# 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。与此同时自然语言领域的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，采用CBOW模型对IT技术问答网站Stack Overflow的文档进行训练，得到了软件工程领域单词的向量表示，以及它们的语义相关词表。为验证SWordMap的有效性，本文从两方面进行衡量：本地代码搜索以及开源代码搜索。本地代码搜索设计了与前人工作SWordNet的对比实验，实验表明采用SWordMap挖掘语义相关词表进行查询扩展能有效提高本地代码搜索精度。一般代码搜索设计了不同的查询扩展模型，并检验了模型的有效性。实验结果表明采用SWordMap进行查询扩展对提高一般代码搜索有一定提升。

关键词：代码搜索；查询扩展；语义相关词；SWordMap

## 绪论

### 1.1 研究背景

现如今，随着软件规模的不断扩大，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，网络上已经有了海量开源代码，涵盖了方方面面，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。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。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，SEWordSim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。

#### 代码搜索

现有的代码搜索工具如Sando[8], Krugle[9]以及Sourcerer[10]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。

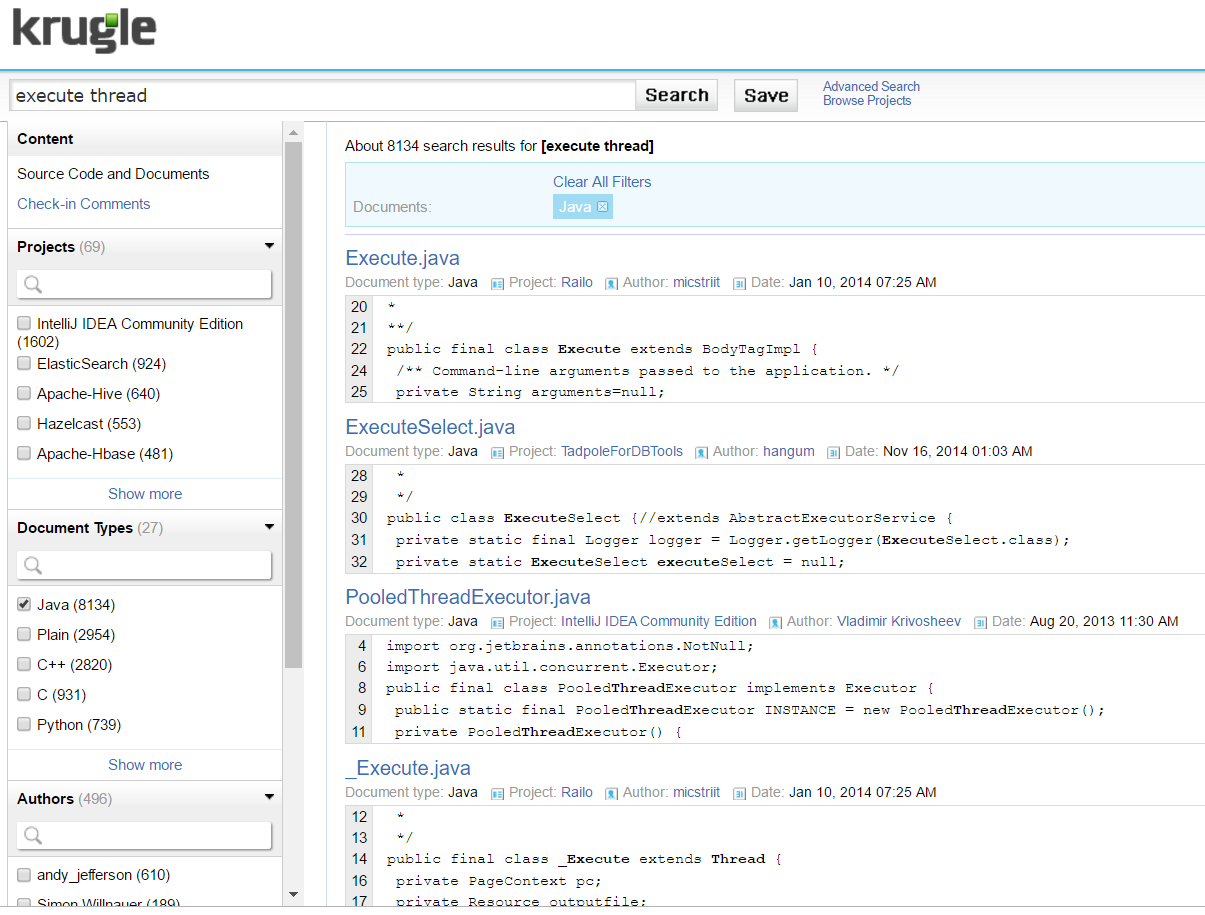


图1 Krugle对“execute thread”查询的搜索结果

#### 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英语词典[11]、WordNet[12]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在英语词典The Merriam-Webster Dictionary and Thesaurus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[13]。

