上海交通大学硕士学位论文

软件工程领域语义相关词的挖掘与应用

硕 士 研 究 生：

学　　号：

导　　师：

申 请 学 位：　工程硕士

学　　科：　软件工程

所 在 单 位：　电子信息与电气工程学院

答 辩 日 期：　2017 年 1 月

授予学位单位：　上海交通大学

Dissertation Submitted to Shanghai Jiao Tong University
for the Degree of Master

RESEARCH ON DIGGING AND APPLICATION OF SEMANTICALLY RELATED WORDS IN SOFTWARE

Candidate：

Student ID:

Supervisor：

Academic Degree Applied for：	Master of Engineering
Speciality：	Software Engineering
Affiliation：	School of Electronic, Information and Electrical Engineering
Date of Defence：	Jan, 2017
Degree-Conferring-Institution：	Shanghai Jiao Tong University

软件工程领域语义相关词的挖掘与应用

摘 要

代码搜索是软件开发及维护过程中的一项常见任务，开发者经常需要进行代码搜索来帮助完成代码学习和重用、代码重构、bug 定位等工作。现有的代码搜索工具大部分是基于关键字文本匹配的搜索方法，与传统信息检索类似，这种方法的一个关键问题在于用户查询关键字与代码文本用词不匹配。因此需要对用户查询做语义相关词扩展以提高搜索精度。

由于软件工程领域的单词语义与自然语言存在很大差异，代码搜索无法使用自然语言的语义相关词做查询扩展，需要软件工程领域的语义相关词表。目前已有的软件工程领域语义相关词挖掘研究大多采用简单的文本相似度检测方法或基于词汇同现的统计方法，具有较大的局限性。而现有的自然语言领域的 Word Embedding 方法在语义相关词挖掘任务上表现良好。基于此，本文设计了一种基于 Word Embedding 的软件工程领域语义相关词挖掘方法 SWordMap，并就 SWordMap 在代码搜索上的应用进行了研究。

SWordMap 采用 CBOW 神经网络语言模型，以 IT 技术问答网站 Stack Overflow 的文档作为训练数据，训练得到 19332 个软件工程领域单词的向量表示及语义相关词表。为研究所得语义相关词表在代码搜索上的应用，根据本地代码搜索及开源代码搜索的不同特点，本文分别设计了针对本地代码搜索及开源代码搜索的查询扩展模型，并基于搜索引擎 Elasticsearch 进行了实现。

本文实验从四个不同角度对 SWordMap 进行评估：SWordMap 挖掘语义相关词表的精确度；SWordMap 对关注定位任务效率的提升；SWordMap 对本地代码搜索精度的提升；SWordMap 对开源代码搜索

精度的提升。实验结果表明，SWordMap 能够有效挖掘软件工程领域语义相关词，能有效提升关注定位任务效率及本地代码搜索精度，对开源代码搜索精度的提升有限。与前人工作的对比实验表明 SWordMap 能挖掘更高精确度的语义相关词，给关注定位任务及本地代码搜索带来可观的帮助。

关键词：代码搜索，查询扩展，语义相关词，SWordMap

RESEARCH ON DIGGING AND APPLICATION OF SEMANTICALLY RELATED WORDS IN SOFTWARE

ABSTRACT

Code search is a common task for software development and maintenance. Developers often need to search code for programming tasks such as code study, code reuse and bug localization. Existing code search tools are usually based on keywords-text matching. The same as traditional information retrieval, the inherent difficulty of keyword based code search is vocabulary mismatch problem between user query and retrieved code. To improve the accuracy of code search, utilizing semantically related words for query expansion is needed.

It is limited to rely on natural language resources such as English dictionary and WordNet to expand code search query because the semantics of words in software differ badly from words in English. A number of techniques have been proposed to identify semantically related words in software, while most of them measure the similarity of words simply by text similarity comparison or statistics of word co-occurrence, the limitation is huge. This paper designs a Word Embedding based method to learn semantically related words in software, and studies its application on code search.

SWordMap obtains semantically related words for 19332 words in software through training the neural network language model CBOW on Stack Overflow documents. To study the application of obtained semantically related words on code search, this paper designs two query expansion models for local code search and large scale open-source code search, and implements them based on search engine Elasticsearch.

This paper designs four experiments to evaluate SWordMap: the precision of the semantically related words obtained by SWordMap; the

improvement on concern location by utilizing SWordMap; the improvement on local code search by utilizing SWordMap; the improvement on open-source code search by utilizing SWordMap. The experiment results show that SWordMap can effectively identify semantically related words in software, improve the concern location performance and local code search accuracy, but has a limited improvement on open-source code search. The results of comparable experiment with previous work show that SWordMap can identify more accurate semantically related words in software and help improve concern location and local code search significantly.

KEY WORDS: code search, query expansion, semantically related words, SWordMap

目 录

图 录

表 录

第一章 绪论

1.1 研究背景

随着软件系统规模的不断扩大，代码行数和参与人数的急剧增长，对软件的开发及维护变得相当困难。通常情况下，一个程序员需要依赖本地代码搜索来帮助快速定位到相关代码片段，大大加速软件开发及维护的工作。同时，随着越来越多的开源软件被开发，网络上已经积累了海量的开源代码，这些海量的代码涵盖了软件开发的方方面面，为帮助程序员理解、学习和重用代码提供了可能。为了在如此庞大的代码库中快速准确地找到相关代码，一个可靠的、自动化的开源代码搜索引擎是关键。

现有的代码搜索工具大部分是基于关键字文本匹配的搜索方法，面临与传统信息检索一样的问题，即用户查询关键字与代码文本用词不匹配[1]。为提高搜索结果的精确度和召回率，需要对用户查询做语义相关词扩展[2]。然而由于软件工程领域的单词语义与自然语言存在很大差异，代码搜索无法使用自然语言的语义相关词做查询扩展，需要软件工程领域的语义相关词表。目前已有的软件工程领域语义相关词挖掘研究如 SWordNet[3]，SEWordSim[4]等，大多采用简单的文本相似度检测方法或基于词汇同现的统计方法，具有较大的局限性，对代码搜索精度的提升有限。

1.1.1 代码搜索

现有的代码搜索工具如 Sando[5], Krugle[6]以及 Sourcerer[7]等都是基于关键字文本匹配的搜索方法，在实际使用中发现它们的搜索精度并不理想。这其中的一个关键问题在于用户查询关键字与代码文本用词不匹配。如图 1-1 所示，假设一个程序员想要搜索怎样执行一个线程的 Java 代码，他的查询关键字是“execute thread”，在 Krugle 搜索引擎中包含 Thread.run( )这个 Java API 调用的正确结果被遗漏。这个时候就需要对用户查询做语义相关词扩展，如使用布尔模型将原始查询扩展为“execute thread OR run thread”，其中的关键在于需要提供 execute 的语义相关词 run。因此需要对软件中的语义相关词进行挖掘。

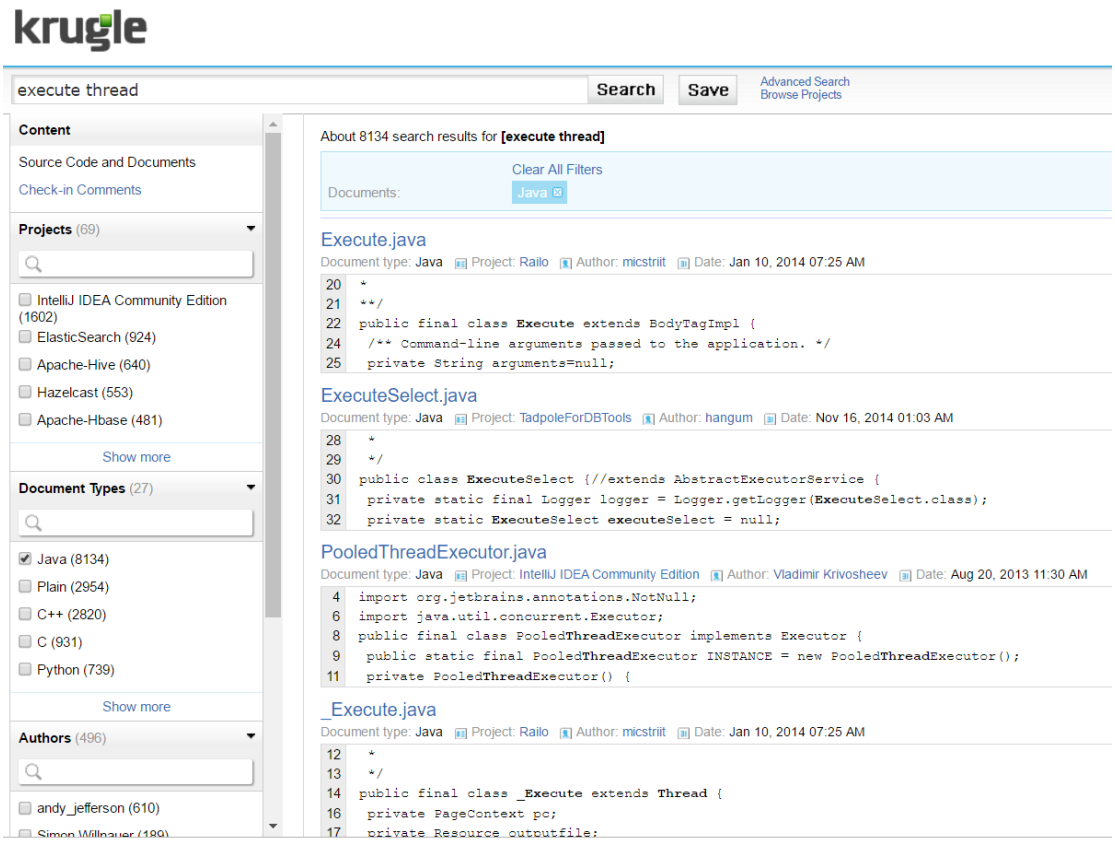

图 1-1 Krugle 对“execute thread”查询的搜索结果
Fig.1-1 Search results of query “execute thread” on Krugle

1.1.2 语义相关词

目前自然语言的语义相关词挖掘工作已经比较成熟，但是代码搜索无法直接使用如 Merriam-Webster 英语词典[8]、WordNet[9]等自然语言的语义相关词表来提高搜索精度。这是因为软件工程领域的单词语义与自然语言有很大不同。如上文提到的 execute 与 run 在 Merriam-Webster 英语词典以及 WordNet 中并不是语义相关词。软件工程领域中还存在大量自然语言中并不存在的缩略词，如 interrupt 和 irq，其中 interrupt 常出现在用户查询中而 irq 则常出现在代码中（如 linux 内核代码中经常出现 mask_irq 这样的函数名）。相对的，一些自然语言的语义相关词也并非软件工程领域的语义相关词，如 disable 和 torture 在 Merriam-Webster 英语词典中是语义相关词，而在软件工程领域中没有关系。Sridhara 等人对基于英语的语义相关词挖掘方法在软件工程任务上的应用做了调查研究，研究表明，采用自然语言的语义相关词进行查询扩展甚至会降低代码搜索的精度[10]。

1.2 国内外研究现状

目前国内外已经有一些针对软件工程领域语义相关词挖掘的相关研究[3-4,11-19]。Shepherd 等人通过自然语言处理方法从软件代码及代码注释中提取相似 verb-DO 对来识别语义相关词[12]。一个 verb-DO（verb-Direct Object）对是指一个动词加上其直接作用名词。具体来说，Shepherd 等人采用自然语言处理方法从类名、函数签名以及代码注释中提取 verb-DO 对，并且将出现在相似 verb-DO 对中的不同单词识别为语义相关词。例如从软件 iReport 的函数签名中找到了两个 verb-DO 对(add, element)和(find, element)，那么 add 和 find 就被识别为一对语义相关词。Hill 细化了 Shepherd 的研究，只从代码中提取 verb-DO 对并且提升了 verb-DO 对提取的准确性，推出了语义相关词识别精度更高的 SWUM[13]。Yang 等人则在 Shepherd 研究的基础上进行一定扩展推出了 SWordNet[3,15]，通过对软件中代码及注释的文本相似度比较挖掘语义相关词的工具。SWordNet 去掉了 SWUM 的自然语言约束，直接从函数签名及注释中推断语义相关词。例如 linux 内核代码的注释中存在“disable all interrupt sources”和“disable all irq sources”这样两条语句，由于具有相同上下文，interrupt 和 irq 在 SWordNet 中被识别为一对语义相关词。上述三种方法均是从软件代码及注释中挖掘语义相关词。如果是从单个软件中挖掘，得到的语义相关词只能用于特定软件，不具备普适性，如 SWordNet 将软件 jBidWatcher 中的 auction 和 entry 识别为一对语义相关词。如果是从多个软件中挖掘，由于不同软件所使用单词不尽相同，得到的语义相关词数量将大大受限。而且如果相似文本中包含自然语言单词，还会导致误报，如 SWordNet 由于一个软件的代码注释中同时存在“we have a match”和“we have a literal”这两条语句而错误地将 match 和 literal 识别成一对语义相关词。Howard 等人采用与 verb-DO 类似的思想从代码注释和函数签名的对应关系中挖掘语义相关词[16]，与 verb-DO 不同的是，他们仅提取代码注释及函数签名中的主要动词，以提高语义相关词的识别精度。Howard 等人的方法仅能挖掘动词的语义相关词。上述四种方法无法挖掘软件相关文档中的语义相关词。Wang 等人分析 Freecode 网站上软件标签之间的语义相似度，通过分析标签标记的具体文档之间的语义相似度对标签之间的语义距离进行衡量[17]。然而由于软件标签的数量较少，该方法能挖掘的语义相关词数量十分有限。Tian 等人推出了 SEWordSim[4,18]，对 Stackoverflow 的文档以基于词汇同现频率的统计方法计算单词之间的语义相似度。由于简单的词汇同现无法体现单词的深层语义，SEWordSim 得到的语义相关词精确度也不够理想。Ye 等人设计了一种基于 Word

Embedding 的软件工程领域文档相关度计算方法，并在 bug 定位以及 Q2API 任务上进行了实验[19]。Q2API（Question-to-API）任务是指给定一个 Stack Overflow 上的问题，查找与其相关的 Java API 文档。然而他们没有对软件工程领域语义相关词进行挖掘，也没有对其在代码搜索上的应用做更多的研究。

1.3 本文研究目的

如本文前面所述，现有的软件工程领域语义相关词挖掘方法存在缺陷，所挖掘语义相关词对代码搜索精度的提升有限。为了能从单词的深层语义着手挖掘具有普适性的语义相关词，提高挖掘语义相关词的精确度，本文设计了一种基于 Word Embedding 的软件工程领域语义相关词挖掘方法 SWordMap。为研究 SWordMap 在代码搜索上的具体应用，本文设计了针对代码搜索的查询扩展模型，以 SWordMap 挖掘语义相关词作为扩展词。根据本地代码搜索与开源代码搜索的不同特点，本文分别设计了针对本地代码搜索及开源代码搜索的查询扩展模型。本文主要研究目的为：

