计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(12):202−210 [doi: 10.15888/j.cnki.csa.008216]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于多源信息融合的 API 知识图谱构建①

马　展, 王　岩, 王微微, 赵瑞莲

(北京化工大学 信息科学与技术学院, 北京 100029)
通讯作者: 王微微, E-mail: wangww@mail.buct.edu.cn

摘　要: API 相关的知识通常分散隐含在多个信息源, 如 API 参考文档、问答网站等非结构化的文本中, 不利于 API 的查询与检索. 为此, 提出一种多源信息融合的 API 知识图谱构建方法, 以提高 API 检索的效率. API 参考文档从设计者角度描述了 API 的功能和结构, Stack Overflow 问答网站从用户角度提供了 API 的使用目的及应用场景, 二者互为补充, 可共同为 API 查询与检索提供支持. 通过分析 API 参考文档, 抽取 API 和领域概念作为实体, 构建 API 和领域概念之间的关联关系; 利用 Stack Overflow 问答网站, 抽取问答 QA 和 API 概念作为实体, 构建问答 QA 和 API 概念之间的关联关系. 在此基础上, 将二者进行知识融合, 构建多源 API 知识图谱, 以实现基于知识图谱的 API 推荐. 为验证本文方法, 分别从知识抽取的准确性和推荐应用两方面对本文构建 API 知识图谱的有效性进行评估. 实验结果表明, 基于知识图谱的 API 推荐, 在推荐效果及效率上均有提升.

关键词: Stack Overflow; 知识抽取; 信息融合; 知识图谱; 信息检索

引用格式: 马展,王岩,王微微,赵瑞莲.基于多源信息融合的 API 知识图谱构建.计算机系统应用,2021,30(12):202–210. http://www.c-s-a.org.cn/1003-3254/8216.html

API Knowledge Graph Construction Based on Multi-Source Information Fusion

MA Zhan, WANG Yan, WANG Wei-Wei, ZHAO Rui-Lian

(College of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: API-related knowledge is usually scattered among multiple information sources, such as API reference documentation, Q&A forum and other unstructured texts, which is not conducive to API query and retrieval. To improve the efficiency of API retrieval, this study proposes an API knowledge graph construction method based on multi-source information fusion. API reference documentation describes the function and structure of the API from a designer's perspective, while Stack Overflow provides the purpose and use scenarios of the API from a user's perspective. API reference documentation and Stack Overflow complement each other, and they can provide support for API query and retrieval together. By means of analyzing API reference documentation, API and domain concepts can be extracted as entities and their relationships can be constructed. With the Stack Overflow website, Q&A and API concepts can be extracted as entities and their relationships can also be constructed. On this basis, these two kinds of information are fused to construct a multi-source API knowledge graph for the API recommendation based on the knowledge graph. To verify the proposed method, this study evaluates the effectiveness of the API knowledge graph in terms of the accuracy of knowledge extraction and the API recommendation. The experimental results show that the recommendation effectiveness and efficiency based on our knowledge graph have been improved.

Key words: Stack Overflow; knowledge extraction; information fusion; knowledge graph; information retrieval

① 基金项目: 国家自然科学基金 (62077003, 61872026)
Foundation item: National Natural Science Foundation of China (62077003, 61872026)
收稿时间: 2021-03-04; 修改时间: 2021-03-29; 采用时间: 2021-04-06

1 引言

应用程序接口 (API) 在软件开发中起着十分重要的作用. 87% 的开发人员经常借助于 API 解决不同的编程问题[1], 但检索查找合适的 API 非常困难. 为此, 为提高 API 检索的效率和质量, 研究人员从 API 参考文档、IT 问答网站等各类资源构建了相应的 API 推荐系统, 以帮助开发人员解决与 API 相关的编程问题. 目前, 已有的 API 推荐系统有: RASH[2]、BIKER[1] 和 RACK[3] 等. RASH 针对 API 参考文档和 Stack Overflow (SO) 问答网站[4], 利用词法相似度为查询推荐 API. 与 RASH 不同, BIKER[1] 利用语义相似度, 结合 API 参考文档和 SO 问答网站, 为查询推荐 API. RACK[3] 则通过建立问答网站标题中的关键字和 API 之间的关联, 为查询推荐 API. 这些方法通过寻找问答网站相似问答涉及的 API, 为查询推荐 API. 然而, 与编程相关的 API 信息常以多种形式关联, 如解决同一问题的 API之间可能存在功能相似的关联. 这种关联可能有助于 API 检索推荐[5]. 但上述方法均未考虑 API 信息之间的关联.

知识图谱是一个能有效表示信息之间语义关联的知识网络[6], 可用于 API 的知识表示. 如, Liu 等人[7] 从 API 参考文档和维基百科中提取相关知识, 构建 API 知识图谱; Li 等人[8] 从 API 参考文档和 API 教程中提取警告语句, 构建 API 警告知识图谱; Ling 等人[5] 以开源项目中的 API 为实体, 以 API 之间的调用、返回、实现等为关系, 构建基于开源项目的 API 知识图谱. 但上述从 API 文档、维基百科或开源项目等构建的 API 知识图谱, 缺少对 API 实际应用场景描述方面的信息, 不利于 API 实际应用知识的融合利用.

