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

(南京航空航天大学计算机科学与技术学院 南京 210016)1

(高安全系统的软件开发与验证技术工信部重点实验室(南京航空航天大学), 南京 211100)2

**摘 要** 在软件开发过程中，复用应用程序编程接口（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。本文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(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，本文从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 **文献标识码** A

**Semantic Similarity based API Usage Pattern** **Recommendation**

ZHANG Yun-fan1 ZHOU Yu1，2\* HUANG Zhi-qiu1，2

(College of Computer Science and Technology，Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China) 1

(Ministry Key Laboratory for Safety-Critical Software Development and Verification (Nanjing University of Aeronautics and Astronautics), Nanjing 211100, China) 2

A**bstract** 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 proposes a novel Semantic Similarity based API Usage Pattern Recommendation approach (SSAPIR). Our approach first adopts hierarchical clustering algorithm to extract API usage patterns, and then calculate 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, we collect Java projects from GitHub, from which we extract the API usage patterns related to the 9 popular third-party API libraries and their description information. Ultimately, we recommend API usage patterns based on natural language queries which are related to the 9 third-party API libraries. To verify the effectiveness of SSAPIR, we measure 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 research 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**

# 研究背景及意义

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和融合到人们生活的各个组成部分，新的软件形态和开发模式不断涌现，其规模和数量正在以惊人的速度膨胀和扩大。有效的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，开发人员常常要实现不熟悉的编程任务，他们要么通过查询和搜索代码示例进行代码复用，要么学习不熟悉的应用程序编程接口（Application Programming Interface，API）的使用方法[1][2]。此外，已有的程序可以帮助开发人员理解其他人如何解决类似的问题，并且开发人员可以在此基础上完成他们需要的功能。数以百万计的软件项目托管在开源社区（如GitHub）中，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他们遇到的问题（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他们需要的API。

为了解决这个问题，一些研究提出了多种方法来推荐API使用模式[4][5][6][7]。API使用模式是指，能够实现一个功能所需要的一组应用程序编程接口（API）调用序列。

以往的研究表明，频繁序列挖掘算法是提取API使用模式的有效途径。Xie等人[4]首先提出了一种经典的API使用模式挖掘算法MAPO。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等人[7]提出一种方法，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。

研究表明，深度学习在API推荐方面也有一定的效果[8][9][10]。Gu等人[8]首先将深度神经网络引入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，最后使用训练得到的模型进行API推荐。Li等人[10]提出了Word2API，该方法使用词嵌入的方式来削减自然语言和API之间的语义鸿沟。通过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等人[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等人通过信息检索（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相关研究工作，所以本文使用该工作作为比较的基线工作。

然而，这些研究的查询示例中[4][6]均使用API作为查询，这并不符合开发人员的开发习惯。当开发人员面临不熟悉的编程任务时，他们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，他们仍然会用自然语言，而不是使用API作为查询输入。针对上述问题，本文提出了SSAPIR(Semantic Similarity based API Recommendation)，SSAPIR[[1]](#footnote-1)以自然语言查询作为输入，对大规模语料库进行语义相似度搜索，得到排名前K个的API使用模式。本文从GitHub下载的高质量的Java项目中，提取出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的Hit@10平均推荐准确率能达到85.02%，高于已有研究方法。

# 准备工作

数据预处理是信息检索和文本挖掘中的重要步骤[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，本文利用了Stanford CoreNLP[15]这一广泛使用的自然语言处理（NLP）工具来进行数据预处理工作。步骤如下：

**(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（例如，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（例如，“a”、“and”、“the”），这些单词只会给自然语言处理工作带来噪声。

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

* 1. 构建向量空间模型

在数据预处理工作之后，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。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。本研究中将使用一个被广泛使用的算法TF-IDF (Term Frequency–Inverse Document Frequency)[17]来计算每个向量空间模型的权重。文档频率TF的计算公式如下：

|  |  |
| --- | --- |
|  | (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数量，代表词袋模型*t*在整个文档中出现的频率。

