代码与文档间关联关系的提取方法研究和改进

赖冠辉,王晓博,刘 超
(北京航空航天大学计算机学院,北京 100191)

摘 要: 在潜在语义模型的基础上融入了软件文档和程序代码的特点,提出了基于类继承关系的代码聚类、代码特征项分类加权、引入相似度词典以及基于文档类型的分类搜索这四种改进策略.实验结果表明,四种策略可以在保持查全率不变的情况下提高查准率 15 %左右.表明在提取代码与文档间可跟踪性链时,考虑它们的固有特点,将有助于提高检索系统的查全率和查准率.

关键词: 信息检索; 可跟踪性链; 程序理解; 逆向工程
中图分类号: TP311.5 **文献标识码**: A **文章编号**: 0372-2112 (2009) 4A-022-09

Analysis and Improvement on Retrieval Methods for Traceability Links between Source Code and Documentation

LAI Guan-hui, WANG Xiao-bo, LIU Chao
(School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

Abstract: Software documentation is usually expressed in natural languages and free text, in which it captures large useful information. Establishing traceability links between documentation and source code can be helpful in Software Engineering Management. Currently, the recovery of traceability links is mostly based on information retrieval techniques, e.g., probabilistic model, vector space model and Latent Semantic Indexing (LSI). But previous work only treats documentation and source code as plain text files without considering the features with respect to Software Engineering. Four enhancing strategies are proposed to improve the traditional LSI method based on the features of software documentation and source code, namely, source code clustering, identifiers classifying, similarity thesaurus and hierarchical structure enhancement. Experimental results show that the four enhancement strategies can increase the precision by about 15 %. So, the special characteristics of documentation and source code should be considered carefully during the recovering traceability links between them.

Key words: information retrieval (IR); traceability recovery; program comprehension; reverse engineering

1 引言

长期以来,软件可跟踪性的研究一直受到广泛的关注,而现有软件工程管理系统的主要缺点之一是缺乏自动或半自动地构建并维护在软件开发各阶段生成的各类文档和程序代码之间的可跟踪性链(Traceability Links)的有效技术手段[1~4].

软件文档大多采用自然语言来编写,包括需求文档、设计文档、用户手册、测试文档、维护文档、系统日志等.这些文档中蕴含着丰富的专业领域知识,并与程序代码之间存在着不同程度的关联.因此,发现和维护代码与文档间的可跟踪性链,对程序理解、软件维护、需求跟踪、变更分析和软件复用等许多软件工程活动都能提供很大的帮助[2].

这种文档和代码间的可跟踪性链表示软件的各类文档和源代码之间的各种关联关系的集合.例如在需求文档中,一项需求与其实现代码间应存在关联性,一个类的说明文档与该类之间存在关联性等.关联性是文档与代码间存在的一种性质,通常表示文档中定义的某个概念或术语与某段代码之间存在关联,比如实现关系、直接或者间接的依赖关系.可跟踪性链是这种关联性的一种表示方法,本文也称之为关联关系.

虽然维护代码和文档间的关联关系十分重要,但是要想提取出这种关联关系却存在诸多困难.一方面,由于文档是使用自然语言编写的,因此对文档难以实现精确的语法和语义分析.另一方面,代码和文档间的关联关系大多是隐性的,即在文档和程序中没有明确的关联性标识,主要原因是文档和代码分别处于不同的抽象层面,以不同的方式来描述所要解决的问题及其求解方法.

本文采用信息检索的方法来提取代码与文档间的

收稿日期:2008-10-22;修回日期:2008-12-10
基金项目:国家 863 项目(No.2006AA01Z176);国家自然科学基金(No.90718018)

这种关联关系,并且融入了代码和文档间适用于提取关联关系的特点,这里称之为基本关联要素特点,目的在于提高恢复关联性的精度. 关联关系的提取方法是基于如下假设:首先,在程序代码中,包名、类名、属性名和方法名等标识符都使用了有意义的名称,而且与需求、设计等文档中出现的概念有相同的语义. 测试中从 Linux 代码中随机抽取了 50 个文件和 50 个商业业务系统的代码文件,发现这些程序中几乎所有类、属性和方法的标识符都具有明确的语义[5]. 其次,程序文件中有必要的注释,且所使用的术语与文档中使用的术语一致. 显然,一个软件满足这两点要求的程度不同,将会反映在关联程度上.