SO 是一个面向编程人员群体的 IT 技术问答网站. 它以解决用户实际问题为出发点, 提供了 API 使用目的和真实应用场景相关信息[9], 可与 API 参考文档相辅相成, 互为补充, 为 API 知识图谱的构建提供真实应用场景的支持. PyTorch 是一个以 Python 优先的深度学习框架, 在 SO 中涉及大量与 API 相关的讨论. 因此, 本文提出一种结合 API 参考文档和 SO 问答网站的 API 知识图谱构建方法 (AKG-CMISF). 从 API 参考文档提取 API、领域概念作为实体, 建立他们之间的包含、继承、重载等关系; 从 SO 问答网站中提取问答 QA、API 概念作为实体, 建立问答 QA 实体和 API 概念实体之间的关联关系. 在此基础上, 建立 API 参考文档中的 API、领域概念和 SO 问答网站中的问答 QA、API 概念之间的关联关系, 将多源知识进行融合, 构建多源信息融合的 API 知识图谱. 具体而言, 针对 API 参考文档, 通过分析半结构化文本提取 API 实体, 通过词典匹配提取领域概念实体; 针对 SO 问答网站, 通过类型识别和摘要生成获取问答 QA 实体, 通过数据驱动识别 API 概念; 然后, 基于语义相似度建立 API 概念和领域概念之间的关联, 利用词共现方法构建 API 和 API 概念之间的关联, 依据启发式策略构建 API 和问答 QA 之间的关联, 将多源信息进行知识融合, 实现 API 知识图谱的构建.

为验证本文提出的方法, 分别从知识抽取的准确度和推荐应用两个角度, 对本文构建 API 知识图谱的有效性进行评估. 实验结果表明, 和现有 API 推荐系统相比, 本文构建的知识图谱的 API 推荐效果和效率均有提升, 表明了 API 知识图谱的有效性以及多源信息融合对 API 推荐效果的支撑.

2 基于多源信息的 API 知识图谱构建

本文从 API 参考文档和 SO 问答网站提取 API 相关知识, 通过两类 API 知识的融合, 构建 API 知识图谱, 其框架如图 1 所示.

2.1 面向 API 参考文档的知识获取

API 参考文档提供所有 API 的功能描述及其相关属性信息 (如参数、返回值等). 本文主要选用 PyTorch 框架的 API 参考文档, 进行知识表示及获取.

2.1.1 API 参考文档的知识表示

本文以 API 参考文档涉及的 API、领域概念以及它们之间的关系作为其知识表示. API 参考文档的功能描述中蕴含 API 的应用领域, 该信息可间接反映 API 和领域概念之间的关联关系. 本文 API 是指模块、类、方法/函数等, 它们之间存在包含、继承和重载等关系. 领域概念是对 API 应用领域的抽象概括. API 和领域概念之间可通过“提及”关联起来. 因此, 本文将 API 和领域概念作为 API 参考文档的实体, 并将它们之间的“提及”作为实体间的关系.

2.1.2 API 参考文档的知识提取

API 参考文档属于半结构化数据, 不同的 HTML 标签表示不同类型的 API 实体, API 标签下含有 API 实体的属性信息, 如功能描述、参数、返回值和返回值类型等. 因此, 本文开发了一个 API 实体及属性分析模块, 分析并识别 API 实体之间的关系. 具体而言, 通

过静态分析并识别 API 类或方法/函数及其它们之间的关系; 根据类的声明规则, 采用正则表达式提取类和基类之间的继承关系, 实现 API 实体及关系的识别和提取.