1. 一种基于 Word Embedding 的软件工程领域语义相关词挖掘方法 SWordMap，该方法能提高挖掘语义相关词的精确度。
2. 一种针对本地代码搜索的查询扩展模型，使用 SWordMap 挖掘语义相关词作为扩展词能有效提升本地代码搜索精度。
3. 一种针对开源代码搜索的查询扩展模型，使用 SWordMap 挖掘语义相关词作为扩展词能有效提升开源代码搜索精度。

1.4 本文研究内容

本文针对现有的软件工程领域语义相关词挖掘方法的缺陷设计了一种基于 Word Embedding 的软件工程领域语义相关词挖掘方法 SWordMap，并且分别针对本地代码搜索及开源代码搜索设计了不同的查询扩展模型，对 SWordMap 在代码搜索上的应用进行研究。本文主要研究内容为：

1. 研究现有自然语言领域语义相关词挖掘方法，设计了一种基于 Word Embedding 的软件工程领域语义相关词挖掘方法 SWordMap，并且以 IT 技术问答网站 Stack Overflow 的文档作为训练数据训练得到了 19332 个单词的向量表示及语义相关词表。

2. 研究现有信息检索模型，针对本地代码搜索以及开源代码搜索分别设计了查询扩展模型，并基于 Elasticsearch 进行了具体实现。
3. 设计实验评估 SWordMap 所挖掘语义相关词的精确度，以及 SWordMap 对关注定位任务效率、本地代码搜索精度及开源代码搜索精度的提升。

1.5 论文结构

本篇论文的结构主要分为以下部分：

第一章：绪论，介绍本文的研究背景，国内外研究现状，研究目的，研究内容以及论文结构。

第二章：介绍本文所使用的 Word Embedding 技术以及信息检索模型的相关知识。

第三章：介绍基于 Word Embedding 的软件工程领域语义相关词挖掘技术 SWordMap 的设计细节和具体实现。

第四章：介绍针对代码搜索设计的查询扩展模型的设计细节和具体实现。

第五章：介绍 SWordMap 评估实验的设计及实验结果分析。

第六章：对全文内容进行总结，并针对实验结果对 SWordMap 进行未来工作的展望。

最后是论文的参考文献、附录及谢辞。

第二章 背景技术介绍

本章对 SWordMap 所使用的 Word Embedding 技术以及信息检索模型的相关知识进行介绍。

2.1 Word Embedding

Word Embedding 是一系列语言模型及特征学习等自然语言技术的统称，这些自然语言技术将词典中的单词映射到固定维度的实数向量[20]。下文对词向量及用于训练词向量的语言模型进行介绍。

2.1.1 词向量

自然语言处理任务的第一步是将单词转换为数学对象，通常是向量。向量的每个维度代表了单词的一个特性，可能具有语义或语法上的含义。这些表征单词的向量被称为词向量。常见的词向量有两种：One-hot Representation 和 Distributed Representation，以下分别对它们进行介绍。

1. One-hot Representation

One-hot Representation 用一个固定长度的 01 向量来表示词典中所有单词。向量维度为词典D的大小N，向量的所有分量中只有一个 1，其他全为 0，1 的位置对应该单词在词典中的索引。这种词向量表示方法十分简单易懂，但其最大的问题在于无法体现单词的语义，无法衡量两个单词之间的语义相似度[21]。比如一个简单的词典中，单词 execute 可能被表示为[1, 0, 0]而 run 被表示为[0, 1, 0]，从它们的向量表示无法判断这两个单词语义是否相似。其次这种方法在词典较大时容易发生维数灾难。

2. Distributed Representation

Distributed Representation 也叫做 Word Embedding，最早由 Hinton 在 1986 年提出[22]。Distributed Representation 的基本思想是通过训练将每个单词映射成固定维度的实数向量，其中每个维度代表了单词的一个潜在特性，预期能捕捉到与单词相关的有用的语法及语义信息。两个单词的语义越相似，在向量空间中的距离也越短，因此通过计算词向量之间的距离即可判断两个单词的语义相似度。比如在一个简单的 Word Embedding 模型中，单词 execute 可能被映射到[0.12, -0.32, 0.01]

而 run 被映射到[0.12, -0.31, 0.02]，从二者的向量距离可以很容易地衡量它们的相似关系。

2.1.2 统计语言模型

传统的统计语言模型（Statistical Language Model）是表示语言基本单位（一般为句子）的概率分布函数，通常基于一个语料库来构建。假设$s = w_1^n := (w_1, w_2, \dots, w_n)$表示由$n$个单词$w_1, w_2, \dots, w_n$按顺序构成的一个句子，根据 Bayes 公式，句子s的概率可以表示为[23]：

$$p(s) = p(w_1^n) = p(w_1, w_2, \dots, w_n) = \prod_{i=1}^{n} p(w_i | w_1, w_2, \dots, w_{i-1}) \quad (2\text{-}1)$$

由于条件概率$p(w_i | w_1, w_2, \dots, w_{i-1})$的参数空间太大，语言模型强制将其转换为一个大概等价的形式[23]：

$$p(s) \cong \prod_{t=1}^{T} p(w_T | Context) \quad (2\text{-}2)$$

其中 *Context* 代表句子中的上下文。根据对 *Context* 的不同定义，统计语言模型可以分为以下几类[24]：

1. 上下文无关模型

该模型定义$Context = NULL$，仅考虑当前单词本身出现的概率。不需要任何学习训练过程，根据对语料库中词频的统计即可估算单词出现的概率。这是一种最简单、易于实现，但没有太大实际应用价值的统计语言模型。

2. N-gram 模型

该模型定义$Context = w_{t-n+1}, w_{t-n+2}, \dots, w_{t-1}$，考虑出现在当前单词之前的 n-1 个单词，n=1 时即上下文无关模型，n=2 时被称为 bigram 语言模型，n=3 时被称为 trigram 模型。N-gram 模型的优点在于包含了前 n-1 个单词能提供的全部信息，这些信息对当前单词出现具有较强约束作用。同时由于仅考虑前 n-1 个单词，在 n 较小时模型的计算效率较高。

3. N-pos 模型

该模型定义$Context = c(w_{t-n+1}), c(w_{t-n+2}), \dots, c(w_{t-1})$，是 N-gram 模型的一种衍生模型。N-gram 模型假设第 t 个单词的出现概率条件依赖于它前面的 n-1 个具体单词，而实际上自然语言中很多单词的出现概率是条件依赖于它前面单词的语法功能的。N-pos 模型将单词按照语法功能进行分类，由这些词类推断下一个单词的出现概率。这样的词类称为词性（part of speech）。$c(w_i)$代表单词w_i的词性。

4. 基于决策树的语言模型

上文提到的三种语言模型都可以以统计决策树的形式表现出来，而统计决策树中每个节点的决策规则是一个上下文相关的问题。这些问题可以是“前一个单

词是w吗”或“前一个单词词性是c吗”等等，也可以是更为复杂的语法语义相关问题。相比上述三种模型，基于决策树的语言模型可以针对训练语料库的实际情况设计决策规则，更为灵活，缺点则在于构建决策树的时间空间开销太大。基于决策树的语言模型是一种更加通用的语言模型。

5. 最大熵模型

最大熵模型（Maxinum Entropy Model）的基本思想为：在学习概率模型时，所有可能的模型中熵最大的模型是最好的模型；若概率模型需要满足一些约束，则最大熵原理就是在对未知情况不做任何主观假设的前提下，从满足已知约束的条件集合中选择熵最大模型。

6. 自适应语言模型

上文所提到的几种语言模型的概率分布都是预先从训练语料库中计算好的，属于静态语言模型。自适应语言模型则是动态学习的过程，能够根据少量新数据动态地调整模型中的概率分布。

2.1.3 神经网络语言模型

前馈神经网络语言模型（Feedforward Neural Net Language Model，简称 NNLM）最早由 Bengio 等人提出[25]。如图 2-1 所示为 NNLM 的模型架构。

NNLM 的概率分布函数为：

$$f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t|w_1^{t-1}) \tag{2-3}$$

其中唯一约束为对任意w_1^{t-1}，满足$\sum_{i=1}^{|V|} f(i, w_{t-1}, \dots, w_{t-n+1}) = 1$，以及$f > 0$。在该模型中，它被分解为两部分：

1. 一个$|V| * m$的映射矩阵C，其中每一行表示一个单词的特征向量，每一列表示一个单词特性。即上文提到的 Distributed Representation。
2. 一个函数g，将输入的一串单词的特征向量$(C(w_{t-n+1}), \dots, C(w_{t-1}))$映射到词典中下一个单词$w_t$出现的条件概率分布。$g$的输出是一个$|V|$维向量，其中第 i 个元素对应估算概率$\hat{P}(w_t = i|w_1^{t-1})$。函数$g$如下：

$$f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1})) \tag{2-4}$$

如果将训练网络的参数记为 ω，那么整个模型的参数为 θ = (*C*, ω)。训练目标为最大化似然函数 *L*：

$$L = \frac{1}{T}\sum_t \log f(w_t, w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta) \tag{2-5}$$

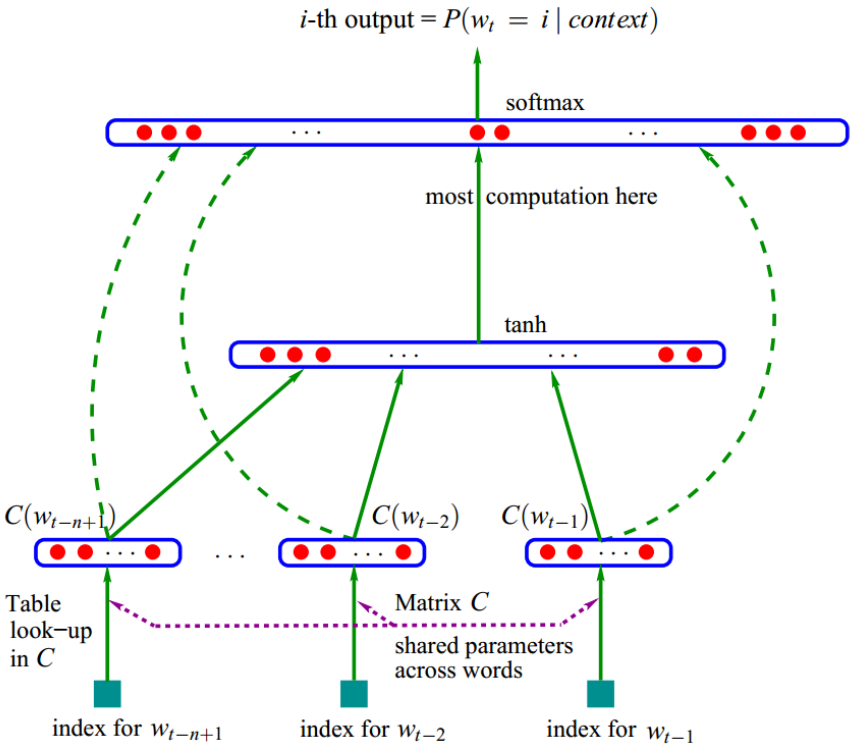

图 2-1 NNLM 模型架构[25]

Fig.2-1 Architecture of NNLM[25]

其中$R(\theta)$为正则化项。NNLM 分为四层：输入层，投影层，隐藏层以及输出层。其中，输出层为 softmax 归一化，保证满足概率分布的约束：

$$\hat{P}(w_t|w_{t-1},\dots,w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{2-6}$$

其中y_i是每个单词未正则化的 log 概率值，通过下式计算得到：

$$y = b + Wx + U\tanh(d + Hx) \tag{2-7}$$

其中双曲正切函数tanh应用在向量的每一个分量上。W通常设置为 0，表示输出层和输入层不存在直接联系。向量x为输入层词向量的拼接组合。假设隐藏层单元数量为h,词向量维度为m。b是隐藏层到输出层的偏值（长度为$|V|$的向量），d是投影层到隐藏层的偏值（长度为h的向量），U为隐藏层到输出层的参数（大小为$|V| * h$的矩阵），H为投影层到隐藏层的参数（大小为$h * (n-1)m$的矩阵）。

NNLM 相比传统 N-gram 模型的优点在于将历史信息映射到一个低维空间，即 Distributed Representation，降低了模型参数复杂度，且从实验结果来看效果非常不错[25]。但 NNLM 仍存在缺点，一方面是隐藏层到输出层的计算量非常大，导致模型训练非常耗时，无法用于大规模数据集的训练；另一方面是由于 NNLM 是典型的前馈神经网络，表征历史信息的上下文长度 n 必须提前指定，无法捕捉更

多的历史信息。

为了解决 NNLM 的上述问题，Mikolov 等人提出了循环神经网络语言模型（Recurrent Neural Net Language Model，简称 RNNLM）[26]。在 RNNLM 中，历史信息用循环相连的神经元进行表示，因此表征历史信息的上下文长度不再受限。同时，RNNLM 去掉了投影层，仅保留输入层、隐藏层及输出层。对于隐藏层到输出层的巨大计算量，RNNLM 采用了一种根据词频将单词分组的方法降低计算复杂度。RNNLM 与 NNLM 的最大区别在于网络会对前面的信息进行记忆并应用到当前输出的计算中，即隐藏层之间的节点不再是无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输入还包括上一时刻隐藏层的输出。

2.1.4 CBOW 模型

上文提到 NNLM 的训练太过耗时，为了降低 NNLM 的计算复杂性，以损失一定训练结果的精确度为前提使其能用于大规模数据集的训练，Mikolov 等人对 NNLM 进行改进提出了 CBOW 模型（Continuous Bag-of-Words Model）和 Skip-gram 模型（Continuous Skip-gram Model）[27]。如图 2-2 所示为它们的模型架构。

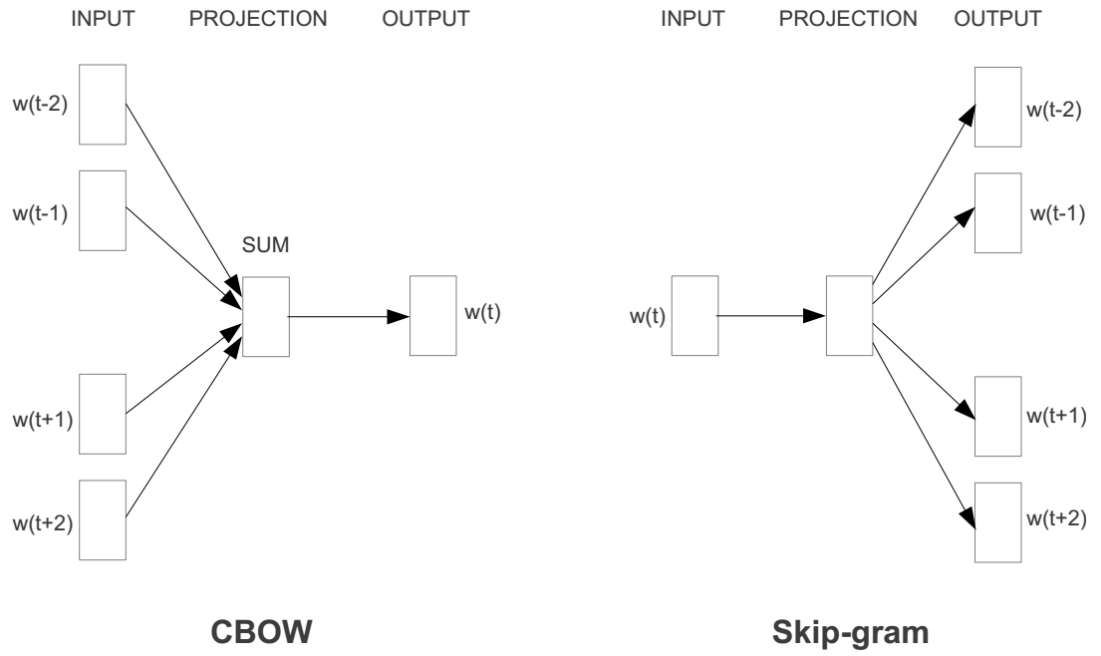

图 2-2 CBOW 和 Skip-gram 的模型架构[27]
Fig.2-2 Architectures of CBOW and Skip-gram[27]