2　相关工作

近年来,人们主要采用两种方法对文档与代码间可追踪性链的建立开展了深入的研究. 一种方法是基于特定的开发方法和支持工具,例如,商业工具 Rational、DOORS、原型工具 TOOR 等[3],采用规范的方法进行元素的命名、模型的构建和转换,以及代码的生成,其优点是在各级文档和代码的转换与生成过程中便自动或者手工地标注了彼此之间的主要关联关系;缺点是导致开发工作量增加,且变更困难. 另一种方法是采用信息检索的方法,例如,Antonial 和 Marcus 等人采用 IR 模型中的概率模型、向量空间模型和潜在语义索引技术进行检索[2,3],主要是把提取代码和文档间的关联关系看成是以代码为查询条件、以文档作为检索文献库的一个 IR 处理过程.

2.1　概率模型(Probabilistic Model)

概率模型用于提取文档和代码间关联关系时,使用代码文件中的特征项作为查询词来检索文档库,所谓的特征项是指关于文本元数据信息,可以由字、词或短语组成. 那么,代码和文档之间的相似度可以按照条件概率解析为:当存在查询时,同时存在文档 i($i = 1, 2 \cdots, n$, n 为文档总数)的概率. 其相似度计算公式是:

$$Similarity(Q, D_i) = P(D_i|Q) = \frac{P(Q|D_i)\,P(D_i)}{P(Q)} \quad (1)$$

其中 D_i 表示第 i 个文档,Q 表示代码,也就是说代码 Q 与文档 D_i 间的相似度等于 Q 出现时,D_i 也同时出现的概率.

通过上述模型,可计算得到每个代码文件与文档集相似度从高到低的列表,然后根据定义的相似度阈值对该列表进行筛选,最后得到代码和文档的关联矩阵.

该模型的优点是简单易于实现、物理含义明确,缺点是无法解决同义词和缩写词的情况,而且在对文档进行预处理的时候需要生成词根,还假定了检索词间是相互独立的.

2.2　向量空间模型(Vector Space Model, VSM)

向量空间模型的主要思想是使用多维向量表示文档和查询,通过空间上的相似性解决语义上的相似性. 从理论上讲,和查询向量越接近、向量间的夹角越小的文档向量所代表的文档和该查询的相似度越高,越接近查询要求[9].

向量空间模型用于提取文档和代码间关联关系时,把代码和文档中的每个特征项看作一个向量,代表空间中的一维,由这些特征项集合可以定义一个向量空间 V. 那么任何一个文档都可以表示为 V 中的一个向量,称为文本向量(Document Vector),例如第 j 个文档可以表示为 $D_j = (d_{1,j}, d_{2,j}, \cdots, d_{m,j})^T$,其中 $d_{i,j}$ 表示特征项 w_i 在文本 D_j 中的权重,m 表示向量空间的维数,也就是不同特征项的个数;同样,代码 Q 也可以表示为 $Q = (q_1, q_2, \cdots q_m)^T$. 那么所有文档和代码的文本向量就可以构造出一个矩阵 D,称为检索项-文本矩阵,矩阵中每一列都是一个文本向量. 在 D 中,使用文本检索中最常用的余弦公式计算文档和代码间的相似度[1],如式(2)所示:

$$Similarity(Q, D_j) = \frac{\sum_{i=1}^{m} d_{i,j}\, q_i}{\sqrt{\sum_{i=1}^{m} d_{i,j}^2 \sum_{i=1}^{m} q_i^2}} \quad (2)$$

该模型的缺点是高维稀疏矩阵不仅带来存储空间上的浪费、影响效率,还会带有很大的噪声. 但向量空间模型实际上是一种通用的文档信息表示方法,任何一种检索模型都可以利用它的概念来表示文档,利用它的方法辅助检索. 而且该模型易于对向量进行修改,从而实现查询反馈技术[6].

Antoniol 等人首次将上述两种 IR 技术用到提取代码与文档间关联关系中[2,7],其实验结果表明,两种模型都适用于提取代码与文档间的关联关系,但概率模型的效果更好[2].

2.3　潜在语义索引模型(Latent Semantic Indexing, LSI)

LSI 模型是基于向量空间模型的一种新型信息检索代数模型. 其基本思想是假设文本中的词与词之间存在某种联系,即存在某种潜在的语义结构,所谓语义结构是指自然语句中存在的语义范畴和语义关系所形成的抽象的语义格式[8].

LSI 模型首先利用了向量空间模型来表示文档,然后使用奇异值分解(Singular Value Decomposition, SVD)的方法对该检索项-文本矩阵进行降维,获得一个新的低维语义空间[6],称为 LSI 子空间. 然后在 LSI 子空间上使用式(2)计算文档和代码的相似度. 最后根据设置的相似度阈值对结果进行筛选,从而得到文档和代码间的

关联关系.

该模型的优点是低维空间过滤了大量噪声,而且节省空间和提高效率. 另外,该模型是基于潜在语义上的检索,不依赖于检索项的匹配来计算相似度,而是通过分析文本之间隐含的"语义结构"来确定关联关系.

