计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190300053

基于语义相似度的 API 使用模式推荐

张云帆[1]　周　宇[1,2]　黄志球[1,2]
1 南京航空航天大学计算机科学与技术学院　南京 210016
2 南京航空航天大学高安全系统的软件开发与验证技术工信部重点实验室　南京 211100
(fan0429@nuaa.edu.cn)

摘　要　在软件开发过程中,复用应用程序编程接口(Application Programming Interface,API)可以提高软件开发效率,但是使用不熟悉的 API 是一项耗时且困难的挑战。已有的研究往往将 API 作为用户输入的查询,通过在语料库中搜索该 API 的使用模式来进行推荐,但这并不符合开发人员的查询习惯。文中提出了一种基于自然语言语义相似度的 API 使用模式推荐方法(Semantic Similazing Based API Recommendation,SSAPIR)。该方法使用层次聚类算法来提取 API 使用模式,然后通过计算查询信息和 API 使用模式来描述信息之间的语意相似度,向开发人员推荐相关度高且被广泛使用的 API 使用模式。为了验证 SSAPIR 的有效性,文中从 GitHub 的高质量 Java 项目中提取 9 个流行的第三方 API 库的 API 使用模式以及 API 使用模式的描述信息,并根据这 9 个流行的第三方 API 库的自然语言查询进行 API 使用模式推荐。通过计算推荐结果的 Hit@K 准确率来验证 SSAPIR 的有效性,实验结果表明,层次聚类能有效提高推荐准确率,且 SSAPIR 在 Hit@10 平均准确率上达到了 85.02%,优于现有研究工作,能够很好地完成 API 使用模式推荐任务,为开发人员输入的自然语言查询提供精准的 API 使用模式。

关键词:API 使用模式推荐;语义相似度;层次聚类
中图法分类号　TP391

Semantic Similarity Based API Usage Pattern Recommendation

ZHANG Yun-fan[1],ZHOU Yu[1,2] and HUANG Zhi-qiu[1,2]
1 College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 210016,China
2 Ministry Key Laboratory for Safety-Critical Software Development and Verification,Nanjing University of Aeronautics and Astronautics,Nanjing 211100,China

Abstract　In the process of software development,reusing application programming interface (API) can improve the efficiency of software development. However,it is difficult and time-consuming for developers to use unfamiliar APIs. Previous researches tend to take APIs as inputs to search corpus and recommend API usage patterns,which does not conform to the habits of developers searching for API usage patterns. This paper proposed a novel Semantic Similarity based API Usage Pattern Recommendation approach (SSAPIR). This approach first adopts hierarchical clustering algorithm to extract API usage patterns,and then calculates the semantic similarity between queries and API usage patterns' description information,aiming to recommend highly relevant and widely used API usage patterns to developers. To verify the effectiveness of SSAPIR,Java projects are collected from GitHub,from which the API usage patterns related to the 9 popular third-party API libraries and their description information are extracted. Ultimately,this paper recommended API usage patterns based on natural language queries which are related to the 9 third-party API libraries. To verify the effectiveness of SSAPIR,this paper measured the Hit@K of the recommendation results. The experimental results demonstrate that SSAPIR can effectively improve the accuracy of recommendation results and achieves an average accuracy of 85.02% in terms of Hit@10,which outperforms the state-of-art work. SSAPIR can complete the API usage pattern recommendation task greatly and provide accurate API usage pattern recommendation for developers by taking natural language queries as inputs.

Keywords　API usage pattern recommendation,Semantic similarity,Hierarchical clustering

到稿日期:2019-03-15　返修日期:2019-06-23　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划项目(2018YFB1003900);中央高校基本科研业务费专项资金(NS2019055);江苏高校"青蓝工程"
This work was supported by the National Key R&D Program of China (2018YFB1003902),Fundamental Research Funds for the Central Universities (NS2019055) and Qing Lan Project.
通信作者:周宇(zhouyu@nuaa.edu.cn)

1 引言

随着计算机应用领域的不断扩大，软件的使用已经逐渐渗透到人们的生活中；新的软件形态和开发模式不断涌现，其规模和数量正以惊人的速度扩大和膨胀，有效的软件复用是提高软件开发效率、降低开发成本的重要方式。早期的软件复用偏重于方法学层面，这些潜在的复用实体往往属于开发者较为熟悉（如内部私有函数库）或较为知名的第三方库（如 JDK 中的应用程序编程接口、函数库等），种类和数量均较为有限。