### 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。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[3]。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对的准确性，推出了语义相关词识别精度更高的SWUM[4]。Yang等人则在Shepherd研究的基础上进行一定扩展推出了SWordNet，一个通过对软件中代码及代码注释的文本相似度比较挖掘语义相关词的工具[5-6]。SWordNet去掉了SWUM的自然语言约束，直接从函数签名及代码注释中推断语义相关词。例如linux内核的代码注释中存在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代码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，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。Tian等人推出了SEWordSim，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[7]。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。

### 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。由于语义相关词的精确度难以达到100%，而采用错误的语义相关词进行查询扩展会给搜索带来噪音，降低代码搜索精度。因此需要设计合理的查询扩展模型，在语义相关词并非完全正确的前提下保证代码搜索精度的提升。由于本地代码搜索与开源代码搜索之间存在差异，本文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。本文的具体研究目的为：

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，使用本文挖掘语义相关词作为该模型的查询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，使用本文挖掘语义相关词作为该模型的查询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，以有效提升代码搜索精度。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. 研究现有查询扩展模型，针对本地代码搜索以及开源代码搜索分别设计了不同的查询扩展模型，并基于Elasticsearch进行了具体实现。
3. 设计实验验证本文所挖掘语义相关词的精确度，以及本文设计的查询扩展模型对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几个部分：

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Word Embedding相关知识，以及查询扩展相关知识。

第三章：介绍SWordMap，以及分别针对本地代码搜索和开源代码搜索设计的查询扩展模型。

第四章：介绍本文设计的实验。

第五章：针对实验结果对SWordMap进行总结和展望。

最后是论文的参考文献、附录及谢辞。

## 背景技术介绍

本章介绍SWordMap所使用的Word Embedding技术，以及现有的查询扩展模型。

### 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 [https://en.wikipedia.org/wiki/Word\_embedding]。

#### 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，以下做简要介绍。

#### 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*，向量的分量只有一个1，其他全为0，1的位置为对应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。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。

#### Distributed Representation

Distributed Representation也叫做Word Embedding，最早由Hinton在1986年提出[Learning distributed representations of concepts, paper]。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。

#### 语言模型

许多不同类型的语言模型都能用来学习词向量，包括著名的LSA（Latent Semantic Analysis）以及LDA（Latent Dirichlet Allocation）。前人的研究工作表明，基于神经网络模型训练的Distributed Representation词向量表现要优于LSA，而LDA在大规模数据集上计算复杂度过高。因此本文主要介绍可用来学习Distributed Representation词向量的语言模型。

#### 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。假设表示由*n*个单词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[Statistical\_Language\_Modeling.pdf]：

由于条件概率的参数空间太大，语言模型强制将其转换为一个大概等价的形式[1]：

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[计算机自然语言处理，book]：

1. 上下文无关模型

该模型定义，仅考虑当前单词本身出现的概率。不需要任何训练过程，根据训练语料库中词频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。

1. N-gram模型

该模型定义，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时也被称为bigram语言模型。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。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。

1. N-pos模型

该模型定义，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）。代表单词的词性。

1. 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“前一个单词是吗”或“前一个单词词性是吗”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空开销太大。基于决策树的语言模型是一种更加通用的语言模型。

1. 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。

1. 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。

#### 神经网络语言模型

#### 前馈神经网络模型

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[a neural probabilistic language model, 2003]。如图2所示为NNLM的模型架构。

