软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(12):3714–3729 [doi: 10.13328/j.cnki.jos.005609]

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

基于代码结构知识的软件文档语义搜索方法*

林泽琦[1,2], 邹艳珍[1,2,3], 赵俊峰[1,2,3], 曹英魁[1,2], 谢 冰[1,2]

[1](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[2](北京大学 信息科学技术学院,北京 100871)

[3](北京大学(天津滨海)新一代信息技术研究院,天津 300450)

通讯作者: 邹艳珍, E-mail: zouyz@sei.pku.edu.cn

摘 要: 自然语言文本形式的文档是软件项目的重要组成部分.如何帮助开发者在大量文档中进行高效、准确的信息定位,是软件复用领域中的一个重要研究问题.提出了一种基于代码结构知识的软件文档语义搜索方法.该方法从软件项目的源代码中解析出代码结构图,并以此作为领域特定的知识来帮助机器理解自然语言文本的语义.这一语义信息与信息检索技术相结合,从而实现了对软件文档的语义检索.在StackOverflow问答文档数据集上的实验表明,与多种文本检索方法相比,该方法在平均准确率(mean average precision,简称 MAP)上可以取得至少 13.77%的提升.

关键词: 软件复用;自然语言文本;代码结构知识;信息检索;语义搜索

中图法分类号: TP311

中文引用格式: 林泽琦,邹艳珍,赵俊峰,曹英魁,谢冰.基于代码结构知识的软件文档语义搜索方法.软件学报,2019,30(12):3714–3729. http://www.jos.org.cn/1000-9825/5609.htm

英文引用格式: Lin ZQ, Zou YZ, Zhao JF, Cao YK, Xie B. Software text semantic search approach based on code structure knowledge. Ruan Jian Xue Bao/Journal of Software, 2019,30(12):3714–3729 (in Chinese). http://www.jos.org.cn/1000-9825/5609.htm

Software Text Semantic Search Approach Based on Code Structure Knowledge

LIN Ze-Qi[1,2], ZOU Yan-Zhen[1,2,3], ZHAO Jun-Feng[1,2,3], CAO Ying-Kui[1,2], XIE Bing[1,2]

[1](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

[2](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[3](Peking University Information Technology Institute (Tianjin Binhai), Tianjin 300450, China)

Abstract: Natural language text is a common form of knowledge representation in various software artifacts. During the practice of software reuse, software developers usually need to search the large amount of textual resource. This paper presents a software text semantic search approach based on code structure knowledge. This approach extracts a code structure graph from software source code and leverages it as a domain-specific knowledge base to analyze the semantic meanings of natural language texts. The semantic information is combined with information retrieval technology to re-rank text search results semantically. Experimental results on StackOverflow dataset show that this approach achieves at least 13.77% improvement in mean average precision (MAP) comparing to several text retrieval approaches.

Key words: software reuse; natural language text; code structure knowledge; information retrieval; semantic search

软件复用是在软件开发中避免重复劳动的解决方案,可以提高软件开发的效率与质量[1].近年来,随着开源

* 基金项目: 国家重点研发计划(2016YFB1000801); 国家杰出青年科学基金(61525201)

Foundation item: National Key Research and Development Program (2016YFB1000801); National Science Fund for Distinguished Young Scholars (61525201)

收稿时间: 2017-10-09; 修改时间: 2018-05-07; 采用时间: 2018-05-25

软件的快速发展,互联网上汇聚了大量可复用的软件项目,例如 Apache Lucene、Apache POI、JFreeChart 等.这些软件项目除了有源代码之外,往往还包含数量庞大、类型各异的自然语言文本形式的文档资源,例如用户手册、邮件归档、缺陷报告、用户论坛讨论等.这些文档中蕴含着丰富的知识,能够帮助软件开发者学习与理解该软件项目,从而更有效地对其进行复用.当软件开发人员在复用一个软件项目的过程中遇到问题,例如不知道一个功能的实现原理,或者是不知道一个需求应该被如何实现时,往往会在这些软件文档中进行搜索,以期找到一些相关的文本段落,其中包含可以用来回答该问题的信息.近年来,为项目内的文档资源提供搜索服务已经成为开源软件项目中最基本的辅助服务之一.如何提高软件文档搜索的准确率,已经成为软件复用领域的重要研究话题[2].

最基本也是最常见的软件文档搜索方法是基于关键词匹配的,这种方法将软件开发者输入的自然语言形式的问题与各个文档进行关键词匹配,并将关键词匹配程度最高的文档作为搜索结果.这种方法没有考虑到不同的词之间是具有语义关联的,因此搜索效果往往不够理想.在信息检索领域中,为了解决这一问题,相关研究工作主要分为两类:其一是基于统计分析的方法,如潜在语义分析(latent semantic analysis,简称 LSA)[3]、潜在狄利克雷分析(latent Dirichlet allocation)[4]、基于神经语言模型的词表示学习技术[5]等,此类方法依赖于训练所用的语料库,受数据稀疏和实际噪声的干扰较大;其二是基于概念知识的方法[6−12],使用概念实体来表示自然语言文本的语义,并以概念实体之间的结构关联关系来度量文本之间的语义相关度,此类方法直观、有效,但需要用到互联网上大型的通用概念知识库(如 WordNet、DBpedia、Freebase、Yago 等).然而,这些通用概念知识库中又往往不包含软件项目中领域特定的概念知识.总的来看,软件文档中的数据稀疏现象较为明显,这导致了基于统计分析的方法对搜索效果的改进是有限的.我们希望通过基于概念知识的方法来提升搜索效果,但其中亟待解决的问题是如何获取软件项目中领域特定的概念知识.

我们在研究中发现,软件项目中涉及的领域特定的概念实体大多都在这个软件的开发过程中被定义成了诸如类、接口等的代码元素,因此软件项目的代码结构图在一定程度上表达了领域特定的概念知识,可以用来帮助机器理解自然语言文本的语义.基于这种思路,本文提出了一种基于代码结构知识的软件文档语义搜索方法.更具体地:(1) 我们将这些代码元素抽取出来用于表示领域特定的概念实体;(2) 抽取这些代码元素之间的结构依赖关系(例如继承关系、调用关系)可以体现出相应的概念实体之间的语义关联关系,并利用它们来推理出文本之间的语义相关度.

与现有工作相比,本文的主要贡献是:

(1) 提出一种基于代码结构知识的文本语义表示方法,将文本的语义表示为代码结构图上的一个带权子图,从而实现了利用软件项目源代码中蕴含的概念知识来“解释”文本的语义;
(2) 提出一种基于代码结构知识的软件文档语义搜索方法,基于多关联数据的表示学习技术,利用代码元素间的结构依赖关系推理出文本间的语义相关度,从而解决了信息检索技术未能考虑不同的词之间的语义关联的问题;
(3) 在 StackOverflow 数据集上对该方法进行了验证,与其他多种信息检索方法(包括向量空间模型、潜在主题建模、神经语言模型等) 相比,本方法在平均准确率(mean average precision,简称 MAP)指标上可以取得至少 13.77%的效果提升.

本文第 1 节概述文本的研究目标与方法框架.第 2 节具体描述基于代码结构知识的软件文档语义搜索方法的实现细节,包括代码结构图提取、文档语义表示、文档语义相关度度量 3 个步骤.第 3 节介绍本文的实证研究,包括实验设计、采用的评测数据集及评价指标以及实验结果分析等.第 4 节与相关工作进行比较.第 5 节总结全文并对未来研究工作进行展望.

1 方法概览

本节首先通过一个实际例子来更为直观地说明本文的目标与基本方法,之后给出基于代码结构知识的软件文档语义搜索方法的总体框架.

1.1 示例说明

文档 *A* 和文档 *B* 都是在线问答社区 StackOverflow 上的问题,它们都与开源软件项目 Apache Lucene 有关.表 1 给出了这两个问题的标题以及详细描述.文档 *A* 的答案中指出,文档 *B* 所对应的问答对可以用于回答文档 *A* 中所提出的问题.因此,我们认为文档 *A* 和文档 *B* 是高度语义相关的.然而文档 *A* 和文档 *B* 中所包含的词汇的匹配程度并不高,因此在 StackOverflow 提供的基于词匹配的信息检索系统中,文档 *B* 被排在了检索结果中较为靠后的位置.

