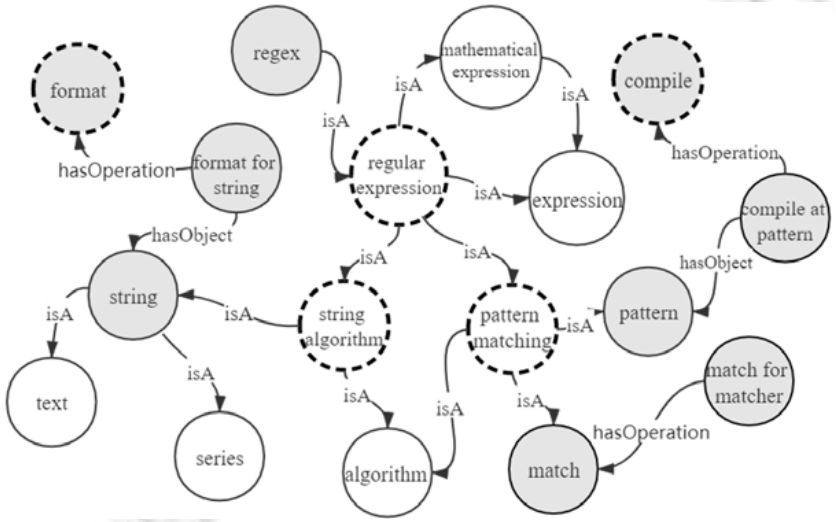

图 3 代码语义标签生成展示

2.2 软件知识图谱构建

软件知识图谱是指由从软件领域文本中抽取软件领域术语构成的知识图谱, 其构建的流程与通用知识图谱构建相似. 将 JDK 1.8 文档、Android 27 文档以及 Stack Overflow 帖子的文本作为语料, 从中抽取出软件领域术语列表, 添加节点关系, 生成软件知识图谱. 在软件术语抽取中主要通过词性分析、句子成分分析和 *N*-Gram 抽取名词、动词、动宾短语. 动词和动宾短语是刻画 API 及代码功能的关键[43], 因此, 本文在软件术语列表抽取中抽取动词和动宾短语, 丰富软件知识图谱的连接. 为了使软件图谱有更强的概念理解能力和更深的语义概括能力, 本文添加了 WikiData[44]节点和来自 Stack Overflow 的标签用于后续的代码语义标签生成. 最终生成如图 4 所示的软件知识图谱. 其中, 软件概念节点主要由名词短语、*N*-Gram 短语和 Stack Overflow 标签组成; 功能短语节点由动宾短语和 *N*-Gram 短语组成; 功能动词由动词抽取得到; 通用概念节点来由软件相关 WikiData 节点组成. 图上的边代表了节点间的关系(图上只标注了单向边, 详见第 2.2.2 节).

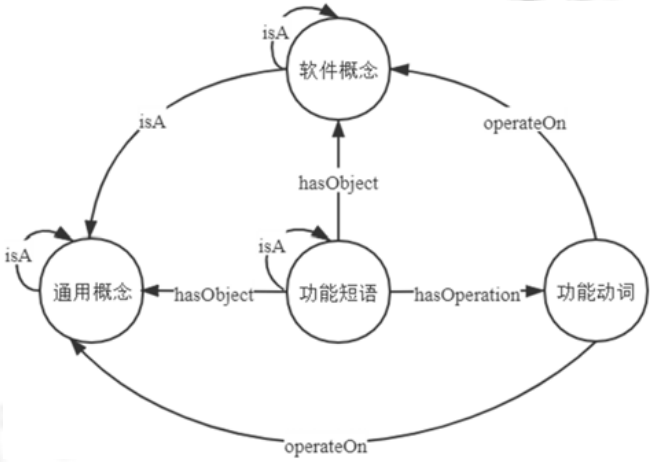

图 4 软件知识图谱高层模式

2.2.1 软件术语列表抽取

本文参考了肖仰华等人[45]的基于无监督学习的领域短语挖掘流程: 使用 N-Gram 生成候选短语、计算统计特征和质量评分排序. 在上述流程的基础上, 本文进行了改进. 图 5 展示了改进后的软件术语列表的抽取过程,实现细节见第 3.2.1 节

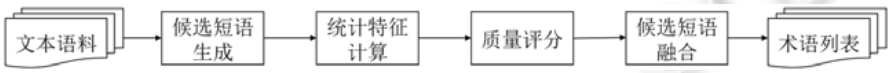

图 5 软件术语列表抽取流程

(1) 候选短语生成

使用 *N*-Gram 可以生成长度为 *N* 的短语, 但不一定符合语法规则, 需要进行过滤. 除使用 *N*-Gram 之外, 还需进行扩展和完善, 以满足术语列表多样化的需求.

a) 使用 Spacy[46]对文本句子进行词性分析, 抽取名词短语、动词、动宾短语.

b) 对文本语料进行预处理, 筛选出高频 *N*-Gram 短语.

c) 将高频 *N*-Gram 短语和 Spacy 短语取并集, 作为候选短语.

(2) 统计特征计算

给候选短语计算 TF(频率)、TF-IDF(频率-逆文档频率)、PMI(点互信息)、左邻字熵和右邻字熵这 5 个统计指标. 其中, TF 和 TF-IDF 是评估短语重要性的指标. TF-IDF 指标与其在语料中出现的频次成正比, 与其在外部语料中出现的频次成反比. 本文引入 NLTK[47]中的 webtext, brown 和 reuters 作为统计 IDF 的外部语料.

PMI 和左、右信息熵也是术语列表抽取中常用的指标: PMI 刻画短语成分间的一致性, 左邻字熵和右邻字熵则可以反映出信息量. PMI 和左、右信息熵通常组合使用, 实现从陌生文本中自动发现新短语.

(3) 质量评分

根据各项指标得分将候选短语从高到低排序.

(4) 候选短语融合

进行同义词合并. 无论是在软件领域[27]还是其他领域[48], 都有研究工作使用 WordNet[49]进行同义词或反义词匹配. 本文同样使用 WordNet 进行同义词匹配. 若是成功匹配多个互为同义词的术语, 保留其中的一个术语, 其余同义词作为该术语的别名存储.

2.2.2 关系抽取

本文使用改进的 Hearst Pattern[42]和启发式规则抽取双向边关系, 包括上下位关系“isA”和“hasA”, 以及自定义类型关系“operateOn”和“hasOperation”, “hasObject”和“objectOf”.

Hearst Pattern[50]是一种基于模式的上下关系抽取方法. 首先, 人工定义一系列模板; 然后, 将模板与文本进行匹配, 提取匹配成功的槽值. 例如, 利用模板“NP_\w+,? such as (NP_\w+(,|and|or)? ?)”可以从“I used programming language such as Java”中抽取出“Java” isA “programming language”. Seitner 等人[42]对 Hearst Pattern 模板进行扩展, 扩展后的模板能够涵盖更多句式. 本文采用扩展后的模板进行上下位关系抽取.