为简化 NNLM，CBOW 模型和 Skip-gram 模型去掉了运算量最大的隐藏层，仅保留三层：输入层，投影层和输出层。Mikolov 等人对这两个模型给出了两套设

计框架，分别基于 Hierarchical Softmax 和 Negative Sampling[28]。其中 Hierarchical Softmax 对罕见词有利，训练速度较慢；Negative Sampling 对常见词有利，训练速度较快。限于篇幅，本文仅对基于 Hierarchical Softmax 的 CBOW 模型进行介绍。

与 NNLM 类似，CBOW 模型的训练目标为最大化对数似然函数 L：

$$L = \sum_{w \in C} \log p(w|Context(w)) \tag{2-8}$$

CBOW 模型三层的功能分别为：

1. 输入层：$Context(w)$定义为出现在单词w之前的 t 个单词及之后的 t 个单词。输入层包含$Context(w)$中出现的 2t 个单词的词向量。
2. 投影层：将输入层的 2t 个向量做求和累加。
3. 输出层：输出层对应一颗二叉树，它是以语料库中出现过的单词为叶子节点，以各单词在语料库中出现的次数为权值构造的哈夫曼树。在这颗哈夫曼树中，叶子节点共$|V|$个，非叶子节点共$|V|-1$个。之后采用 Hierarchical Softmax 对词典中下一个单词出现的条件概率进行计算。Hierarchical Softmax 最早由 Morin 和 Bengio 提出[29]，是对传统 softmax 的一个有效近似。

相比 NNLM，CBOW 模型去掉了隐藏层，输出层改用了哈夫曼树，采用 Hierarchical Softmax 代替传统的 softmax 归一化运算，大大降低了计算复杂性。CBOW 模型可以有效训练大规模数据集，且耗时较短。

CBOW 模型基于上下文对当前单词进行预测，而 Skip-gram 模型则基于当前单词预测上下文。Skip-gram 模型的设计细节与 CBOW 模型类似，这里不再赘述。

2.2 信息检索模型

信息检索（Information Retrieval，简称 IR）是指从信息资源的集合中查找所需文献及相关信息内容的过程[30]。信息检索领域发展至今已经有了一系列成熟的检索模型，按照它们的数学理论基础可以分为基于集合论、基于线性代数以及基于概率和统计的模型。以下对几种常见信息检索模型做简要介绍。

2.2.1 布尔模型

布尔模型（Boolean Model）是基于集合论和布尔代数的经典信息检索模型[31]。在该模型中，查询以布尔表达式形式表示，满足布尔表达式的文档即为相关文档。例如用户查询为“(run OR execute) AND thread”，如果一个文档中包含 run 或 execute，

同时包含 thread，那么这个文档即为相关文档。

布尔模型的最大缺点在于其检索策略基于二元判定标准，即检索结果只有相关和不相关，无法衡量文档与查询之间的相关度。

2.2.2 向量空间模型

向量空间模型（Vector Space Model，简称 VSM）是基于线性代数的经典信息检索模型[32]。相比布尔模型的严格二元判定，它提出了部分匹配的检索策略。向量空间模型将文档和查询均表示为向量:

$$\boldsymbol{d_j} = (w_{1,j}, w_{2,j}, \dots, w_{t,j}) \tag{2-9}$$

$$\boldsymbol{q} = (w_{1,q}, w_{2,q}, \dots, w_{n,q}) \tag{2-10}$$

其中每个分量代表了一个单词的权重。如果一个单词在文档d_j中出现，那么它对应的分量大于 0。有许多不同方法可以用来计算单词的权重，其中最常用的是 tf-idf（term frequency – inverse document frequency）权重机制。

tf-idf 是两个统计量的乘积：词频（term frequency）和逆文档频率（inverse document frequency）。其中词频表示该单词在文档中的出现次数，词频越高表示该单词越能表征这个文档。逆文档频率表示该单词在文档库的多少个文档中出现，出现次数越多，说明该单词越普遍，越不能表征一个文档。一种常见的计算方式如下：

$$tfidf(t,d,D) = tf(t,d) * idf(t,D) = f_{t,d} * \log\frac{N}{n_t} \tag{2-11}$$

其中$f_{t,d}$为单词 t 在文档 d 中出现频率的正则化值，N 为文档库中文档总数，n_t为文档库中包含该单词的文档数量。

在计算得到文档和查询的向量表示后，采用余弦距离即可计算文档和查询之间的相似度：

$$sim(d_j, q) = \frac{\boldsymbol{d_j} * \boldsymbol{q}}{||\boldsymbol{d_j}||||\boldsymbol{q}||} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2}\sqrt{\sum_{i=1}^{N} w_{i,q}^2}} \tag{2-12}$$

2.2.3 扩展布尔模型

扩展布尔模型（Extended Boolean Model）是为了解决布尔模型的缺点提出来的[33]。如上文所述，布尔模型无法考虑查询中单词的权重，无法衡量文档与查询的相关度，检索结果粒度太粗。扩展布尔模型综合了布尔模型和向量空间模型的特点，将向量空间模型中的单词权重与布尔代数结合起来，实现了细粒度的布尔检索。类似向量空间模型，扩展布尔模型同样将文档和查询表示为向量，其中单

词权重$w_{i,j}$的常用计算公式为：

$$w_{i,j} = f_{i,j} * \frac{idf_i}{max_i idf_i} \tag{2-13}$$

对一个或的查询$q_{or} = (t_1 \vee t_2)$，假设对应的在文档 d 中的单词权重为w_1和w_2，则二者相关度的计算公式为：

$$sim(q_{or}, d) = \sqrt{\frac{w_1^2 + w_2^2}{2}} \tag{2-14}$$

相应的对一个与的查询$q_{and} = (t_1 \wedge t_2)$，二者相关度的计算公式为：

$$sim(q_{and}, d) = 1 - \sqrt{\frac{(1-w_1)^2 + (1-w_2)^2}{2}} \tag{2-15}$$

上述是 2 维扩展布尔模型且未考虑查询中单词权重，采用欧几里得距离以及 p 正则化可以推广至 k 维。推广的 k 维查询表示为：

$$q_{or} = (t_1 \vee^p t_2 \vee^p \ldots \vee^p t_k) \tag{2-16}$$

$$q_{and} = (t_1 \wedge^p t_2 \wedge^p \ldots \wedge^p t_k) \tag{2-17}$$

对应的相关度计算公式为：

$$sim(q_{or}, d) = \sqrt[p]{\frac{w_{1,d}^p w_{1,q}^p + w_{2,d}^p w_{2,q}^p + \cdots + w_{k,d}^p w_{k,q}^p}{w_{1,q}^p + w_{2,q}^p + \cdots + w_{k,q}^p}} \tag{2-18}$$

$$sim(q_{and}, d) = 1 - \sqrt[p]{\frac{(1-w_{1,d})^p w_{1,q}^p + (1-w_{2,d})^p w_{2,q}^p + \cdots + (1-w_{k,d})^p w_{k,q}^p}{w_{1,q}^p + w_{2,q}^p + \cdots + w_{k,q}^p}} \tag{2-19}$$

举例：对一个查询$q = (t_1 \wedge t_2) \vee t_3$，假设查询中单词权重均为 1，其与文档$d$的相关度计算公式为：

$$\mathrm{sim(q, d)} = \sqrt[p]{\frac{\left(1 - \sqrt[p]{\frac{(1-w_1)^p + (1-w_2)^p}{2}}\right)^p + w_3^p}{2}} \tag{2-20}$$

2.2.4 其他模型

除上述三种经典模型外还有许多其他检索模型，如概率模型和语言模型。概率模型通过估算查询与文档相关的概率对查询与文档之间的相关度进行衡量，语言模型则依靠大规模数据训练进行统计建模用于信息检索。限于篇幅，本文不再对它们一一介绍。

2.3 本章小结

本章分别对 SWordMap 所使用的 Word Embedding 技术以及设计查询扩展模

型用到的信息检索模型进行了简要介绍，其中重点介绍了 CBOW 模型以及扩展布尔模型，为下文对 SWordMap 及本文设计查询扩展模型的阐述做好了铺垫。

第三章 SWordMap

本章对基于 Word Embedding 的语义相关词挖掘方法 SWordMap 的设计细节及具体实现进行介绍。

3.1 SWordMap 整体流程

如图 3-1 所示为 SWordMap 的整体流程。SWordMap 主要分为三步：（1）语料库构建；（2）模型训练；（3）语义相关词表生成。下文对这三个步骤作详细阐述。

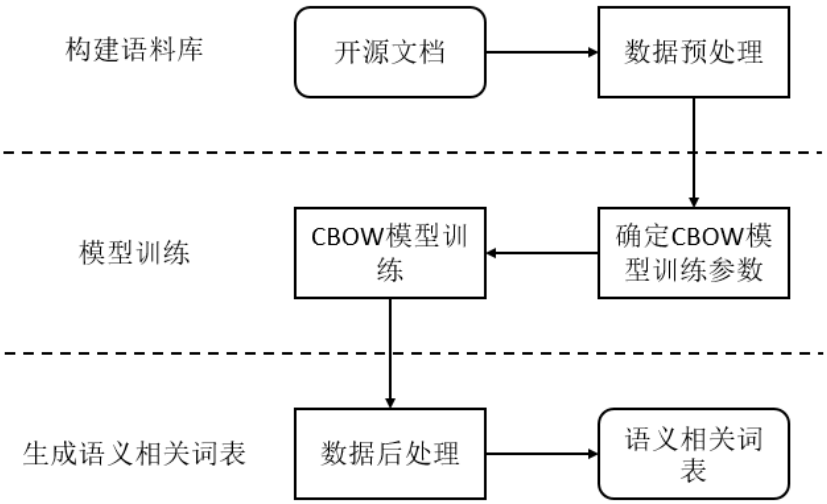

图 3-1 SWordMap 整体流程
Fig.3-1 Overall flow chart of SWordMap

3.2 语料库构建

Word Embedding 技术训练所得词向量的质量很大程度上取决于训练语料库的质量，因此构建一份高质量的语料库十分重要。

3.2.1 语料库选取

由于语言模型一般是句子的概率分布函数，在训练中以句子作为单词的上下文环境，因此语料库中文档需要能划分成类似句子的基本单元。其次，语料库中

文档内容需要软件工程领域相关，且覆盖面广泛，具有通用性。基于这两条准则，本文初步选取了 MSDN 博客[34]，Bing 搜索日志[35]以及 IT 技术问答网站 Stack Overflow[36]作为语料库来源。经过观察发现，MSDN 博客中夹杂大量非软件工程领域相关的英语，不适合用作训练数据；Bing 搜索日志中仅包含一两个单词的短句子太多，不能提供有效上下文，且单词拼写错误较多，也不适合用作训练数据；Stack Overflow 作为著名的 IT 技术问答网站，有着十分规范的管理（如用户提问会有人进行规范，用户提问需备注标签等等），其上的文档内容质量较高，软件工程领域相关，并且覆盖面相当广泛，适合作为训练数据。因此最终本文以 Stack Overflow 作为训练语料库来源。

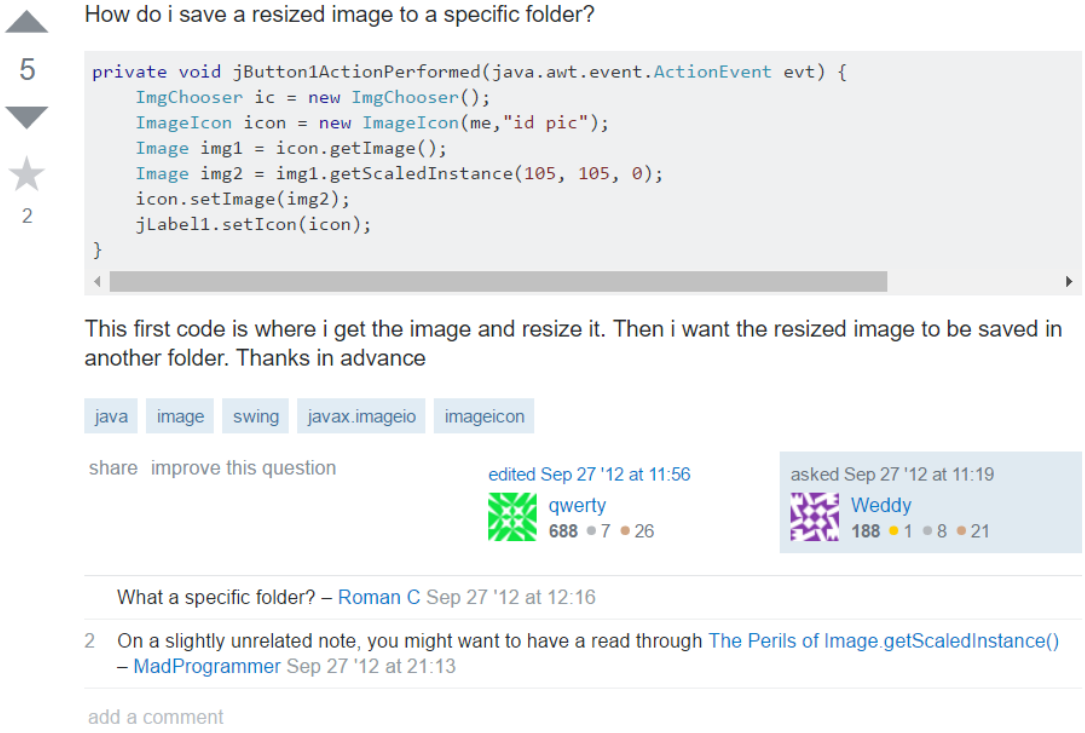

图 3-2 Stack Overflow 网站页面
Fig.3-2 A page on Stack Overflow

如图 3-2 所示为 Stack Overflow 网站上一个典型的问题页面。提问者在所提问题中会用一些语句来做详细阐述，同时还可能包含一些问题相关的代码片段。问题下面列有标签，表明问题所属类别。最底下还有一些浏览者的评论，对问题进行补充说明。问题左边还有投票数，表明问题热度。一个问题下面通常会有一些回答，格式与问题类似，这里不再举例说明。