逆文档频率IDF的计算公式如下：

|  |  |
| --- | --- |
|  | (2) |

其中，*|D|*代表整个文档中不同词汇出现的数量，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*的权重的计算公式如下：

|  |  |
| --- | --- |
|  | (3) |

根据以上内容，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。下面的公式展示了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计算公式：

|  |  |
| --- | --- |
|  | (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。和分别代表二者向量空间模型中的词袋模型*t*的权重之和。

# 整体框架

图1给出了SSAPIR的整体架构图。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主要由三个部分组成：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 The overview framework of SSAPIR |

# 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，首先需要知道哪些第三方API库是开发人员经常使用的。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。这些项目使用Maven项目管理工具进行项目管理。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，即可知道在这个项目中使用哪些第三方API库。

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>compile</scope>

<optional>true</optional>

</dependency>

|  |
| --- |
| 图2 阿帕奇Ant项目的POM文件中的依赖信息  Fig.2 A dependency from Apache ant project POM file |

图2展示了Apache Ant项目[[2]](#footnote-2)的POM文件中的一个真实存在的依赖信息。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。

# 提取API调用序列

根据提取上述项目中第三方API库的使用频率，最终构建了一个包含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[[3]](#footnote-3)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，分析该项目是否为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。其中，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。最终，只保留JDK和第三方API库的API调用序列。

# 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。由官方定义可知，Javadoc[[4]](#footnote-4)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Eclipse JDT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2.1节中提到的数据预处理方法来处理注释信息和方法签名信息。最终，本文将获取到的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>。

|  |
| --- |
| /\*\*  \* Will create whichever of SummaryInformation  \* and DocumentSummaryInformation (HPSF) properties  \* are not already part of your document.  \* This is normally useful when creating a new  \* document from scratch.  \* If the information properties are already there,  \* then nothing will happen.  \*/  public void createInformationProperties() {  if (!initialized) {  readProperties();  }  if (sInf == null) {  sInf = PropertySetFactory.newSummaryInformation();  }  if (dsInf == null) {  dsInf = newDocumentSummaryInformation();  }  }  Call Sequences  Annotation  Method Signature  API call sequences: poi.hpsf.PropertySet.readProperties()->  poi.hpsf.PropertySetFactory.newSummaryInformation ()->  poi.hpsf.PropertySetFactory.newDocumentSummaryInformation()  Method signature: create information property  Annotation: create summary information document summary information property part document  图3 Apache POI项目中提取的元数据结构示例 |
| Fig.3 An example of Metadata from Apache POI project |

如图3所示，为Apache POI项目中CreateInformationProperties方法中提取的元数据结构示例。最终，本文得到一个由190393个元数据结构组成的数据集[[5]](#footnote-5)，所有的数据均都存储在MySQL数据库中。

* 1. 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之前，需要定义相似度度量法则。本文的相似度度量法则的定义如下：

**方法名**：图3展示了Apache POI项目中的一个方法，它的方法名是createInformationProperties，将该方法名使用第2节中提到的数据预处理方法进行处理，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了几乎相同的含义，可以看出方法名在一定程度上也存储了一个Java方法的功能信息。本文通过计算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如下：

|  |  |
| --- | --- |
|  |  |
| (5) | |

其中是两个单词的相似度得分，表示。是当时为0，否则为1的指示函数。

方法名的相似度计算公式如下：

|  |  |
| --- | --- |
|  | (6) |

其中，*n*代表方法名中出现的单词的个数，代表这两个单词的相似度得分。

**被调用的API**：除了方法名之外，本文使用元数据结构中被调用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，他们之间的相似度计算公式如下：

|  |  |
| --- | --- |
|  | (7) |

其中，和代表各个元数据结构中API的集合，表示两个元数据结构中API的并集，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到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，最终分数计算得分如下：

|  |  |
| --- | --- |
|  | (8) |

其中，是两个元数据结构的方法名相似度得分，是两个元数据结构的API调用序列相似度得分，和为两个得分的权重。代表得分通过归一化计算后的值。此处，我们经验性地将设为0.4，设为0.6（见4.4节实验二）。