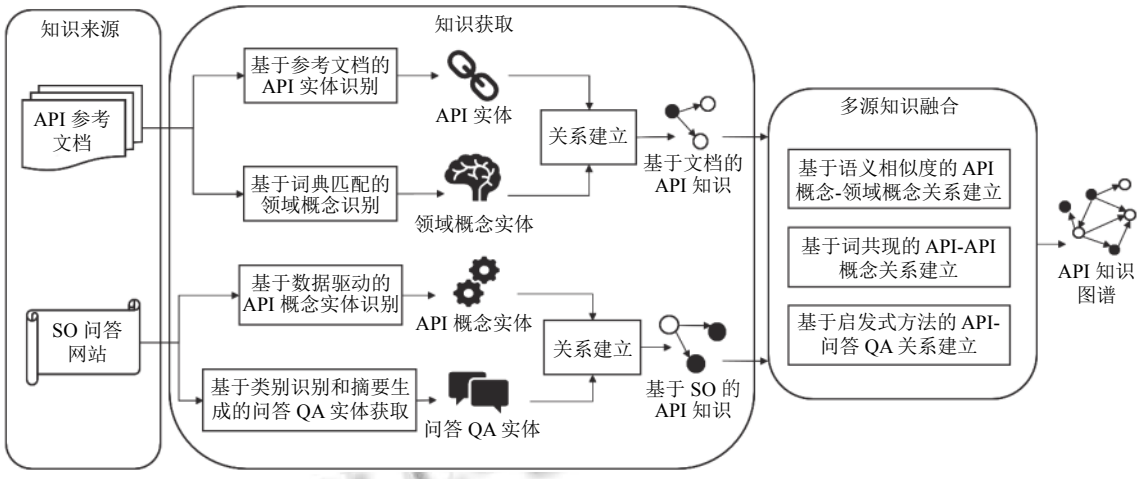

图 1 基于多源信息的 API 知识图谱构建框架

针对 API 功能描述中隐含的领域概念, 可通过目前已有的领域概念词典进行识别. 由于本文关注的是 PyTorch 框架, 因此, 采用文献 [10] 提供的领域概念开源词典[11], 匹配并识别 API 功能描述中的领域概念. 即将领域概念词典加入工具的自定义命名实体识别列表中, 根据 API 功能描述, 利用自然语言处理 spacy 匹配并识别 API 功能描述中的领域概念.

每个 API 都对应一个功能描述, 功能描述隐含领域概念, 因此, 从这种对应结构中可提取 API 和领域概念之间的“提及”关系. 如 API “torch.normal”的功能描述中包含领域概念“standard deviation”. 当领域概念“standard deviation”被识别后, 则可在 API“torch.normal”和领域概念“standard deviation”之间建立“提及”关系.

2.2 面向 SO 问答网站的知识获取

SO 问答网站含有 API 的具体应用场景信息, 可与 API 参考文档互为补充. 本文主要对 SO 问答网站标签为“PyTorch”的问答进行知识的表示及获取.

2.2.1 SO 问答网站的知识表示

SO 问答网站的文本描述中包含了与 API 相关的术语, 如“download”“update”等. 这些术语可能只与软件开发相关, 不限于某领域, 但往往和 API 出现在同一句话, 同一段落或者同一篇问答中. 由于段落是问答描述的基本逻辑单元, 故本文将与 API 处于同一段落的相关术语称为 API 概念实体.

SO 问答存在目的不明确和信息过载等问题. 目的不明确是指, SO 问答因未考虑用户提问的目的, 使得 SO 问答网站很难把握用户提问的原因, 从而找出相应答案[12,13]; 信息过载是指, SO 问答的一个提问可能有多个回答. 有统计表明, 超过 37% 的 SO 问答包含一个以上的回答, 每个回答的平均字数超过 789 个单词[14], 这增加了 SO 问答网站获取关键信息的难度. 本文依据提问目的对 SO 问答自动分类, 以识别目的类型, 明确提问目的; 并依据 SO 问答特征生成回答的摘要, 从而减少信息过载. 本文将包含目的类型和摘要的 SO 问答称为问答 QA 实体. 继而可建立问答 QA 实体和 API 概念实体之间的“提及”关系. 并将 API 概念、问答 QA, 以及它们之间的关系作为 SO 问答网站知识表示.

2.2.2 SO 问答网站的知识提取

为从 SO 问答网站提取 API 概念实体, 首先需识别 SO 问答网站中涉及的 API. SO 问答网站中的 API 通常由<code>标签标注, 当<code>标签标注的元素部分匹配或完全匹配 API 参考文档中的 API 全限定名时, 可识别该元素为 SO 问答中的 API. API 概念与 API 处于同一段落, 它可能是由多单词组成的概念, 如“convolutional layer”等. 为识别多单词 API 概念, 本文采用一种基于数据驱动的 API 概念识别方法, 即在 SO 问答网站中寻找经常连续出现但不常单独出现的单词, 将其作为 API 概念. 为此, 可通过自然语言处理技术, 对 SO 问答进行分词及去停用词, 形成单字符概念. 根据连续单词出现的频率, 计算词组得分 (*score*), 提取

API 概念. 词组得分如式 (1) 所示:

$$score(w_iw_j)=\frac{count(w_iw_j)-\delta}{count(w_i)\times count(w_j)} \tag{1}$$

其中, $count(w_iw_j)$ 表示两个连续的单词 w_i 和 w_j 在全文档中出现的次数, $count(w_i)$ 和 $count(w_j)$ 分别表示单词 w_i 和 w_j 出现的次数, δ 是一个阈值. 当两个连续出现的单词 w_i 和 w_j 的频率小于 δ 时, w_i 和 w_j 不能构成双词短语. 当形成双词概念时, 重复式 (1) 可检测三单词短语. 由于 3 个以上单词构成的 API 概念不常见, 故本文仅识别三单词以内的短语作为 API 概念.

为提取 SO 网站的问答 QA 实体, 本文通过机器学习算法对提问的目的进行分类, 获得目的类型, 在此基础上, 进行基于特征提取的回答摘要生成, 实现问答 QA 实体的提取.