然而，在软件开发过程中，开发人员常常需要实现不熟悉的编程任务，他们既要通过查询和搜索代码示例来进行代码复用，又要学习不熟悉的 API 的使用方法[1-2]。此外，已有程序可以帮助开发人员理解他人是如何解决类似问题的，并且开发人员可以在此基础上完成他们需要的功能。一方面，数以百万计的软件项目被托管在开源社区（如 GitHub），这些是开发人员复用代码和 API 的重要资源；另一方面，在线问答论坛（如 Stack Overflow）也帮助开发人员解决问题（包括代码和 API）[3]。但是，由于各种因素，开发人员仍然需要花费大量的时间来选择 API。为此，研究者提出了多种方法来推荐 API 使用模式[4-7]，API 使用模式是指能够实现一个功能所需要的一组 API 调用序列。

已有研究表明，频繁序列挖掘算法是提取 API 使用模式的有效途径。Xie 等[4]首次提出了一种经典的 API 使用模式挖掘算法 MAPO（Mining API usages from Open source repositories）。MAPO 使用聚类方法对相似 API 进行聚类，并通过 SPAM 算法挖掘 API 使用模式。Niu 等[7]提出了一种方法，该方法将源代码表示为 Object 对象的网络，再根据 Object 对象之间的共存关系对数据进行聚类，从而自动提取 API 使用模式。

亦有研究表明，深度学习在 API 推荐方面也有一定的效果[8-10]。Gu 等[8]首次将深度神经网络（Deep Neural Network，DNN）引入 API 推荐。该方法通过将每个方法体中的 API 序列和注释信息输入到 RNN Encoder-Decoder 模型中进行模型训练，再使用训练得到的模型进行 API 推荐。Li 等[10]提出了 Word2API，该方法基于词嵌入的方式来削减自然语言和 API 之间的语义鸿沟。将一个方法注释中出现的单词与方法体中出现的 API 合并得到一个元组，将得到的元组使用 Word2vec 进行训练，最终使用输出的词向量来进行 API 推荐。但深度学习存在一定的局限性，如过于依赖数据质量，模型可解释性较差[11-12]。Tung 等[13]提出了一种推荐任务，该任务通过旧的 Feature Requests 的自然语言描述信息，为新的 Feature Requests 推荐相关的 API 方法。Tung 等通过信息检索（Information Retrieval，IR）技术来解决这一任务。这是与本文推荐任务最相近的且最先进的研究，因此本文将该工作作为比较的基线。

然而，文献[4-6]均使用 API 作为查询，这并不符合开发人员的习惯。开发人员在面临不熟悉的编程任务时，难以确定输入什么样的 API 作为查询，即使开发人员对问题有一定的了解，但他们仍然会用自然语言，而不是用 API 作为查询输入。针对上述问题，本文提出了 SSAPIR，SSAPIR[1)] 以自然语言查询为输入，对大规模语料库进行语义相似度搜索，得到排名前 K 的 API 使用模式。本文从 GitHub 下载的高质量的 Java 项目中提取了 9 个流行的第三方 API 库的 API 使用模式，来验证本文方法的有效性。实验结果表明，SSAPIR[1)] 的 Hit@10 平均推荐准确率能达到 85.02%，高于已有研究方法。

2 准备工作

数据预处理是信息检索和文本挖掘的重要步骤[14]。数据预处理操作旨在去除数据中的噪声来提高文本挖掘算法的准确性，本文利用 Stanford CoreNLP[15] 这一广泛使用的自然语言处理（Natural Language Processing，NLP）工具来进行数据预处理。

（1）拆分驼峰式命名：驼峰式命名法在编程任务中被广泛使用。该步骤将方法签名中出现的驼峰式命名拆分成为原子单词，如将 MethodName 变成 Method Name。

（2）小写化：该步骤将出现在注释信息和方法签名中的大写字母转换成小写字母。

（3）删除停止词：该步骤删除了注释信息和方法签名中出现在 WordNet 英语停止词列表[16] 中的单词。该列表中包含英语中常见的停止词（如“a”“and”“the”），这些单词只会给 NLP 工作带来噪声。

（4）词干提取：该步骤将单词转换成其词根（如将 got 或 getting 转换成 get），减少了单词的数量并降低了自然语言处理工作的复杂性。

在数据预处理工作后，将所有 API 使用模式的描述信息（见第 3.2 节）和自然语言查询构建成向量空间模型，再根据向量空间模型中出现的单词构建词袋模型。此时，文本数据就转换成了计算机可以处理的结构化数据，通过计算两个向量之间的余弦相似度，来计算两个文档之间的语义相似度。