Marcus 等人[3]针对查全率和查准率低的问题,提出了采用 LSI 模型的方法计算相似度. 最后采用与文献[2]完全相同的实验数据集,从实验结果上看,LSI 的查全率与查准率均高于概率模型和空间向量模型.

最后,从文献[2,3]的实验结果上看,虽然上述三种模型都可以用于提取代码与文档间的关联关系,但普遍存在查准率低的情况. 其根本原因在于,使用常规的 IR 模型简单地把代码和文档当作一般的文本进行处理,没有考虑软件工程中代码和文档中的固有特点. 因此,本文的主要工作在于两个方面:第一、使用 LSI 技术支持关联关系的提取;第二、结合软件文档和程序代码间基本关联要素的特点来提高查准率,包括:

(1) 采用相似度词典的方法来解决同义词和缩写词的问题.

(2) 使用基于类继承关系的代码聚类和代码特征项分类加权策略来提高模型的检索精度.

(3) 基于文档类型的分类搜索,引入了迭代处理的过程以借助"较低层次"的文档(如详细设计文档),来逐步建立高层文档(如需求文档)与代码间的关联性.

3 关联关系提取的基本流程

从第 2 节的介绍中不难看出,LSI 模型在揭示文档和代码间的关联关系时,大体上分为 3 大部分:预处理模块、LSI 执行模块和结果处理模块. 如图 1 所示.

3.1 预处理模块

LSI 模型的核心思想是使用文本空间的相似性来表示语义上的相似性,所以预处理模块主要是分别构造代码和文档的文本向量表示形式.

对于代码文件,以类为单位,首先是提取标识符,包括类名、函数名、属性名和注释中的相关词汇等. 接下来将进行标识符分割,例如将函数名标识符 list.polygon 分割为 list,polygon. 最后进行大小写转换、去掉停用词和代码中的关键字等操作,形成一个代码段.

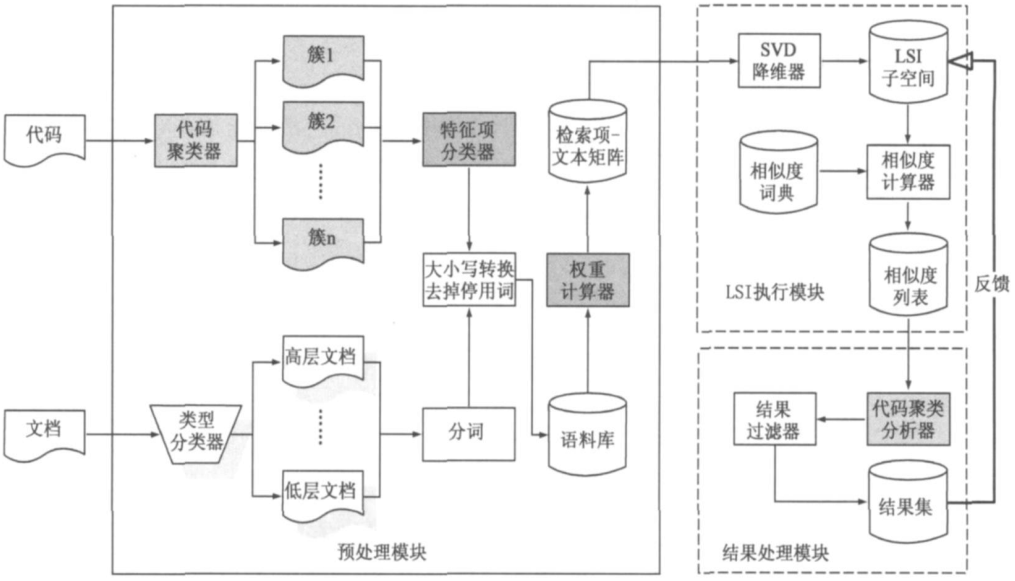

图1 改进后的LSI模型处理流程

对文档的预处理,首先是将 Word、HTML、PDF 等各种格式的文件转换为文本格式,为了达到较好的效果,还要将文本进行分割,一般按照段/节进行分割,因为文档中通常是按照段/节进行组织的,即一段/节通常是表述一个完整内容的最小单位. 然后进行分词、去掉停用词、大小写转换等操作,形成一个文档段.

接着,按照 2.2 节中的方法,将上述提取出来的特征项组成向量空间 V,然后将代码段和文档段表示为文本向量的形式,分别称为代码向量和文档向量. 在计算词的权重时,这里采用了人们常用的 $tf-idf$[8]方法计算.