代码中的函数体属于结构化文本，本身不具备类似句子的特征，而且一般包

含大量诸如循环变量等无明确语义的单词，因此不适合用作训练数据。代码中的函数名则具有类似短语句的特征，且一般表征了函数的具体功能[14]，适合用作训练数据。为同时考虑软件代码及相关文档中的语义相关词，本文选取 Stack Overflow 文档中的语句文本以及代码片段中的函数名作为训练数据。

考虑到程序语言特性，不同的程序语言一般具有不同的专有名词，它们的语义相关词也不同。例如 jdbc 是 Java 特有的数据库访问接口 API，仅在 Java 语言中 jdbc 与 database 为一对语义相关词。本文主要针对 Java 代码搜索，因此在选取语料库时仅选取带有 Java 标签的 Stack Overflow 问题。最终本文收集了共 708473 个 Stack Overflow 问题作为训练语料库。

3.2.2 数据预处理

在拿到 Stack Overflow 问题的原始数据之后需要做进一步处理。具体步骤如下：

1. 对原始数据中的超链接进行过滤。
2. 以标点符号“,”、“.”、“;”以及换行符等为分隔符对原始数据进行划分。
3. 原始数据中包含许多很短的口语化短句子，如“thank you”、“thanks in advance”等等，不具备训练价值。经过观察发现这些句子中停词数量占句中单词总数的比重一般大于 50%，因此本文对停词数量比重大于 50%的句子进行过滤。
4. 原始数据的句子中包含的函数名及 API 名还有代码中的函数名，如“saveImage”、“save_image”以及“Thread.run”，分别需要按照驼峰式命名规则、下划线命名规则以及符号“.”进行分词。

关于词干提取，本文在实际实验中发现，现有的词干提取工具不够精确，如著名的 Porter stemmer[37]会错误地将 adding 的词干提取为 ad，在代码搜索时会带来大量噪音；同时 CBOW 模型训练得到的结果能有效识别一个单词的词形变换。因此本文未对原始数据做词干提取。

经过上述处理后语料库中包含共 6828664 条句子，共 76352417 次单词出现。

3.3 模型训练

本文采用 CBOW 模型对语料库进行训练。CBOW 模型的实现采用了 word2vec[38]。CBOW 模型在开始训练前需要设置一些训练参数，如表 3-1 所示。其中比较关键的参数是单词向量维度以及上下文窗口大小。为选择最优的训练参

数，本文从上一节构建的语料库中随机选取了 100000 条语句作为样本，单词向量维度以 100 为间隔从区间[100, 500]取值，上下文窗口大小以 1 为间隔从区间[3, 8]取值，共得到 30 种不同的参数组合进行训练，使用常见软件工程领域语义相关词检查训练所得词向量的质量。根据采样实验结果，本文最终确定单词向量维度为 200，上下文窗口大小为 5。本文最终使用的训练参数为：size=5，min-count=5，window=5，negative=5，hs=0，sample=1e-4，threads=20，iter=15。

表 3-1 CBOW 模型训练设置参数
Table 3-1 Setting parameters of CBOW

参数	说明
size	单词向量维度大小
min-count	单词最低出现频率，低于该词频的单词会被过滤
window	上下文窗口大小
negative	如果大于 0，表示使用 Negative Sampling 方法，且该值表示采样单词数量
hs	如果等于 1，表示采用 Hierarchical Softmax 方法
sample	采样单词频率阈值，词频越高的单词越可能被采样
threads	训练线程数量
iter	训练迭代次数，迭代次数越多训练结果越精确

实际训练所使用的机器配置如表 3-2 所示。训练共耗时 5 小时 49 分钟。训练得到共 31123 个单词的向量表示。

表 3-2 训练机器配置
Table 3-2 Configuration of the machine used for training

配置	参数
CPU	2.4 GHz Intel Core i7
内存	64GB 1600 MHz DDR3
操作系统	Ubuntu 14.04 LTS 64bit

3.4 语义相关词表生成

3.4.1 数据后处理

在得到 CBOW 模型的训练结果之后还需要做进一步处理。具体步骤如下：

1. 训练结果的单词中包含大量无意义的数字符号，如 1.0f，1e-3 等等。本文对所有这些数字符号进行过滤。

2. 训练结果的单词中包含一些停词（stop words），所谓停词是指英语中经常使用的无具体语义的单词，如 a、the、or 等等。本文根据 SWordNet 使用的停词表以及英语中的常用停词表[39]构建了一份停词表，并根据该表对所有停词进行过滤。

3. 训练结果中存在一些单词，由于在语料库中出现频率过低，训练得到的词向量不够精确。这些单词通常是一些冷门单词或拼写错误，在训练前为保证句子上下文的完整性没有去掉，在训练完成后需要过滤。经过观察发现这些单词的词频一般低于 30。因此本文对所有词频低于 30 的单词进行过滤。

经过上述处理后得到共 19332 个单词的词向量。本文采用 t-SNE 算法[40]绘制了从所得结果中随机采样 3000 个词向量的 2D 投影图，如图 3-3 所示。图 3-4 为局部放大之后的词向量的 2D 投影图。从图 3-4 中可以看出，语义相似的单词在向量空间中距离相近，如图中心的 clean、build 以及 linking，这三个单词均与代码编译过程相关。

3.4.2 语义相关词表

在得到词向量之后本文采用余弦距离计算任意两个单词之间的语义相似度。经过观察发现，距离一个单词最近的前 40 个单词之外的其他单词与其已几乎没有语义上的联系。因此本文初步选取距离每个单词最近的前 40 个单词作为其语义相关词。由于不同单词的语义相关词数量也不同，对某些单词来说距离最近的前 40 个单词并非都是语义相关词。根据采样实验，本文最终确定以相似度 0.4 为阈值对所得语义相关词表进行过滤。最终得到 19332 个单词的语义相关词表，共 325583 对不重复的语义相关词。

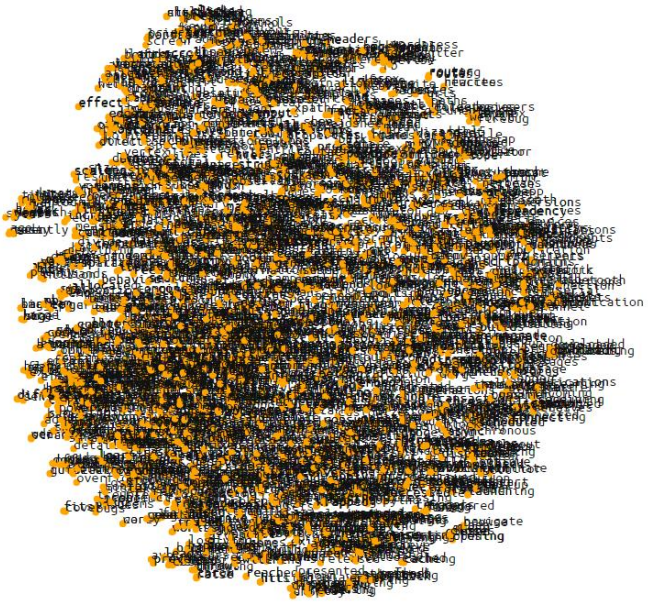

图 3-3 词向量的 2D 投影图
Fig.3-3 A 2D projection of word embeddings

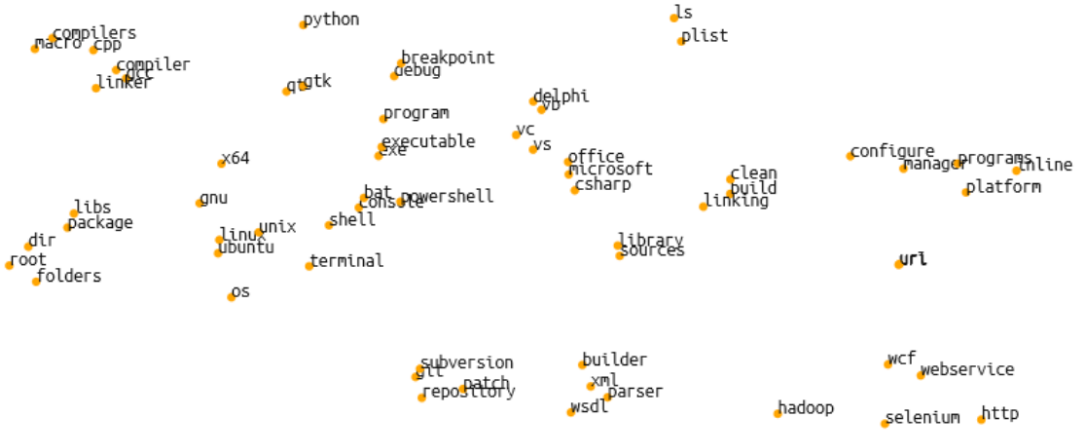

图 3-4 局部放大的词向量的 2D 投影图
Fig.3-4 A local amplification of 2D projection of word embeddings

如表 3-3 所示为从所得语义相关词表中截取的几个示例。从该表可以看出，本文方法可以有效挖掘不同形式的语义相关词：

1. 词形变换。如 save 和 saves，saving，saved。
2. 同义词。如 delete 和 remove。
3. 反义词。如 delete 和 insert。
4. 缩略词。如 image 和 img。
5. 语义相关词。如 excel 和 xls，xlsx，csv（这些均为 excel 的文档格式）。

注意到本文方法还能识别单词拼写错误，如 conection 和 connection。3.2.1 节中提到的 jdbc 也被成功识别为 db 的语义相关词。

表 3-3 语义相关词示例
Table 3-3 Examples of semantically related words

单词	语义相关词（按相似度由高到低排序）
save	saves, saving, saved, store, backup…
delete	deletes, remove, deleting, deleted, insert…
directory	folder, directories, dir, subdirectory, folders…
image	images, picture, png, bitmap, img…
db	database, databse, sql, databases, jdbc…
conection	connection, connexion, connections…
excel	xls, xlsx, csv, spreadsheet, ods…

3.5 本章小结

本章阐述了本文提出的基于 Word Embedding 的软件工程领域语义相关词挖

掘技术 SWordMap 的整体流程及处理细节，并对 SWordMap 挖掘的语义相关词进行了举例说明。根据观察，SWordMap 能有效挖掘不同形式的软件工程领域语义相关词，甚至是单词拼写错误。下文将对如何应用所得语义相关词进行阐述。

第四章 查询扩展模型

在得到软件工程领域语义相关词表后，剩下的关键问题是如何将其应用到代码搜索上以提高搜索精度。本章对本文针对代码搜索设计的查询扩展模型进行介绍。

4.1 代码搜索整体流程

如图 4-1 所示为代码搜索的整体流程。查询扩展是代码搜索中的一个步骤，用于对原始用户查询作语义相关词扩展，生成新的查询作为搜索引擎的输入，以提升代码搜索精度。本文根据本地代码搜索和开源代码搜索的不同特点分别设计了两种查询扩展模型。下文对这两种查询扩展模型作详细阐述。

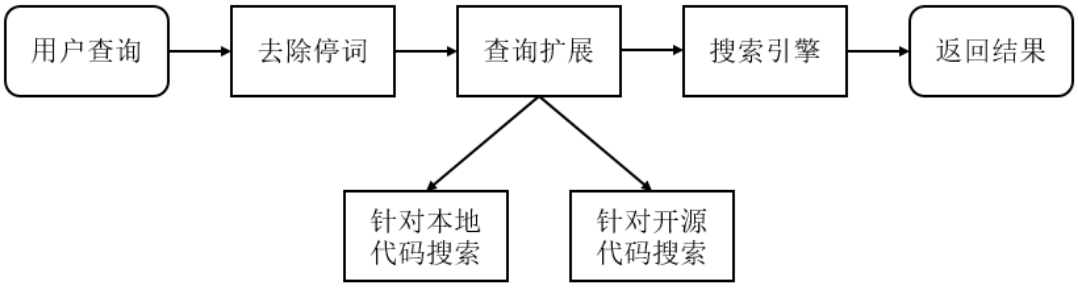

图 4-1 代码搜索整体流程
Fig.4-1 Overall flow chart of code search

4.2 针对本地代码搜索的查询扩展模型

本地代码搜索在许多软件任务中均有应用，如关注定位、bug 定位以及代码重构等[41]。在这些任务中，以本地代码搜索的返回结果作为起点，能大大提升用户的工作效率。

本地代码搜索具有如下特点：

1. 相比精确度更注重召回率。由于本地代码搜索仅搜索一个软件项目中的所有代码，查询的相关文档集合一般较小，返回结果集合也不大，因此用户可以很容易判断所有返回结果的相关性。而如果返回结果中有遗漏，用户

则需要花费数倍的时间浏览查阅整个软件项目以找到相关代码[13]。

2. 搜索返回结果的排序不太重要。本地代码搜索的目的是找到某个查询的所有相关代码，而不是最相关的部分代码，即对返回结果的相关性用户可以做出严格的二元判断。如上文所述，本地代码搜索返回结果集合较小，因此用户很容易检查所有返回结果的正确性。

根据上述特点，针对本地代码搜索的查询扩展一个朴素的想法即布尔模型。因此本文基于布尔模型设计针对本地代码搜索的查询扩展模型。

一个包含 n 个单词的查询Q定义如下：

$$Q = (t_1, t_2, \dots, t_n) \tag{4-1}$$

根据前人研究[3,14-15]，由于一个函数的函数签名（包括函数名及其所属类名）一般对其具体功能进行了概括，而函数体则可能使用了不相关单词对其进行实现，在进行本地代码搜索时仅考虑函数签名即可获得不错的效果。因此对一个代码片段，本文仅考虑它的一个特性f：函数名及其所属类名。因此原始查询Q可以表示为：

$$Q = (q_1\ AND\ q_2\ AND\ \dots AND\ q_n) \tag{4-2}$$

其中$q_i = (f{:}\ t_i)$。

假设单词t_i有n_i个语义相关词，其列表为$(t_{i1}, t_{i2}, \dots, t_{in_i})$，本文将该单词及其所有语义相关词以或操作符进行联结。经过查询扩展后的查询Q_e为：

$$Q_e = (q_{e1}\ AND\ q_{e2}\ AND\ \dots AND\ q_{en}) \tag{4-3}$$

其中：

$$q_{ei} = (f{:}\ t_i\ OR\ f{:}\ t_{i1}\ OR\ f{:}\ t_{i2}\ OR\ \dots OR\ f{:}\ t_{in_i}) \tag{4-4}$$

示例：假设一个用户查询为“How to execute a thread?”，在去除停词后查询$Q = (execute, thread)$。假设 execute 的相关词列表为$(run, invoke)$，thread 的相关词列表为$(runnable, task)$，那么经过查询扩展后的查询Q_e为：

$$Q_e = (f{:}\ execute\ OR\ f{:}\ run\ OR\ f{:}\ invoke)\ AND\ (f{:}\ thread\ OR\ f{:}\ runnable\ OR\ f{:}\ task) \tag{4-5}$$