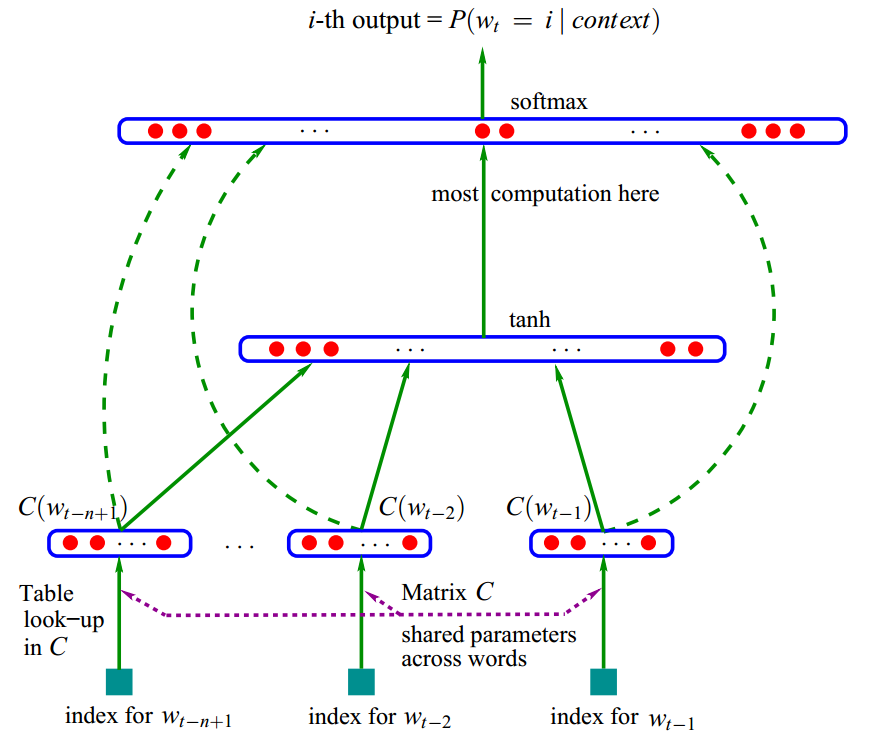


图2 NNLM模型架构

NNLM的概率分布函数为：

其中唯一约束为对任意，满足，以及。在该模型中，它被分解为两部分：

1. 一个映射矩阵，其中每一行表示一个单词的特征向量，每一列为一个单词特性。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映射到词典中下一个单词出现的条件概率分布。*g*的输出是一个维向量，其中第i个元素对应估算概率。函数*g*如下：

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*：

其中为正则化项。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：

其中是每个单词非正则化的log概率值，通过下式计算得到：

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（大小为的矩阵），*H*为投影层到隐藏层的参数（大小为的矩阵）。

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。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）[Recurrent neural network based language model, In: Proceedings of Interspeech]。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。

#### CBOW模型和Skip-gram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）。如图3所示为它们的模型架构。

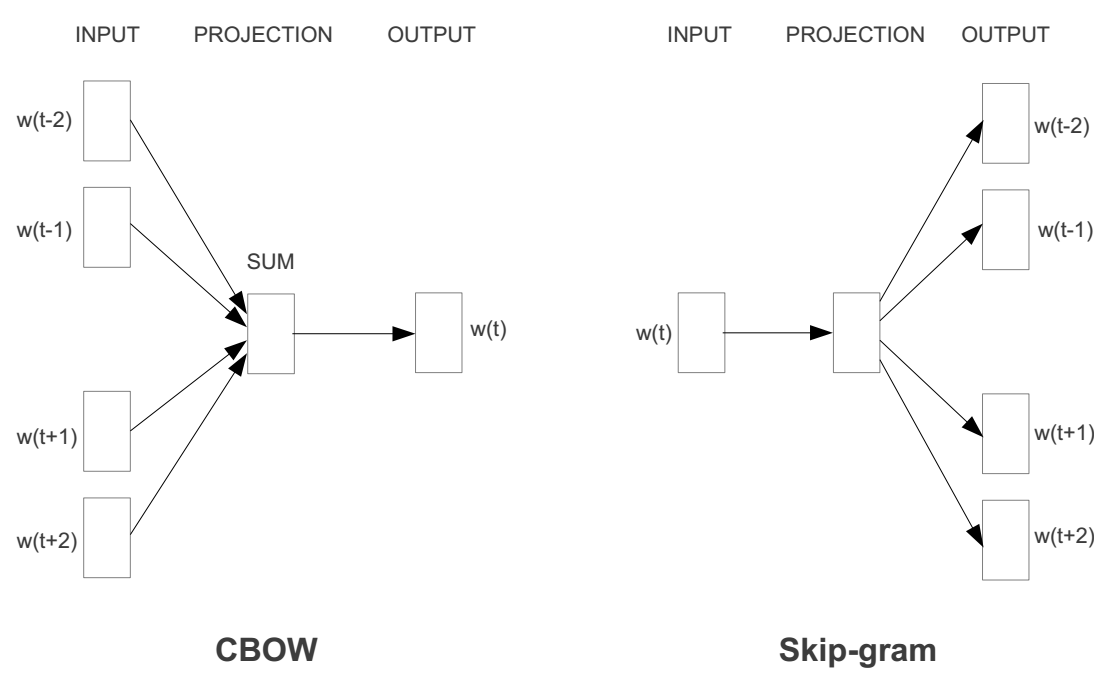


图3 CBOW和Skip-gram的模型架构

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。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*：