Table 1 Two sample documents selected from StackOverflow

表 1 从 StackOverflow 中选取的两个示例文档

文档 *A*:Lucene query parser to use filters for wildcard queries (新提出的问题)
问题描述:My problem is how to parse wildcard queries with Lucene that the query term is passed through a TokenFilter. I'm using a custom Analyzer with several filters (e.g. ASCIIFoldingFilter). My problem is that whether Lucene's QueryParser detects that one of the sub-queries is a WildcardQuery
文档 *B*:How to get a Token from a Lucene TokenStream (一个与文档 *A* 具有高度语义相关性的问题)?
问题描述:I'm trying to use Apache Lucene for tokenizing, and I am baffled at the process to obtain Tokens from a TokenStream. The worst part is that I'm looking at the comments in the JavaDocs of *TokenStream.incrementToken*(·) that address my question. Somehow, an AttributeSource is supposed to be used, rather than Tokens. I'm totally at a loss. Can anyone explain how to get token-like information from TokenStream?

为了解决这一问题,我们利用 Apache Lucene 项目源代码中的概念知识作为桥梁来建立文档 *A* 和文档 *B* 间的语义关联关系.这一想法主要分为两个步骤:首先建立起文本中的词汇与源代码中的代码元素之间的对应关系,从而能够用源代码中的概念知识来"解释"文本的语义;之后,利用代码元素在源代码中的结构依赖关系,度量文本之间的语义相关度.以文档 *A* 中的句子:"My problem is how to **parse wildcard queries** with Lucene that the **query term** is passed through a **TokenFilter** …"为例.该句中直接提及了 Apache Lucene 项目中的类 TokenFilter.同时,该句中还包含关键词:parse, wildcard,query 和 term.根据这些关键词,我们可以合理地猜测出这个句子的语义与 Apache Lucene 项目中的类 WildcardQuery、类 QueryParser 和类 Term 等代码元素有关.基于这一思想,这个句子的语义可以用这些代码元素的集合来表示.类似地,我们可以将文档 *B* 的语义表示为由类 Token、类 TokenStream、方法 *TokenStream.incrementToken*(·)、类 AttributeSource 等代码元素构成的集合.我们基于 TF-IDF 技术[13]为这些代码元素赋上权值,则每个文档的语义就可以分别被表示为代码结构图上的一个带权子图.

图 1 解释了我们如何利用源代码中的概念知识作为桥梁来建立文档 *A* 和文档 *B* 之间的语义关联关系.图中展示了 Apache Lucene 项目的代码结构图的一小部分.在代码结构图中,每个结点代表一个代码元素,而不同类型的有向边则代表代码元素间各种类型的结构依赖关系.与文档 *A* 相关的代码元素在图中以网格线标记,而与文档 *B* 相关的代码元素则在图中以浅灰色标记.由于在 Apache Lucene 项目的整个代码结构图中,两类结点是很接近的,因此,即使文档 *A* 和文档 *B* 中所包含的词汇的匹配程度并不高,我们也能根据代码结构图上的结构依赖关系来推理出文档 *A* 和文档 *B* 具有很高的语义相关度.

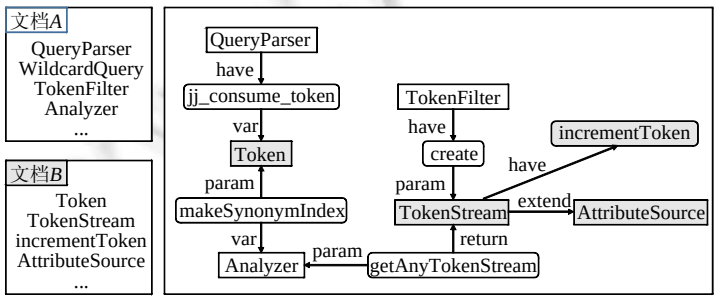


Fig.1 A brief illustration of how we can use conceptual knowledge in source code to capture the semantic relatedness between two documents

图 1 如何利用软件项目源代码中的概念知识来度量文档间的语义相似度的一个示例说明

综上,为了实现基于代码结构知识的软件文档语义搜索,需要解决两个方面的关键问题:一是如何准确地将文本的语义表示为代码结构图上的一个带权子图;二是如何利用这些子图在代码结构图上的结构依赖关系来计算文本的语义相关度.

1.2 方法框架

本文提出了一种基于代码结构知识的软件文档语义搜索方法.图 2 给出了该方法的整体流程.该方法主要由 3 部分构成.

(1) 代码结构图提取:这一部分用于从软件项目的源代码中提取出一个代码结构图.一个代码结构图由代码元素(例如类、接口、方法等)以及这些代码元素之间的结构依赖关联(例如调用、继承等)组成.在我们的方法中,代码结构图被作为软件项目特定的概念知识库来使用.

(2) 文档语义表示:这一部分用于将文档的语义表示为代码结构图上的一个带权子图.更具体地,我们基于词匹配和文本上下文分析建立文档中的词汇与代码结构图中的代码元素之间的关联关系,并基于 TF-IDF 技术为这些代码元素赋上权值.

(3) 文档语义相关度度量:这一部分用于度量 2 个文档之间的语义相关度.首先,我们基于多关联数据的表示学习技术,利用代码元素之间的结构依赖关联,计算不同的代码元素在结构上的相似度;随后,我们利用代码元素在结构上的相似度来计算代码结构图上的 2 个带权子图在结构上的相似度,从而用这一相似度来度量 2 个文档之间的语义相关度.在软件文档检索系统中,我们使用得到的文档语义相关度来对检索结果进行重排序:如果一个文档和查询语句之间具有较高的语义相关度,则该文档在检索结果中应该被排在更加靠前的位置.

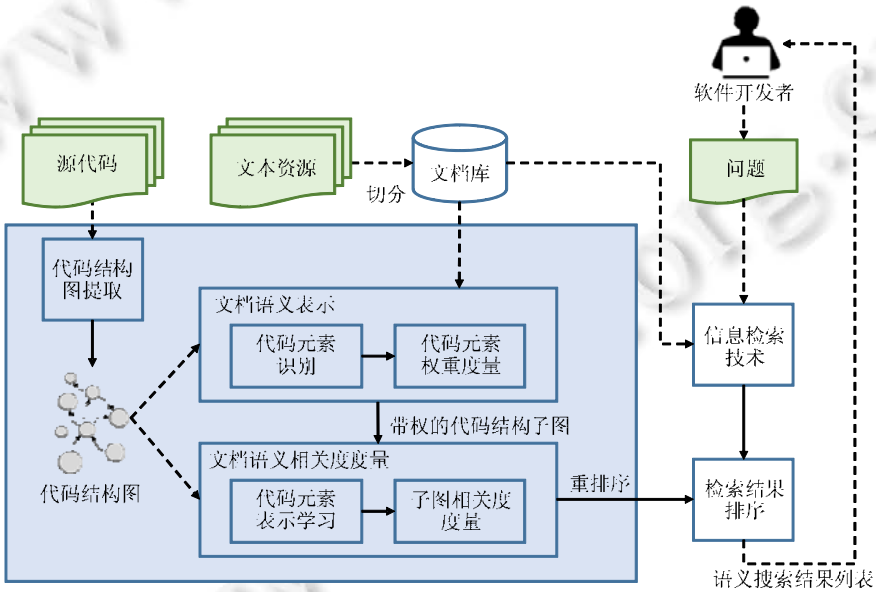


Fig.2 Workflow of the software text semantic search approach based on code structure knowledge

图 2 基于代码结构知识的软件文档语义搜索方法整体流程

2 方法实现

本节具体描述基于代码结构知识的软件文档语义搜索方法的实现细节,包括代码结构图提取、文档语义表示、文档语义相关度度量这 3 个步骤.

2.1 代码结构图提取

我们使用抽象语法树解析器 org.eclipse.jdt.core.ASTParser,从 Java 软件项目的源代码中提取代码结构图.在一个代码结构图中,结点代表源代码中的代码元素,结点间不同类型的有向边代表这些代码元素之间不同类型的结构依赖关系.在本方法中,我们考虑了 4 种类型的代码元素:类、接口、方法和域.我们考虑了这些类型的代码元素之间的 11 种类型的结构依赖关系,见表 2.