本文针对提问目的, 在 Beyer 等人[15] 对 SO 问答归类的基础上, 将 SO 问答分为如下 7 类: “API USAGE”类的提问目的为: 寻求实现某功能或 API 的使用建议; “DISCREPANCY”类的提问目的为: 请求解决非预期结果的代码段; “ERRORS”类的提问目的为: 请求修复错误或处理异常; “REVIEW”类的提问目的为: 请求最佳解决方案; “CONCEPTUAL”类的提问目的为: 询问 API 的原理或背景; “API CHANGE”类的提问目的为: 寻求解决因 API 版本变化所产生的问题的方法; “LEARNING”类的提问目的为: 询问文档或教程来学习工具或语言.

XGBoost (eXtreme Gradient Boosting) 机器学习算法具有快速学习和预测的功能[16]. 它基于决策树, 采用残差拟合的方式对残差进行顺序学习, 具有较高的分类准确性, 因此本文采用 XGBoost 算法训练分类器对 SO 问答分类. 主要步骤如下:

(1) 将 SO 问答标注为 7 种类型中的一种;

(2) 通过自然语言处理过程 (分词、删除停用词、词形还原等), 将提问转换成相应的单词列表;

(3) TF-IDF 通过字词出现的频率反映了其重要性, 本文以提问的 TF-IDF 作为其文本特征. 在此基础上, 利用 XGBoost 算法识别所有 SO 问答的提问类型.

针对 SO 问答信息过载, 本文以回答中的重要段落为依据, 生成其摘要. 即根据 SO 问答网站的特点, 通过问题相关特征、内容相关特征和用户特征, 综合为各段落计算其特征值, 取前 M(10) 个段落作为其摘要. 具体特征分析如下:

(1) 问题相关特征: 若段落中包含提问中的关键词, 该段落被认为是与提问相关的段落. 段落包含的关键词越多, 则相关度越大. SO 的 Tag 标签作为关键词集, 以回答段落涉及的关键词占提问与回答共同涉及的关键词的比例, 计算每个回答段落和提问的相关度. 当提问和回答中没有共同的关键词时, 则该段特征值置为 0.

(2) 内容相关特征: 该特征从是否包含 API、信息熵、语义模板 3 个子特征评估段落内容的重要性. ① 若段落中至少出现一个 API, 则子特征值置为 1, 否则置为 0. ② 单词的逆文本频率指数 *IDF* 值可用来衡量其信息熵, 可通过式 (2) 计算. 其中 p 表示段落总数, p'表示包含某个单词的段落数. *IDF* 值越高, 表明单词出现的频率越低, 则重要程度越高. 段落的熵值可用其单词的 *IDF* 值之和表示, 并将熵值归一化为 (0,1], 作为子特征值. ③ 本文结合 API 推荐的句式特征和带有总结性的句式特征对 Xu 等人[17] 的段落语义模板进行扩充, 结果如表 1 所示. 若段落中至少包含一个句式, 子特征的值置为 1, 否则置为 0. 内容特征值为 3 个子特征值之和.

$$IDF=\lg\left(\frac{p}{p'+1}\right) \tag{2}$$

表 1 表明突出段落的语义模板

编号	句式	编号	句式	编号	句式
1	Please check XX	9	I’d recommend XX	17	If you want to XX
2	Pls check XX	10	In summary, XX	18	Simply out XX
3	You should XX	11	Keep in mind that XX	19	If you use XX
4	You can try XX	12	I suggest that XX	20	What’s important XX
5	You could try XX	13	You can use XX	21	When you call XX
6	Check out XX	14	I prefer XX	22	By popular demand XX
7	In short, XX	15	XX needs to be called	23	You would need to XX
8	The most important is XX	16	Note that XX	24	You must XX

(3) 用户特征: SO 中每个回答都有相应的投票, 投票越高, 表明该回答的质量越高. 因此, 采用回答的票数作为当前回答中段落的票数.

对于上述 3 种特征, 为了避免特征得分为 0, 通过添加平滑因子 0.0001, 将所有特征归一化为 (0,1]. 将每个特征的归一化值相乘, 作为每个段落的总分. 最终取前 M(10) 个段落作为问答 QA 的摘要.

通过对 SO 问答进行提问的类型识别以及回答的摘要生成, 最终获得有效的问答 QA 实体. SO 问答通常会提及多个 API 概念, 当识别了 API 概念后, 可分别在 API 概念和当前问答 QA 之间建立“提及”关系.

2.3 知识融合

为对 API 参考文档和 SO 问答网站的 API 知识进行融合, 构建完整的 API 知识图谱, 本文分别从 API 实体和 API 概念实体、API 概念实体和领域概念实体以及 API 实体和问答 QA 实体之间, 进行知识融合.

2.3.1 基于词共现的 API 实体和 API 概念实体之间的融合

词共现是指 API 和 API 概念具有相同的上下文, 即出现在同一段落. API 概念对 API 的功能作用进行抽象概括, 这是建立两者之间语义关联的关键. 共现频率是指 API 和 API 概念出现在同一段落的次数. 因此本文通过计算 API 和 API 概念的共现频率, 捕获具有语义关联的 API 和 API 概念.

$$freq(A_i \rightarrow Ac_j) \geq \alpha \tag{3}$$

如式 (3) 所示, $freq(A_i \rightarrow Ac_j)$ 表示 API A_i 和 API 概念 Ac_j 的共现频率, α 为其阈值. 当共现频率不低于阈值 α 时, API A_i 和 API 概念 Ac_j 之间可建立“提及”的关系.