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，叶子节点共个，非叶子节点共个。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[Hierarchical probabilistic neural network language model]，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

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。[https://en.wikipedia.org/wiki/Information\_retrieval]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。

#### 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[https://en.wikipedia.org/wiki/Standard\_Boolean\_model]。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。

#### 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[https://en.wikipedia.org/wiki/Vector\_space\_model]。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:

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：

其中为单词*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，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：

#### 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[https://en.wikipedia.org/wiki/Extended\_Boolean\_model]。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，其中单词权重的常用计算公式为：

对一个或的查询，假设对应的在文档*d*中的单词权重为和，则二者相关度的计算公式为：

相应的对一个与的查询，二者相关度的计算公式为：

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：

对应的相关度计算公式为：

举例：对一个查询，假设查询中单词权重均为1，其与文档的相关度计算公式为：

#### 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进行检索，语言模型则依靠大规模数据训练进行统计建模用于检索。限于篇幅，本文不再对它们一一介绍。

### 本章小结

本章对SWordMap所使用的Word Embedding技术及信息检索模型等本文相关背景知识做了简要介绍，为下文的阐述做好铺垫。

## 第三章 SWordMap

本章对本文提出的基于Word Embedding的语义相关词挖掘技术SWordMap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）构建语料库；（2）模型训练；（3）生成语义相关词表。下文对这三个步骤作详细阐述。