Table 2 Eleven different edge types in code structure graphs

表 2 代码结构图中的 11 种结构依赖关系类型

结构依赖关系类型	描述
extend	从一个子类/子接口指向其父类/父接口
implement	从一个类指向这个类所实现的接口
haveMethod	从一个类/接口指向这个类/接口中声明的方法
haveField	从一个类指向这个类中声明的域
throw	从一个方法指向这个方法所抛出的异常类型
fieldType	从一个域指向这个域的类型
parameterType	从一个方法指向这个方法的输入参数类型
variableType	从一个方法指向这个方法内部的变量的类型
returnType	从一个方法指向这个方法的输出类型
methodInvocation	从一个方法指向这个方法调用的方法
fieldInvocation	从一个方法指向这个方法调用的域

2.2 文档语义表示

文档语义表示用于将文档的自然语言文本表示为代码结构图上的一个带权子图.在这里,“文档”并不限于软件文档中的一个文本段落,还包括软件开发者输入的文本形式的查询语句.文档语义表示的实现分为 2 个步骤:代码元素识别与代码元素权重度量.

2.2.1 代码元素识别

文档语义表示中,最关键的部分是识别出该文档的语义该由该软件项目中的哪些代码元素来表示:首先,我们使用开源工具 Recodoc[14]来识别出文档中直接提及的代码元素;其次,我们基于词匹配对代码元素进行扩充.

Recodoc 是一个基于文本上下文分析的用于识别出文档中直接提及的代码元素的工具.例如,对于句子“My problem is how to parse wildcard queries with Lucene that the query term is passed through a TokenFilter …”,Recodoc 能够从中识别出类 TokenFilter.然而,仅靠类 TokenFilter 是不足以表达这个例句的语义的.例如,由于句子中出现了 wildcard 和 query 这两个关键词,因此我们可以合理地猜测,这个句子和类 WildcardQuery 也是相关的.类似地,根据句子中的 query,parser,term 等关键词,我们还可以识别出类 QueryParser 和类 Term 等代码元素.尽管这些代码元素并未在句子中被显式地提及,但它们能够体现出潜藏在句子的关键词中的语义信息,因此它们也应该被加入到这个句子的语义表示中去.基于这一思想,我们采用了基于词匹配的方法对代码元素进行扩充.

首先,我们对每个文档进行词法预处理.我们使用空白字符与标点符号将文档切分为词袋,并使用停用词表将其中的停用词过滤掉.我们使用 Porter 词根化算法(https://tartarus.org/martin/PorterStemmer/)对剩下的词进行词根化处理,最终得到了每个文档的关键词集合.

对于一个文档中的每一个关键词,我们从软件项目的源代码中找到所有标识符中包含这个关键词的代码元素,并将这些代码元素的集合称为这个关键词的候选代码元素集合.一个候选代码元素集合可能会包含大量的候选代码元素.例如,对于关键词 english,其候选代码元素集合包含 55 个代码元素,包括类 EnglishStemmer、类 EnglishAnalyzer、类 EnglishPossessiveFilterFactory,等等.如果我们直接将这样一个候选代码元素集合用于文档的语义表示,则计算代价巨大,且其中包含的大量噪音也会极大地影响计算效果.因此,基于对候选代码元素集合的观察,我们提出了 3 条启发式规则,以对集合中的候选代码元素进行过滤.

(1) 基于关键词对候选代码元素进行过滤.提出这一启发式规则所基于的假设是:若一个代码元素能够匹

配上更多的关键词,则说明这个代码元素更可能与这个文档是语义相关的.例如,filter 是一个文档中的一个关键词,类 TokenFilter 和类 StopFilter 是这个关键词的 2 个候选代码元素.该文档还包含关键词 token,但不包含关键词 stop.类 TokenFilter 的标识符中的关键词比率高于类 StopFiler,这说明文档中的关键词 filter 更有可能是指类 TokenFilter,而不是类 StopFiler.因此在这种情况下,我们过滤掉类 StopFilter,而留下类 TokenFilter.

(2) 基于代码元素类型对候选代码元素进行过滤.我们认为,在软件的源代码中,软件项目特定的概念知识主要是以类/接口的形式定义在项目的源代码中的,而方法/域则主要用于描述这些概念中的具体细节与动作.因此,如果一个关键词的候选代码元素集合中既包含类/接口,也包含方法/域,则我们会过滤掉所有方法/域,留下所有类/接口.

(3) 基于代码元素标识符长度对候选代码元素进行过滤.提出这一启发式规则所基于的假设是:若一个代码元素的标识符中包含的无法与文档中的关键词相匹配的信息越多,则可以认为该代码元素与这个文档的语义相似度越低.例如,english 是一个文档中的一个关键词,类 EnglishAnalyzer 和类 EnglishMinimalStemFilterFactory 是这个关键词的两个候选代码元素.在这种情况下,由于类 EnglishMinimalStemFilterFactory 的标识符的长度远大于类 EnglishAnalyzer,我们会过滤掉类 EnglishMinimalStemFilterFactory 而留下类 EnglishAnalyzer.在本方法中,我们将“长度远大于”定义为超过 5 个字母或是超过 1 倍.

对于每个文档,我们对其中的每个关键词的候选代码元素集合应用上述 3 条启发式规则进行过滤,所得到的代码元素集合即是用于表示该文档的语义的子图.

2.2.2 代码元素权重度量

我们基于 TF-IDF 技术为从文档中识别出的代码元素进行权重度量,从而将文档的语义表示为代码结构图上的一个带权子图.首先,我们使用 TF-IDF 技术度量出文档中的各个关键词的权重;随后,我们将该权重重新分配给这些关键词所对应的代码元素,如公式(1)与公式(2)所示.

$$w_c = \frac{w'_c}{\sum_{c' \in c} w'_{c'}} \tag{1}$$

$$w'_c = \sum_{t \in T_c} \frac{tfidf_t}{|C_t|} \tag{2}$$

在公式(1)和公式(2)中,w_c 表示代码元素 c 的权重,C 表示在文档中识别到的所有代码元素的集合,T_c 表示与代码元素 c 对应的所有关键词的集合,C_t 表示与关键词 t 对应的所有代码元素的集合.

经过代码元素识别与代码元素权重度量,我们为每个文档生成了代码结构图上的一个带权子图.我们使用这个带权子图来表示文档的语义.

2.3 文档语义相关度度量

文档语义相关度度量部分通过计算文档所对应的带权子图在代码结构图上的结构相关度来度量文档之间的语义相关度,从而能够对文本检索的结果进行重排序.具体分两步:代码元素表示学习和子图相关度度量.

2.3.1 代码元素表示学习

代码元素表示学习用于计算不同的代码元素之间的相似度,这一步骤是在线下完成的.

在本方法中,我们将代码元素间的相似度定义为:在代码结构图上,两个代码元素的上下文的相似度.这里的“上下文”是指代码元素以何种类型的结构依赖关系关联到哪个其他的代码元素.例如,如果类 A 和类 B 都是类 C 的子类,则我们认为类 A 和类 B 比较相似;如果类 A 和类 B 还共同作为方法 m 的输入参数,则我们认为类 A 和类 B 应该具有更高的相似度.此外,我们认为代码元素间的相似度是具有传递性的.例如,假设类 A 和类 B 具有很高的相似度,方法 x 和方法 y 分别用到了类 A 和类 B,则我们认为方法 x 和方法 y 也具有较高的相似度.