2.3.2 基于语义相似度的 API 概念实体和领域概念实体之间的融合

API 概念和领域概念的融合是指建立 API 概念和领域概念之间的关联关系, 这种关联关系有助于建立 API 之间的间接关联, 以提高检索到相关 API 的可能.

由于 API 概念和领域概念由词组构成, 因此, 可通过结合词法和语义相似度来确定两者之间的关系. 当二者的相似度高于给定阈值时, 可建立它们的“相关”关系. 相似度计算公式如式 (4) 所示:

$$sim(n_1,n_2) = w_1 \times sim_{\text{lex}}(n_1,n_2) + w_2 \times sim_{\text{con}}(n_1,n_2) \tag{4}$$

$$sim_{\text{lex}}(n_1,n_2) = \frac{|Token(n_1) \cap Token(n_2)|}{|Token(n_1) \cup Token(n_2)|} \tag{5}$$

$$sim_{\text{con}}(n_1,n_2) = \frac{sim_{\cos}(V_p(n_1),V_p(n_2)) + 1}{2} \tag{6}$$

式 (4) 从词法和语义两个方面度量领域概念和 API 概念的相似度, n_1 和 n_2 代表候选的领域概念和 API 概念. 词法相似度 sim_{lex} 通过 Jaccard 相似度进行计算, 如式 (5) 所示, 其中 $Token(n)$ 代表组成概念的单词. 在 SO 问答和 API 参考文档两类语料库的基础上, 本文利用 Word2Vec[18] 训练词嵌入模型, 将概念转换为词向量. 通过式 (6) 计算 n_1 和 n_2 之间的语义相似度. $V_p(n)$ 表示概念实体向量, $sim_{\cos}$ 表示两个向量的余弦相似度. 一般来说, 语义相似度的重要性大于词法相似度, 因此, 设置 $w_1 < w_2$.

2.3.3 基于启发式的 API 实体和问答 QA 实体之间的融合

API 实体和问答 QA 实体之间的融合是指建立 API 实体和问答 QA 之间的关联关系. 这种关联关系的建立使得 API 参考文档中 API 的通用知识 (功能描述、参数、返回值等) 可以和 API 在具体场景下的具体知识 (如何解决具体问题) 联系在一起, 可为开发人员提供更全面的 API 信息.

一般而言, SO 问答中的 API 并不总是以全限定名的形式存在, 例如 SO 问答“What does model.train() do in PyTorch”中提及 API“forward()”. 然而“forward()”可以与多个 API 关联. 为了将问答 QA 实体无歧义地和相应的 API 实体进行关联, 本文设计了启发式策略: SO 问答中代码元素的出现有局部性[19], 即同一问答出现的 API 通常属于同一模块或类. 通过解析 HTML 的<code>标签, 识别问答 QA 提及 API 的模块或类. 通过制定正则表达式识别代码块中的模块或类确定其 API. 当识别了无歧义的 API 后, 便可在问答 QA 和 API 之间建立“提及”的关联关系.

3 实验设计及结果分析

3.1 实验设计

本文从 SO 官方数据 (截至 2020 年 6 月发布的数据)[20] 中提取标注为“pytorch”的问答, 通过剔除其没有回答和回答中评分小于 1 的问答, 共收集了 3361 个 SO 问答作为研究对象. 从 PyTorch 的 API 参考文档中获取 27 个模块, 314 个类, 1570 个函数或方法, 以及它们对应的功能描述和属性作为研究对象, 进行 API 知识图谱的构建. API 概念提取时, 实验中设置阈值 δ 为 5, 以避免不常见短语的识别. API 和 API 概念之间融

合时, 实验中将共现频率阈值α设置为 3, 来捕获 API 实体和 API 概念实体之间的语义关联. 融合 API 概念和领域概念时, 考虑到语义相似度的重要性大于词法相似度, 因此实验中设置权重 w_1 和 w_2 分别为 0.3 和 0.5. 最终构建的 API 知识图谱含 28 730 个实体, 142 578 个关系. 其中 API 实体 1912 个, API 概念实体 16 216 个, 领域概念实体 7116 个, 问答 QA 实体 3361 个. 图 2 是构建 API 知识图谱的部分抽象表示.