本文将使用一种被广泛使用的 TF-IDF（Term Frequency-Inverse Document Frequency）[17] 算法来计算每个向量空间模型的权重。文档频率 TF 的计算公式为：

$$TF_t=\frac{T_t}{\sum_n T_i} \tag{1}$$

其中，t 代表一个词袋模型，n 代表这个词袋模型中不同词汇出现的次数，T_t 代表词袋模型 t 在整个文档中出现的频率。

逆文档频率 IDF 的计算式为：

$$IDF_t=\log\frac{|D|}{1+|\{j:t_i\in d_j\}|} \tag{2}$$

其中，$|D|$ 代表整个文档中不同词汇出现的次数，$|\{j:t_i\in d_j\}|$ 代表整个文档中包含词袋模型 t 的数量。

文档 D 中的词袋模型 t 的权重计算公式为：

$$Weight_t=TF_t\times IDF_t \tag{3}$$

给定语料库 P 中的 API 使用模式的描述信息 D 的向量空间模型，即可得到自然语言查询中每个词袋模型 t 的权重。

余弦相似度常用于度量两个向量之间的相似程度。式(4)给出了 API 使用模式的描述信息和自然语言查询之间的余弦相似度：

1) https://github.com/Fan0429/SSAPIR

$$CosSim(P,Q)=\frac{\sum_{i=1}^{n}(P_i\times Q_i)}{\sqrt{\sum_{i=1}^{n}P_i^2}\sqrt{\sum_{i=1}^{n}Q_i^2}} \quad (4)$$

其中，P 代表 API 使用模式中的描述信息的向量空间模型，Q 代表处理后的用户输入的自然语言查询的向量空间模型，Q_i 和 P_i 分别代表二者向量空间模型中的词袋模型 t 的权重之和。

3 SSAPIR 的整体框架

图 1 给出了 SSAPIR 的整体架构。SSAPIR 的推荐场景是当开发人员进行 API 相关问题的查询时，向其推荐相应的 API 使用模式。SSAPIR 主要由 3 个部分组成：元数据结构提取、API 使用模式挖掘和基于语义相似度的 API 使用模式推荐。

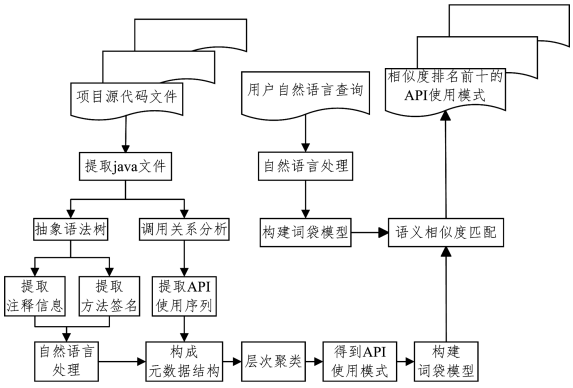


图 1 SSAPIR 的整体架构

Fig. 1 Overview framework of SSAPIR

3.1 元数据结构提取

为了实现基于语义相似度的 API 使用模式推荐，在挖掘 API 使用模式之前，需要知道开发人员经常使用的第三方 API 库，显然开发人员更倾向于使用常用 API 库中的方法。本文从 GitHub 上按照其 Star 数排序，收集了 585 个高质量的 Java 项目，使用 Maven 项目管理工具对其进行管理。在 Maven 项目中，POM(Project Object Model)文件记录了项目的依赖信息和配置信息。POM 文件中的 Artifact ID 和 dependency 标签记录每个项目的第三方 API 库的依赖关系。通过提取 POM 文件中的依赖关系，即可知道项目中使用的第三方 API 库。

为了收集第三方 API 库的使用频率，本文从上述使用 Maven 管理的高质量项目的 POM 文件中提取第三方 API 库的依赖信息，以得到第三方 API 库的使用频率。Apache Ant 项目[1]的 POM 文件中的一个真实存在的依赖信息如下：

```
〈dependency〉
〈groupId〉junit〈/groupId〉
〈artifactId〉junit〈/artifactId〉
〈version〉4.12〈/version〉
〈scope〉compile〈/scope〉
〈optional〉true〈/optional〉
〈/dependency〉
```

3.1.1 提取 API 调用序列

首先，根据提取到的上述项目中第三方 API 库的使用频率，来构建一个包含 81 个最常用第三方 API 库的本地语料库；然后，将本地第三方 API 语料库的路径配置到每个项目的.CLASSPATH 文件中，以确保后续方法可以提取这些第三方 API 库的 API 调用序列。SSAPIR 的 API 调用序列提取模块，是基于 Eclipse 的 Call Hierarchy[2]模块的插件项目。首先，使用 Call Hierarchy 模块来解析工作区中的所有项目，分析其是否为 Java 项目；然后，分析和提取各项目中的每个类的方法调用关系，本文忽略调用项目内方法的方法调用，因为此类方法调用不能被其他开发人员复用；最后，只保留 JDK 和第三方 API 库的 API 调用序列。

3.1.2 提取注释信息和方法签名

本文提取每个 Java 方法的 Javadoc 的第一个句子作为 API 调用序列的注释信息，而 Javadoc[3]的第一个句子是对该 Java 方法功能的总结。注释提取模块是基于 EclipseJDT 组件编写的 Java 项目，它将 Java 代码构建为抽象语法树，并从抽象语法树的 JavaDoc 节点获取相应 Java 方法的注释信息。在提取注释信息的同时，使用同样的方法提取该 Java 方法的方法签名。本文忽略没有注释信息的 Java 方法，并使用第 2 节提到的数据预处理方法来处理注释信息和方法签名信息。最后，本文利用获取的 API 调用序列、注释信息和方法签名，构成元数据结构〈API 调用序列，注释信息，方法签名〉。