由于模板的有限, 仍然存在部分关系未被抽取出来. 本文自定义了两种双向边关系“operateOn”和“hasOperation”、“hasObject”和“objectOf”. 针对这部分未抽取出的上下位关系和自定义的关系, 本文采取启发式规则, 利用关系短语的前后缀补全关系边. 当两个短语都是名词时, 则添加上下位关系“isA”和“hasA”. 例如, “list”是“array list”的后缀, 添加上下位关系“array list” isA “list”. 当存在功能动词且前后缀匹配, 则添加“operateOn”和“hasOperation”. 例如, “add”是“add to list”的前缀, 添加关系“add to list” hasOperation “add”. 当存在功能短语且另一个术语不是功能动词, 则添加“hasObject”和“objectOf”. 例如, “list”是“add to list”的后缀, 添加关系“add to list” hasObject “list”. 为防止引入过多噪音, 本文对基于前后缀添加节点关系进行以下限制.

(1) 前缀和后缀必须是一个完整的单词, 且单词长度大于 3.

(2) 前缀和后缀不在停用词列表中.

(3) 前缀和后缀中不包含任何特殊符号和数字.

(4) 前缀和后缀必须是词干形式, 任何其他形式必须先词干化.

2.2.3 外部知识融合

(1) 软件领域知识融合

图 6 展示的 Stack Overflow 标签信息中包含标签名字、同义标签、Wikipedia 标签描述等信息. 添加 Stack Overflow 标签信息可以丰富软件知识图的节点. Stack Overflow 网站虽然已经人工合并部分同义标签, 但是仍存在部分低质量标签. 因此, 本文首先对 Stack Overflow 的标签进行一个预处理, 合并同义的 Stack Overflow 标签(第 3.1.3 节), 例如, 将“sorted”“sorting”和“sort”合并为“sort”. 同时, 将 Stack Overflow 标签中 Wikipedia 信息对应的 WikiData 节点直接添加到软件知识图谱中.

图 6 Stack Overflow 标签信息举例

(2) 通用知识融合

与之前 Liu 等人的工作相似[26], 本文首先使用神经网络训练分类器从 WikiData 节点中获取软件领域概念节点, 然后通过比较节点相似度将其链接到软件知识图谱中. Word2Vec[51]是只有一个隐层的全连接神经网络, 用其训练的词向量能够包涵语义信息, 向量的夹角余弦能表示单词的相似度. 因此, 本文使用 Word2Vec 为每个单词训练词向量, 然后将节点名称中每个单词的平均词向量作为节点词向量. 在融合前, 计算软件术语节点和 WikiData 节点的词向量相似度, 如果词向量相似度大于阈值α, 则将这两个点融合, 并将该 WikiData 节点自身的上下位节点加入图中. 最后, 使用第 2.1.2 节的关系抽取方法为所有新加入的节点更新关系.

2.3 代码语义标签生成

我们分 3 节来阐述代码标签生成的原理: 代码概念提及抽取、候选标签生成、候选标签排序.

2.3.1 代码概念提及抽取

在一段可以执行的代码中, 有类、方法、变量. 本文从类和方法的签名以及方法的实例进行代码概念提及抽取. 代码概念提及抽取流程包含 3 个部分: API 抽取以及 API 库的增量式构建、API 签名分析、API 实例分析.

(1) API 抽取以及 API 库的增量式构建: 这是一个离线处理操作, 即从代码语料中抽取出所有 API 和 API 实例预先构建一个 API 库. Baker[52]是一个实现 API 全限定名精准链接的方法, 例如, “String”链接到“java.lang.String”. 本文复现了 Baker, 将代码片段中的 API 链接到它们的全限定名 API, 完成 API 抽取. 然后, 使用正则表达式逐行进行 API 实例抽取. 将所有 API 和 API 实例增量地添加到 API 库中, 并计算和保存它们的 TF-IDF 值, 用于后续的 API 过滤.

(2) API 签名分析: 对于一个需要生成语义标签的代码片段, 使用步骤(1)中的 Baker 抽取 API 签名. 利用 API 库进行对比, 结合 TF-IDF 指标过滤无意义且不重要的 API, 例如“*System.out.Print*(·)”等. 将筛选后的 API 按照驼峰式命名进行拆开, 并保留原名作为别名.

(3) API 实例分析: 首先抽取 API 实例, 然后按照步骤(2)中结合 TF-IDF 指标的方式进行过滤, 保留有意义的方法实例. 受到 Software Word Usage Model (SWUM)[53]获取方法操作的启发, 本文在不考虑参数的前提下实现了一个操作捕获方法. 例如, “list.add”抽取出“add to list”, “array.size”抽取出“get size of array”, 并保留原名作为别名.

2.3.2 候选标签生成

通过上一节, 从代码中获取代码概念提及列表. 接下来需要进行概念映射、高层软件概念扩展和候选标签筛选, 即将代码概念提及链接到软件知识图谱中. 然后以链接上的概念为起点向外游走, 扩展概念, 筛选候选标签. 通过该步骤, 可以得到在代码中没有出现但与其密切相关的、更加抽象的概念, 例如“regular expression”的高层软件概念“pattern-matching”.

(1) 概念映射

概念映射的实质, 是将字符串映射到知识库对应的实体上. 当存在同名不同义的实体时, 需要进行实体消歧. 在构建软件知识图谱时, 本文使用的文本语料都是关于软件领域的描述文本, 还融合了部分同义词节点. 因此, 在软件知识图谱概念映射过程中不需要进行实体消歧. 本文采用文本匹配的方式进行概念映射, 即判断代码术语提及和软件概念名字或软件概念别名或去除停用词的软件概念名字是否一样. 对于没有映射成功的软件术语提及, 它们可能不包含重要信息导致链接失败, 且不能在图上完成高层概念节点扩展.

(2) 概念扩展

概念扩展指从映射成功的软件概念出发, 按照特定的游走规则寻找高层软件概念. 在游走的过程中, 设置判断停止条件来判断游走状态. 本文使用相似度线性函数 y=0.8+0.04×*step* 作为停止条件, 其中, *step* 为跳数,其意义是往外游走跳数越大, 对节点与其所在群体的平均相似度要求越高.

- 基础定义

a) 定义软件知识图谱为 *G*, 节点集合 *N*, 图上的单个软件概念节点 *n*.

b) 代码概念提及列表 L, 映射到软件知识图谱的软件概念节点列表 M, 节点列表 M 之间的关系 R.

c) 节点 n 的词向量表示 $V(n)$.

d) 节点 n_1 和节点 n_2 的余弦相似度 $Simcos(n_1,n_2)$.

e) 节点 n 所在的联通子图 C_n.