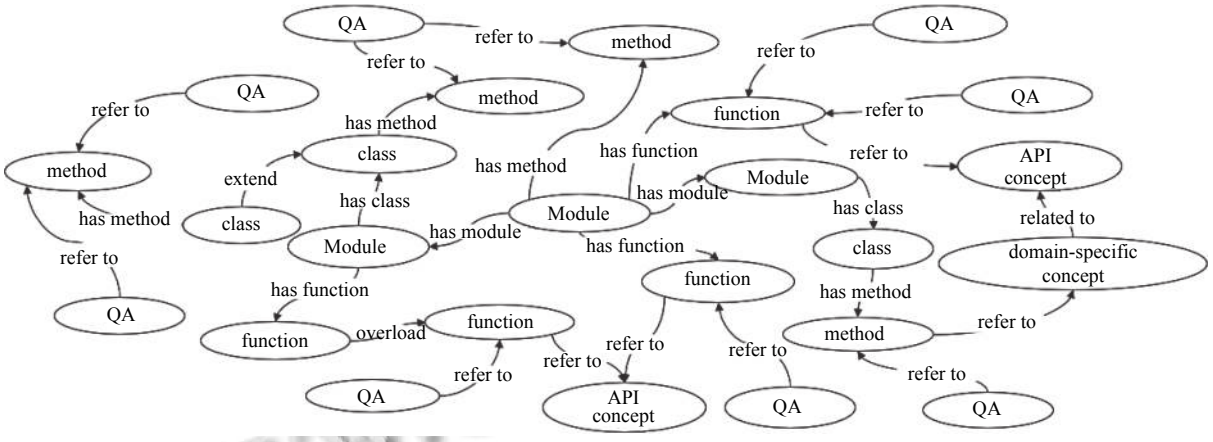

图 2 基于多源信息融合的 API 知识图谱

本文构建的 API 知识图谱以 Neo4j 图数据库存储, 以便利用 Stanford CoreNLP[21] 自然语言处理工具, 通过语法分析提取查询中的关键词. 对于每个关键词, 在 API 知识图谱中搜索查找一个与之语义相近的概念实体. 与概念实体具有“提及”关系的 API 实体作为候选 API, 与候选 API 关联的问答 QA 为该候选 API 在具体使用场景下的信息. 由于候选 API 可能存在多个, 因此依据候选 API 和查询的语义相关性对候选 API 进行排序. 即通过计算查询和每个候选 API 功能描述的语义相似度 (见式 (6)) 对 API 进行排序, 取前 K 个 API 推荐给用户. 本文从 SO 问答网站中随机抽取 10 个至少有一次投票且回答中含有明确 API 的问题作为实验对象, 通过搜索 API 知识图谱, 获得与其相关的 API, 以此验证基于知识图谱的 API 推荐的有效性.

3.2 实验结果与分析

本文从知识提取以及 API 推荐效果两个角度评估 API 知识图谱的质量.

3.2.1 API 知识提取的有效性分析

本实验主要从实体和关系提取的有效性说明知识提取的有效性. API 实体以及它们之间的关系来自半结构化 API 参考文档, 根据特定 HTML 标签和声明, 一般都会提取与之相关的实体和关系. 因此本文主要评估从非结构化文本中提取实体的有效性, 即概念实体和问答 QA 实体, 以及 API 概念和领域概念之间关系提取的有效性. 概念实体以及关系的数量超过上万个, 检查所有实体和关系需要大量时间, 因此采用随机采样. 在确保 95% 置信度下, 从构建 API 知识图谱的实体或关系中随机抽取 5% 的样本, 样本估计精度具有 0.05 的误差幅度. 问答 QA 实体通过提问目的类型识别和回答摘要生成获得, 因此通过分析分类对类型识别的准确性以及摘要生成的质量, 评估问答 QA 实体的有效性. 查询问题及相关 API 个数如表 2 所示.

(1) 针对 API 概念和领域概念有效性的评估, 通过人工识别抽样结果的准确与否. 准确是指提取的概念是正确且有意义的. 经过随机采样, 从领域概念实体中获得的 356 个领域概念的准确度可达 95.6%, 误差主要来源于领域概念词典自身; 从 API 概念实体采样的 800 个 API 概念的准确度为 97.8%, 影响其准确度的原因主要是 SO 问答中常提到一些数字指标, 如“200k images”, 这些术语不应被识别为 API 概念. 针对 API 概念和领域概念关系提取的有效性评估, 人工识别抽样结果的准确性, 即 API 概念和领域概念之间的语义相关性是否准确. 经过随机采样, 从 API 知识图谱中获得 4000 个关系, 其中 94.3% 的 API 概念和领域概念语义被识别为是相关的, 未被正确识别的 API 概念或领域概念影响关系提取的有效性.

(2) 针对问答 QA 实体有效性的评估, 主要通过分析 SO 问答提问的分类效果和回答摘要生成的质量加

以说明.

① 问答分类效果评估: 本文采用 10 折交叉验证的方法, 验证 SO 问答分类的有效性. 即用精度 (precision)、召回率 (recall)、*F*1 值和准确度 (accuracy) 对分类效果进行评估. 为了验证基于 XGBoost 的提问分类效果, 以常用的 SVM(支持向量机) 和 RF(随机森林) 作为对比. 结果如表 3 所示, XGBoost 的精度相比 SVM 和 RF 分别提高了 14.6% 和 5.7%, 准确度分别提高了 5.9% 和 4.6%. 因此, 可以看出, 采用 XGBoost 算法的分类效果优于其它方法.