当给定两个元数据结构，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数平均值，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获得一系列的簇，每个簇即为API使用模式，每个簇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。

* 1. 基于语义相似度的API使用模式推荐

在本节中将描述如何计算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。

为了提取语义信息，本方法首先将获取到的开发人员输入的自然语言查询信息，利用2.1节中描述的处理方法进行数据预处理，并转换成向量空间模型。同样，API使用模式的描述信息也以相同的方式进行处理。

数据预处理后，给定一个与API相关的查询*Q*和API使用模式*P*的描述信息之间的语义相似度的计算公式如下：

|  |  |
| --- | --- |
|  | (9) |

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

# 实验分析

为了验证本文提出方法的有效性，我们设计了三个实验。实验一验证了层次聚类在API使用模式推荐方法中的作用。实验二验证了不同的相似度度量法则对层次聚类效果的影响。实验三通过横向对比最先进的方法来验证本工作的有效性。

本文的实验平台为Intel i7-8700K 3.7GHz处理器、32.0GB 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。

# 实验准备

本节主要介绍数据集的构建方式。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。为了验证本文提出方法的有效性，我们首先创建了一个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 数据集中项目的统计信息 | | | |
| Table 1 Statistics for projects message in our dataset | | | |
| 项目数量 | 文件数量 | 行数 | 元数据结构数 |
| 585 | 91259 | 8571730 | 190393 |

表1展示了语料库的详细信息，表2展示了元数据结构库中的API调用序列和注释长度的详细信息。平均而言，一条注释信息包含8个词，并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2 API调用序列、注释信息的统计信息 | | | | | |
| Table 2 Statistics for API sequences and comments length | | | | | |
| API 调用序列长度 | | | | | |
| 平均数 | 众数 | 中位数 | <5 | <10 | <20 |
| 2.47 | 1 | 2 | 85.76% | 91.21% | 96.87% |
| 注释信息长度 | | | | | |
| 平均数 | 众数 | 中位数 | <15 | <25 | <35 |
| 8.28 | 8 | 11 | 75.50% | 88.79% | 97.43% |

|  |  |
| --- | --- |
| 表3 9个第三方API库中API使用模式数量 | |
| Table 3 Number of API Usage Patterns in our 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 |

为了验证本文实验的有效性，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9个第三方API库作为提取API使用模式的库，与Thung等人[13]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3所示。

该基线工作从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。通过上述自然语言信息来计算新的Feature Requests和已有Feature Requests的历史相似性得分。再通过新的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计算最终相似性得分，并以此推荐相关的API。

# 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。为了评估本文提出的方法，将使用Hit@K作为评估度量指标[20]。Hit@K准确率被广泛用于评价推荐系统[21]。在本文实验中，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。具体地说，Hit@K定义为Top-K推荐结果[22]中的真阳性的结果。即推荐的结果包含在API使用模式中，即判定为准确。

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。具体而言，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，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。

# 实验一

在API使用模式挖掘的部分中，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,以此减少推荐的冗余结果。实验一将通过比较未进行层次聚类的结果和层次聚类后的结果，来探讨层次聚类对本方法性能的影响。

|  |
| --- |
|  |
| 图4 SSAPIR和未聚类方法在9个第三方API库下的Hit@10推荐结果  Fig.4 Hit@10 Result for SSAPIR and no clustering approach for nine third-party API libraries |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表4 SSAPIR和未聚类方法在9个第三方API库下的Hit@10准确率结果 | | | | | |
| Table 4 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 |
| 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 | |

**方法**：本实验使用上述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3所示。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。

**结果**：图4展示了实验一的实验结果。表4为图4结果的详细说明。通过表4可以看到，层次聚类后的数据展现出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%。

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。

**结论**：通过层次聚类减少API调用序列的冗余程度，使推荐获得最佳的结果。

# 实验二

通过实验一的结果，验证了层次聚类的有效性。而相似性度量法则是一个聚类算法好坏的决定性因素之一。在实验二中，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得到更好的结果。本实验使用与实验一相同的数据作为实验数据。

|  |
| --- |
|  |
| 图5 SSAPIR和其两种变体在9个第三方API库下的Hit@10推荐结果  Fig.5 Hit@10 Result for SSAPIR and its variants for nine third-party API libraries |