- 游走判断条件

从软件概念节点列表 M 出发, 向外扩展, 寻找语义相近高层概念. 本文训练词向量模型, 使用词向量表示节点, 具体操作与第 2.2.3 节一致. 用余弦相似度表示高层软件概念节点 k 与图上节点 n 的相似度. 相似度值越大, 节点间语义越接近. 而高层软件概念节点 k 能否被保留, 取决于高层软件概念节点 k 与它的子节点 n 所在的连通图 C_n 的相似度 $Sim(k,C_n)$. 如果相似度 $Sim(k,C_n)$满足游走条件, 则添加高层概念节点 k, 并以高层概念节点 k 为起点继续向外扩展, 直到满足停止条件.

a) 定义 $Simcos(k,n)$为高层软件概念节点 k 与图上软件概念节点 n 的余弦相似度, 其计算方式为

$$Simcos(k,n)=\frac{V(k)\times V(n)}{\|V(k)\|^2\times\|V(n)\|^2} \tag{1}$$

b) 定义 $Sim(k,C_n)$为高层软件概念节点 k 与子概念节点 n 所在连通图 C_n 的相似度, 其计算方式为

$$Sim(k,C_n)=\frac{\sum_{n\in C_n}Simcos(k,n)}{Size(C_n)} \tag{2}$$

其中, $Size(C_n)$表示联通子图 C_n 的软件概念节点数目.

c) 满足以下两个条件, 高层软件概念节点 k 保留, 否则停止游走:

$$\begin{cases}Sim(k,C_n)>0.8+0.04\times Step\\ \min(Simcos(k,n))>0.6\ (n\in C_n)\end{cases} \tag{3}$$

其中, $Step$ 为游走的跳数.

(3) 候选标签筛选

在完成节点概念扩展后, 下一步生成候选标签列表. 经典的方式是用聚类的方式从所有簇中筛选出最具代表的节点. 本文根据自身需求, 以联通子图为单位进行排序, 利用图本身的结构进行排序筛选. 要求同一个联通子图中筛选出来的概念节点必须最能代表该子图中的所有节点的语义, 而不同联通子图中筛选出来的概念节点又必须满足差异最大, 从而保证使用最少的标签概括代码意图.

联通子图内概念节点的排序由两部分得分构成: 与该子图内所有其他节点的平均相似度(公式(2))、节点热门程度. 与子图内所有其节点的平均相似度越大, 则代表该节点与联通子图内的所有节点语义最相似; 节点热门程度使用节点的入度来表示, 节点入度越大, 则概括性越强. 联通子图内概念节点得分计算方如下:

a) 定义 $InDegree(n)$为节点 n 的入度, $InDegreeScore(n)$为节点入度得分, 将 $InDegreeScore(n)$进行 min-max 标准化后, 其计算方式为

$$InDegreeScore(n)=\frac{(InDegree(n)-InDegreeMIN)}{(InDegreeMAX-InDegreeMIN)} \tag{4}$$

其中, $InDegreeMAX$ 和 $InDegreeMIN$ 分别表示联通子图内节点入度的最大值和最小值.

b) 定义 $Score(n)$为节点 n 在联通子图 C_n 内的得分, 其计算方式为

$$Score(n)=w\times Simc(n,C_n)+(1-w)\times InDegreeScore(n) \tag{5}$$

其中, w=0.7 表示权重.

每个联通子图内的概念节点, 按照联通子图自身规模的比例, 筛选出候选标签节点.

2.3.3 候选标签排序

来自不同的联通子图候选标签 n_1 和 n_2, 其差异性用余弦相似度表示 $Sim_{cos}(n_1,n_2)$, 计算公式如公式(1)所示. 将每个联通子图内节点的候选标签节点按照组合的方式, 计算出相似度之和最小的组合, 即为最终的标签节点. 节点的名字即为最终的代码语义标签.

完整的代码标签生成算法如算法 1 所示.

算法 1. 标签生成算法.

输入: 代码片段 *C*.

输出: 代码语义标签.

```
1.  function TagGeneration(C)
2.  代码概念提及抽取和概念映射, 得映射概念节点列表 M 和节点关系 R
3.  需要往外扩展节点列表 N=M
4.  获取当前图结构的所有联通子图 connectedGraphs
5.  while N≠∅:
6.    upperNodes                                  //新加的往外扩展节点集合
7.    newRelation                                 //新加的节关系
8.    for current_node in N:
9.      获取当前节点的父节点 parent_nodes
10.       for parent_node in parent_nodes:
11.         if parent_node add into upperNodes:   //判断父节点是否保留(式 3)
12.           upperNodes.add(parent_node)
13.           newRelation.add(current_node,parent_node)
14.         end if
15.       end for
16.     end for
17.     更新联通子图 connectedGraphs
18.     N=upperNodes
19.   end while
20.   candidateForConnectedGraph                  //各联通子图的候选节点
21.   finalNodes                                  //最终输出节点
22.   for graph in connectedGraphs:
23.     for node in graph:
24.       计算节点得分 score(公式(5))
25.     end for
26.     获取当前联通子图中的候选节点 currentCandidates
27.     candidateForConnectedGrapha.update(currentCandidates)
28.   end for
29.   从各联通子图的候选节点中筛选标签节点 finalNodes
30. return finalNodes
```

3 方法实现

基于第 2 节阐述的方法, 本节描述软件知识图谱构建和代码语义标签生成的相关细节.

3.1 数据准备

3.1.1 数据获取

本文获取 JDK 1.8 文档[28]、Android 27 文档[29]以及按照标准筛选符合要求的 Stack Overflow 帖子. 共获取 2 121 个 JDK 页面、4 619 个 Android 页面、250 821 个 Stack Overflow 帖子. 将上述 3 个部分数据中的文本作为构建软件知识图谱的文本语料, 帖子中的所有代码作为构建 API 库的代码语料. 帖子筛选标准如下.

(1) 标题“How to”开头.

(2) 具有“java”或者“android”标签.

(3) 至少有一个得分大于 0 的回答.

(4) 创建时间为 2016 年 3 月以前.

(5) 包含不大于 50 行代码片段.

3.1.2 数据处理

本文获取的原始数据都是 html 格式, 因此使用 BeautifulSoup[54]进行解析. 将其中的代码部分取出, 作为代码语料. 然后用占位符“—CODE—”代替代码, 不影响句子结构, 从而防止使用 Spacy 时发生错误. 由于文本中通常包含 API 或者特殊的学术短语, 容易造成 Spacy 分句错误. 例如, 将 post id 为 18158891 的帖子的标题“How to call String.Split that takes string as separator?”误分为“How to call String.”和“Split that takes string as separator?”. 本文在 Spacy 中添加了新的分句规则来避免这种错误, 从而提高准确率.