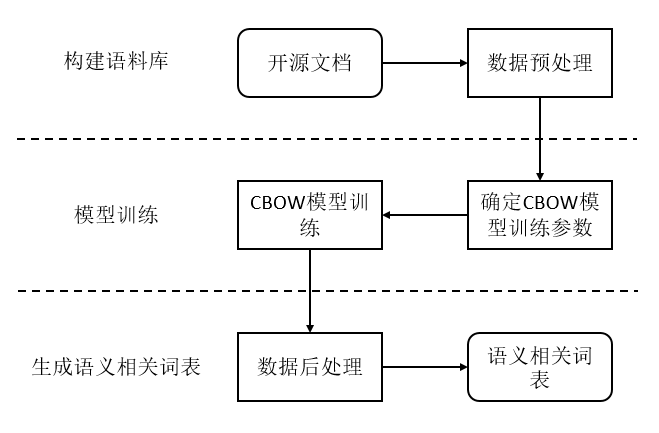


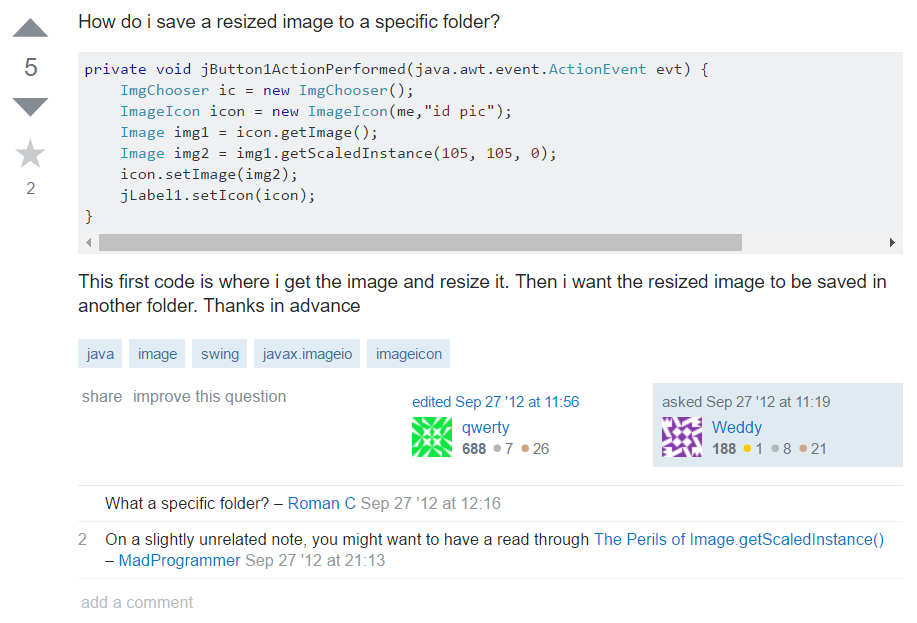
图3-1 SWordMap整体流程图

### 3.2 语料库构建

Word Embedding技术学习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，以句子作为单词的上下文环境，因此语料库中文档需要满足能以句子为单元进行划分。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[https://blogs.msdn.microsoft.com/]，Bing搜索日志[http://cn.bing.com/]以及IT技术问答网站Stack Overflow[http://stackoverflow.com/]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，问题需备注标签等等），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4 Stack Overflow网站页面

如图4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。

代码属于结构化文本，本身不具备类似句子的特征，而且一般会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。前人的研究中曾将代码中的函数签名用作单词上下文[SWUM, SWordNet]，然而一般函数命名遵循简洁明了的原则，所使用的单词数量很少，能提供的上下文信息十分有限。因此本文在选取训练数据时仅考虑语句文本。

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。为了对照，本文也对所有Stack Overflow问题进行了训练。最终本文共收集了708473个带有java标签的Stack Overflow问题作为训练语料库。

#### 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大多单词数不超过3，因此本文对单词数小于等于3的短句子进行过滤。
4. 原始数据的句子中包含函数名及API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会错误地将adding的词干提取为ad[https://tartarus.org/martin/PorterStemmer/]，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5,913,765条语句，共73,396,324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[https://code.google.com/archive/p/word2vec/]。CBOW模型在开始训练前需要设置一些训练参数，见表3-1。

|  |  |
| --- | --- |
| 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 |

表3-1 CBOW模型训练参数

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100,000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。

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22分钟。训练得到共30,992个单词的向量表示。

|  |  |
| --- | --- |
| 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 |

表3-2 训练机器配置

### 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[http://xpo6.com/list-of-english-stop-words/]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。

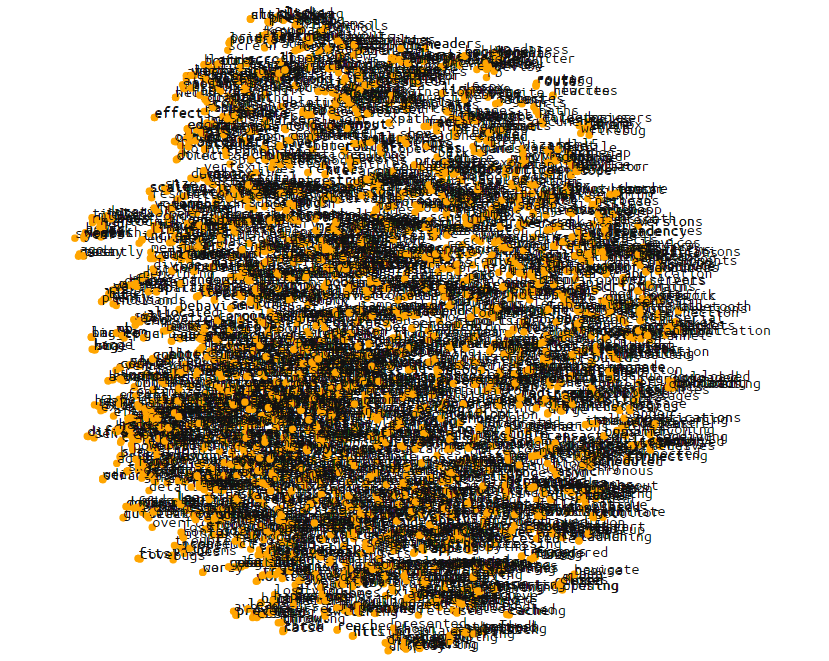
经过上述处理后得到共19,332个单词的词向量。本文采用t-SNE算法[Accelerating t-sne using tree-based algorithms]绘制了从所得结果中随机采样3000个词向量的2D投影图，如图3-1所示。图3-2为局部放大之后的词向量的2D投影图。从图3-2中可以看出，语义相似的单词在向量空间中距离相近，如图中心的clean、build以及linking，这三个单词均与编译过程相关。