我们使用多关联数据表示学习算法 TransR[15]来计算不同的代码元素之间的相似性,其基本思想是:通过机器学习的方法,将代码结构图中的每个代码元素表示为一个低维、实值的向量,两个向量之间的距离越近,则说

明这两个向量所对应的代码元素的相似度越高.TransR 的基本假设是:存在一个 K_0 维的向量空间,使得代码结构图中的每一个结点都可以表示为该空间中的一个向量.两个向量之间的距离越近,则说明这两个向量所对应的代码元素的相似度越高.我们称这个向量空间为主向量空间.同时,对于每种类型的结构依赖关系 r,都存在一个与之对应的 K_1 维的向量空间,我们称其为面向关系类型 r 的向量空间.对于代码结构图中的任意一个结点 e,我们将它在主向量空间中对应的向量记为 $\boldsymbol{e}$.我们使用 3 元组$\langle h,r,t\rangle$来表示一条始于结点 h、指向结点 t、类型为 r 的边.对于这样一条边,TransR 算法首先将其首尾两个结点通过一个与 r 有关的线性变换 $\boldsymbol{M}_r$ 映射到面向关系类型 r 的向量空间中去.

$$\boldsymbol{h}_r=\boldsymbol{h}\boldsymbol{M}_r \tag{3}$$

$$\boldsymbol{t}_r=\boldsymbol{t}\boldsymbol{M}_r \tag{4}$$

随后,TransR 算法在面向关系类型 r 的向量空间中对结点 h 与结点 t 所对应的向量进行线性约束,即

$$f_r(h,t)=|\boldsymbol{h}_r+\boldsymbol{r}-\boldsymbol{t}_r| \tag{5}$$

其中,$\boldsymbol{r}$ 是一个与 r 有关的平移变换.$f_r(h,t)$是用于进行学习的似然函数,即如果 3 元组$\langle h,r,t\rangle$存在于代码结构图中,则 $f_r(h,t)$应该尽可能地小;否则,$f_r(h,t)$应该尽可能地大.综合考虑所有可能的 3 元组的 $f_r(h,t)$,就得到了总体的最小似然优化函数:

$$L=\sum_{\langle h,r,t\rangle\in G,\langle h',r',t'\rangle\notin G}\frac{1}{1+e^{-(f_r(h,t)-f_{r'}(h',t'))}} \tag{6}$$

我们使用随机批量梯度下降法,以 L 为目标进行最小似然优化,从而训练出模型参数.这些参数包括每个结点对应的一个 K_0 维的向量以及每种类型的结构依赖关系 r 所对应的一个 $K_0\times K_1$ 的线性变换 $\boldsymbol{M}_r$ 和一个 K_1 维的平移变换 $\boldsymbol{r}$.在训练过程中,我们限制所有向量的长度不超过 1.

图 3 对 TransR 的基本思想进行了简单解释.在图 3 中,我们考虑方法 m_1 以及被方法 m_1 所调用的 3 种方法 m_2,m_3 和 m_4.如图所示:对于这种情况,在面向关系类型 methodInvocation 的向量空间中,TransR 倾向于将方法 m_2,m_3 和 m_4 所对应的向量约束在一个以 $\boldsymbol{m}_1\boldsymbol{M}_r+\boldsymbol{r}$ 为中心的小邻域内,即 $\boldsymbol{m}_2\boldsymbol{M}_r\approx\boldsymbol{m}_3\boldsymbol{M}_r\approx\boldsymbol{m}_4\boldsymbol{M}_r\approx\boldsymbol{m}_1\boldsymbol{M}_r+\boldsymbol{r}$.相应地,在主向量空间中,方法 m_2,m_3 和 m_4 所对应的向量就被约束在同一个流型附近.因此,两个结点在代码结构图中的上下文越相似,则用来约束它们的流型越多,它们的位置也就会越倾向于靠近.通过以 L 为最小似然函数的整体优化,即可生成每一个结点在主向量空间中所对应的向量.两个向量之间的距离越近,说明这两个向量所对应的代码元素的相似度越高.我们将两个代码元素 a 和 b 之间的相似度定义为 $1-|\boldsymbol{a}-\boldsymbol{b}|$.

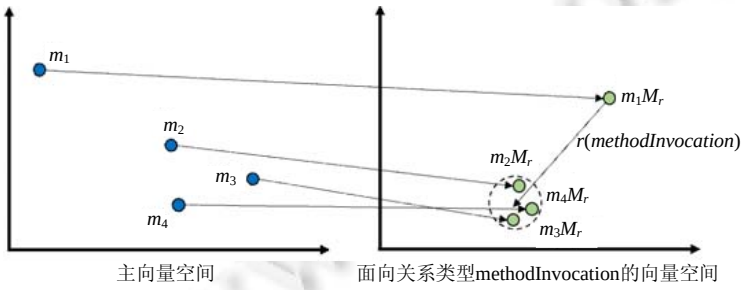


Fig.3 A brief illustration of TransR

图 3 对 TransR 的简单解释

2.3.2 子图相关度度量

在之前的文档语义表示步骤中,我们将每个文档的语义表示成为了代码结构图上的一个带权子图.在子图相关度度量步骤中,我们将在代码结构图上计算两个不同带权子图的相关度,以此来度量两个文档间的语义相关度.

首先,我们将代码元素 c 到一个带权子图 C 的相似度定义为代码元素 c 到 C 中的各个代码元素的相似度的最大值,如公式(7)所示.

$$sim(c,C)=\max_{c'\in C} sim(c,c') \tag{7}$$

记 C_q 为查询语句所对应的带权子图,C_d 为一个候选文档所对应的带权子图,我们定义 C_q 相对于 C_d 的相似度为

$$sim(C_q \to C_d)=\sum_{c\in C_d} sim(c,C_q)\times w_c \tag{8}$$

类似地,我们可以定义 C_d 相对于 C_q 的相似度 $sim(C_d \to C_q)$:把公式(8)中的 C_q 和 C_d 进行交换即可.

之后,我们将 C_q 和 C_d 之间的相似度定义为

$$sim(C_q,C_d)=\frac{sim(C_d \to C_q)+sim(C_q \to C_d)}{2} \tag{9}$$

我们使用 $sim(C_q,C_d)$来度量查询语句 q 和候选文档 d 之间的语义相关度,并使用它来对文本检索系统的检索结果进行重排序,从而实现语义搜索的效果,如公式(10)所示.

$$S(d)=\alpha\cdot S_0(d)+(1-\alpha)\cdot sim(C_q,C_d) \tag{10}$$

在公式(10)中,α是一个取值范围在[0,1]区间上的实值超参数;$S_0(d)$表示文本检索系统对文档 d 的原始评分,一般而言,这一评分是基于该文档与查询语句的表层文本相似度的.通过正则化操作,我们将 $S_0(d)$的取值范围限定在[0,1]区间上.$sim(C_q,C_d)$的取值范围也是[0,1].之后,我们将 $S_0(d)$与 $sim(C_q,C_d)$通过超参数α进行凸组合,从而实现对文本检索系统的语义化重排序.

3 实验与分析

为了验证本文所提出的方法的有效性,我们基于 StackOverflow 数据集设计了一系列实验.这些实验用于回答下列研究问题.

- 研究问题 1:本文所提出的方法是否能够有效地改进软件文档检索的效果?

我们关心本文所提出的方法在对文本检索系统的检索结果进行重排序后,是否能够将有价值的文档排在更靠前的位置.为了验证这一点,首先,我们研究本方法中的超参数的选取;随后,我们将本方法与多种已有的文本检索方法进行对比.

- 研究问题 2:本文中所使用的文本语义表示方法的有效性如何?

在本文所提出的方法中,我们将每个文档的语义表示为代码结构图上的一个带权子图.我们关心这样的一个带权子图是否能够有效地表达文本内容中蕴含的语义信息,从而改进软件文档检索的效果.因此,我们设计了相应的实验,将本文所使用的文本语义表示方法与若干变种方法进行对比,以验证文中所述的方法的有效性.

- 研究问题 3:本文中所使用的文本语义相关度度量方法的有效性如何?

在本文所提出的方法中,我们将代码结构图中的所有代码元素映射到同一个低维、实值的向量表示空间中,再通过这些代码元素的表示向量之间的距离作为桥梁来度量不同的文本之间的语义相关度.其中,我们将不同的代码元素之间的相似度定义为在代码结构图上,两个代码元素的上下文的相似度.我们关心这样的定义方式在文档的语义搜索任务中是否有效.因此,我们设计了相应的实验,将文本所使用的基于多关联数据表示学习技术的代码元素相似度计算方法与若干其他代码元素相似度计算方法进行对比,以验证本文中的文本语义相关度度量部分的有效性.

3.1 实验设计

我们使用 3 个著名的 Java 开源软件项目:Apache Lucene、Apache POI 和 JFreeChart 作为我们的实验对象.实验任务为软件文档检索,是在这 3 个项目上分别完成的.实验设计的细节列举如下.