3.1.3 Stack Overflow 标签预处理

首先对 Stack Overflow 标签页面中明确指出的标签同义词(图 6 所示)进行合并. Stack Overflow 标签中还存在一些同义但表现形式不一的标签, 例如“sorted”“sorting”和“sort”. 本文将 Stack Overflow 标签以“-”和空格为分隔符进行分词, 使用 NLTK 对每一个单词进行词干化处理, 合并同名标签, 例如, 将“sorted”“sorting”和“sort”合并为“sort”. 由于 NLTK 不能对所有标签进行词干化处理, 经统计, 在 NLTK 词干化处理后, 仍然有 4 020 个以“ing”“ed”“s”结尾的标签. 在这 4 020 个标签中, 有 408 个以“ing”结尾; 154 个以“ed”结尾, 其中 2 个以“ied”结尾; 3 458 个以“s”结尾, 其中 294 个以“es”结尾. 鉴于这 4 020 个标签中交叉着需要词干化的普通单词和无须词干化的专有名词, 情况复杂, 无论是否软件专有名词, 本文统一对所有标签进行分词, 将末尾单词长度大于 7 的标签按照英文语法规则去除末尾的“ing”“ed”“s”. 对所有处理后的标签再次进行合并, 其余未合并的标签仍然以 Stack Overflow 上原有的标签名字存储.

3.2 软件知识图谱构建实现

3.2.1 软件术语列表抽取实现

将文本语料预处理后, 使用 Spacy 进行词性分析, 抽取出名词短语、动词和动宾短语. 计算 902 850 个名词短语、动词和动宾短语的长度, 其平均值为 2.8, 中位数是 3. 因此, 去除文本中的标签和其他特殊符号, 生成 N=3 的 N-Gram 短语. 对所有短语进行统一处理.

(1) 删除首位和末尾停用词和介词, 例如, “of array list”去除首位停用词后为“array list”.

(2) 删除中首位和末尾的数字, 例如, “str1”去除末尾数字后为“str”.

(3) 词干化处理.

(4) 过滤介词和副词开头结尾的短语.

然后统计特征评分, 排序筛选候选短语. 本文计算了 TF、TF-IDF、PMI、左邻字熵和右邻字熵这 5 个统计指标. 保留所有 TF 大于 5 的动词. 剩余的候选短语, 保留 TF-IDF 值前 20%、PMI 值前 20%且左邻字熵和右邻字熵均大于 5 的交集. 融合 5 914 个同义词后, 最终获得 206 468 个软件术语.

3.2.2 关系抽取实现

从开源仓库[55]获取 Hearst Pattern 的源代码, 完成第 1 轮上下位关系抽取. 然后按照第 2.2.2 节的启发式规则, 完成第 2 轮基于前后缀添加节点关系.

3.2.3 外部知识融合实现

在第 3.1.3 节已经完成 Stack Overflow 标签预处理. 本文将标签对应的 WikiData 节点直接添加到软件知识图谱中, 对于不存在 WikiData 节点的标签, 本文也将其直接加入到软件知识图谱中, 更新关系, 完成软件领域知识融合. 最终添加 54 973 个 Stack Overflow 节点. 与 Liu 等人[26]的工作相似, 本文使用神经网络得到 78 182 个 WikiData 软件概念节点. 然后从开源仓库[56]获取 100 维词向量, 添加本文的文本语料, 使用 Gensim 库[57]再次训练, 生成最终的词向量模型. 在第 2.2.3 节、第 2.3.3 节和第 2.3.4 节中, 节点的词向量表示均为节

点和它所有别名的平均向量. 本文将 26 826 个相似度α大于 0.8 的 WikiData 节点加入软件知识图谱, 并更新节点关系, 完成通用知识融合. 通过上述步骤, 最终生成节点规模为 288 267、关系规模为 1 018 919 的软件知识图谱. 将软件知识图谱存储在 Neo4j[58]图数据库中.

3.3 代码标签生成实现

代码语义标签生成过程分为 3 步: 代码概念提及抽取、候选标签生成和候选标签排序(算法部分见第 2.3 节). 在代码概念提及抽取中, 用于过滤非重要 API 的 API 库是从 250 821 个 Stack Overflow 帖子的代码片段中抽取构成. Baker 则是在 Mikolov 等人[52]思想的基础上复现得到的. 在复现过程中, 本文对 Baker 进行了两点修改.

(1) 将所有来自 JDK1.8 和 Android27 的 API 作为数据库, 实现所有 Java 和 Android 的 API 的链接.

(2) 优化节点信息抽取方式. 针对不可编译且不能生成抽象语法树的代码, 根据 API 的命名规则进行切块提取信息, 然后与 API 库进行匹配.

4 实验分析

本节通过实验分析构建软件知识图谱过程的合理性和代码语义标签的有效性. 下面分别从软件术语抽取、关系抽取、外部知识融合这 3 个步骤的合理性进行实验分析, 用来评估软件知识图谱的质量. 然后, 将 3 个对比方法生成标签与本文方法 KGCodeTagger 生成的代码标签进行比较, 用来评估代码语义标签的有效性. 最后, 讨论部分指出了 KGCodeTagger 的几点局限性.

4.1 构建软件知识图谱实验

为了确保观察到的样本比值在一定的置信区间内, 且在一定的置信水平上推广到总体. Singh 等人[59]得出 95%置信水平、置信区间为 5 时, 所需样本容量为 384 的结论. 为了评估软件知识图谱的质量, 本文分别从软件术语抽取、关系抽取、外部知识融合这 3 个步骤中随机筛选了 384 个软件术语、384 个节点边、384 个融合链接结果作为实验数据. 然后, 挑选了 6 位具有 Java 和 Android 开发经验的硕士研究生, 将他们平均分为 3 组, 每组做一个步骤的实验. 实验人员在每条实验数据后面进行准确性判断, 认为正确则标注 1, 认为不正确则标注 0. 当每组的两位实验人员标注不一致时, 由其中一位作者进行仲裁.

本文统计了科恩 Kappa 系数[60]和准确率来判断软件知识图谱的质量, 其中,

- Kappa 系数是一种统计测量分类项目评分者间一致性的统计指标. 当科恩 Kappa 系数在[0.61–0.8]之间时, 认为评分者基本一致; 当科恩 Kappa 系数在[0.81–1]之间时, 表示评分者几乎完全达成一致.
- 准确率是根据仲裁结果计算.

实验结果见表 1.

表 1 软件知识图谱质量评估实验结果

项目名称	Kappa	准确率
软件术语质量	0.73	0.90
关系抽取质量	0.68	0.88
外部知识融合质量	0.78	0.93

由实验可知: 3 个实验项目的科恩 Kappa 系数都在[0.61–0.8]之间, 最高为 0.78, 实验人员评判基本一致; 而准确率都在 0.88 以上, 最高为 0.93. 本文对标注不一致或者实验人员都认为不合理的实验数据进行分析, 发现在过滤和融合后, 软件术语抽取仍会不可避免地引入噪音. 在软件术语中, 依然存在诸如“owner and environment”“public protect”之类排序得分较高的噪音, 从而影响后续的节点间关系. 而节点关系除了受软件术语的影响, 启发式规则关系抽取也有一定的局限, 例如, 生成“string algorithm” isA “string”和“string algorithm” isA “algorithm”这两个节点关系. 但是前者是错误的, 后者是正确的. 在外部知识融合中, 实验人员标注为 0 的有“project”融合“activity”等. 但这些噪音数量较小, 并且在后续的标签生成算法中不一定能够完

成概念映射, 对算法的影响在可接受的范围之内. 因此, 本文的构建软件知识图谱的每个步骤总体上是有效的, 图谱是高质量的.