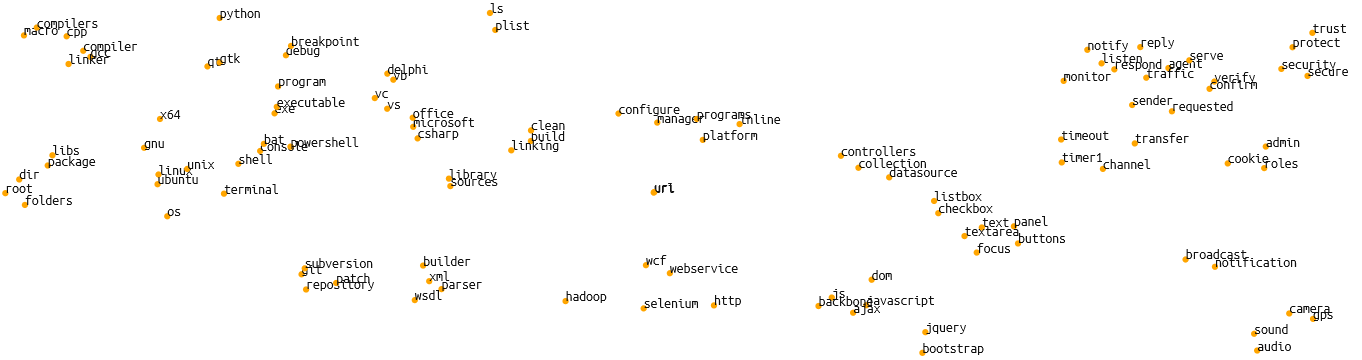
图3-1 词向量的2D投影图

图3-2 局部放大的词向量的2D投影图

#### 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19,332个单词的语义相关词表，共325,583对不重复的语义相关词。

|  |  |
| --- | --- |
| 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… |

表3-3 语义相关词表示例

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。3.1节中提到的jdbc也被成功识别为db的语义相关词。

### 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，并且对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。本章对本文提出的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。本文根据本地代码搜索和开源代码搜索的特点分别设计了两种不同的查询扩展模型。下文对这两种查询扩展模型作详细阐述。

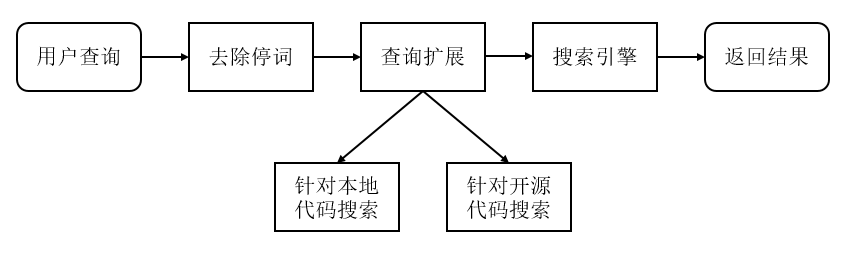


图4-1 代码搜索流程图

### 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[An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks]。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，搜索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。而如果搜索返回结果中有遗漏，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。[Swordnet: Inferring semantically related words from software context]
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个单词的查询定义如下：

根据前人研究，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了实现，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 [Using Natural Language Program Analysis to Locate and Understand Action-Oriented Concerns，Improving Source Code Search with Natural Language Phrasal Representations of Method Signatures]。因此对一个代码片段，本文仅考虑它的一个特性：函数名及其所属类名。因此原始查询可以表示为：

其中：

假设单词有个语义相关词，其列表为，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为：

其中：

示例：假设一个用户查询为“How to execute a thread?”，在去除停词后查询。假设execute的相关词列表为，thread的相关词列表为，那么经过查询扩展后的查询为：

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，搜索返回结果集合也很大。在这种情况下，对搜索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，因此搜索返回结果的排序直接影响本次搜索的有效性。在搜索返回结果中越相关的代码应该越靠前。
3. 对搜索的时效性要求较高。由于开源代码搜索的代码库十分庞大，搜索耗时将成为影响代码搜索引擎实用性的一个关键因素。如果一次搜索耗时太久，即使搜索返回结果精度很高，对用户的实用意义也不大。

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个单词的查询定义如下：

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[Using natural language program analysis to locate and understand action-oriented concerns, Integrating natural language and program structure information to improve software search and exploration，SWordNet， CodeHow]，常把函数作为代码搜索的最小单位。本文同样以函数作为代码搜索的最小单位，并考虑一个函数的如下特性：