图 2 给出了 Apache POI 项目中 CreateInformationProperties 方法中提取的元数据结构示例。

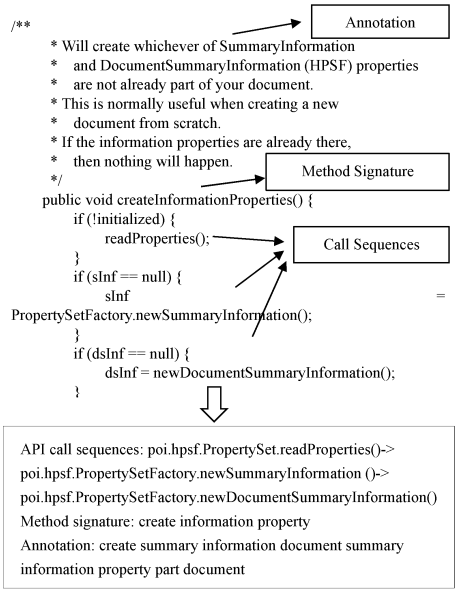


图 2 Apache POI 项目中提取的元数据结构示例

Fig. 2 Example of metadata from Apache POI project

最终，本文得到一个由 190393 个元数据结构组成的数据集[4]，所有数据均存储于 MySQL 数据库中。

[1] https://ant.apache.org/antlibs/srcdownload.cgi

[2] http://eclipse-tools.sourceforge.net/

[3] https://www.oracle.com/technetwork/articles/java/index-137868.html

[4] https://github.com/Fan0429/SSAPIR/tree/master/data

3.2 API 使用模式挖掘

为了从元数据结构中提取 API 使用模式，本文使用广泛用于提取 API 使用模式的层次聚类算法[4-5,7]。聚类算法[18]是一种无监督的机器学习算法，它通过计算数据之间的相似度，将未标记的数据分类成有意义的簇，簇之间的相似度度量法则决定了聚类结果的质量。因此，在对元数据结构进行层次聚类前，需要定义相似度度量法则，具体定义如下。

(1)方法名。图 3 给出了 Apache POI 项目中的一个方法，方法名是 CreateInformationProperties，使用第 2 节中的数据预处理方法对其进行处理，处理后的结果为 Create Information Property。通过观察可以发现，经过处理的方法名与经过处理的注释信息表达相同的含义，方法名在一定程度上也存储了 Java 方法的功能信息。本文将两个元数据结构中的方法名中出现的单词的平均相似度作为最终的相似度。其中，两个单词之间的相似度的计算法则公式为：

$$s_{w_1,w_2}(i,j)=\begin{cases}\max(i,j), & \min(i,j)=0\\ \min\begin{cases}s_{w_1,w_2}(i-1,j)+1\\ s_{w_1,w_2}(i,j-1)+1\\ s_{w_1,w_2}(i-1,j-1)+I_{w_{1i}\neq w_{2j}}\end{cases}, & \min(i,j)\neq 0\end{cases} \tag{5}$$

其中，$s_{w_1,w_2}(i,j)$是两个单词的相似度得分，表示 w_1 的 i 字母到 w_2 的 j 字母的距离；$I_{w_{1i}\neq w_{2j}}$ 是当 $w_{1i}=w_{2j}$ 时为 0，否则为 1 的指示函数。

方法名的相似度计算式为：

$$Score_{name}=\frac{\sum_{1}^{n}S_w}{n} \tag{6}$$

其中，n 代表方法名中出现的单词的个数，S_w 代表这两个单词的相似度得分。

(2)被调用的 API。除了方法名以外，本文使用元数据结构中的被调用 API 作为相似度度量法则的第二部分。对于两个元数据结构中的 API 调用序列，它们之间的相似度计算式为：

$$Score_{api}=\frac{A_1\cap A_2}{A_1\cup A_2} \tag{7}$$

其中，A_1 和 A_2 代表各个元数据结构中 API 的集合，$A_1\cup A_2$ 表示两个元数据结构中 API 的并集，$A_1\cap A_2$ 表示两个元数据结构中 API 的交集。

在得到每个元数据结构的两个相似度得分后，首先将它们归一化到 0～1 的范围内，然后计算两个相似度得分之间的算术平均值作为最终得分。

$$Score_{\text{final}}=\alpha_1\overline{Score_{\text{name}}}+\alpha_2\overline{Score_{\text{api}}} \tag{8}$$