4.2 代码语义标签对比实验

本文采用 3 个对比方法: Stack Overflow 帖子标签、TextRank[13]关键词抽取和 TagRCNN[8]模型. Stack Overflow 帖子标签, 它是用户在发布帖子时人工标注的标签, 有一定的概括性和主题. TextRank 算法是基于 PageRank, 为文本生成关键字和摘要的算法. 本文用 TextRank 算法为代码生成 5 个关键词作为其标签. 第 3 个对比方法 TagRCNN 算法是 Zhou 等人[8]对比 4 种深度学习和 3 种传统标签推荐算法时得出的性能较优的算法.

“image”“button”“arrays”“url”这 4 个标签在 Java 和 Android 较热门, 且差异较大. 因此, 本文在 Stack Overflow 上按照这 4 个标签分别筛选了 5 个带有小于 50 行代码片段代码的、不在软件图谱构建数据集中的帖子. 使用实验方法和 3 个方法分别为每段代码生成 5 个标签. 将这 20 个帖子的 Stack Overflow 标签作为第 1 个对比实验数据. 然后, 将 20 个帖子的代码按照标点符号和空格分开组成文本, 使用 TextRank 抽取 5 个关键词, 作为第 2 个对比方法实验数据. 接着, 使用 250 821 个 Stack Overflow 帖子训练 TagRCNN 模型, 预测每段代码的标签, 作为第 3 个对比实验数据. 最后, 使用 KGCodeTagger 为每段代码生成 5 个标签作为实验数据.

由于实验数据是 Java 和 Android 代码, 通过观察发现, 在标签中存在语言标签“java”和“android”. 在特定编程语言的情况下, 语言标签不具有语义, 在实验前将这类语言标签删除. 因此, 在综合 4 种方法生成的标签后, 每段代码存在不多于 20 个实验标签. 然后, 将 20 段实验代码分为两组, 每组 10 段代码. 对于每段代码, 将 4 种方式生成的标签混合. 实验人员并不知道某个具体标签是用何种方式生成的, 以保证实验的公平性和可靠性. 本文挑选了 6 位具有 Java 和 Android 研究背景和开发经验的硕士研究生, 将他们分为两组, 每组 3 人, 分别进行实验. 实验人员需要根据是否同意每一个标签是否符合代码意图且能够帮助理解代码语义进行打分: 1 分表示极度不同意, 标签完全不符合代码意图且不能帮助理解代码语义; 2 分表示有点不同意, 标签不符合代码意图但能帮助理解代码语义; 3 分表示有点同意, 标签基本符合代码意图且能帮助理解代码语义; 4 分表示特别同意, 标签完全符合代码意图且能够帮助理解代码语义. 本文统计了每段代码每种方法生成标签的平均分值和每个得分的频数, 最后求得所有代码的平均分值和每个得分的频数. 表 2 为实验统计结果, 图 7 为箱形图.

表 2 代码语义标签对比实验结果

Post id	Stack Overflow-Tag					TextRank				
	1 分	2 分	3 分	4 分	Avg.	1 分	2 分	3 分	4 分	Avg.
64270486	0	0	3	3	3.5	0	3	7	5	3.1
64215146	3	0	2	4	2.8	0	4	6	5	3.1
64010369	0	0	5	4	3.4	6	2	4	3	2.3
64002760	0	0	3	3	3.5	1	2	6	6	3.1
64783529	2	1	4	2	2.7	1	4	7	3	2.8
65172910	2	1	0	3	2.7	5	3	4	3	2.3
65222728	3	3	3	0	2.0	0	5	4	6	3.1
42266264	0	0	1	5	3.8	0	7	7	1	2.6
45237866	0	0	3	3	3.5	0	0	8	7	3.5
60786976	3	3	2	1	2.1	3	6	3	3	2.4
65128035	0	3	2	4	3.1	3	3	5	4	2.7
65222710	6	3	1	2	1.9	3	4	1	7	2.8
9572795	0	1	1	7	3.7	5	4	6	0	2.1
65089256	2	2	2	6	3.0	5	7	3	0	1.9
65220576	0	2	3	7	3.4	4	3	3	5	2.6
65123092	2	1	3	3	2.8	1	7	6	1	2.5
63996569	0	1	2	6	3.6	0	5	3	7	3.1
63660144	3	0	4	5	2.9	3	6	4	2	2.3
63618827	0	0	2	10	3.8	4	9	3	0	1.9
63589303	0	1	3	8	3.6	0	3	5	7	3.3
Sum / Avg.	26	22	49	86	3.1	44	86	95	75	2.7

表 2 代码语义标签对比实验结果(续)

Post id	TagRCNN					KGCodeTagger				
	1 分	2 分	3 分	4 分	Avg.	1 分	2 分	3 分	4 分	Avg.
64270486	5	1	5	4	2.5	1	0	9	5	3.2
64215146	9	2	3	1	1.7	3	1	7	4	2.8
64010369	12	0	1	2	1.5	0	2	6	7	3.3
64002760	7	1	4	3	2.2	5	2	5	3	2.4
64783529	12	0	2	1	1.5	0	3	7	5	3.1
65172910	8	2	1	4	2.1	1	2	7	5	3.1
65222728	5	2	4	4	2.5	3	2	5	5	2.8
42266264	3	3	6	3	2.6	3	3	5	4	2.7
45237866	2	6	4	3	2.5	0	4	4	7	3.2
60786976	10	3	2	0	1.5	1	2	9	3	2.9
65128035	2	4	5	4	2.7	0	3	7	5	3.1
65222710	0	0	7	8	3.5	1	3	5	6	3.1
9572795	0	4	4	7	3.2	0	5	5	5	3.0
65089256	2	4	5	4	2.7	1	9	5	0	2.3
65220576	0	2	4	9	3.5	0	0	2	10	3.8
65123092	2	6	7	0	2.3	6	4	4	1	2.0
63996569	8	2	3	2	1.9	0	1	4	10	3.6
63660144	7	2	4	2	2.1	4	1	8	2	2.5
63618827	4	5	6	0	2.1	4	3	3	5	2.6
63589303	0	4	1	10	3.4	0	4	4	7	3.2
Sum / Avg.	98	53	78	71	2.4	33	54	111	99	2.9