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. ：所属类名，与该函数具体功能相关。
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，使其无法较好表征函数功能。
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。

因此原始查询可以表示为：

其中：

假设单词有个语义相关词，其列表为，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为：

其中：

其中：

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

由2.2.3节可知查询与文档的相似度计算公式。其中关键在于项在查询中的权重以及在文档中的权重的计算。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，本文假设原始查询中的各项平权，因此在计算时项的权重为。

对于，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相似度。假设扩展词的相似度为，则在计算时各项权重为：

对于和，根据上述4个特性的特点可知，函数名最能体现函数的具体内容，所属类名次之，函数体噪点较多，而函数注释则具有不确定性，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，本文配置计算及时各项权重为：

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。

对于和中各项在文档中的权重采用tf-idf公式直接计算：

### 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。由于开源代码搜索对时效性要求较高，在实现时需要着重考虑搜索耗时。经过对现有搜索引擎的调查研究，本文最终决定基于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[https://www.elastic.co/guide/index.html]。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[http://www.learnes.net/]。

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。

1. 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。

1. 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。

1. 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相同字段（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。

1. 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。

{

"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",

}

代码4-1 Elasticsearch文档格式示例

1. 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搜索引擎的建立包括如下步骤：

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。

1. 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。

PUT twitter/tweet/1

{

"user" : "kimchy",

"post\_date" : "2009-11-15T14:12:12",

"message" : "trying out Elasticsearch"

}

代码4-2 RESTful API示例

#### 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。

4.4.2.1 索引创建

4.4.2.1.1 索引及类型配置

本文代码搜索的文档是函数，除此之外没有其他类型数据，因此仅需要一个索引及一个类型。在开始创建索引之前需要对索引及类型进行配置。如代码4-1所示为索引的配置，其中配置了分词器、停词、碎片及副本数量。

{

"settings": {

"index": {

"analysis": {

"filter": {

"custom\_word\_delimiter": {

"split\_on\_numerics": "false",

"generate\_word\_parts": "true",

"type": "word\_delimiter",

"stem\_english\_possessive": "true",

"generate\_number\_parts": "true"

},

"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",

"version": {

"created": "1030299"

},

"uuid": "yKuZZBclSAKWk13kGQzdhg",

"number\_of\_replicas": "1"

}

}

}

代码4-1 索引配置文件

如代码4-2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，参与检索过程；后5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代码片段。

{

"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\_stopwords"

},

"filepath": {

"type": "string"

},

"projectLink": {

"type": "string"

},

"projectName": {

"type": "string"

},

"license": {

"type": "string"

},

"repositoryOrigin": {

"type": "string"

},

}

}

}

代码4-2 类型配置文件

4.4.2.1.2 开源代码索引创建

在上一步完成集群中的索引及类型配置后，就可以开始创建开源代码索引。如图4-2所示为索引创建的整体流程。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）[https://projects.eclipse.org/projects/eclipse.jdt]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进行了如下过滤处理：

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。
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是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。

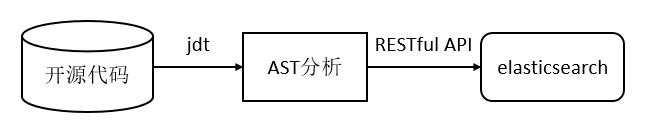


图4-2 索引创建流程图

* + - 1. 索引搜索

索引搜索即采用既定检索模型对所创建索引进行搜索、计算相关度并排序的过程。Elasticsearch默认不支持扩展布尔模型，需要自己实现。幸运的是Elasticsearch提供了十分友好的扩展插件模式，因此只需要实现扩展布尔模型的插件，并集成到Elasticsearch中供代码搜索使用即可。Elasticsearch对底层的分布式并行计算进行了封装，本文仅需要实现基于扩展布尔模型的相关度打分机制。由于插件的具体实现较复杂，这里不再赘述。

### 4.4 本章小结

本章阐述了本文提出的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各自的特点，本文分别设计了两种查询扩展模型，并且基于Elasticsearch做了具体实现。下文将设计实验对SWordMap及所实现的查询扩展模型进行测试，并对实验结果进行分析说明。

## 第五章 实验与评估

本章对SWordMap的实验进行设计和评估。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评估；（2）SWordMap对关注定位任务效率的提升评估；（3）SWordMap对本地代码搜索精度的提升评估；（4）SWordMap对开源代码搜索精度的提升评估。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