其中，$Score_{\text{name}}$ 是两个元数据结构的方法名相似度得分，$Score_{\text{api}}$是两个元数据结构的 API 调用序列相似度得分，α_1 和 α_2 为两个得分的权重。$\overline{Score_{\text{name}}}$代表得分 $Score_{\text{name}}$通过归一化计算后的值。此处，将 α_1 设为 0.4，α_2 设为 0.6(见 4.4 节实验 2)。

当给定两个元数据结构时，本文首先计算两者基于方法名的相似性得分，再计算两者 API 调用序列的相似度得分，通过计算两个相似度得分归一化后的算术平均值，得到相似度度量法则的最终得分。基于最终得分，本文使用层次聚类算法[19]对元数据结构进行层次聚类以获得一系列的簇，每个簇即为 API 使用模式，是多个相似的元数据结构的集合，其中 API 使用模式的描述信息是指对应元数据结构中的注释信息的集合。

3.3 基于语义相似度的 API 使用模式推荐

本节介绍了 API 使用模式的描述信息和开发人员的自然语言查询之间的语义相似度的计算方法。

为了提取语义信息，该方法首先利用第 2 节的处理方法对获取的开发人员输入的自然语言查询信息进行数据预处理，并将其转换成向量空间模型。同时，以相同的方式对 API 使用模式的描述信息进行处理。数据预处理后，给定一个与 API 相关的查询 Q，其与 API 使用模式 P 的描述信息之间的语义相似度为：

$$SimScore(P_d,Q)=CosSim(VSM(P_d),VSM(Q)) \tag{9}$$

其中，Q 代表提取后的用户输入的自然语言查询，P_d 代表 API 使用模式中的描述信息，$VSM(P_d)$代表 P_d 的向量空间模型。

4 实验及结果分析

为了验证本文方法的有效性，我们设计了 3 个实验。实验 1 验证了层次聚类在 API 使用模式推荐方法中的作用，实验 2 验证了不同的相似度度量法则对层次聚类效果的影响，实验 3 通过横向对比最先进的方法来验证该方法的有效性。

实验环境为 Intel i7-8700K 3.7 GHz 处理器、32.0 GB RAM 的台式机电脑，搭载 Windows 10 64-bit 操作系统，使用 JKD1.8.0 版本，并使用 Eclipse Oxygen.3 作为代码实现的开发 IDE。数据库的版本为 MySQLV5.7。

4.1 实验准备

本节主要介绍了数据集的构建方式。首先介绍 Java 项目的选择方法，然后介绍第三方 API 库的选择方法。为了验证所提方法的有效性，本文首先创建了一个 Java 项目组成的大规模语料库，这些项目均是 GitHub 上 Star 数排名靠前的 Java 项目；然后，通过判断项目中是否存在 POM.xml 文件对项目进行过滤，得到用 Maven 进行项目管理的 Java 项目；最终，得到 585 个高质量的 Java 项目，并从这些项目中提取 API 使用模式。

表 1 列出了语料库的详细信息，表 2 列出了元数据结构库中的 API 调用序列和注释长度的详细信息。其中，每一条注释信息包含 8 个词，且大多数注释信息的长度小于 10；此外，大多数 API 调用序列的长度小于 5。

表 1 数据集中项目的统计信息

Table 1 Statistics for projects message in dataset

项目数量	文件数量	行数	元数据结构数
585	91259	8571730	190393

表 2 API 调用序列的统计信息

Table 2 Statistics for API sequences

平均数	众数	中位数	<5	<10	<20
2.47	1	2	85.76%	91.21%	96.87%

表 3 注释信息的长度

Table 3 Comment length

平均数	众数	中位数	<15	<25	<35
8.28	8	11	75.50%	88.79%	97.43%

为了验证实验的有效性，我们通过计算语料库中第三方 API 库的使用频率，对所选用的第三方 API 库进行筛选。据统计，语料库的所有项目中共使用了 3654 个第三方 API 库，其中 2727 个库的使用次数少于 5 次。本文最终选择了 9 个第三方 API 库作为提取 API 使用模式的库，与 Thung 等[13]的工作基本相同，这同时也是本文对比的最先进的相关工作，并将其作为基线工作。其中，本文将基线工作中的 slf4j-api 和 slf4j-log4j12 这两个第三方 API 库的调用序列合并提取。这 9 个第三方 API 库中提取到的 API 使用模式的信息如表 4 所列。

表 4 9 个第三方 API 库中 API 使用模式的数量

Table 4 Number of API usage patterns in datasets for nine third-party API Libraries

第三方 API 库名称	API 使用模式数量
commons-codec	309
commons-io	202
commons-lang	304
commons-logging	394
easymock	76
junit	736
log4j	152
servlet-api	985
slf4j	795
总计	3953

基线工作从 JIRA 中的 Feature Requests 中提取描述、摘要信息、作者信息等自然语言信息，通过这些信息来计算新的 Feature Requests 和已有 Feature Requests 的历史相似性得分；然后，通过新的 Feature Requests 和 API 文档之间的相似性，来计算文档相似性得分；最后，根据一定的权重，使用上述两个相似性得分来计算最终相似性得分，并以此推荐相关的 API。