3.1.1 文档集

StackOverflow 是一个著名的面向软件开发者的在线问答社区,其上积累了大量与软件开发有关的问答形式的文档.因此,我们从 StackOverflow 数据集中抽取出我们的实验验证所需的文档集.对于上述的两个用于实验的开源软件项目中的每一个项目,我们从 StackOverflow 上下载标签中包含该项目名称的所有相关的问答对,

并将每个问题以及它所对应的被采纳的答案定义为一个文档.若一个问题没有被采纳的答案,则这个问题并不会被收录到我们的文档集中.这一做法是为了对文档集中的文档质量进行保证.这样,就分别为这 3 个软件项目构造了一个文档集.这 3 个文档集的一些基本统计数据,包括文档数量、平均关键词数量等,可在表 3 中找到.

Table 3 Various statistics from our dataset

表 3 实验验证数据集基本统计信息

项目名	源代码版本	代码元素数量	结构依赖关联数量	文档数量	文档平均关键词数量	文档平均关联到的代码元素数量
Apache Lucene	6.3.0	19 701	65 216	4 753	64.13	85.96(7.48)
Apache POI	3.14	37 594	87 629	1 348	70.85	90.10(18.36)
JFreeChart	1.0.19	12 655	32 974	1 078	78.98	38.89(21.60)

3.1.2 代码结构图

对于每个软件项目,我们的方法从它的源代码中提取出了一个代码结构子图.一个软件项目往往会有很多个不同的版本,我们选取其中最新的版本进行代码结构图的提取.与代码结构图相关的统计信息也在表 3 中进行了展示,包括源代码版本、代码元素数量、关联关系数量.在一台配置了 3.40GHz 双核处理器的服务器上,解析生成 3 个项目的代码结构图的时间开销分别为 18min、31min 和 12min.

在我们的方法中,我们将每个文档的语义表示为代码结构图上的一个带权子图.表 3 中的“文档平均关联到的代码元素数量”这一列给出了这些带权子图中的平均结点数量.例如,“85.96(7.48)”是指,平均而言,每个文档会被关联到 85.96 个代码元素,其中,有 7.48 个代码元素是在文档中被直接提及的.

3.1.3 测试集

为了验证我们方法的有效性,对于每个软件项目,我们从文档集中随机抽取出 100 个问题作为测试输入.在被抽中后,这些问题所对应的文档被我们从文档集中删去.我们的实验任务是:对于每个问题,在文档集中寻找可以用于回答该问题的文档.

我们使用基于 TF-IDF 的向量空间模型实现了一个经典的文本检索系统.我们对每个测试用的问题进行关键词提取,形成查询语句输入到该文本检索系统中,从而得到与各个问题在表层文本上具有较高相似度的文档.对于每个问题,我们选取检索结果中的前 20 个文档作为候选文档.我们的目标是对这 20 个文档进行重排序,使得有价值的文档能够被重排序到更靠前的位置.为了得到测试集,我们让 3 位计算机专业的高年级本科生对这些候选文档进行标注.这些标注人员都有 1 年以上的 Java 编程经验,且都有过复用这 3 个开源软件项目的经历.对于每个测试问题,我们将该问题以及这个问题的被采纳的答案显示给标注者.随后,对于 20 个候选文档,标注者分别将它们与已有的被采纳的答案进行对比,以判断这些文档是否对解决该问题有价值.对于一个问题-文档对,标注者可以对其标注“是”或者“否”.若 3 个标注者都将一个文档标注为“是”,则我们认为该问题-文档对是一个正样本;否则,我们认为这个问题-文档对是一个负样本.图 4 给出了各个问题所拥有的正样本的数量分布.平均而言,对于每个问题,会有 2.76 个文档被认为是对解决该问题有帮助的.

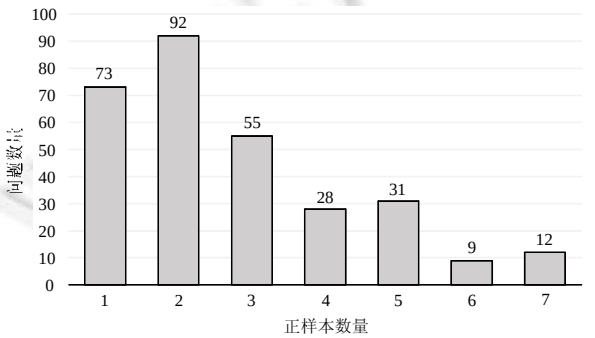


Fig.4 Distribution of the number of ground truth related documents

图 4 各个问题所拥有的正样本的数量分布

3.1.4 比较对象

我们将我们的方法记为 CK(conceptual knowledge).我们将该方法与如下 4 个文本检索方法进行对比.

(1) TF-IDF.这一文本检索方法首先将每个文档表示为词向量空间中的一个 TF-IDF 向量,之后使用两个向量之间的余弦相似度来度量文档之间的相似度[13].

(2) Okapi BM25.这一文本检索方法是 TF-IDF 方法的概率化改进,在信息检索领域有着广泛的应用[16].我们使用 Apache Lucene 中提供的 Okapi BM25 模块来实现这一方法.

(3) LDA.LDA 是一种在文本处理领域被广泛使用的潜在主题生成模型,这一方法将每个文档表示为若干个潜在主题上的一个分布[4].我们使用开源软件项目 mallet 中提供的 LDA 模块来计算每个文档的主题分布(经过手工调优,我们将 LDA 模型中的各个超参数的取值设置如下:主题个数 k=100,α=0.5,β=0.01).之后,我们将每个主题分布视为一个向量,并使用 KL 距离来度量两个文档之间的相似度.

(4) Word2vec.Word2vec 是一种在文本处理领域被广泛使用的神经语言模型,这一方法能够从文本语料库中的上下文信息中对各个单词的语义进行表示学习,从而将各个单词映射为一个低维、实值的向量表示空间中的词向量[17].不同词向量之间的距离越近,意味着所对应的两个单词之间的语义相似度越高.我们采用 Ye 等人提出的方法[18],以 word2vec 词向量作为基本单元来计算两个文档间的相似度.

3.1.5 评测指标

我们使用信息检索领域中一个被广泛使用的评测指标,即平均准确率(MAP)来验证我们的方法的有效性.简单来说,对于多个测试点,平均准确率是指方法在每个测试点上的准确率的平均值.更具体地,公式(11)和公式(12)给出了平均准确率的定义.

$$MAP=\frac{\sum_{q=1}^{Q}AveP(q)}{Q} \tag{11}$$

$$AveP(q)=\frac{\sum_{k=1}^{n}P(k)\times rel(k)}{number\text{-}of\text{-}related\text{-}documents} \tag{12}$$

其中,

- k 是指检索结果列表中的名次.
- $P(k)$是指在检索结果的前 k 名中,正样本所占的比例.
- $rel(k)$的取值为 0 或 1.当其取值为 1 时,说明排在第 k 位的文档是一个正样本;否则,其取值为 0.

例如,考虑一个查询语句,有两个文档与其相关,它们分别被排在检索结果中的第 2 位和第 5 位.在这种情况下,平均准确率为 $\frac{1/2+2/5}{2}=0.45$.

3.2 实验结果

本节主要分析研究问题 1:本文所提出的方法是否能够有效地改进软件文档检索的效果?

在我们的方法中,我们将文档和代码之间的表层文本相似度和代码结构上的语义相关度进行了线性结合,从而对文本检索的结果进行重排序.α是一个超参数,代表表层文本相似度在其中的占比.若α=0,则说明我们只考虑代码结构上的语义相关度,而不考虑表层文本相似度;若α=1,则我们的方法退化为原始的文本检索方法(即 TF-IDF 方法).图 5 对于我们所选取的 3 个实验用的软件项目分别给出了不同的α取值对平均准确率的影响.

从图 5 中可以看出,我们方法的效果明显好于 TF-IDF 方法.这一提升是显著且普遍的,以 Apache Lucene 项目为例,当α=0.7 时,我们的方法达到了最高的平均准确率,为 0.314;而 TF-IDF 方法的平均准确率为 0.245(α=1).