这种模型设计虽然简单，但符合本地代码搜索的特点，实际应用中对提升本地代码搜索精度有着不错的效果。

4.3 针对开源代码搜索的查询扩展模型

开源代码搜索主要用来帮助程序员理解、学习和重用代码。开源代码搜索需要解决的关键问题是如何在海量源代码中快速准确地查找到与用户查询最为相关

的代码。

开源代码搜索具有如下特点：

1. 精确度与召回率难以衡量。开源代码搜索的对象往往是海量开源代码，查询的相关文档集合可能很大，返回结果集合也很大。在这种情况下，对返回结果的精确度及召回率计算几乎不可能。用户也往往仅对搜索返回比较靠前的部分结果感兴趣，不会花费时间去查看更多结果。
2. 搜索返回结果的排序十分重要。开源代码搜索的目的是找到与某个查询最相关的部分代码，因此返回结果的排序直接影响本次搜索的实际价值。在返回结果中越相关的代码应该越靠前。

根据上述特点可知经典的布尔模型不再适用。向量空间模型能够计算查询与文档之间的相似度，但是无法处理布尔查询。向量空间模型的相似度计算依赖于单词的 tf-idf 值，如果某个文档仅包含查询中一个单词，但是该单词词频很高，则可能出现该文档排序高于包含查询中多个单词的文档的情况，使搜索结果精度下降。扩展布尔模型结合了二者的特点，能够处理布尔查询，也能计算查询与文档之间的相似度。因此本文基于扩展布尔模型设计针对开源代码搜索的查询扩展模型。

同上，一个包含 n 个单词的查询Q定义如下：

$$Q = (t_1, t_2, \dots, t_n) \tag{4-6}$$

不同于本地代码搜索，开源代码搜索需要考虑更多的代码片段特性，仅考虑函数签名会导致相关度计算粒度太粗，排序机制丧失其应有的意义。比如针对一个用户查询“How to save an image”，代码库中一般存在大量函数名类似“saveImage”的函数，需要考虑更多代码片段特性对这些结果进行排序，从中选出与查询最相关的结果。由于软件中的函数一般对应一个具体功能的实现，在前人研究中[3,12-15,42]，常把函数作为代码搜索的基本单位。本文同样以函数作为代码搜索的基本单位，并考虑一个函数的如下特性：

1. f_1：函数名，一般是函数的简短描述，最能表征一个函数具体功能。
2. f_2：所属类名，与该函数具体功能相关。
3. f_3：函数体，该函数功能的具体实现，一般含有大量无具体语义的单词如循环变量等，导致其无法较好表征函数功能。
4. f_4：函数注释，写在函数签名之前，一般是对函数功能的说明，也包括对函数形参的说明。由于开源软件中具有良好规范的函数注释并不多，且函数注释的质量完全取决于代码作者，函数注释对函数功能的表征作用有限。

因此原始查询Q可以表示为：

$$Q = (q_1\ AND\ q_2\ AND\ \dots AND\ q_n) \tag{4-7}$$

其中:

$$q_i = (f_1{:}\,t_i\ OR\ f_2{:}\,t_i\ OR\ f_3{:}\,t_i\ OR\ f_4{:}\,t_i) \tag{4-8}$$

假设单词t_i有n_i个语义相关词，其列表为$(t_{i1}, t_{i2}, \dots, t_{in_i})$，与上文类似，本文仍将该单词及其所有语义相关词以或操作符进行联结。经过查询扩展后的查询Q_e为：

$$Q_e = (q_{e1}\ AND\ q_{e2}\ AND\ \dots AND\ q_{en}) \tag{4-9}$$

其中：

$$q_{ei} = (q_{oi}\ OR\ q_{ei1}\ OR\ \dots OR\ q_{ein_i}) \tag{4-10}$$

$$q_{oi} = (f_1{:}\,t_i\ OR\ f_2{:}\,t_i\ OR\ f_3{:}\,t_i\ OR\ f_4{:}\,t_i) \tag{4-11}$$

$$q_{eij} = \left(f_1{:}\,t_{ij}\ OR\ f_2{:}\,t_{ij}\ OR\ f_3{:}\,t_{ij}\ OR\ f_4{:}\,t_{ij}\right) \tag{4-12}$$

示例：假设一个用户查询为“How to save an image?”，在去除停词后查询$Q = (save, image)$。假设 save 的相关词列表为$(write)$，image 的相关词列表为(img)，那么经过查询扩展后的查询Q_e为：

$$\begin{aligned} Q_e = (&((f_1{:}\,save\ OR\ f_2{:}\,save\ OR\ f_3{:}\,save\ OR\ f_4{:}\,save\ )\ OR \\ &(f_1{:}\,write\ OR\ f_2{:}\,write\ OR\ f_3{:}\,write\ OR\ f_4{:}\,write))\ AND \\ &((f_1{:}\,image\ OR\ f_2{:}\,image\ OR\ f_3{:}\,image\ OR\ f_4{:}\,image\ )\ OR \\ &(f_1{:}\,img\ OR\ f_2{:}\,img\ OR\ f_3{:}\,img\ OR\ f_4{:}\,img))) \end{aligned} \tag{4-13}$$

由 2.2.3 节可知查询Q_e与文档d的相似度计算公式。其中关键在于项t_i在查询中的权重$w_{i,q}$以及在文档中的权重$w_{i,d}$的计算。注意这里存在三层扩展布尔模型的嵌套，需分开进行讨论：

对于Q_e，本文假设原始查询中的各项平权，因此在计算Q_e时各项q_{ei}的权重为$w_{q_{ei},q} = 1$。

对于q_{ei}，本文假设原单词的权重为 1，扩展词的权重可以直接采用其与原单词的相似度。假设扩展词t_{ij}的相似度为$s_{i,j}$，则在计算q_{ei}时各项权重为：

$$w_{t,q} = \begin{cases} 1, & if\ \mathrm{t} = q_{oi} \\ s_{i,j}, & if\ \mathrm{t} = q_{eij} \end{cases} \tag{4-14}$$

对于q_{oi}和q_{eij}，根据上述 4 个特性的特点可知，函数名f_1最能体现函数的具体内容，所属类名f_2次之，函数体f_3噪点较多，而函数注释f_4则具有不确定性，据对开源网站 GitHub 的实际考察，开源软件中的大部分函数没有函数注释，而带注释的函数其注释一般能较好说明函数内容。基于此，本文配置计算q_{oi}及q_{eij}时各项权重为：

$$w_{t,q} = \begin{cases} 1.5, & if\ t = (f_1 : t_i) \\ 1, & if\ t = (f_2 : t_i) \\ 0.8, & if\ t = (f_3 : t_i) \\ 1.2, & if\ t = (f_4 : t_i) \end{cases} \tag{4-15}$$

注意这样会导致搜索对带注释的函数带有偏好，这也是本文希望得到的结果，因为一个带注释的函数会比不带注释的函数更容易让用户理解。

对于q_{oi}和q_{eij}中各项在文档中的权重采用如下 tf-idf 公式直接计算：

$$w_{t,d} = \left(0.5 + 0.5 * \frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}}\right) * log\frac{N}{n_t} \tag{4-16}$$

4.4 查询扩展模型实现

本小节对上述两种查询扩展模型的具体实现进行阐述。由于开源代码搜索的对象数量十分巨大，在实现时需要考虑系统性能。经过对现有搜索引擎的调查研究，本文最终决定基于搜索引擎 Elasticsearch 实现上述查询扩展模型。

4.4.1 Elasticsearch

Elasticsearch 是一个分布式可扩展的实时搜索分析引擎，能够快速地处理大量数据的存储、搜索及分析。Elasticsearch 建立在全文搜索引擎 Apache Lucene 的基础上，是一个十分高效的全功能开源搜索引擎框架[43]。Lucene 不是一个完整的全文搜索引擎，而是一个全文搜索引擎的架构，要充分利用它的功能，需要在开发者自己的程序中进行集成。而且 Lucene 十分复杂，学习成本较高。Elasticsearch 使用 Lucene 作为内部引擎，提供了统一完整的 API 接口，实现了对 Lucene 复杂细节的封装。同时 Elasticsearch 还支持分布式实时文件存储，实时数据分析以及高可扩展性，能够处理 PB 级别的结构化或非结构化数据[44]。

Elasticsearch 中存在一些核心的基本概念：集群（Cluster），节点（Node），索引（Index），类型（Type），文档（Document），碎片及复制（Shards & Replicas）。以下分别对它们做简要介绍：

1. 集群

集群是一个或多个节点（服务器）的集合，这些节点存储了应用中的所有数据，并且协同工作，共同提供了数据的索引及搜索功能。集群拥有唯一标识名，一个节点仅能通过该标识名加入一个集群。

2. 节点

节点是集群中的一台服务器，负责存储数据，以及与其他节点协同提供数据的索引及搜索功能。节点拥有唯一标识名。

3. 索引

索引是一系列具有相同特性的文档的集合。例如在一个订单存储系统中我们可以为客户信息及订单详情分别建立索引。索引拥有唯一标识名。

4. 类型

类型是索引中数据的逻辑上的类别划分。通常一个类型用来区分具有不同字段（field）的文档。例如在一个博客管理平台中我们可以仅建立一个索引，并给用户信息、博客文章以及用户评论分别分配类型。类型包含一个名字及一个映射（mapping），该映射描述了对应类型的文档可能拥有的字段或属性、每个字段的数据类型以及这些字段会被 Lucene 如何索引和存储。

5. 文档

文档是能被索引的基本信息单元。Elasticsearch 是面向文档型数据库，支持直接对文档进行索引、搜索、排序及过滤。一个索引的一个类型中允许存储不限数量的文档。Elasticsearch 使用 JSON（JavaScript Object Notation）作为文档序列化的格式。JSON 是一种轻量级的数据交换格式，简洁且易于阅读。如代码 4-1 所示为 Elasticsearch 中文档的一个具体实例。

代码 4-1 Elasticsearch 文档示例
Code 4-1 An example of Elasticsearch document

```
{
    "email":        "john@smith.com",
    "first_name":   "John",
    "last_name":    "Smith",
    "about": {
        "bio":          "Eco-warrior and defender of the weak",
        "age":          25,
        "interests": [ "dolphins", "whales" ]
    },
    "join_date": "2014/05/01",
}
```

6. 碎片及副本

一个索引中存储的数据量可能会超过硬件的限制，并且在一个节点上存储过多数据会导致该节点对搜索服务的响应变慢。为解决该问题，Elasticsearch 支持将

索引分片，即碎片。一个碎片可以存储到集群中的任意节点上，且具有完整的与索引同样的功能。碎片机制使得我们可以横向地扩展划分存储，并且支持更细粒度的分布式及并行操作，大大提高系统性能。

副本机制则是为了提高系统可用性及容错性。副本机制使得在集群中某个碎片或节点失效时系统仍具有高可用性，同时也使得搜索能在多个副本上并行执行，提高系统性能。

一个 Elasticsearch 集群可以包含多个索引，每个索引中可以包含多个类型。这些类型中包含许多文档，每个文档又包含了许多字段。如代码 4-2 所示，Elasticsearch 提供了 REST（Representational State Transfer）风格的交互 API——RESTful API。一个 Elasticsearch 搜索引擎的构建包括如下步骤：

1. 创建索引（indexing）

提取原始数据中的信息，按照既定的文档类型进行存储，并建立反向索引（inverted index）以加速数据的取回速度。

2. 搜索索引（search）

即根据用户的请求，按照既定检索模型搜索创建的索引，返回根据相关度排序的搜索结果。

代码 4-2 Elasticsearch RESTful API 示例
Code 4-2 An example of Elasticsearch RESTful API

```
1 PUT twitter/tweet/1
2 {
3       "user" : "kimchy",
4       "post_date" : "2009-11-15T14:12:12",
5       "message" : "trying out Elasticsearch"
6 }
```

4.4.2 系统实现

针对本地代码搜索的查询扩展模型的实现与针对开源代码搜索的查询扩展模型的实现类似，且后者更具代表性，因此本文仅对针对开源代码搜索的查询扩展模型的实现进行阐述。

如图 4-2 所示为 Elasticsearch 的工作流程。首先需要对收集的开源代码做信息提取，将提取的函数相关信息索引到 Elasticsearch 集群中；其次在搜索过程中，对给定用户查询，首先使用 SWordMap 挖掘语义相关词根据上文设计的查询扩展模型进行扩展，然后采用扩展布尔模型对索引中每个文档计算与查询的相关度，最后根据相关度进行排序并返回搜索结果。下文分别对这两个步骤进行介绍。

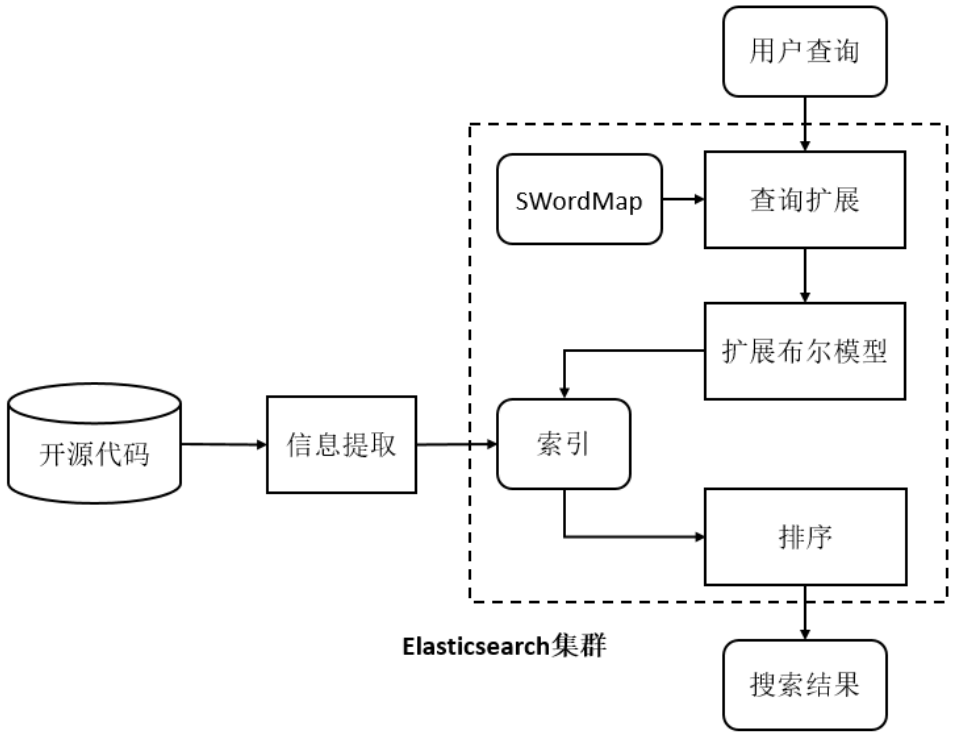

图 4-2 Elasticsearch 工作流程
Fig.4-2 Overall flow chart of Elasticsearch

1. 索引创建

本文代码搜索的文档是函数，在开始索引创建之前需要对 Elasticsearch 集群的索引及类型进行配置。如代码 4-3 所示为索引的配置，其中配置了分词器、停词表、碎片及副本数量。如代码 4-4 所示为类型的配置，其中前 4 个字段为上文提到的函数特性，参与检索模型计算；后 4 个字段为该函数所在文件、所属项目名等相关信息，不参与检索过程，仅作为给用户提供的额外信息，帮助用户更好地理解搜索结果。