4.2 实验设置

理想的推荐结果应该获得更多的相关 API 使用模式，并将它们放在结果的顶部。为了评估本文方法，将 Hit@K 作为评估度量指标[20]，Hit@K 准确率被广泛应用于评价推荐系统[21]。本文实验通过计算 Hit@K(K 设定为 5 和 10)来推荐 Top5 和 Top10 的 API 使用模式。具体地，Hit@K 定义为 Top-K 推荐结果[22]中的真阳性的结果，即若推荐的结果包含在 API 使用模式中，则判定为准确。

实验中，为了判断推荐结果是否准确，我们使用上述提取的 9 个第三方 API 库的 API 使用模式构建标准数据集。在此基础上，针对每个第三方 API 库进行分别测试，计算平均准确率。具体地，针对每个第三方库的 API 使用模式，依次选取 API 使用模式的描述信息作为输入，将 API 使用模式作为标准推荐结果。将输入与其余数据分别进行语义相似度计算，最终得到 Top-K 推荐结果。

4.3 实验 1

在 API 使用模式挖掘部分，本文采用层次聚类算法对元数据结构进行聚类。聚类的目的是通过上述相似度度量法则将相似的元数据结构合并成一个簇，以减少推荐的冗余结果。实验 1 将通过比较进行层次聚类前后的结果，来探讨层次聚类对本文方法性能的影响。

实验将上述 9 个第三方 API 库的元数据结构和通过层次聚类得到的 API 使用模式作为实验数据。层次聚类的结果如表 4 所列。通过比较 API 使用模式与元数据结构分别的推荐结果，来观察层次聚类是否有效。

图 3 给出了实验 1 的实验结果，表 5 列出了图 3 结果的详细说明。由表 5 可以看出，层次聚类后的数据展现了比未进行层次聚类的数据更好的结果。SSAPIR 在 9 个第三方 API 库下的 Hit@10 平均准确达到了 85.02%，而未进行层次聚类的数据 Hit@10 平均准确率仅为 70.56%。

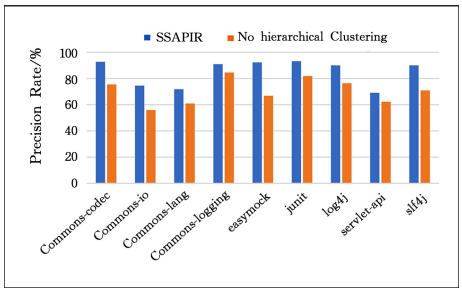


图 3 SSAPIR 和未聚类方法在 9 个第三方 API 库下的 Hit@10 推荐结果

Fig. 3 Hit@10 results for SSAPIR and no clustering approach for nine third-party API libraries

表 5 SSAPIR 和未聚类方法在 9 个第三方 API 库下的 Hit@10 准确率

Table 5 Hit@10 precision rate details for SSAPIR and no clustering approach for nine third-party API libraries

第三方 API 库	SSAPIR	未聚类方法	提升值
commons-codec	0.9265	0.7527	0.2309
commons-io	0.7477	0.5596	0.3361
commons-lang	0.7190	0.6082	0.1822
commons-logging	0.9104	0.8475	0.0742
easymock	0.9241	0.6703	0.3786
junit	0.9304	0.8194	0.1354
log4j	0.8994	0.7628	0.1790
servlet-api	0.6926	0.6211	0.1152
slf4j	0.9019	0.7090	0.2721
AVG	0.8502	0.7056	0.2049

使用层次聚类得到更好结果的原因可能是，本实验使用的数据源是开源的高质量项目。其中一些方法调用具有相同的名称，只改变部分参数(即方法重载)，导致 API 调用序列产生冗余结果。如果不进行聚类，这些冗余的 API 调用序列在推荐过程中将变成噪声，因此层次聚类是必要的。

通过层次聚类减小了 API 调用序列的冗余程度，获得了最佳的推荐结果。

4.4 实验 2

实验 1 的结果验证了层次聚类的有效性，而相似性度量法则是一种聚类算法好坏的决定性因素之一。在实验 2 中，将分别用方法名相似度得分和 API 调用序列相似度得分作为相似度度量法则进行层次聚类，来验证 SSAPIR 是否能够

获得比单独使用其中一种相似度得分更好的结果。本实验采用与实验 1 相同的实验数据。

实验 2 将相似度度量法则的两个部分分别作为单独的相似度度量法则来进行层次聚类，并计算最终的推荐准确率，以验证 SSAPIR 的有效性。

图 4 给出了实验 2 的最终结果，表 6 列出了图 4 所示结果的详细说明。由图 4 可知，通过 SSAPIR 的相似度度量法则，可以得到最佳的推荐结果。在大部分情况下，只使用方法名作为相似度度量法则比只使用 API 调用序列作为相似度度量法则的准确率低。