此外,图 5 还说明了将文档和代码之间的表层文本相似度和代码结构上的语义相关度进行线性结合是有必要的.如果我们只考虑代码结构上的语义相关度,而不考虑表层文本相似度(α=0),平均准确率就会明显降低.

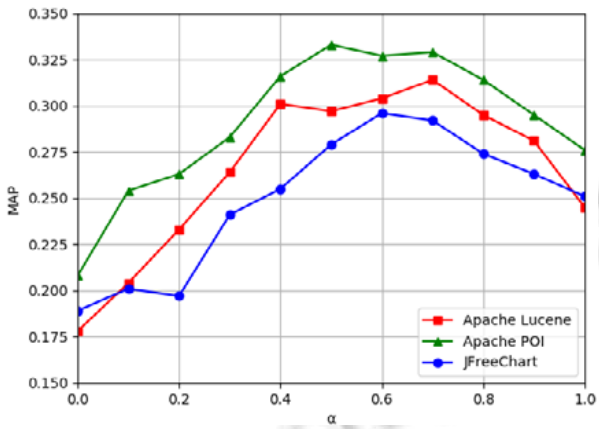


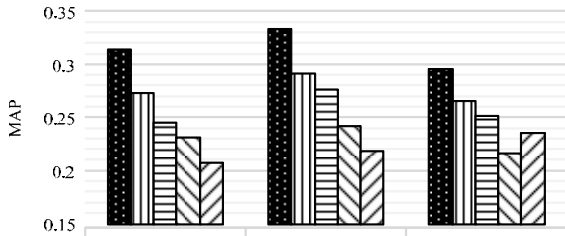
Fig.5 Effect of varying α on the MAP of our approach

图 5 调整本方法中的超参数α对平均准确率 MAP 的影响

对于 3 个软件项目,α的最佳取值分别为 0.7,0.5 和 0.6.在下文中,我们保持这样的取值.

在我们的方法中,代码元素表示学习的步骤上还有两个超参数:K_0 和 K_1,代表向量空间的维度.我们在实验过程中对这两个超参数在 100~500 的范围内进行调整,发现它们对实验结果的影响是很有限的.因此,我们在实验的全过程中统一将它们设置为 200.

图 6 以平均准确率为指标,将我们的方法与 TF-IDF 方法、Okapi BM25 方法、LDA 方法和 word2vec 方法进行了对比.



	Apache Lucene	Apache POI	JFreeChart
CK	0.314	0.333	0.296
Word2vec	0.273	0.291	0.266
TF-IDF	0.245	0.276	0.251
LDA	0.231	0.242	0.216
Okapi BM25	0.207	0.218	0.235

Fig.6 Comparison of our approach and four other text retrieval approaches

图 6 本方法与其他 4 种文本检索方法的平均准确率效果对比

表 4 给出了各种方法在这 3 个项目上的平均 MAP 得分,以及它们相对 TF-IDF 方法的提升比率.

Table 4 Average MAP results across all the three software projects

表 4 各种方法在 3 个项目上平均 MAP 值

方法	CK	word2vec	TF-IDF	BM25	LDA
MAP	0.314	0.276	0.257	0.230	0.220
MAP Improvement (w.r.t. TF-IDF) (%)	22.2	7.4	0	−10.5	−14.4

从图 6 和表 4 中可以看出,我们的方法相对于 TF-IDF 方法能够取得 22.2%的效果提升,而表现第 2 好的 word2vec 方法的提升率仅为 7.4%.因此,与其他文本检索方法相比,我们提出的基于代码结构知识的软件文档语义搜索方法的有效性是显著的.

3.3 文本语义表示方法比较分析

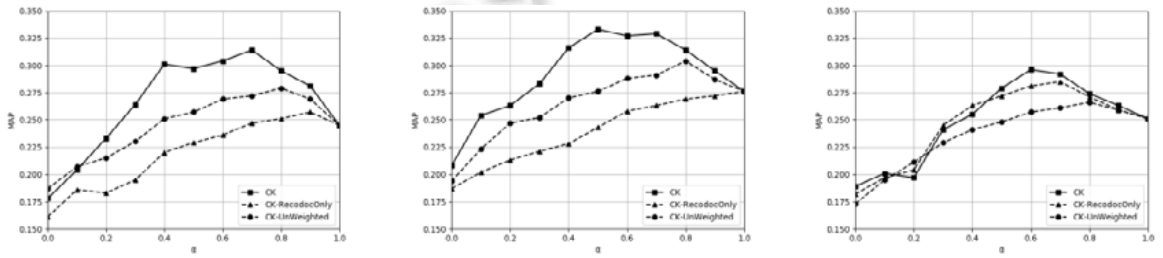
本节主要分析研究问题 2:本文中所使用的文本语义表示方法的有效性如何?

为了验证本文所使用的文本语义表示方法的有效性,将我们的方法与如下两种方法进行对比.

(1) CK-RecodocOnly.这一方法在我们方法的基础上,对文本语义表示步骤进行了修改:用于表示文本语义的子图中仅仅包含由 Recodoc 识别出的在文本中直接被提及的代码元素,而不再包含基于词匹配识别出的其他代码元素.将我们的方法与这一方法进行比较分析的目的是验证基于词匹配对代码元素进行扩充是有必要的.

(2) CK-UnWeighted.这一方法在我们方法的基础上,对文本语义表示步骤进行了修改:不再为子图中的代码元素计算权重.将我们的方法与这一方法进行比较分析的目的是验证代码元素权重度量是有必要的.

图 7 给出了我们的方法与这两种方法的比较结果.可以看到,我们的方法达到的平均准确率显著高于这两种方法.例如,在 Apache Lucene 项目中,我们的方法在α=0.7 时达到了最高的平均准确率 0.314;对于 CK-RecodocOnly 方法,其最高准确率为 0.257(α=0.9);对于 CK-UnWeighted 方法,其最高准确率为 0.279(α=0.8).



(a) Apache Lucene (b) Apache POI (c) JFreeChart

Fig.7 Comparison between different conceptual document representation strategies

图 7 文本语义表示方法比较分析结果

这一结果说明:

(1) 使用 Recodoc 识别出的文档中直接被提及的代码元素不足以用来表达文档的语义.因此,基于词匹配对代码元素进行扩充是有必要的.

(2) 基于 TF-IDF 技术对子图中的代码元素进行权重度量能够很好地提升文本语义表示的有效性.

3.4 文本语义相关度度量方法比较分析

本节主要分析研究问题 3:本文中所使用的文本语义相关度度量方法的有效性如何?

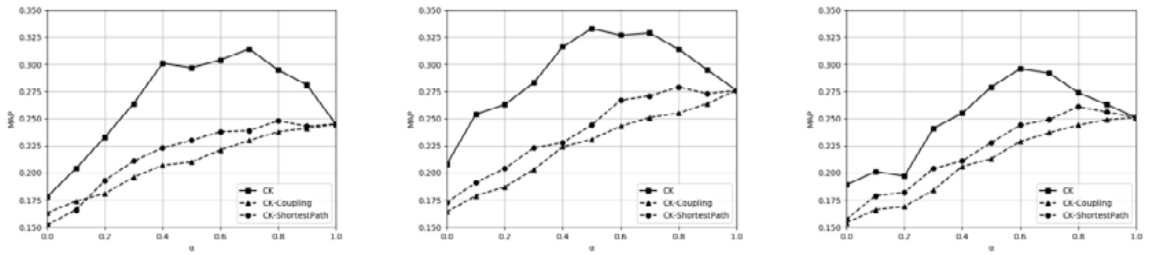
为了验证本文所使用的文本语义相关度度量方法的有效性,将我们的方法与如下两种方法进行对比.

(1) CK-Coupling.这一方法在我们的方法的基础上,对文本语义相关度度量步骤进行了修改:在计算两个代码元素之间的相似度时,使用由 Briand 等人提出的基于耦合度的算法[19],而不是我们的方法中提出的基于多关联数据表示学习的算法.

(2) CK-ShortestPath.这一方法在我们的方法的基础上,对文本语义相关度度量步骤进行了修改:在计算两个代码元素之间的相关度时,在代码结构图上使用基于 Dijkstra 最短路径长度的算法,而不是我们的方法中提出的基于多关联数据表示学习的算法.