该实验主要目标为评估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

5.1.1.1 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1740个不重复的语义相关词对作为样本数据。

5.1.1.2 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。

5.1.1.3 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，因此召回率无法衡量。

#### 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77.7%，说明SWordMap挖掘语义相关词词表的精确度平均在77.7%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[15]。与SWordNet挖掘语义相关词的平均精确度56.9%相比提升了约36.6%。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74.8%，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。

|  |  |
| --- | --- |
| 语义相关词对总数 | 1740 |
| 正确语义相关词对 | 1352 |
| 精确度（%） | 77.7 |
| 不在英语词典或WordNet中的正确结果数量 | 1011 |
| 不在英语词典或WordNet中的正确结果比例（%） | 74.8 |

表5-1 语义相关词对正确性验证结果

### 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

5.2.1.1. 实验数据

关注定位的对象为表5-2所示的4个java软件。测试数据集如表5-3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[18]。其中关注定位搜索的最小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。

|  |  |  |  |
| --- | --- | --- | --- |
| 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5-2 关注定位测试软件

|  |  |
| --- | --- |
| 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 |

表5-3 关注定位测试数据

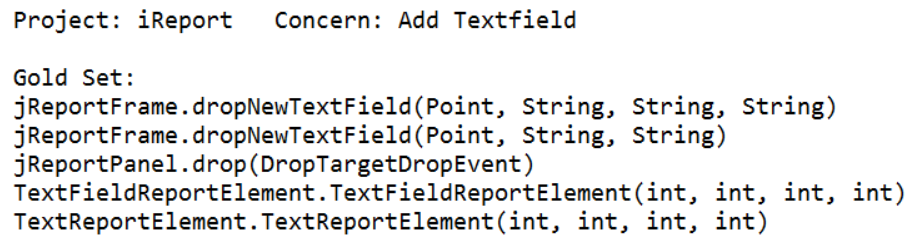


图5-1 关注定位任务Add Textfield

5.2.1.2 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.\*”。

5.2.1.3 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：

#### 5.2.2 实验结果

实验结果如表5-4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），SWM表示本文工作SWordMap。SWordNet的实验结果直接取自Yang的论文[6]。

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）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[3]。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[6]。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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** |

表5-4 关注定位实验结果

### 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[Sando]。

#### 5.3.1 实验设定

5.3.1.1 实验数据

本地代码搜索的对象与关注定位实验相同，为表5-2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5-5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在代码结构上与查询相关的函数进行检索。

|  |  |
| --- | --- |
| 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 |

表5-5 本地代码搜索测试数据

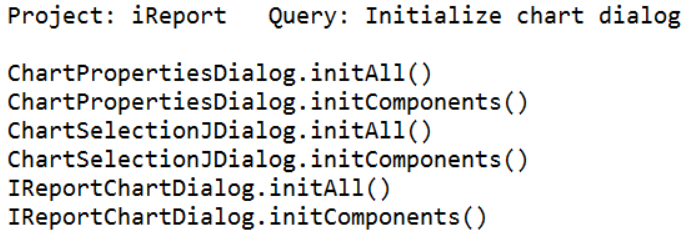


图5-2 本地代码搜索任务Initialize Chart Dialog

5.3.1.2 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exporter，excel的语义相关词是csv，那么查询扩展为“(export OR exporter) AND (excel OR csv)”。

5.3.1.3 实验评估标准

与5.2.1.3节相同，采用精确度及召回率作为评估标准。

#### 5.3.2 实验结果

实验结果如表5-6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。

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。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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** |

表5-6 本地代码搜索实验结果

#### 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

5.4.1.1 实验环境

由于开源代码搜索对时效性要求较高，实验环境会影响到最终的实验结果。本文采用的实验环境为两台服务器，配置如表5-7所示。

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

表5-7 实验环境配置

5.4.1.2 实验数据

开源代码搜索的对象为开源社区Github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22GB，包含23333333行代码，23333个函数及23333个java文件。

测试采用的查询与CodeHow相同，为表5-8中的34条查询。其中，第1-4条取自前人研究工作[Leveraging usage similarity for effective retrieval of examples in code repositories]，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。

|  |  |
| --- | --- |
| 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 |

表5-8 开源代码搜索测试数据

5.4.1.3 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“export to excel”，则去除停词后原始查询为“remove 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)”。

5.4.1.4 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：

其中表示第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：

其中表示第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。

5.4.2 实验结果

实验结果如表5-9所示。