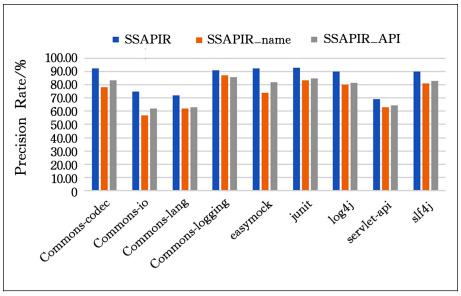


图 4 SSAPIR 和其两种变体在 9 个第三方 API 库下的 Hit@10 推荐结果

Fig. 4 Hit@10 results for SSAPIR and its variants for nine third-party API libraries

表 6 SSAPIR 和其两种变体在 9 个第三方 API 库下的 Hit@10 准确率

Table 6 Hit@10 precision rate details for SSAPIR and its variants for nine third-party API libraries

第三方 API 库	SSAPIR	SSAPIR_name	SSAPIR_API
commons-codec	0.9265	0.7815	0.8324
commons-io	0.7477	0.5703	0.6196
commons-lang	0.7190	0.6209	0.6327
commons-logging	0.9104	0.8710	0.8574
easymock	0.9241	0.7415	0.8190
junit	0.9304	0.8318	0.8504
log4j	0.8994	0.7995	0.8139
servlet-api	0.692	0.6319	0.6437
slf4j	0.9019	0.8096	0.8275
AVG	0.8502	0.7398	0.7663

SSAPIR 在 9 个第三方 API 库下的 Hit@10 平均准确率达到了 85.02%，只使用方法名作为相似度度量法则的 Hit@10 平均准确率仅为 73.98%，而只使用 API 序列作为相似度度量法则的 Hit@10 准确率为 76.63%。对比实验 1 的实验结果可知，使用层次聚类后的 3 种方法的结果均比未使用层次聚类的结果高，再次说明了层次聚类的有效性。

在计算簇之间的相似性时，API 调用序列比方法名更重要，且 SSAPIR 比单独方法名相似度得分或 API 调用序列相似度得分作为相似度度量法则，有更高的推荐准确率。

4.5 实验 3

基线研究[13]是使用自然语言描述信息推荐 API 的最新研究。在实验 3 中，在基线研究中使用的 9 个第三方 API 库的数据上，对比 SSAPIR 和基线研究的推荐准确率。

本文通过利用 Thung 等提到的方法进行对比实验。两次实验使用本文所提取的数据集并在相同的环境中运行。需要说明的是，在实验 3 与基线工作的对比实验中，由于基线工作使用了 Feature Requests 的作者信息以及 API 文档信息等，而本文所提取的数据并不包含该类信息，因此将其设置为缺省值。表 7 列出了 SSAPIR 和基线研究在推荐 API 相关问题时的详细结果。

表 7 SSAPIR 和基线方法在 9 个第三方 API 库下的 Hit@5、Hit@10 准确率

Table 7 Precision rate details for SSAPIR and baseline approach in different Hit@K for nine third-party API libraries

第三方 API 库	方法	Hit@5	Hit@10	Hit@10 增益/%
commons-codec	SSAPIR	0.8758	0.9265	57.26
	Baseline	0.5287	0.5891	
commons-io	SSAPIR	0.5570	0.7477	57.48
	Baseline	0.3360	0.4748	
commons-lang	SSAPIR	0.5865	0.7190	47.65
	Baseline	0.2375	0.4870	
commons-logging	SSAPIR	0.8487	0.9104	13.28
	Baseline	0.6148	0.8037	
easymock	SSAPIR	0.8642	0.9241	112.93
	Baseline	0.3832	0.4340	
junit	SSAPIR	0.8840	0.9304	53.96
	Baseline	0.5291	0.6043	
log4j	SSAPIR	0.8072	0.8994	83.96
	Baseline	0.1099	0.4889	
servlet-api	SSAPIR	0.6006	0.6926	98.35
	Baseline	0.2147	0.3492	
slf4j	SSAPIR	0.7965	0.9019	45.54
	Baseline	0.5420	0.6197	
AVG	SSAPIR	0.7578	0.8502	57.75
	Baseline	0.3884	0.5163	

由表 7 可以看出，SSAPIR 在 Hit@5 和 Hit@10 上的平均准确率分别为 75.78%和 85.02%。而基线研究在 Hit@5 和 Hit@10 上的平均准确率仅为 38.84%和 51.63%。SSAPIR 在 Hit@10 的平均增益为 57.75%，较基线工作有了较高的提升。