图 8 给出了我们的方法与这两种方法的比较结果,可以看到,我们的方法达到的平均准确率显著高于这两种方法.例如,在 Apache Lucene 项目中,CK-ShortestPath 方法在α=0.8 时达到了最高的平均准确率 0.248,而这一结果只是略微高于 TF-IDF 方法;而对于 CK-Coupling 方法,达到最高平均准确率的α值为 1,这意味着其效果是低于 TF-IDF 方法的.这一实验结果表明,我们将代码元素之间的相似度定义为它们在代码结构图上的上下文相

似度的做法是适合于文本语义相关度度量这一任务的.与我们的做法相比,其他代码元素相似度的计算方法并不适合文本语义相关度度量任务.



(a) Apache Lucene (b) Apache POI (c) JFreeChart

Fig.8 Comparison between different code element pairwise similarity calculation strategies

图 8 文本语义相关度度量方法比较分析结果

4 相关研究工作

本文的相关研究工作可分为两类:(1) 软件文档检索技术;(2) 基于概念知识的语义搜索技术.

4.1 软件文档检索

在软件工程领域中,信息检索技术已被成功应用在各种类型的软件文档上,包括源代码文件[20−25]、软件文档[26−29]、缺陷报告[30,31]、在线问答社区[32,33]等.

如何挖掘不同的词之间的语义相关度,并用其来改进检索效果,是信息检索领域中的重要问题.在软件工程中,从软件文档中挖掘不同的词之间的语义相关度的代表性工作包括:Tian 等人[34]基于点互信息 PPMI 对 StackOverflow 上的问答文档进行挖掘,从而生成软件项目特定的词汇相似度数据库;Ye 等人[18]使用神经语言模型对软件项目的相关文档进行分析,生成各个词汇的低维实值词向量,从而能够度量不同的词汇之间的相似度;Howard 等人[35]和 Yang 等人[36]通过分析源代码中的上下文来挖掘不同的词汇之间的相似度;Wang 等人[37]则通过分析 FreeCode 代码库中的标签信息来挖掘不同的词汇之间的相似度;Haiduc 等人[2]提出了一种基于有监督的机器学习的自动查询重构方法 Refoqus,以对查询语句进行扩充;Panichella 等人[38]提出了一种结合潜在主题建模技术和遗传算法的软件文档检索方法;等等.然而,上述方法都是基于对软件文档语料库的统计分析的,其改进效果依赖于词汇的特定模式在语料库中的频繁出现.对于在语料库中出现次数并不够多的词汇而言,这些方法难以挖掘出它们与其他词汇的相似度.

此外,研究者们提出了多种基于非文本特征的软件文档检索方法.例如,Ponzanelli 等人[32]提出了一种面向 StackOverflow 问答文档的检索方法,在该方法中,问题评分、答案评分、用户等级、问题标签等非文本特征是检索结果重排序的重要因素;Ye 等人[25]提出了一种面向缺陷报告的源代码文件检索方法,该方法考虑了最近缺陷修复日期、缺陷修复频率等非文本特征;Zou 等人[39]提出了一种面向问题的软件文档检索方法,该方法考虑了查询语句中的疑问词(例如“how”“why”,which”,等等)对检索策略的影响.这些方法在特定的文本检索任务中能够有效地提升检索效果,但这些特定的非文本特征不具有在各种类型的文档中的可扩展性.例如,问题评分、答案评分、用户等级、问题标签等非文本特征能够很好地改进针对 StackOverflow 问答文档的检索效果,但其他类型的文档(如需求/设计文档、缺陷报告、邮件归档等)并不具有这些特征,无法使用这些特征.

4.2 基于概念知识的语义搜索

近年来,随着互联网上大规模的通用概念知识库(例如 WordNet、DBpedia、Freebase、谷歌知识图谱、Yago 等)的飞速发展,研究者们提出了多种基于概念知识的语义搜索方法[6−12],以在信息检索中发现并利用不同词之间的语义相关度.在这里,概念知识是指现实世界中存在的各种实体以及这些实体之间的各种语义关联关系.概念知识一般被表示为 3 元组,例如⟨Michael_Jackson,publish_song,Billi_Jean⟩,⟨Barack_Obama,born_in,Honolulu⟩

等.基于概念知识的语义搜索的基本过程是:首先,识别出文档中提及的实体;其次,利用实体之间的关联关系来度量文档之间的语义相关度;最后,利用得到的语义相关度对文本检索的结果进行重排序.这些研究工作证明了引入额外的概念知识来帮助机器理解文本的语义能够有效地解决词汇间隙问题,改进文本检索的效果.受到这些工作的启发,我们希望基于概念知识来实现软件文档的语义搜索.然而,软件文档中往往涉及到大量的软件项目特定的概念知识,而互联网上常用的大型通用知识库中却往往并不包含这些知识.因此,在软件文档上直接使用互联网上常用的大型通用知识库进行语义搜索所能达到的效果提升是非常有限的,我们需要寻找该软件项目特定的概念知识来源.一些研究工作将专家构造的本体作为软件项目特定的概念知识来源,以解决文本检索中的词汇间隙问题.然而,由专家构造本体的成本过高,这导致了此类方法的可用性不足.

本文提出,软件项目特定的概念知识可以从该软件项目的源代码中抽取得到,并用于实现软件文档的语义搜索.在软件工程的研究社区中,软件源代码中的结构依赖信息已经被广泛地应用在了多种自动软件工程任务中[40−46],以帮助软件开发人员在软件的开发、维护和复用过程中对软件项目进行学习与理解.McMillan 等人[41]和 Panichella 等人[45]研究了如何将软件源代码中的结构依赖信息用作软件项目特定的概念知识,从而在文本检索任务中对查询语句进行语义扩充.本文进一步研究如何利用从软件项目源代码中抽取出的软件项目特定的概念知识来度量文档之间的语义相关度,从而在语义层面上对文本检索的结果进行重排序.

5 总结与展望

本文提出了一种基于代码结构知识的软件文档语义搜索方法.在该方法中,我们的基本思想是:从软件项目的源代码中提取出代码结构图,以此作为先验知识来帮助机器理解无结构的自然语言文本.更具体地,我们将每个文档的语义表示为该代码结构图上的一个带权子图,并通过计算带权子图在代码结构图上的相似度来度量文档之间的语义相关度.我们利用语义相关度对文本检索系统返回的检索结果进行重排序,从而实现了语义搜索.我们在 StackOverflow 数据集上对本方法的有效性进行了验证.以平均准确率(MAP)作为评测指标,结果表明,本方法相对于其他文本检索方法可以取得至少 13.77%的效果提升.

我们下一步的工作计划包括:研究如何改进本方法中的文本语义表示步骤,获取能够更准确地表示文本语义的带权子图;在更多类型的软件文档上验证本方法的通用性和有效性,例如缺陷报告、需求文档、设计文档、邮件归档等等.

References:

[1] Yang FQ, Mei H, Li KQ. Software reuse and software component technology. Chinese Journal of Electronics, 1999;27(2):68−75 (in Chinese with English abstract).

[2] Haiduc S, Bavota G, Marcus A, Oliveto R, De Lucia A, Menzies T. Automatic query reformulations for text retrieval in software engineering. In: Proc. of the 2013 Int'l Conf. on Software Engineering. Deerwester: IEEE Press, 2013. 842−851.

[3] Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 1990,41(6):391−407.

[4] Blei DM, Ng AY, Jordan MI. Latent dirichlet allocation. Journal of Machine Learning Research, 2003,3(1):993−1022.

[5] Bengio Y, Ducharme R, Vincent P, Jauvin C. A neural probabilistic language model. Journal of Machine Learning Research, 2003, 3(2):1137−1155.

[6] Gabrilovich E, Markovitch S. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: Proc. of the IjcAI, Vol.7. 2007. 1606−1611.

[7] Egozi O, Markovitch S, Gabrilovich E. Concept-based information retrieval using explicit semantic analysis. ACM Trans. on Information Systems (TOIS), 2011,29(2):Article No.8.

[8] Schuhmacher M, Ponzetto SP. Knowledge-based graph document modeling. In: Proc. of the 7th ACM Int'l Conf. on Web Search and Data Mining. ACM Press, 2014. 543−552.