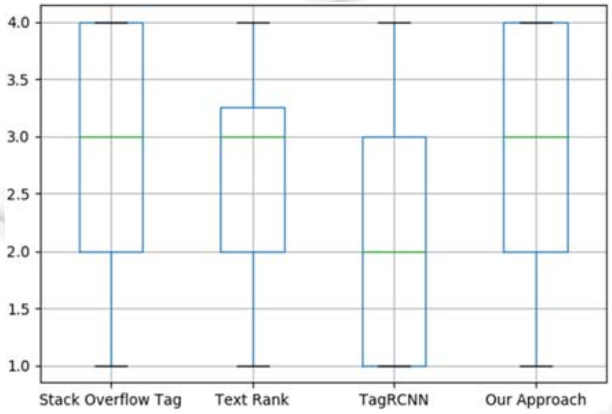

图 7 代码语义标签对比实验箱形图

通过表 2 和图 7 可知: Stack Overflow 标签中有 47.0% (86/183)评分 4 分, 73.8% (135/183)评分 3 分以上, 平均得分为 3.1 分; TextRank 标签中有 25.0% (75/300)评分 4 分, 56.7% (170/300)评分 3 分以上, 平均得分为 2.7 分; TagRCNN 标签中有 23.7% (71/300)评分 4 分, 49.7% (149/300)评分 3 分以上, 平均得分为 2.4 分; KGCodeTagger 标签有 33.3% (99/297)评分 4 分, 70.7% (210/297)评分 3 分以上, 平均得分为 2.9 分. 由此可见, Stack Overflow 标签和 KGCodeTagger 生成的语义标签比 TextRank 和 TagRCNN 算法生成的标签更准确, 语义更强, 而 TextRank 算法生成的标签比 TagRCNN 算法生成的标签更有意义.

由于 Stack Overflow 标签是用户人工生成, 且部分帖子的原始标签不满 5 个, 仅从数目上来说, 本文的自动代码语义标签生成方法能够生成更多有意义的标签. 为了更好地比较两者生成标签的区别, 经统计发现, KGCodeTagger 对 20 段代码生成的 99 个标签中仅 19 个与 Stack Overflow 标签一致. 由此可见, 人工打上的标签和从代码里自动抽取的标签有很大的不同.

为了比较 Stack Overflow 和 KGCodeTagger 标签得分之间的差距, 本文选取了其中两段标签差距较大的代码标签进行分析. 图 8 为 Post id 为 65089256 的帖子代码以及 Stack Overflow 标签和 KGCodeTagger 生成的标签. 该代码的意思是为每个孩子随机发放礼物. Stack Overflow 标签中, “repeat”得分最低, 该标签是该帖子发布者希望改进这段代码功能所提出的标签, 仅从代码中无法体现出来. 而本文生成的“contain”获得 3 个 2 分,

“write”和“file”分别获得两个 2 分. 这是因为即使它们出现在代码中, 也不代表核心功能. 通过分析生成的流程发现: 由于图结构的局限性, 只能映射代码中出现的词, 不能进行概念扩展, 因此需要更加完善和优化图上的关系和节点.

```
public static void main(String[] args) throws IOException {
    String path = "Christmas.txt";
    String line = "";
    ArrayList<String> kids = new ArrayList<>();
    FileWriter fw = new FileWriter("Deliveries.txt");
    SantasFactory sf = new SantasFactory();
    try (Scanner s = new Scanner(new FileReader("Christmas.txt"))) {
      while (s.hasNext()) {
        kids.add(s.nextLine());
      }}
    for (String boys : kids) {
      ArrayList<String> btoys = new ArrayList<>();
      int x = 0;
      while (x < 3) {
        if (!btoys.contains(sf.getRandomBoyToy().equals(sf.getRandomBoyToy()))){
          btoys.add(sf.getRandomBoyToy());
          x++;
        }}
      if (boys.endsWith("M")) {
        fw.write(boys + " (" + btoys + ")\n\n");
      }}
    fw.close();
  }}
```

Stack Overflow Tags: arrays, string, random, repeat
Our Tags: ArrayList, file, write, getRandomBoyToy, contain

图 8 帖子 65089256 案例分析

图 9 是 Post id 为 63618827 的帖子代码以及 Stack Overflow 标签和 KGCodeTagger 生成的标签.

```
public static void readFileFromURL(SLSearch slSearch){
  String currentLine = "";
  BufferedReader br = null;
  try{
    String webUrl = slSearch.getDomainURL();
    String name = slSearch.getUserId();
    String password = slSearch.getDailyPassword();
    String authString = name + ":" + password;
    byte[] authEncBytes = Base64.encodeBase64(authString.getBytes());
    String authStringEnc = new String(authEncBytes);
    URL url = new URL(webUrl);
    HttpURLConnection urlConnection =(HttpURLConnection)url.openConnection();
    urlConnection.setRequestProperty("Authorization", "Basic " + authStringEnc);
    InputStream is = urlConnection.getInputStream();
    InputStreamReader isr = new InputStreamReader(is);
    br = new BufferedReader(isr);
    while((currentLine = br.readLine()) != null){
      if(currentLine.indexOf("2020-08-27 20:37") > 0){
        System.out.println(currentLine);
      }}
  }catch(ConnectException e){
    e.printStackTrace();
  }catch(MalformedURLException e){
    e.printStackTrace();
  }catch(IOException e){
    e.printStackTrace(); } }
```

Stack Overflow Tags: file, url, inputStream, urlConnection
Our Tags: dns, get inputStream, computer network protocol, HttpUrlConnection, BufferedReader

图 9 帖子 63618827 案例分析

该代码的意思是，使用 URL 和身份验证访问放置在服务器上的日志文件. Stack Overflow 生成的标签得分均在 3 分以上，本文生成的标签“dns”和“computer network protocol”分别获得两个 1 分. 经过采访实验人员，他们认为“httpUrlConnection”与这两者有交叉含义，且“computer network protocol”太宽泛. 说明本文算法的游走停止条件较宽泛，导致游走步数过大，游走和生成候选标签还有改进空间.

4.3 讨 论

上述两个实验验证了本文提出的基于软件知识图谱的代码语义标签自动生成方法 KGCodeTagger 能够为代码生成有意义的语义标签. 然而，本文的实验和方法还存在以下几点不足.

- 第一，本文寻找到的符合要求的实验参与者人数较少，且均是在校研究生，与实际从事软件开发的从业人员相比，在实践经验等方面存在一定的差距；且由于实验开销限制，本文只将 4 种标签类型的 20 个数据做实验分析，这导致实验结果缺乏足够的代表性.
- 第二，本文方法比较依赖软件知识图谱的质量，但是在图谱中仍然存在一些噪音，对实验结果可能产生影响；且代码标签生成算法的参数仍可以优化.
- 最后，目前只在 Java 和 Android 领域进行了实验验证，对其他语言或者其他领域还没有进行充分验证，未证明其泛用性，但是至少提出了一种可解决的思路.

5 总 结

代码语义标签对大范围代码语义标注、辅助代码理解和搜索具有重要作用，但是当前缺少针对纯代码生成语义标签的工作. 本文提出了基于软件知识图谱的代码语义标签自动生成方法 KGCodeTagger，从代码 API 调用和概念提及出发，通过构建软件知识图谱，链接外部资源节点，为代码生成标签语义. 实验结果验证了软件知识图谱的合理性以及 KGCodeTagger 方法的有效性. 并且从实验参与者的反馈情况来看，这种自动化的方法产生的结果的确可以作为代码语义的重要参考，尤其是在需要快速理解代码语义的情况下，将极大地减轻开发人员的负担.