由实验结果可以得出，SSAPIR 在推荐结果上获得了较高的准确率。与 Thung 等提出的方法相比，SSAPIR 在 Hit@5 和 Hit@10 上的平均推荐准确率都获得了提高。因此，SSAPIR 在开发人员输入自然语言查询的推荐场景中，能够更准确地推荐 API 使用模式。

结束语 本文提出了一种基于语义相似度的 API 使用模式推荐方法 SSAPIR。通过对比开发人员输入和自然语言查询和语料库中已有的 API 使用模式的描述信息之间的语义相似度，对 API 使用模式进行推荐。实验结果表明，SSAPIR 的 Top10 平均准确率能够达到 85.02%，优于现有方法。未来将对以下两个方面进行扩展：一方面，拓展 SSAPIR 适用的编程语言，以提高推荐范围；另一方面，将考虑在 SSAPIR 中自动地进行 API 参数推荐，以完善推荐场景。

参 考 文 献

[1] PiCCIONI M, FURIA C A, MEYER B. An empirical study of API usability[C]// 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 2013: 5-14.

[2] ZHOU Y, WANG C, YAN X, et al. Automatic Detection and

Repair Recommendation of Directive Defects in Java API Documentation[J]. IEEE Transactions on Software Engineering, 2018.

[3] ZHANG J X, JIANG H, REN Z L, et al. Recommending APIs for API Related Questions in Stack Overflow[J]. IEEE Access, 2018, 6: 6205-6219.

[4] ZHONG H, XIE T, ZHANG L, et al. MAPO: Mining and recommending API usage patterns[C]// Proceedings of the 23 rdEuropean Conference on ECOOP 2009—Object-Oriented Programming. Berlin: Springer, 2009: 318-343.

[5] BUSE R P L, WEIMER W. Synthesizing API usage examples [C]// Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012: 782-792.

[6] WANG J, DANG Y N, ZHANG H Y, et al. Mining succinct and high-coverage API usage patterns from source code[C]// Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, 2013: 319-328.

[7] NIU H, KEIVANLOO I, ZOU Y. API usage pattern recommendation for software development[J]. Journal of Systems and Software, 2017, 129(C): 127-139.

[8] GU X D, ZHANG H Y, ZHANG D M, et al. Deep API learning [C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2016: 631-642.

[9] HUANG Q, XIA X, XING Z, et al. API method recommendation without worrying about the task-API knowledge gap[C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, 2018: 293-304.

[10] LI X C, JIANG H, KAMEI Y, et al. Bridging Semantic Gaps between Natural Languages and APIs with Word Embedding[J]. arXiv: 1810. 09723, 2018.

[11] HELLENDOORN V J, DEVANBU P. Are deep neural networks the best choice for modeling source code? [C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017: 763-773.

[12] LU Y, HSIAO I H. Exploring Programming Semantic Analytics with Deep Learning Models[C]// Proceedings of the 9th International Conference on Learning Analytics & Knowledge. ACM, 2019: 155-159.

[13] THUNG F, WANG S, LO D, et al. Automatic recommendation of API methods from feature requests[C]// Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, 2013: 290-300.

[14] MANNING C, RAGHAVAN P, SCHÜTZE H. Introduction to information retrieval[J]. Natural Language Engineering, 2010, 16(1): 100-103.

[15] MANNING C, SURDEANU M, BAUER J, et al. The Stanford CoreNLP natural language processing toolkit[C]// Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Baltimore: ACL, 2014: 55-60.

[16] WordNet English Stopword List [EB/OL]. http://www.d.umn.edu/~tpederse/Group01/WordNet/wordnet-stoplist.html.

[17] RAMOS J. Using tf-idf to determine word relevance in document queries[C]// Proceedings of the First Instructional Conference on Machine Learning. 2003: 133-142.

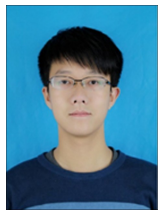
[18] JAIN A K, MURTY M N, FLYNN P J. Data clustering: a review[J]. ACM computing surveys (CSUR), 1999, 31(3): 264-323.

[19] PUDI V. Data mining: concepts and techniques[M]. New York: Oxford University Press, 2011.

[20] XU C Y, SUN X B, LI B, et al. MULAPI: Improving API method recommendation with API usage location[J]. Journal of Systems and Software, 2018, 142: 195-205.

[21] AVAZPOUR I, PITAKRAT T, GRUNSKE L, et al. Dimensions and metrics for evaluating recommendation systems[M]// Recommendation Systems in Software Engineering. Berlin: Springer, 2014: 245-273.

[22] MCMILLAN C, POSHYVANYK D, GRECHANIK M, et al. Portfolio: Searching for relevant functions and their usages in millions of lines of code[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2013, 22(4): 37.



ZHANG Yun-fan, postgraduate. His research interests include software evolution analysis, artificial intelligence, and mining software repositories.

ZHOU Yu, postdoctor, professor. His research interests mainly include software evolution analysis, mining software repositories, software architecture, and reliability analysis.