**方法**：实验二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。以此验证SSAPIR的有效性。

**结果**：图5展示了实验二的最终结果，表5展示了图5结果的详细说明。由实验结果可知，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来计算聚类相似度的最终结果比只使用API调用序列作为相似度度量法则的结果准确率低。

SSAPIR在9个第三方API库下的Hit@10平均准确达到了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%。对比实验一的实验结果可知，使用层次聚类后的三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。

**结论**：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，有更高推荐准确率。

|  |  |  |  |
| --- | --- | --- | --- |
| 表5 SSAPIR和其两种变体在9个第三方API库下的Hit@10准确率结果 | | | |
| Table 5 Hit@10 Precision Rate details for SSAPIR and its variants for nine third-party API libraries | | | |
| 第三方API库名称 | 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 | 0.6926 | 0.6319 | 0.6437 |
| slf4j | 0.9019 | 0.8096 | 0.8275 |
| AVG | 0.8502 | 0.7398 | 0.7663 |

# 实验三

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表6 SSAPIR和基线方法在9个第三方API库下的Hit@5、Hit@10准确率结果 | | | | | |
| Table 6 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 |

正如4.1节描述的，基线研究[13]是使用自然语言描述信息推荐API的最新研究。在实验三中，通过基线研究中使用的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。

**方法**：本文通过复现Thung等人论文中提到的方法，进行对比实验。两次实验使用本文所提取到的数据集并在相同的环境中运行。需要说明的是，在实验三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，而本文的所提取的数据并不包含该类信息，故将其设置为缺省值。表6展示了SSAPIR和基线研究在推荐API相关问题的详细结果。

**结果**：通过表6展示的实验结果可以看出，SSAPIR Hit@5和Hit@10的平均准确率分别为75.78%和85.02%。而基线研究的Hit@5和Hit@10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。

**结论**：基于实验三的实验结果，可以得出SSAPIR在推荐结果上获得了较高的准确率。通过与Thung等人提出的方法相比，SSAPIR在Hit@5和Hit@10上的平均推荐准确率都获得了提高。可以得出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。实验结果标明，SSAPIR的Top10平均准确率能够达到85.02%，优于现有方法。对于未来的工作，将在以下两个方面进行扩展。首先，拓展SSAPIR适用的编程语言，以提高推荐范围。其次将考虑在SSAPIR中自动的进行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. Jingxuan Zhang; He Jiang; Zhilei Ren; and Xin Chen, Recommending APIs for API Related Questions in Stack Overflow, IEEE Access, 2018, Vol. 6, pp. 6205-6219.
4. Zhong H, Xie T, Zhang L, et al. MAPO: Mining and recommending API usage patterns[C]//European Conference on Object-Oriented Programming. Springer, Berlin, Heidelberg, 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, Zhang H, 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: 127-139.
8. Gu X, Zhang H, Zhang D, et al. Deep API learning[C]//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 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, Jiang H, Kamei Y, et al. Bridging Semantic Gaps between Natural Languages and APIs with Word Embedding[J]. IEEE Transactions on Software Engineering, 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. 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. 2014: 55-60.
16. “WordNet English Stopword List,” 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, 242: 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. Han J, Pei J, Kamber M. Data mining: concepts and techniques[M]. Elsevier, 2011.
20. Xu C, Sun X, 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. Springer, Berlin, Heidelberg, 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.

1. https://github.com/Fan0429/SSAPIR [↑](#footnote-ref-1)
2. https://ant.apache.org/antlibs/srcdownload.cgi [↑](#footnote-ref-2)
3. http://eclipse-tools.sourceforge.net/ [↑](#footnote-ref-3)
4. https://www.oracle.com/technetwork/articles/java/index-137868.html [↑](#footnote-ref-4)
5. https://github.com/Fan0429/SSAPIR/tree/master/data [↑](#footnote-ref-5)