由于高层语义标签的质量很大程度依赖于软件知识图谱的质量，节点本身的质量以及外部链接资源节点的质量十分重要，所以未来的一个改进方向是不断扩大和优化软件知识图谱的质量，从而扩大应用的范围，为代码片段生成语义更丰富的标签.

References:

[1] Shivakumar SK. A survey and taxonomy of intent-based code search. Int’l Journal of Software Innovation, 2021, 9(1): 69−110. [doi: 10.4018/IJSI.2021010106]

[2] Stack overflow. 2020. https://stackoverflow.com/

[3] Lv F, Zhang H, Lou JG, Wang S, Zhang D, Zhao J. CodeHow: Effective code search based on API understanding and extended boolean model. In: Proc. of the 30th IEEE/ACM Int’l Conf. on Automated Software Engineering. IEEE Computer Society, 2015. 260−270. [doi: 10.1109/ase.2015.42]

[4] Wang S, Lo D, Vasilescu B, Serebrenik A. EnTagRec: An enhanced tag recommendation system for software information sites. In: Proc. of the 30th Int’l Conf. on Software Maintenance and Evolution. IEEE Computer Society, 2014. 291−300. [doi: 10.1109/icsme.2014.51]

[5] Liu J, Zhou P, Yang Z, Liu X, Grundy J. FastTagRec: Fast tag recommendation for software information sites. Automated Software Engineering, 2018, 25(4): 675−701. [doi: 10.1007/s10515-018-0239-4]

[6] Wang H, Wang B, Li C, Xu L, He JJ, Yang MN. SOTagRec: A combined tag recommendation approach for stack overflow. In: Proc. of the 4th Int’l Conf. 2019. [doi: 10.1145/3325730.3325751]

[7] Chen W, Zhou JH, Zhu JX, Wu GQ, Wei J. Semi-supervised learning based tag recommendation for docker repositories. Journal of Computer Science and Technology, 2019, 34(5): 957−971. [doi: 10.1007/s11390-019-1954-4]

[8] Zhou P, Liu J, Liu X, Yang Z, Grundy J. Is deep learning better than traditional approaches in tag recommendation for software information sites? Information and Software Technology, 2019, 109(5): 1−13. [doi: 10.1016/j.infsof.2019.01.002]

[9] Zhou P, Liu J, Yang Z, Zhou G. Scalable tag recommendation for software information sites. In: Proc. of the 24th Int'l Conf. on Software Analysis, Evolution and Reengineering. IEEE Computer Society, 2017. 272−282. [doi: 10.1109/SANER.2017.7884628]

[10] Xia X, Lo D, Wang X, Zhou B. Tag recommendation in software information sites. In: Proc. of the 10th Working Conf. on Mining Software Repositories. IEEE Computer Society, 2013. 287−296. [doi: 10.1109/msr.2013.6624040]

[11] Yang Y, Li Y, Yue Y, Wu Z, Shao W. CUT: A combined approach for tag recommendation in software information sites. In: Proc. of the 9th Int'l Conf. on Knowledge Science, Engineering and Management. 2016. 599−612. [doi: 10.1007/978-3-319-47650-6_47]

[12] Wu Y, Yao Y, Xu F, Tong H, Lu J. Tag2Word: Using tags to generate words for content based tag recommendation. In: Proc. of the 25th Int'l Conf. on Information and Knowledge Management. 2016. 2287−2292. [doi: 10.1145/2983323.2983682]

[13] Rada M, Paul T. TextRank: Bringing order into text. In: Proc. of the Conf. on Empirical Methods in Natural Language Processing. 2004. 404−411. https://www.aclweb.org/anthology/W04-3252

[14] Dwarakanath A, Ramnani RR, Sengupta S. Automatic extraction of glossary terms from natural language requirements. In: Proc. of the 21st IEEE Int'l Requirements Engineering Conf. IEEE, 2013. 314−319. [doi: 10.1109/re.2013.6636736]

[15] Chen C, Xing Z, Wang X. Unsupervised software-specific morphological forms inference from informal discussions. In: Proc. of the 39th Int'l Conf. on Software Engineering. 2017. 450−461. [doi: 10.1109/icse.2017.48]

[16] Wang C, Peng X, Liu M, Xing Z, Bai X, Xie B, Wang T. A learning-based approach for automatic construction of domain glossary from source code and documentation. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2019. 97−108. [doi: 10.1145/3338906.3338963]

[17] Wang Y, Qu J, Liu J, Chen J, Huang Y. What to tag your microblog: Hashtag recommendation based on topic analysis and collaborative filtering. In: Proc. of the Asia-Pacific Web Conf. 2014. 610−618. [doi: 10.1007/978-3-319-11116-2_58]

[18] Zhao F, Zhu Y, Jin H, Yang LT. A personalized hashtag recommendation approach using LDA-based topic model in microblog environment. Future Generation Computer Systems, 2016, 65: 196−206. [doi: 10.1016/j.future.2015.10.012]

[19] Nguyen HTH, Wistuba M, Grabocka J, Drumond LR, Schmidt-Thieme L. Personalized deep learning for tag recommendation. In: Proc. of the Pacific-Asia Conf. on Knowledge Discovery and Data Mining. 2017. 186−197. [doi: 10.1007/978-3-319-57454-7_15]

[20] Gupta M, Li R, Yin Z, Han J. Survey on social tagging techniques. ACM SIGKDD Explorations Newsletter, 2010, 12(1): 58−72. [doi: 10.1145/1882471.1882480]

[21] Arora C, Sabetzadeh M, Briand L, Zimmer F. Automated extraction and clustering of requirements glossary terms. IEEE Trans. on Software Engineering, 2017, 43(10): 918−945. [doi: 10.1109/TSE.2016.2635134]

[22] Wikipedia. 2020. https://www.wikipedia.org/

[23] Karthik S, Medvidovic N. Automatic detection of latent software component relationships from online Q&A sites. In: Proc. of the 7th Int'l Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. IEEE, 2019. 15−21. [doi: 10.1109/raise.2019.00011]

[24] Zhao X, Xing Z, Kabir MA, Sawada N, Li J, Lin SW. HDSKG: Harvesting domain specific knowledge graph from content of webpages. In: Proc. of the 24th Int'l Conf. on Software Analysis, Evolution and Reengineering. IEEE Computer Society, 2017. 56−67. [doi: 10.1109/saner.2017.7884609]

[25] Wang F, Liu JP, Liu B, Qian TY, Xiao YH, Peng ZY. Survey on construction of code knowledge graph and intelligent software development. Ruan Jian Xue Bao/Journal of Software, 2020, 31(1): 47−66 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5893.htm [doi: 10.13328/j.cnki.jos.005893]

[26] Liu M, Peng X, Marcus A, Xing Z, Xie W, Xing S, Liu Y. Generating query-specific class API summaries. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2019. 120−130. [doi: 10.1145/3338906.3338971]