[9] Liu XT, Fang H. Latent entity space: A novel retrieval approach for entity-bearing queries. Information Retrieval Journal, 2015, 18(6):473−503.

[10] Xiong CY, Callan J. Esdrank: Connecting query and documents through external semi-structured data. In: Proc. of the 24th ACM Int'l Conf. on Information and Knowledge Management. ACM Press, 2015. 951−960.

[11] Raviv H, Kurland O, Carmel D. Document retrieval using entity-based language models. In: Proc. of the 39th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM Press, 2016. 65−74.

[12] Ni Y, Xu QK, Cao F, Mass Y, Sheinwald D, Zhu HJ, Cao SS. Semantic documents relatedness using concept graph representation. In: Proc. of the 9th ACM Int'l Conf. on Web Search and Data Mining. ACM Press, 2016. 635−644.

[13] Salton G, Christopher B. Term-weighting approaches in automatic text retrieval. Information Processing & Management, 1988, 24(5):513−523.

[14] Dagenais B, Robillard MP. Recovering traceability links between an API and its learning resources. In: Proc. of the 2012 34th Int'l Conf. on Software Engineering (ICSE). IEEE, 2012. 47−57.

[15] Lin YK, Liu ZY, Sun MS, Liu Y, Zhu X. Learning entity and relation embeddings for knowledge graph completion. In: Proc. of the AAAI, Vol.15. 2015. 2181−2187.

[16] Robertson SE, Walker S, Jones S, Hancock-Beaulieu MM, Gatford M. Okapi at TREC-3. National Institute of Standards and Technology (NIST), 1994. 109−126.

[17] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301. 3781, 2013.

[18] Ye X, Shen H, Ma X, Bunescu R, Liu C. From word embeddings to document similarities for improved information retrieval in software engineering. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 404−415.

[19] Briand L, Devanbu P, Melo W. An investigation into coupling measures for C++. In: Proc. of the 19th Int'l Conf. on Software Engineering. ACM Press, 1997. 412−421.

[20] Marcus A, Sergeyev A, Rajlich V, Maletic JI. An information retrieval approach to concept location in source code. In: Proc. of the 11th Working Conf. on Reverse Engineering. IEEE, 2004. 214−223.

[21] Poshyvanyk D, Guéhéneuc YG, Marcus A, Antoniol G, Rajlich V. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Trans. on Software Engineering, 2007,33(6):420−432.

[22] Jian Z, Zhang HY, Lo D. Where should the bugs be fixed?—More accurate information retrieval-based bug localization based on bug reports. In: Proc. of the 34th Int'l Conf. on Software Engineering. IEEE Press, 2012. 14−24.

[23] Dit B, Revelle M, Poshyvanyk D. Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. Empirical Software Engineering, 2013,18(2):277−309.

[24] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: Proc. of the 2013 IEEE/ACM 28th Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2013. 345−355.

[25] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2014. 689−699.

[26] Antoniol G, Canfora G, Casazza G, De Lucia A. Information retrieval models for recovering traceability links between code and documentation. In: Proc. of the Int'l Conf. on Software Maintenance. IEEE, 2000. 40−49.

[27] Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E. Recovering traceability links between code and documentation. IEEE Trans. on Software Engineering, 2002,28(10):970−983.

[28] Marcus A, Maletic JI. Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Proc. of the 25th Int'l Conf. on Software Engineering. IEEE, 2003. 125−135.

[29] Oliveto R, Gethers M, Poshyvanyk D, De Lucia A. On the equivalence of information retrieval methods for automated traceability link recovery. In: Proc. of the 2010 IEEE 18th Int'l Conf. on Program Comprehension (ICPC). IEEE, 2010. 68−71.

[30] Wang XY, Zhang L, Xie T, Anvik J, Sun JS. An approach to detecting duplicate bug reports using natural language and execution information. In: Proc. of the ACM/IEEE 30th Int'l Conf. on Software Engineering (ICSE 2008). IEEE, 2008. 461−470.

[31] Nguyen AT, Nguyen TT, Nguyen TN, Lo D, Sun CN. Duplicate bug report detection with a combination of information retrieval and topic modeling. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM Press, 2012. 70−79.

[32] Ponzanelli L, Bavota G, Di Penta M, Oliveto R, Lanza M. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In: Proc. of the 11th Working Conf. on Mining Software Repositories. ACM Press, 2014. 102−111.

[33] Ghaderi R. Improving the Retrieval of Related Questions in StackOverflow. Irvine: University of California, 2015.

[34] Tian Y, Lo D, Lawall J. Automated construction of a software-specific word similarity database. In: Proc. of the 2014 Software Evolution Week—IEEE Conf. on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE). IEEE, 2014. 44−53.

[35] Howard Matthew J, Gupta S, Pollock L, Vijay-Shanker K. Automatically mining software-based, semantically-similar words from comment-code mappings. In: Proc. of the 10th Working Conf. on Mining Software Repositories. IEEE Press, 2013. 377−386.

[36] Yang JQ, Lin T. Inferring semantically related words from software context. In: Proc. of the 2012 9th IEEE Working Conf. on Mining Software Repositories (MSR). IEEE, 2012. 161−170.

[37] Wang SW, Lo D, Jiang LX. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In: Proc. of the 2012 28th IEEE Int'l Conf. on Software Maintenance (ICSM). IEEE, 2012. 604−607.

[38] Panichella A, Dit B, Oliveto R, Di Penta M, Poshynanyk D, De Lucia A. How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In: Proc. of the 2013 35th Int'l Conf. on Software Engineering (ICSE). IEEE, 2013. 522−531.

[39] Zou YZ, Ye T, Lu YY, Mylopoulos J, Zhang L. Learning to rank for question-oriented software text retrieval. In: Proc. of the 2015 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2015. 1−11.

[40] Warr FW, Robillard MP. Suade: Topology-based searches for software investigation. In: Proc. of the 29th Int'l Conf. on Software Engineering. IEEE Computer Society, 2007. 780−783.

[41] McMillan C, Poshyvanyk D, Revelle M. Combining textual and structural analysis of software artifacts for traceability link recovery. In: Proc. of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering. IEEE Computer Society, 2009. 41−48.

[42] Scanniello G, Marcus A. Clustering support for static concept location in source code. In: Proc. of the 2011 IEEE 19th Int'l Conf. on Program Comprehension (ICPC). IEEE, 2011. 1−10.

[43] Chan WK, Hong C, Lo D. Searching connected API subgraph via text phrases. In: Proc. of the ACM SIGSOFT 20th Int'l Symp. on the Foundations of Software Engineering. ACM Press, 2012. Article No.10.

[44] McMillan C, Poshyvanyk D, Grechanik M, Xie Q, Fu C. Portfolio: Searching for relevant functions and their usages in millions of lines of code. ACM Trans. on Software Engineering and Methodology (TOSEM), 2013,22(4):Article No.37.

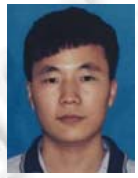
[45] Panichella A, McMillan C, Moritz E, Palmieri D, Oliveto R, Poshyvanyk D, De Lucia A. When and how using structural information to improve IR-based traceability recovery. In: Proc. of the 2013 17th European Conf. on Software Maintenance and Reengineering (CSMR). IEEE, 2013. 199−208.

[46] Scanniello G, Marcus A, Pascale D. Link analysis algorithms for static concept location: An empirical assessment. Empirical Software Engineering, 2015,20(6):1666−1720.

附中文参考文献:

[1] 杨芙清,梅宏,李克勤.软件复用与软件构件技术.电子学报,1999,27(2):68−75.

林泽琦(1992－),男,福建莆田人,博士生,主要研究领域为软件工程,软件复用,知识工程,数据挖掘.



曹英魁(1993－),男,博士生,主要研究领域为软件工程,软件复用,信息挖掘.

邹艳珍(1976－),女,博士,副教授,CCF 专业会员,主要研究领域为软件工程,软件复用,信息检索.

谢冰(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,形式化方法,软件复用.

赵俊峰(1974－),女,博士,副教授,CCF 高级会员,主要研究领域为软件工程,软件复用,Web 服务,云计算.