在完成索引及类型配置后，就可以开始创建开源代码索引。如图 4-3 所示为索引创建的具体流程。索引创建主要分为三步：（1）收集开源代码；（2）对代码做抽象语法树（Abstract Syntax Tree，简称 AST）分析，提取代码中的函数信息；（3）通过 RESTful API 将提取的函数信息存储到 Elasticsearch 集群。其中第二步采用 JDT（Java development tools）[45]对 Java 代码进行分析。为提高集群中的索引质量，第二步对开源代码中的函数做如下过滤处理：

1. 忽略 main 函数。由于 main 函数内容通常是对软件中其他函数的调用，本身不包含具体功能的实现，缺乏索引价值。

代码 4-3 Elasticsearch 索引配置
Code 4-3 Index setting of Elasticsearch

```
{
   "settings": {
      "index": {
         "analysis": {
            "filter": {
               "custom_word_delimiter": {
                  "split_on_numerics": "false",
                  …
               },
               "my_stopwords": {
                  "type": "stop",
                  "stopwords": ["a", "about", "above", "after",…]
               },
            },
            "analyzer": {
               "custom_analyzer”: {
                  "filter": ["custom_word_delimiter", "lowercase", "my_stopwords"],
                  "type": "custom",
                  "tokenizer": "classic"
               }
            }
         },
         "number_of_shards": "1",
         …
         "number_of_replicas": "1"
      }
   }
}
```

2. 忽略测试用例函数。开源软件中通常存在一些测试用例函数，与 main 函数一样不包含具体功能的实现，缺乏索引价值。测试用例函数可以通过检查函数批注或函数名中是否包含单词“test”进行大致判断。
3. 忽略字符数及长度超过限定区间的函数。开源代码中存在一些字符数太多或长度太长的函数，对代码搜索用户的作用有限；字符数太少或长度太短的函数一般没有具体内容，缺乏索引价值。

代码 4-4 Elasticsearch 类型配置
Code 4-4 Type setting of Elasticsearch

```
{
   "snippet": {
      "properties": {
         "methodname": {
            "type": "string", "analyzer": "custom_analyzer"
         },
         "classname": {
            "type": "string", "analyzer": "custom_analyzer"
         },
         "body": {
            "type": "string", "analyzer": "custom_analyzer"
         },
         "comment": {
            "type": "string", "analyzer": "custom_analyzer"
         },
         "filepath": {
            "type": "string"
         },
         "projectLink": {
            "type": "string"
         },
         "projectName": {
            "type": "string"
         },
         "license": {
            "type": "string"
         }
      }
   }
}
```

图 4-3 Elasticsearch 索引创建流程
Fig.4-3 Overall flow chart of indexing of Elasticsearch

2. 搜索过程

搜索算法如算法 4-1 所示。对输入用户查询Q，首先对查询Q中单词根据 SWordMap 挖掘语义相关词表查找语义相关词，并构造扩展查询Q_e；然后对索引中每个文档d_i（以函数为基本单位的代码片段）采用 4.3 节设计的扩展布尔模型计算与扩展查询Q_e的相关度$sim(Q_e, d_i)$；最后对所有文档按相关度由高到低进行排序，返回排序后的文档列表$L = (d_1, d_2, \dots, d_m)$。

算法 4-1：搜索算法 **输入**：用户查询$Q = (t_1, t_2, \dots, t_n)$ **输出**：按相关度排序的搜索返回结果列表$L = (d_1, d_2, \dots, d_m)$	
1	对原始查询Q中每个单词t_i：
2	构造$q_{oi} = (f_1{:}\, t_i\ OR\ f_2{:}\, t_i\ OR\ f_3{:}\, t_i\ OR\ f_4{:}\, t_i)$
3	根据 SWordMap 查找t_i的语义相关词列表$(t_{i1}, t_{i2}, \dots, t_{in_i})$，对每个$t_{ij}$：
4	构造$q_{eij} = \left(f_1{:}\, t_{ij}\ OR\ f_2{:}\, t_{ij}\ OR\ f_3{:}\, t_{ij}\ OR\ f_4{:}\, t_{ij}\right)$
5	构造$q_{ei} = (q_{oi}\ OR\ q_{ei1}\ OR \dots OR\ q_{ein_i})$
6	构造扩展查询$Q_e = (q_{e1}\ AND\ q_{e2}\ AND \dots AND\ q_{en})$
7	对索引中每个文档d_i：
8	采用扩展布尔模型计算d_i与查询Q_e的相关度$sim(Q_e, d_i)$
9	根据相关度$sim(Q_e, d_i)$由高到低对文档进行排序，得到文档列表$L = (d_1, d_2, \dots, d_m)$
12	返回$L = (d_1, d_2, \dots, d_m)$

4.5 本章小结

本章阐述了本文针对代码搜索设计的查询扩展模型的设计思路及具体实现。针对本地代码搜索及开源代码搜索的不同特点，本文分别设计了两种查询扩展模型，并且基于搜索引擎 Elasticsearch 做了具体实现。下文将设计实验对 SWordMap 挖掘语义相关词及其在代码搜索上的应用进行验证评估，并分析实验结果。

第五章 实验与评估

本章对 SWordMap 挖掘语义相关词及其在代码搜索上的应用设计实验进行验证评估。本文共设计了四个不同实验：（1）SWordMap 挖掘语义相关词表的精确度；（2）SWordMap 对关注定位任务效率的提升；（3）SWordMap 对本地代码搜索精度的提升；（4）SWordMap 对开源代码搜索精度的提升。下文分别对这四个实验进行介绍。

5.1 语义相关词表的精确度

该实验主要目的为评估 SWordMap 挖掘语义相关词表的精确度。这直接影响所得语义相关词表的实际应用价值。

5.1.1 实验设定

1. 实验数据

从所得语义相关词表的 19332 个单词中随机选取 100 个单词，共 1872 个不重复的语义相关词对作为样本数据。

2. 实验方法

对样本数据中语义相关词对的正确性进行人工验证。为减少主观性的影响，验证由 3 名软件工程方向研究生完成，且采取较严格的评判标准，即三个验证者必须同时标记某个语义相关词对正确，才将该词对标记为正确。

在确认正确语义相关词对之后人工检查这些正确结果是否出现在 Merriam-Webster 英语词典或 WordNet 的语义相关词库中。选择 Merriam-Webster 英语词典是因为其权威性。

3. 实验评估标准

与 SWordNet 类似，本实验采用评估标准为精确度（precision），即样本中正确语义相关词对数量占样本总数的比重。由于一个单词的语义相关词一共有多少难以统计，因此召回率难以衡量。

5.1.2 实验结果

实验结果如表 5-1 所示。从表中数据可以看出样本的精确度为 74.7%，说明

SWordMap 挖掘语义相关词表的精确度平均在 74.7%左右，根据 Mikolov 等人在自然语言文档上训练 CBOW 模型的实验结果可知这是一个比较理想的数值[27]。与 SWordNet 挖掘语义相关词的平均精确度 56.9%相比提升了约 31.3%。同时，在所有正确语义相关词对中，不在 Merriam-Webster 英语词典或 WordNet 中的占 75.3%，进一步验证了 1.1.2 节中提到的单词语义在软件工程领域与自然语言之间的差异性。可以看出这种差异是相当大的。

表 5-1 语义相关词正确性验证结果
Table 5-1 Experiment results of correctness verification of semantically related words

语义相关词对总数	1872
正确语义相关词对	1398
精确度（%）	74.7
不在英语词典或 WordNet 中的正确结果数量	1053
不在英语词典或 WordNet 中的正确结果比例（%）	75.3

5.2 关注定位

该实验主要目的为评估 SWordMap 挖掘语义相关词表是否能有效提升关注定位（concern location）任务效率。关注定位是指用户对软件中某一个功能或模块感兴趣，需要通过用户提供的关键字查找到软件中所有与该功能或模块相关的代码。为与前人工作 SWordNet 进行比较，本文采用与 SWordNet 同样的实验数据及实验方法。

5.2.1 实验设定

1. 实验环境

本实验采用的实验环境配置如表 5-2 所示。

表 5-2 关注定位实验环境配置
Table 5-2 Experiment environment of concern location

配置	参数
CPU	2.4 GHz Intel Core i7
内存	8 GB 1600 MHz DDR3
操作系统	Windows 8.1 64bit

2. 实验数据

关注定位的对象为表 5-3 所示的 4 个 Java 软件。测试数据集如表 5-4 所示，为 Shepherd 提供的与这 4 个 Java 软件相关的 8 个关注定位任务及结果[46]。其中

关注定位搜索的基本单位为代码中的函数。如图 5-1 所示为软件 iReport 的关注定位任务“add textfield”的所有相关结果。

表 5-3 测试软件
Table 5-3 Evaluated Java softwares

软件名称	软件描述	代码行数（LOC）	函数个数
iReport	报表可视化设计器	74506	7587
javaHMO	多媒体服务器	25988	1787
jBidWatcher	eBay 拍卖监控软件	23502	1918
jajuk	音乐管理播放软件	20679	2132

表 5-4 关注定位测试数据
Table 5-4 Concern location tasks

软件名称	关注定位任务
iReport	add textfield
	compile report
javaHMO	gather music files
	load movie listing
jBidWatcher	add auction
	save auction
	set snipe price
jajuk	play song

```
Project: iReport   Concern: Add Textfield

Gold Set:
jReportFrame.dropNewTextField(Point, String, String, String)
jReportFrame.dropNewTextField(Point, String, String)
jReportPanel.drop(DropTargetDropEvent)
TextFieldReportElement.TextFieldReportElement(int, int, int, int)
TextReportElement.TextReportElement(int, int, int, int)
```

图 5-1 关注定位任务 add textfield
Fig.5-1 Concern location task – add textfield

3. 实验方法

与 SWordNet 的测试方法一样，搜索仅考虑函数的函数签名（函数名及其所属类名），搜索采取正则表达式匹配的搜索策略。具体来说，假设用户查询是“gather file”，那么原始查询的正则表达式是“.*gather.*file.*”以及“.*file.*gather.*”。在进行查询扩展时，假设 collect 是 gather 的一个语义相关词，那么查询扩展为“.*gather.*file.*”，“.*collect.*file.*”，“.*file.*gather.*”以及“.*file.*collect.*”。

4. 实验评估标准

与 SWordNet 的评估标准一样，该实验采用精确度（precision）及召回率（recall）作为评估标准。同时为了更直观地判断实验结果的好坏，本文还采用了 F-measure 评价指标。精确度即代码搜索返回结果中正确结果数量占所有返回结果总数的比重。召回率即代码搜索返回结果中正确结果数量占所有正确结果总数的比重。F-measure 评价指标综合考虑了精确度及召回率，是二者的调和平均。F-measure 的计算公式如下：

$$F = 2 * \frac{precision * recall}{precision + recall} \tag{5-1}$$

5.2.2 实验结果

实验结果如表 5-5 所示。其中 SWN 表示 SWordNet，SWN-T 表示 Transitive SWordNet（考虑了语义相关词对传递性的 SWordNet，即假设 a 与 b 是一对语义相关词，b 与 c 是一对语义相关词，那么能推断 a 与 c 也是一对语义相关词），SWM 表示本文工作 SWordMap。SWordNet 的实验结果直接取自 Yang 的论文[3]。注意在同样实验数据上 SWordNet 的表现要优于 SWUM，限于篇幅本文不再列举 SWUM 的实验结果。

表 5-5 关注定位实验结果
Table 5-5 Experiment results of concern location

关注定位任务	精确度（%）			召回率（%）			F-measure		
	SWN	SWN-T	**SWM**	SWN	SWN-T	**SWM**	SWN	SWN-T	**SWM**
add textfield	14.30	5.30	0	40	60	0	0.211	0.097	0
compile report	4	0.17	6.52	25	87.50	37.50	0.069	0.003	0.111
gather music files	28.60	0.50	28.57	50	75	50	0.364	0.010	0.364
load movie listing	0	0	0	0	0	0	0	0	0
add auction	3.70	0.94	9.41	100	100	72.73	0.071	0.019	0.167
save auction	1.61	0.66	7.41	33.30	77.80	66.67	0.031	0.013	0.133
set snipe price	0	0	2.60	0	0	16.67	0	0	0.045
play song	0	0	11.54	0	0	75	0	0	0.200
平均	6.53	0.95	**8.26**	31.04	50.04	**39.82**	0.093	0.018	**0.127**

从表中列 F-measure 的数据可以看出，在 8 个搜索任务中有 5 个 SWordMap 的表现优于 SWordNet，有 6 个优于 Transitive SWordNet。根据 F-measure 指标，SWordMap 相比 SwordNet 及 Transitive SWordNet 分别提升了 36.6%及 600%。

从表中数据可以看出，SWordMap 可以识别 SWordNet 不能识别的语义相关词从而提高代码搜索召回率，如查询“play song”中，SWordMap 成功将查询中的

song 和相关结果中的 playlist 识别成一对语义相关词而 SWordNet 没有。这是由于 SWordNet 采用简单的文本相似度比较挖掘语义相关词，无法真正理解单词的深层语义。而且 SWordNet 挖掘对象仅限于软件代码及注释，无法挖掘软件文档中的语义相关词。SWordMap 还能有效提升代码搜索精确度，8 个搜索任务中有 5 个 SWordMap 精确度优于 SWordNet，有 6 个优于 Transitive SWordNet。

SWordMap 仍存在缺陷。在查询“add textfield”中，SWordMap 未能找到任何相关结果。这是因为该查询中相关结果包含关键字为 drop，SWordMap 未能识别 add 和 drop 为语义相关。add 和 drop 这对语义相关词是软件 iReport 特有的，不具备普适性，因此 SWordMap 未能识别。在查询“load movie listing”中，SWordMap 与 SWordNet 表现一样，未能找到任何相关代码。这是因为该查询中相关结果包含关键字为 reload 和 container，SWordMap 成功识别了 load 和 reload 为语义相关，未能识别 listing 和 container。这说明 SWordMap 仅使用 Stack Overflow 的文档作为训练语料库是不充分的，需要从不同渠道收集更多更广泛的训练数据。SWordMap 的平均召回率要低于 Transitive SWordNet。注意虽然 Transitive SWordNet 的召回率较高，但它的精确度极低。实验中发现，使用 Transitive SWordNet 做查询扩展会导致代码搜索返回结果数量剧增（个别查询返回结果数量接近软件中所有函数），极大影响搜索的实用价值。