表 2 10 个查询问题以及相关 API 个数

SO编号	查询	相关API的个数
44524901	How to do product of matrices in PyTorch?	6
54716377	How to do gradient clipping in pytorch?	2
48152674	How to check if pytorch is using the GPU?	7
50544730	How do I split a custom dataset into training and test datasets	1
55546873	How do I flatten a tensor in pytorch?	4
53841509	How does adaptive pooling in pytorch work?	4
53266350	How to tell Pytorch to not use the GPU?	2
53879727	PyTorch-How to deactivate dropout in evaluation mode?	2
47197885	How to do fully connected batch norm in PyTorch?	4
51136581	How to create a normal distribution in pytorch?	4

表 3 不同分类算法的分类对比

方法	精度	召回率	*F*1值	准确度
XGBoost	0.871	0.847	0.834	0.910
SVM	0.760	0.851	0.785	0.850
RF	0.824	0.903	0.849	0.870

② 摘要生成质量评估: 本文从相关性、有用性和多样性[17] 衡量摘要的质量. 相关性表示摘要是否与问题相关, 有用性表示摘要内容能否解决问题, 多样性表示能否从多个角度回答问题. 本文设定每个指标最高分为 5 分 (分别代表高度相关、有用和多样), 最低分为 1 分 (分别代表不相关、无用和单一). 为了评估摘要质量, 邀请具有两年 PyTorch 使用经验的研究生参与评估. 表 4 表示评估者对表 2 的 10 个 SO 问答回答摘要的评价情况, 其中每个查询的 3 个评价指标均在 3 分到 5 分之间, 10 个问题的平均结果分别为 3.6、3.4 和 3.7, 表明基于特征提取的摘要生成方法是有效的.

表 4 SO 问答摘要得分的分布情况

查询	相关性	有用性	多样性
How to do product of…	3	3	3
How to do gradient…	3	3	3.6
How to check if…	4	3.5	3.5
How do I split…	3.5	3	4
How do I flatten…	3.6	3	3.6
How does adaptive pooling…	3.5	4	3.5
How to tell Pytorch…	4.5	4	4.5
PyTorch-How to deactivate…	3	3	3.5
How to do fully connected…	4	3.5	4
How to create a…	4.5	4	4
AVERAGE	3.6	3.4	3.7

3.2.2 本文 AKG-CMISF 对 API 的推荐效果分析

本实验选择信息检索常用的 3 个评价指标, HR (Hit Ratio) 值、MRR (Mean Reciprocal Rank) 和 MAP (Mean Average Precision) 对 API 推荐效果进行评估. HR 评价的是在前 *K* 个搜索结果中, 正确结果占所有正确结果的百分比; MRR 是指出现第一个正确结果的位置; MAP 为所有正确结果的排名. 由于与查询相关的 API 个数在 10 以内, 因此本文设置 *K*=10. 以最新 API 推荐系统——BIKER[1] 作为基线, 进行 API 推荐效果的对比, 结果如表 4 所示.

由表 5 可知, 本文 AKG-CMISF 的 HR 比 BIKER 提高了 49%, MRR 提高了 22%. 表明在搜索的前 10 个结果中, AKG-CMISF 能够搜索到更多与查询相关的 API, 并且比 BIKER 更早找到第一个正确 API. 由此可见, AKG-CMISF 相比于 BIKER, 检索效率有所提高. 表 6 表示在时间成本方面, AKG-CMISF 的构建时间较长, 主要集中在分类器的训练和摘要生成, BIKER 的构建时间较短, 主要集中在词嵌入模型的训练. 虽然本文方法构建的时间成本高于 BIKER, 但查询时间缩短了一倍, 提高了 API 的检索效率.

表 5 和基线 API 推荐方法对比

方法	HR	MAP	MRR
BIKER	0.520	0.521	0.573
AKG-CMISF	0.774	0.558	0.701
Improve. BIKER (%)	49	7	22

3.2.3 多源信息融合对 API 推荐的有效性分析

本实验对比多信息源融合的 API 知识图谱和单信

息源的 API 知识图谱对 API 的推荐效果, 以此验证多源信息融合对 API 推荐的有效性. 单信息源的 API 知识图谱构建分别从 API 参考文档和 SO 问答网站中提取 API 相关知识. 从 API 参考文档提取的知识包括 API 实体、领域概念实体以及它们之间的关系; 从 SO 问答网站提取的知识包括 API 实体、API 概念实体、问答 QA 实体以及它们之间的关系体.

由表 7 可知, 多信息融合的推荐效果优于单信息源. 信息融合将与问题相关的 API 通过提及的概念间接关联起来, 提高了推荐效果. 值得注意的是, 基于 API 参考文档和基于 SO 问答网站的推荐效果相比差距较大. 分析原因可能是: API 参考文档提供的功能描述不涉及具体应用场景, 因此它包含的领域概念很难和具体问题中的关键词对应起来, 导致仅使用 API 参考文档的推荐效果不尽人意.