[27] Liu Y, Liu M, Peng X, Treude C, Xing Z, Zhang X. Generating concept based API element comparison using a knowledge graph. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. IEEE, 2020. [doi: 10.1145/3324884.3416628]

[28] Sun J, Xing Z, Chu R, Bai H, Wang J, Peng X. Know-how in programming tasks: From textual tutorials to task-oriented knowledge graph. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE, 2019. 257−268. [doi: 10.1109/ICSME.2019.00039]

[29] JDK 1.8. 2020. https://docs.oracle.com/javase/8/docs/api

[30] Android 27. 2020. https://developer.android.com/reference/packages

[31] Sridhara G, Hill E, Muppaneni D, Pollock L, Vijay-Shanker K. Towards automatically generating summary comments for Java methods. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2010. 43−52. [doi: 10.1145/1858996.1859006]

[32] Sridharac G, Pollock L, Vijay-Shanker K. Automatically detecting and describing high level actions within methods. In: Proc. of the 33th Int'l Conf. on Software Engineering. ACM, 2011. 101−110. [doi: 10.1145/1985793.1985808]

[33] Wang X, Pollock L, Vijay-Shanker K. Automatically generating natural language descriptions for object-related statement sequences. In: Proc. of the 24th Int'l Conf. on Software Analysis, Evolution and Reengineering. IEEE, 2017. 205−216. [doi: 10.1109/SANER.2017.7884622]

[34] Hassan M, Hill E. Toward automatic summarization of arbitrary Java statements for novice programmers. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. IEEE Computer Society, 2018. 539−543. [doi: 10.1109/ICSME.2018.00063]

[35] Wong E, Liu T, Tan L. Clocom: Mining existing source code for automatic comment generation. In: Proc. of the 22th Int'l Conf. on Software Analysis, Evolution and Reengineering. IEEE, 2015. 380−389. [doi: 10.1109/SANER.2015.7081848]

[36] Vassallo C, Panichella S, Penta MD, Canfora G. CODES: Mining source code descriptions from developers discussions. In: Proc. of the 22th Int'l Conf. on Program Comprehension. ACM, 2014. 106−109. [doi: 10.1145/2597008.2597799]

[37] Iyer S, Konstas I, Cheung A, Zettlemoyer L. Summarizing source code using a neural attention model. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics. 2016. [doi: 10.18653/v1/p16-1195]

[38] Hu X, Li G, Xia X, Lo D, Jin Z. Deep code comment generation. In: Proc. of the 26th Int'l Conf. on Program Comprehension. ACM, 2018. 200−210. [doi: 10.1145/3196321.3196334]

[39] Hu X, Li G, Xia X, Lo D, Lu S, Jin Z. Summarizing source code with transferred API knowledge. In: Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence. 2018. 2269−2275. [doi: 10.24963/ijcai.2018/314]

[40] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Proc. of the Advances in Neural Information Processing Systems. 2014. 3104−3112. [doi: 10.5555/2969033.2969173]

[41] Leclair A, Jiang S, Mcmillan C. A neural model for generating natural language summaries of program subroutines. In: Proc. of the 41th Int'l Conf. on Software Engineering. IEEE, 2019. 795−806. [doi: 10.1109/ICSE.2019.00087]

[42] Seitner J, Bizer C, Eckert K, Faralli S, Meusel R, Paulheim H, Ponzetto S. A large database of hypernymy relations extracted from the Web. In: Proc. of the 10th Int'l Conf. on Language Resources and Evaluation Conf. 2016. http://www.lrec-conf.org/proceedings/lrec2016/pdf/204_Paper.pdf

[43] Xie W, Peng X, Liu M, Treude C, Xing Z, Zhang X, Zhao WY. API method recommendation via explicit matching of functionality verb phrases. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 1015−1026. [doi: 10.1145/3368089.3409731]

[44] WikiData. 2020. https://www.wikidata.org/

[45] Xiao YH, *et.al*. Knowledge Graph Concepts and Techniques. Beijing: Publishing House of Electronics Industry, 2020 (in Chinese).

[46] Spacy. 2020. https://spacy.io/

[47] Nltk. 2020. https://www.nltk.org/

[48] Zhang J, Deng B, Li X. Concept based query expansion using WordNet. In: Proc. of the Int'l e-Conf. on Advanced Science and Technology. IEEE Computer Society, 2009. 52−55. [doi: 10.1109/ast.2009.24]

[49] Miller GA. Wordnet: A lexical database for English. Communications of the ACM, 1995, 38(11): 39−41. [doi: 10.1145/219717.219748]

[50] Hearst MA. Automatic acquisition of hyponyms from large text corpora. In: Proc. of the 14th Conf. on Computational Linguistics. 1992. 539−545. [doi: 10.3115/992133.992154]

[51] Mikolov T, Sutskever I, Chen K, Corrado G, Dean J. Distributed representations of words and phrases and their compositionality. In: Proc. of the Advances in Neural Information Processing Systems (NIPS 2013). 2013. 3111−3119. https://arxiv.org/abs/1310.4546v1

[52] Subramanian S, Inozemtseva L, Holmes R. Live API documentation. In: Proc. of the 36th Int'l Conf. on Software Engineering. ACM, 2014. 643−652. [doi: 10.1145/2568225.2568313]

[53] Hill E, Pollock L, Vijay-Shanker K. Automatically capturing source code context of NL-queries for software maintenance and reuse. In: Proc. of the 31th Int'l Conf. on Software Engineering. IEEE, 2009. 232−242. [doi: 10.1109/icse.2009.5070524]

[54] Beautiful soup. 2020. https://www.crummy.com/software/BeautifulSoup/

[55] https://github.com/mmichelsonIF/hearst_patterns_python.git/

[56] https://github.com/3Top/word2vec-api/

[57] Gensim. 2020. https://radimrehurek.com/gensim/

[58] Neo4j. 2020. https://neo4j.com/

[59] Singh R, Mangat NS. Elements of Survey Sampling. Springer Science & Business Media, 1996.

[60] McHugh ML. Interrater reliability: The kappa statistic. Biochemia Medica: Biochemia Medica, 2012, 22(3): 276−282.

附中文参考文献:

[25] 王飞, 刘井平, 刘斌, 钱铁云, 肖仰华, 彭智涌. 代码知识图谱构建及智能化软件开发方法研究. 软件学报, 2020, 31(1): 47−66. http://www.jos.org.cn/1000-9825/5893.htm [doi: 10.13328/j.cnki.jos.005893]

[45] 肖仰华, 等. 知识图谱概念与技术. 北京: 电子工业出版社, 2020.

邢双双(1996－), 女, 硕士, 主要研究领域为智能化软件开发.

刘名威(1994－), 男, 博士, CCF 学生会员, 主要研究领域为智能化软件工程与系统.

彭鑫(1979－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为智能化软件开发, 云原生与智能化运维, 泛在计算软件系统.