注意到虽然 SWordMap 在代码搜索的精确度及召回率上有一定提升，但平均精确度及召回率仍不高。这是由于以下两个原因：

1)测试数据为关注定位任务，关注定位要求给定查询关键字，返回项目中所有与查询内容相关的代码。然而由于代码搜索的固有局限性，代码搜索能够返回的仅仅是与该关键字在文本层面上直接相关的部分代码，更深层次的代码结构上的相关代码（比如在直接相关函数中调用的辅助函数，一般不大可能在文本语义上与查询相关）需要用户以搜索的返回结果作为起点使用代码浏览工具进行检索[12]。如图 3，函数 jReportPanel.drop(DropTargetDropEvent)与查询“add textfield”在文本层面没有直接关联，但是在软件 iReport 中是实现“add textfield”功能的一个辅助函数。因此代码搜索的召回率难以达到 100%。

2)搜索算法采用的是正则表达式匹配，而且考虑了所有可能的语义相关词的组合。这是基于保守的思想，因为在该实验设计中覆盖率比精确度更重要。对用户来说，检查搜索结果的正确性比起猜测软件中可能使用的语义相关词要简单得多[3]。

5.3 本地代码搜索

该实验主要目的为评估 SWordMap 挖掘语义相关词表是否能有效提升本地代码搜索精度。许多软件任务均从本地代码搜索开始，代码搜索的返回结果给程序员提供了一个起点[47]。

5.3.1 实验设定

1. 实验环境

本实验采用的实验环境配置同 5.2.1 节，如表 5-2 所示。

2. 实验数据

本地代码搜索的对象与关注定位实验相同，为表 5-3 中的 4 个 Java 软件。测试数据集的构建由 3 名软件工程方向研究生共同完成。测试数据中的查询任务由其中 1 名研究生提出，该研究生事先没有对测试软件的任何了解，在阅读过测试软件的相关文档后提出了这些与软件具体功能相关的问题。为减少主观性的影响，其他 2 名研究生对这些查询的客观性进行验证，检查它们是否带有对某项技术的偏好。最终选取了共 25 个查询。之后 3 名研究生对这些查询的相关结果的真实数据（ground truth）集进行构建。对于一个查询，这 3 名研究生首先分别阅读软件源码，查找它的相关结果，然后对 3 人查找结果进行汇总：首先取 3 人查找结果的交集，对 3 人查找结果并集中的其他结果由 3 人一起进行验证，只有在 3 人均同意该结果为真的时候才将其加入真实数据集。测试数据集如表 5-6 所示。

如图 5-2 所示为软件 iReport 的查询任务“initialize chart dialog”的相关结果。与关注定位不同的是，本地代码搜索仅查找与用户查询直接相关的函数。用户可以以代码搜索的返回结果为起点对更多与查询相关的函数进行检索。

```
Project: iReport   Query: Initialize chart dialog

ChartPropertiesDialog.initAll()
ChartPropertiesDialog.initComponents()
ChartSelectionJDialog.initAll()
ChartSelectionJDialog.initComponents()
IReportChartDialog.initAll()
IReportChartDialog.initComponents()
```

图 5-2 本地代码搜索任务 initialize chart dialog
Fig.5-2 Local code search task – initialize chart dialog

3. 实验方法

采用 4.2 节中针对本地代码搜索设计的查询扩展模型，分别使用 SWordNet 及 SWordMap 挖掘的语义相关词进行代码搜索。同时设立不做查询扩展的对照组。具体来说，假设用户查询为“export to excel”，则去除停词后原始查询为“export AND excel”。在进行查询扩展时，假设 export 的语义相关词是 import，excel 的语义相关词是 csv，那么查询扩展为“(export OR import) AND (excel OR csv)”。

4. 实验评估标准

与 5.2.1 节相同，采用精确度及召回率作为评估标准。

表 5-6 本地代码搜索测试数据
Table 5-6 Local code search tasks

软件名称	查询任务
iReport	initialize a chart dialog
	get the chosen text
	get db connection credentials
	export to excel
	set attributes of a chart
	copy a dataset
javaHMO	load audio data
	collect all music files
	get the category of a music
	get the thumbnail of a movie
	set the singer of a song
	extract network domain information
	configure server ip address
	find image files
jBidWatcher	delete an auction
	get the best bid price
	update an auction
	buy an item
	login to ebay
	translate between currency
jajuk	get playable songs
	add a playlist
	remove a song from playlist
	get the author of a track
	silence the player

5.3.2 实验结果

实验结果如表 5-7 所示。其中 Initial 表示不做查询扩展的对照组，SWN 表示 SWordNet，SWN-T 表示 Transitive SWordNet（考虑了语义相关词对传递性的 SWordNet），SWM 表示本文工作 SWordMap。

从表中数据可以看出，采用 SWordMap 查询扩展能有效提升本地代码搜索精

度。SWordMap 的精确度为 37%，覆盖率为 77.22%，相比原始查询分别提升了 147.8% 及 683.2%。SWordMap 相比 SWordNet 和 Transitive SWordNet 也有显著提升。

表 5-7 本地代码搜索实验结果
Table 5-7 Experiment results of local code search

查询任务	精确度（%）				召回率（%）			
	Initial	SWN	SWN-T	**SWM**	Initial	SWN	SWN-T	**SWM**
initialize a chart dialog	0	0	0	54.55	0	0	0	100
get the chosen text	0	0	0	31.82	0	0	0	100
get db connection credentials	0	0	0	40	0	0	0	100
export to excel	0	0	0	100	0	0	0	100
set attributes of a chart	0	0	0	14.81	0	0	0	57.14
copy a dataset	16.67	0.76	0.13	9.46	14.29	14.29	100	100
load audio data	100	25	0.26	66.67	20	60	60	40
collect all music files	0	0	0	0	0	0	0	0
get the category of a music	0	0	0	9.09	0	0	0	100
get the thumbnail of a movie	0	4.76	0.20	7.69	0	100	100	100
set the singer of a song	0	0	0	8.70	0	0	0	100
extract network domain information	0	0	0	100	0	0	0	100
configure server ip address	0	0	0	16.67	0	0	0	100
find image files	0	2.82	0.17	25	0	100	100	100
delete an auction	66.67	0.61	0.38	10	33.33	83.33	100	100
get the best bid price	0	0	0	100	0	0	0	100
update an auction	50	1.31	0.66	17.74	45.45	100	100	100
buy an item	0	0.56	0.26	0	0	75	100	0
login to ebay	0	0	0	0	0	0	0	0
translate between currency	0	0	0	100	0	0	0	100
get playable songs	0	0	0	0	0	0	0	0
add a playlist	40	10.17	2.60	4.17	33.33	100	100	33.33
remove a song from playlist	0	0	0	8.70	0	0	0	100
get the author of a track	100	0.51	0.25	100	100	100	100	100
silence the player	0	0	0	100	0	0	0	100
平均	14.93	1.86	0.20	**37**	9.86	29.30	34.40	**77.22**

注意到对许多查询 SWordNet 找不到任何相关结果而 SWordMap 可以。这是由于 SWordNet 仅能挖掘出现在当前搜索的软件中的语义相关词，若一个语义相关词未在当前搜索的软件中出现，则 SWordNet 无能为力。如软件 iReport 的查询“export to excel”中，excel 与 csv 为一对语义相关词，由于 excel 没有出现在软件 iReport 中，SWordNet 未能成功识别。SWordMap 能识别的语义相关词范围取决于

训练语料库，只要训练语料库范围足够广泛就不会有这个问题。同时在所有查询上 SWordNet 的精确度均低于原始查询，说明 SWordNet 挖掘的语义相关词的精确度较低，简单的文本相似度比较导致了较多误报，虽然在召回率上有提升，但精确度的大幅下降导致搜索的实用价值极低。相比 SWordNet，SWordMap 在具有高召回率的同时还有着较高的精确度，能够给本地代码搜索的用户带来可观的帮助。

SWordMap 仍有其局限性。在查询“buy an item”中，SWordMap 未能识别软件 jBidWathcer 特有的语义相关词对 item 和 auction。在查询“login to ebay”中，SwordMap 未能识别语义相关词对 login 和 sign，再次说明了 SWordMap 训练语料库不够充分。在查询“get playable songs”中，一个有趣的事实是软件 jajuk 的代码中错误地将 playable 拼写成了 playeable，导致 SWordMap 及 SWordNet 均未能找到相关结果。这也证实了拼写错误在代码中的确存在，因此识别一个单词的拼写错误对代码搜索也是有帮助的。在查询“collect all music files”中，相关结果中包含的关键字为 gather 和 directory，SWordMap 成功识别了语义相关词对 collect 和 gather 以及 files 和 directory，但是由于相关结果不包含 music 或其语义相关词，搜索未能返回任何相关结果。这实际上是检索模型的不足。

5.4 开源代码搜索

该实验主要目的为评估 SWordMap 挖掘语义相关词表是否能有效提升开源代码搜索精度。开源代码搜索为程序员理解、学习和重用代码提供了帮助，让程序员可以快速学习新的 API，借鉴前人工作来解决软件开发中遇到的问题。

5.4.1 实验设定

1. 实验环境

本实验采用的实验环境为两台服务器，配置如表 5-8 所示。每台服务器上部署了两个 Elasticsearch 节点，一共四个节点。

表 5-8 开源代码搜索实验环境配置
Table 5-8 Experiment environment of open-source code search

配置	参数
CPU	2.4 GHz Intel Core i7
内存	256 GB 1600 MHz DDR3
操作系统	Windows Server 2012 64bit

2. 实验数据

开源代码搜索的对象为开源社区 Github[48]的 Java 项目。Github 作为全球流行的开源社区，其上有着海量的具有高关注度及高可靠性的开源项目。Github 上的项目具有收藏数（star），一般来说收藏数越大说明该项目越受关注，代码质量越高。本文从 Github 上面下载了收藏数大于 900 的共 1023 个 Java 项目，共约 20.6GB，包含 457265 个 Java 代码文件，3547798 个函数，其中经过过滤后被索引到 Elasticsearch 集群中的函数共 1304201 个。

表 5-9 开源代码搜索测试数据
Table 5-9 Open-source code search queries

来源	查询
前人研究	copy paste data from clipboard
	open url in html browser
	track mouse hover
	highlight text range in editor
Stack Overflow	convert utc time to local time
	converting String to DateTime
	get current date and time
	get file name without extension
	how can I decode HTML characters
	how can I download HTML source
	how do I round a decimal value to 2 decimal places
	how to change RGB color to HSV
	how to convert an IPv4 address into a integer
	how to delete all files and folders in a directory
	how to execute a sql select
	how to generate random int number
	how to get Color from Hexadecimal color code
	how to get temporary folder for current user
	if a folder does not exist create it
	ping a hostname on the network
	Process.start: how to get the output
	sending email through Gmail
Bing 搜索日志	append string to file
	calculate md5 checksum
	how to deserialize XML document
	how to get mac address
	how to play a sound
	how to save image in png format
	read file line by line
	remove cookie
	verify folder exists
	how to reverse a string
	quick sort
	how to split string into words

测试采用的查询与 CodeHow 相同，为表 5-9 中的 34 条查询[42]。其中，第 1-

4 条取自前人研究工作[49]，第 5-22 条取自 IT 技术问答网站 Stack Overflow，第 23-34 条取自 Bing 搜索日志。

3. 实验方法

采用 4.3 节中针对开源代码搜索设计的查询扩展模型，使用 SWordMap 挖掘的语义相关词进行代码搜索。同时设立不做查询扩展的对照组。具体来说，假设用户查询为“remove cookie”，则去除停词后原始查询为“remove AND cookie”。在进行查询扩展时，假设 remove 的语义相关词是 delete，cookie 的语义相关词是 session，那么查询扩展为“(remove OR delete) AND (cookie OR session)”。

4. 实验评估标准

与 CodeHow 的评估标准一样，该实验采用 Precision@k 及 MRR（Mean Reciprocal Rank）指标作为评估标准。Precision@k 用来衡量返回结果的前 k 个中相关结果的数量，是所有查询的前 k 个返回结果中相关结果所占比重的平均。Precision@k 的计算公式如下：

$$Precision@k = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{|relevant_i|}{k} \tag{5-2}$$

其中$relevant_i$表示第 i 个查询的前 k 个返回结果中相关结果数量。Precision@k 越大说明代码搜索精度越高。本文选取的 k 值为 1，5，10 及 20。多数情况下用户不会关注排在第 20 之后的结果。MRR 用来衡量返回结果中出现的第一个相关结果的排序，是所有查询的返回结果中第一个相关结果排序倒数的平均。MRR 的计算公式如下：

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{5-3}$$

其中$rank_i$表示第 i 个查询的返回结果中第一个相关结果的排序。MRR 越大说明代码搜索精度越高。

返回结果的相关性判断由 3 名软件工程方向研究生共同完成。对一个返回结果，只有当 3 人均认为其与查询相关时才被标记为相关。

5.4.2 实验结果

实验结果具体数据如表 5-10 所示。评估标准实验结果如表 5-11 所示。其中 Initial 表示不做查询扩展的对照组，SWordMap 表示做查询扩展的实验组。

从表 5-10 中数据可以看出，在所有查询中的 18 条查询上 SWordMap 表现优于原始查询，7 条查询上 SWordMap 表现与原始查询相当，9 条查询上 SWordMap 表现比原始查询差。对其中一些查询，SWordMap 能较大提升返回结果精度。如查询“how can I decode HTML characters”，SWordMap 能识别 decode 的语义相关词

表 5-10 开源代码搜索实验结果
Table 5-10 Experiment results of open-source code search

查询任务	前 k 个返回结果中相关结果数量							
	Initial				SWordMap			
	k=1	k=5	k=10	k=20	k=1	k=5	k=10	k=20
copy paste data from clipboard	1	5	9	15	1	5	9	17
open url in html browser	1	2	6	11	1	4	8	14
track mouse hover	1	4	7	12	1	2	5	9
highlight text range in editor	1	4	7	13	1	2	3	7
convert utc time to local time	0	3	6	11	1	4	6	13
converting String to DateTime	1	4	4	8	1	4	5	9
get current date and time	1	3	6	13	1	5	8	15
get file name without extension	1	5	9	13	1	5	9	15
how can I decode HTML characters	1	4	4	4	1	5	9	14
how can I download HTML source	1	2	3	7	0	1	1	5
how do I round a decimal value to 2 decimal places	0	2	4	7	0	3	7	14
how to change RGB color to HSV	1	4	5	8	1	2	6	7
how to convert an IPv4 address into a integer	0	2	3	5	0	4	7	15
how to delete all files and folders in a directory	1	5	9	10	1	5	9	12
how to execute a sql select	0	2	5	7	0	4	8	10
how to generate random int number	1	4	8	17	1	4	8	17
how to get Color from Hexadecimal color code	0	2	4	7	1	2	3	7
how to get temporary folder for current user	0	2	3	5	1	2	3	6
if a folder does not exist create it	1	4	7	15	1	4	7	15
ping a hostname on the network	1	4	8	13	1	4	8	15
Process.start: how to get the output	0	3	8	9	0	3	7	9
sending email through Gmail	1	3	5	6	1	4	7	16
append string to file	1	5	6	13	1	5	6	11
calculate md5 checksum	0	2	5	9	0	3	5	7
how to deserialize XML document	1	4	9	12	0	3	8	10
how to get mac address	1	2	3	11	1	2	2	3
how to play a sound	1	4	8	15	1	4	8	16
how to save image in png format	1	5	9	18	1	5	8	16
read file line by line	1	3	5	10	1	4	5	10
remove cookie	0	2	3	5	0	2	3	5
verify folder exists	0	3	7	12	1	3	8	13
how to reverse a string	1	5	8	16	1	5	8	16
quick sort	1	5	10	18	1	5	10	18
how to split string into words	1	4	9	18	1	4	9	18