表 6 和基线方法在时间成本方面的对比

方法	方法构建时间 (min)	查询时间 (s/query)
AKG-CMISF	16	1
BIKER	5	2

表 7 多源信息融合和单信息源推荐结果对比

方法	HR	MAP	MRR
SO	0.726	0.490	0.583
API Doc	0.322	0.181	0.149
AKG-CMISF	0.774	0.558	0.701
改进SO (%)	6	13	20
改进API Doc (%)	140	208	370

4 结论

本文提出了一种基于多源信息融合的 API 知识图谱构建方法 (AKG-CMISF). 该方法融合了 API 的功能和结构信息以及 SO 问答网站中 API 在具体场景下的使用信息, 提高了 API 检索的准确性. 实验从信息提取和 API 推荐效果两个方面验证了 API 知识图谱的有效性. 与现有 API 推荐系统相比, 本文基于 API 知识图谱的推荐方法在常见指标上均有提升. 同时通过实验验证了基于多信息源的 API 推荐效果相较于单信息源有明显提升.

下一步工作将基于该方法研发 API 知识图谱构建工具, 并将其应用在多个项目的 API 推荐.

参考文献

1 Huang Q, Xia X, Xing ZC, *et al*. API method recommendation without worrying about the task-API knowledge gap. 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). Montpellier: IEEE, 2018. 293–304.

2 Zhang JX, Jiang H, Ren ZL, *et al*. Recommending APIs for API related questions in stack overflow. IEEE Access, 2017, 6: 6205–6219. [doi: 10.1109/ACCESS.2017.2777845]

3 Rahman MM, Roy CK, Lo D. RACK: Automatic API recommendation using crowdsourced knowledge. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). Osaka: IEEE, 2016. 349–359. [doi: 10.1109/SANER.2016.80]

4 Stack Overflow. https://stackoverflow.com. [2020-06-07]

5 Ling CY, Zou YZ, Lin ZQ, *et al*. Graph embedding based API graph search and recommendation. Journal of Computer Science and Technology, 2019, 34(5): 993–1006. [doi: 10.1007/s11390-019-1956-2]

6 杨玉基, 许斌, 胡家威, 等. 一种准确而高效的领域知识图谱构建方法. 软件学报, 2018, 29(10): 2931–2947. [doi: 10.13328/j.cnki.jos.005552]

7 Liu MW, Peng X, Marcus A, *et al*. Generating query-specific class API summaries. Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Tallinn: ACM, 2019. 120–130. [doi: 10.1145/3338906.3338971]

8 Li HW, Li SR, Sun JM, *et al*. Improving API caveats accessibility by mining API caveats knowledge graph. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). Madrid: IEEE, 2018. 183–193. [doi: 10.1109/ICSME.2018.00028]

9 Treude C, Robillard M P. Augmenting api documentation with insights from stack overflow. Proceedings of the 38th International Conference on Software Engineering. Texas: ACM, 2016. 392–403.

10 Wang C, Peng X, Liu MW, *et al*. A learning-based approach for automatic construction of domain glossary from source code and documentation. Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Tallinn: ACM, 2019. 97–108. [doi: 10.1145/3338906.3338963]

11 Research-ESEC-FSE2019-DomainGlossary. https://github.com/FudanSELab/Research-ESEC-FSE2019-DomainGlossary, (2019-06-23).

12 Ponzanelli L, Bacchelli A, Lanza M. Seahawk: Stack

overflow in the IDE. Proceedings of the 2013 International Conference on Software Engineering. San Francisco: ACM, 2013. 1295–1298.

13 Ponzanelli L, Bavota G, Di Penta M, *et al*. Mining StackOverflow to turn the IDE into a self-confident programming prompter. Proceedings of the 11th Working Conference on Mining Software Repositories. Hyderabad: ACM, 2014. 102–111.

14 Nadi S, Treude C. Essential sentences for navigating stack overflow answers. 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 229–239. [doi: 10.1109/SANER48275.2020.9054828]

15 Beyer S, Macho C, Pinzger M, *et al*. Automatically classifying posts into question categories on stack overflow. Proceedings of the 26th Conference on Program Comprehension. Gothenburg: ACM, 2018. 211–221. [doi: 10.1145/3196321.3196333]

16 Ryu SE, Shin DH, Chung K. Prediction model of dementia risk based on XGBoost using derived variable extraction and hyper parameter optimization. IEEE Access, 2020, 8: 177708–177720. [doi: 10.1109/ACCESS.2020.3025553]

17 Xu BW, Xing ZC, Xia X, *et al*. AnswerBot: Automated generation of answer summary to developers' technical questions. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). Urbana: IEEE, 2017. 706–716. [doi: 10.1109/ASE.2017.8115681]

18 Mikolov T, Sutskever I, Chen K, *et al*. Distributed representations of words and phrases and their compositionality. Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2. Lake Tahoe: ACM, 2013. 3111–3119.

19 李文鹏, 王建彬, 林泽琦, 等. 面向开源软件项目的软件知识图谱构建方法. 计算机科学与探索, 2017, 11(6): 851–862. [doi: 10.3778/j.issn.1673-9418.1609026]

20 Files for Stackexchange. https://archive.org/download/stackexchange. [2020-06-07].

21 Manning CD, Surdeanu M, Bauer J, *et al*. The stanford CoreNLP natural language processing toolkit. Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Baltimore: Association for Computational Linguistics, 2014. 55–60.