表 5-11 开源代码搜索评估标准实验结果
Table 5-11 Experiment results of evaluation metrics of open-source code search

	Initial	SWordMap
Precision@1	0.706	0.767
Precision@5	0.688	0.724
Precision@10	0.624	0.656
Precision@20	0.549	0.594
MRR	0.814	0.855

encode、parse 等等，html 的语义相关词 webpage、css、tag 等等，以及 characters 的语义相关词 chars、symbols 等等，因此能够大幅度提升相关结果的排序。又如查询“how to convert an IPv4 address into a integer”，SWordMap 能识别 ipv4 的语义相关词 ipv6、ip 等等，以及 address 的语义相关词 addr、hostname 等等，因此前 20 个返回结果中相关结果数量有较大提升。对其中一些查询，SWordMap 仅有较小提升或没有提升。如查询“how to reverse a string”、“quick sort”等等，这是因为原始查询中的单词在代码中十分常见，本身即能较好匹配到大量相关结果，查询扩展对最终返回结果的排序影响不大。而对余下查询，SWordMap 表现要差于原始查询。如查询“how to get mac address”，由于 mac 本身具有两种不同语义：（1）苹果电脑的操作系统；（2）网卡物理地址，SWordMap 识别的 mac 的语义为第一种，所识别的语义相关词为 osx、ubuntu、windows 等等，导致搜索结果精度大大下降。这说明两个单词的相关性是受上下文限制的，不同上下文中两个单词的语义相关性也不同。

从表 5-11 中数据可以看出，总的来说 SWordMap 相对原始查询在代码搜索精度上有一定提升。其中，Precision@1 提升了 8.64%，Precision@5 提升了 5.23%，Precision@10 提升了 5.13%，Precision@20 提升了 8.20%，MRR 提升了 5.04%。注意虽然结果有提升，但提升并不显著，相比本地代码搜索实验结果的提升是很小的。这是由于在本地代码搜索中，搜索对象数量很少，仅是一个软件中的所有代码，因此在查询扩展时部分错误的语义相关词对搜索结果不会产生很大影响（大部分错误的语义相关词不会在软件中出现）。而在开源代码搜索中，搜索对象数量巨大，在查询扩展时错误的语义相关词也会在许多代码片段中出现，因而搜索结果会受到较大影响。因此还需要提升 SWordMap 挖掘语义相关词表的精确度，使其能对开源代码搜索带来更大帮助。同时由于单词的语义相关词受上下文限制，即使是对的语义相关词也不一定会带来好的搜索结果。这意味着在做查询扩展时还需要考虑原始查询的上下文。

5.5 本章小结

本章阐述了针对 SWordMap 设计的四个实验及实验结果分析。第一个实验评估 SWordMap 挖掘语义相关词表的精确度，实验结果表明 SWordMap 挖掘语义相关词表精确度较高。第二个实验评估 SWordMap 对关注定位任务效率的提升，实验结果表明 SWordMap 能有效提升关注定位任务效率。第三个实验评估 SWordMap 对本地代码搜索精度的提升，实验结果表明 SWordMap 能有效提升本地代码搜索精度。第四个实验评估 SWordMap 对开源代码搜索精度的提升，实验结果表明 SWordMap 对开源代码搜索精度有一定提升，但效果不显著。总的来说，SWordMap 挖掘语义相关词具有一定应用价值，相比开源代码搜索更适用于本地代码搜索。

第六章 总结与展望

6.1 本文工作内容总结

本文主要贡献为：提出了一种基于 Word Embedding 的软件工程领域语义相关词挖掘方法 SWordMap，并设计了针对代码搜索的查询扩展模型对 SWordMap 挖掘语义相关词在代码搜索上的应用进行了研究。

本文首先分析了现有的软件工程领域语义相关词挖掘方法，并指出了它们的优点和不足，然后针对它们的不足本文提出了基于 Word Embedding 的软件工程领域语义相关词挖掘方法 SWordMap。之后本文对 SWordMap 使用的 Word Embedding 技术及信息检索模型的相关背景知识进行介绍。介绍完相关背景知识后本文对 SWordMap 及针对代码搜索的查询扩展模型的设计及具体实现进行详细阐述。通过对 SWordMap 挖掘语义相关词表的观察可以看出 SWordMap 的有效性。由于本地代码搜索及开源代码搜索各自的特点，本文分别设计了针对本地代码搜索及开源代码搜索的查询扩展模型，并基于搜索引擎 Elasticsearch 做了具体实现。最后本文设计实验对 SWordMap 挖掘语义相关词及其在代码搜索上的应用进行验证评估。为充分评估 SWordMap，本文设计了四个实验，从不同角度评估 SWordMap 的有效性。实验结果表明 SWordMap 挖掘语义相关词精确度较高，能有效提升关注定位任务效率及本地代码搜索精度，对开源代码搜索精度的提升有限。

6.2 未来工作展望

根据 SWordMap 的实验结果，SWordMap 目前还存在一些缺陷。针对这些缺陷 SWordMap 有着下述改进空间：

1. 扩充训练数据

关注定位及本地代码搜索的实验结果均表明 SWordMap 挖掘的语义相关词存在遗漏，说明 SWordMap 仅使用 Stack Overflow 的文档作为训练数据不够充分。在将来的工作中 SWordMap 需考虑更多的训练数据来源，如软件开发者手册、API 文档、开源软件的代码及文档等等。

2. 改进训练模型

目前的 CBOW 训练模型仅能对具有句子格式的软件文档进行训练，对代码的

训练结果较差。因此在将来的工作中可以考虑对 CBOW 模型进行改进，使得其能够对代码这种结构化文本进行训练，进一步提升挖掘语义相关词的精确度。

3. 改进查询扩展模型

目前的查询扩展模型未考虑查询中的上下文，导致在开源代码搜索中的表现不佳。因此可以对查询扩展模型进行改进，在选取扩展词时考虑查询上下文，确保扩展词的确与查询语义上相关。一个简单的想法如直接计算词库中单词与查询中所有单词的向量距离，选取词库中与查询中所有单词距离的均值最近的单词作为扩展词[50]。同时目前的查询扩展模型仅考虑了代码文本信息，未考虑代码结构信息。在将来的工作中可以考虑将代码结构也作为一个代码特性进行搜索[51]。

综上，本文提出的 SWordMap 方法能够有效挖掘软件工程领域语义相关词，能有效提升关注定位任务效率及本地代码搜索精度，但是对开源代码搜索精度的提升有限。SWordMap 仍存在一些提升空间，可以提高其对开源代码搜索的应用价值。

参 考 文 献

[1] Furnas G W, Landauer T K, Gomez L M, et al. The vocabulary problem in human-system communication[J]. Communications of the ACM, 1987,30(11):964-971.

[2] Haiduc S, Bavota G, Marcus A, et al. Automatic query reformulations for text retrieval in software engineering[C]. In proceedings of the 2013 International Conference on Software Engineering, 2013:842-851.

[3] YANG Jinqiu, LIN Tan. Swordnet: Inferring semantically related words from software context[J]. Empirical Software Engineering, 2014,19(6):1856-1886.

[4] Tian Y, Lo D, Lawall J. Sewordsim: software-specific word similarity database[C]. In proceedings of the 36th International Conference on Software Engineering, 2014:568-571.

[5] Sando. Sando code search tool[EB/OL]. [2016-01-05]. http://sandosearch.weebly.com/.

[6] Aragon Consulting Group, Inc. Krugle code search[EB/OL]. [2016-01-05]. http://www.krugle.com/.

[7] Linstead E, Bajracharya S, Ngo T, et al. Sourcerer: mining and searching internet-scale software repositories[J]. Data Mining and Knowledge Discovery, 2009,18(2):300-336.

[8] Merriam-Webster, Incorporated. Merriam-webster english dictionary and thesaurus[EB/OL]. [2016-01-05]. http://www.merriam-webster.com/.

[9] Princeton University. WordNet[EB/OL]. [2016-01-05]. http://wordnet.princeton.edu/.

[10] Sridhara G, Hill E, Pollock L, et al. Identifying word relations in software: A comparative study of semantic similarity tools. In proceedings of the 16th IEEE International Conference on Program Comprehension, 2008:123-132.

[11] Shepherd D, Pollock L, Vijay-Shanker K. Towards supporting on-demand virtual remodularization using program graphs[C]. In proceedings of the 5th international conference on Aspect-oriented software development, 2006:3-14.

[12] Shepherd D, FRY Z P, HILL E, et al. Using natural language program analysis to locate and understand action-oriented concerns[C]. In proceedings of the 6th International Conference on Aspect-oriented Software Development, 2007:212-224.

[13] Hill E. Integrating natural language and program structure information to improve software search and exploration[Dissertation]. University of Delaware, 2010.

[14] Hill E, Pollock L, Vijay-Shanker K. Improving source code search with natural language phrasal representations of method signatures[C]. In proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, 2011:524-527.

[15] YANG Jinqiu, LIN Tan. Inferring semantically related words from software context[C]. In proceedings of the 9th IEEE Working Conference on Mining Software Repositories, 2012:161-170.

[16] Howard M J, Gupta Samir, Pollock L, et al. Automatically mining software-based, semantically-similar words from comment-code mappings[C]. In proceedings of the 10th Working Conference on Mining Software Repositories, 2013:377-386.

[17] Wang S, Lo D, Jiang L. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging[C]. In proceedings of the 28th IEEE International Conference on Software Maintenance, 2012:604-607.

[18] Tian Y, Lo D, Lawall J. Automated construction of a software-specific word similarity database[C]. In proceedings of the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, 2014:44-53.

[19] Ye X, Shen H, Ma X, et al. From word embeddings to document similarities for improved information retrieval in software engineering[C]. In proceedings of the 38th International Conference on Software Engineering, 2016:404-415.

[20] Word embedding[EB/OL]. [2016-10-27]. https://en.wikipedia.org/wiki/Word_embedding.

[21] Turian J, Ratinov L, Bengio Y. Word representations: a simple and general method for semi-supervised learning[C]. In proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, 2010:384-394.

[22] Hinton G E. Learning distributed representations of concepts[C]. In proceedings of the 8th annual conference of the cognitive science society, 1986:1-12.

[23] Clark A, Fox C, Lappin S. The Handbook of Computational Linguistics and Natural Language Processing[M]. Wiley-Blackwell, 2010:74-104.

[24] 王晓龙，关毅. 自然语言处理[M]. 北京：清华大学出版社，2005：47-65.

[25] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model[J]. Journal of Machine Learning Research. 2003, 3:1137-1155.

[26] Mikolov T, Kombrink S, Deoras A, et al. Rnnlm-recurrent neural network language modeling toolkit[C] In proceedings of the 2011 IEEE Workshop on Automatic Speech Recognition & Understanding, 2011:196-201.

[27] Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations

in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

[28] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[C]. In proceddings of the 2013 Neural Information Processing Systems. 2013:3111-3119.

[29] Morin F, Bengio Y. Hierarchical Probabilistic Neural Network Language Model[C]. In proceedings of the international workshop on artificial intelligence and statistics, 2005:246-252.

[30] Information retrieval[EB/OL]. [2016-10-17]. https://en.wikipedia.org/wiki/Word_embedding.

[31] Standard Boolean model[EB/OL]. [2016-10-01]. https://en.wikipedia.org/wiki/Standard_Boolean_model.

[32] Vector space model[EB/OL]. [2016-10-17]. https://en.wikipedia.org/wiki/Vector_space_model.

[33] Extended Boolean model [EB/OL]. [2016-10-19]. https://en.wikipedia.org/wiki/Extended_Boolean_model.

[34] MSDN Blogs[EB/OL]. [2016-01-05]. https://blogs.msdn.microsoft.com/.

[35] Bing Search[EB/OL]. [2016-01-05]. http://global.bing.com/.

[36] Stack Overflow[EB/OL]. [2016-01-05]. http://stackoverflow.com/.

[37] The Porter Stemming Algorithm[EB/OL]. [2016-01-05]. https://tartarus.org/martin/PorterStemmer/.

[38] word2vec[EB/OL]. [2016-01-05]. https://code.google.com/archive/p/word2vec/.

[39] List of English Stop Words[EB/OL]. [2016-01-05]. http://xpo6.com/list-of-english-stop-words/.

[40] Maaten L V D. Accelerating t-sne using tree-based algorithms[J]. The Journal of Machine Learning Research, 2014,15(1):3221-3245.

[41] Ko A J, Myers B A, Coblenz M J, et al. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks[J]. IEEE Transactions on Software Engineering, 2006:32(12):971-987.

[42] Lv F, Zhang H, Lou G, et al. Codehow: Effective code search based on API understanding and extended boolean model[C]. In proceedings of the 30th International Conference on Automated Software Engineering, 2015: 260-270.

[43] Elastic Stack and Product Documentation[EB/OL]. [2016-10-05]. https://www.elastic.co/guide/index.html.

[44] Elasticsearch 权威指南[EB/OL]. [2016-10-05]. http://www.learnes.net/.

[45] JDT[EB/OL]. [2016-10-05]. https://projects.eclipse.org/projects/eclipse.jdt.

[46] Shepherd D. Action-oriented concerns[EB/OL]. [2016-01-05].

https://www.eecis.udel.edu/~gibson/context/action_oriented_concerns.txt.

[47] Shepherd D, Damevski K, Ropski B, et al. Sando: an extensible local code search framework[C]. In proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, 2012:1-2.

[48] GitHub[EB/OL]. [2016-01-05]. https://github.com/.

[49] Bajracharya S K, Ossher J, Lopes C V. Leveraging usage similarity for effective retrieval of examples in code repositories[C]. In proceedings of the 8th ACM SIGSOFT international symposium on Foundations of software engineering, 2010:157-166.

[50] Mihalcea R, Corley C, Strapparava C. Corpus-based and knowledge-based measures of text semantic similarity[C]. In proceedings of the 21st national conference on Artificial intelligence, 2006:775-780.

[51] Kim J, Lee S, Hwang S, et al. Towards an intelligent code search engine[C]. In proceedings of the 24th AAAI Conference on Artificial Intelligence, 2010:1358-1363.

攻读硕士学位期间已发表或录用的论文

[1] 第一作者. Android 应用中 Fragment 组件的污点分析[J]. 计算机与现代化，2017 年 8 期（已录用）