最后由所有文档向量和代码向量组成一个 $m \times n$ 的检索项-文本矩阵 D,如式(3)所示,其中 n 表示代码段和文档段的总数量. 将 D 作为预处理模块的输出,交给 LSI 执行模块进行下一步的处理.

$$D = (D_1 D_2 \cdots D_n) = \begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ d_{m,1} & d_{m,2} & \cdots & d_{m,n} \end{bmatrix} \quad (3)$$

3.2 LSI 执行模块

LSI 模型区别于向量空间模型主要在于对检索项-文本矩阵进行降维操作,使用新的低维空间表示原文档和代码.LSI 执行模块的主要功能就是进行降维,并且在新的低维空间中计算代码和文档的相似度.

首先,LSI 模型对检索项-文本矩阵 D 进行奇异值分解,计算 D 的 k 阶近似矩阵 D_k($k \ll \min(m,n)$).经过奇异值分解,矩阵 D 可表示为三个矩阵的乘积:

$$D = U\Sigma V^T \quad (4)$$

其中,矩阵 U($m \times r$ 矩阵)和 V($n \times r$ 矩阵)分别是与 D 的奇异值对应的左右奇异矩阵,矩阵 D 的奇异值按照递减排列构成对角阵 Σ($r \times r$ 矩阵),r 是矩阵 D 的秩.

然后,取前 k 个最大的奇异值及其对应的奇异向量构成一个 k 阶矩阵 D_k 来近似表示原检索项—文本矩阵 D,也称 D_k 为 LSI 子空间,有:

$$D_k = U_k \Sigma_k V_k^T \quad (5)$$

文献[9]证明了奇异值分解不仅保持了矩阵的检索能力,而且提高了检索性能.

最后,在 LSI 子空间 D_k 中,使用余弦公式(2)计算代码和文档间的相似度.

3.3 结果处理模块

经过 LSI 执行模块后,得到每个代码段与文档段的相似度从高到低的列表.但其规模较大,不利于进一步的手工分析,所以有必要对该列表进行过滤.常用的过滤方法有以下两种:

第一、Cut-Point 法[2,3].该方法设置一个结果提取限额 C,对于每一个代码段,将与其最相关的前 C 个文档段提取出来作为最终结果,即相似度排在前 C 位的关联关系.

第二、相似度阈值法[3].该方法设置一个相似度阈值 S,只提取相似度高于该阈值的关联关系.S 的值一般设置在 50 %～70 %间.

通过上述的方法对相似度列表进行过滤,最后得到所有代码段和文档段的关联关系列表.

本节中介绍了使用 LSI 模型提取代码与文档间关联关系的基本流程,下面将针对软件文档和代码的特点,阐述在该模型的基础上进行改进的策略,目的是提高检索的查准率.

4 关联关系提取方法的改进

上述常规的 LSI 模型在提取文档和代码间的关联关系时,将文档和代码看作普通的文本进行检索,因此存在以下缺陷:

(1) 文档和代码中所有词的权重都使用相同的方法进行计算,那么特征项间的不同价值就不能体现出来,例如类名的价值应该大于变量名;

(2) 不能完全解决同义词和缩写词的问题.虽然 LSI 可以解决部分同义词的问题,但无法解决缩写词的问题,而代码中经常含有大量的缩写词,这将直接影响检索的效果;

(3) 不能使用反馈来对结果进行二次处理,由于不可能一次性使得查全率和查准率都达到理想的效果,所以应该利用每次的结果作为反馈进行迭代处理,使其逐步提升.

本节针对文档和代码的特点,提出 4 种改进策略,并结合第 3 节的处理过程说明如何改进 LSI 模型.

4.1 基于类继承关系的代码聚类

代码中的一个重要特点是包含各种调用关系和继承关系.我们可以利用继承关系对代码进行聚类分析,即将继承于同一父类的所有子类和该父类都划分为同一个簇(Cluster),因为继承是一种联结类的层次模型,其父类和子类间是"*is-a*(是一个)"关系,也就是说同一继承体系中的类一般表示相近的概念.对同一簇中的代码进行二次分析,如图 2 所示,如果文档 d 和代码 q_i 已经存在关联关系,那么 d 和 q_i 所在簇中的其它代码存在关联的概率应该增大.

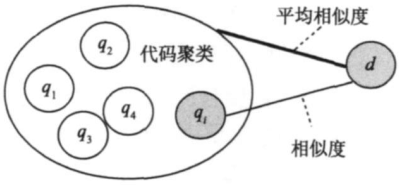

图2 代码聚类图

Input: source code clusters C, similarity list L
Output: similarity list after applying source code clustering

```
foreach Sim(d_i, q) ∈ L do
    if Sim(d_i, q) > threshold_high then
        retrieve the traceability link between d_i and q;
    else
        if Sim(d_i, q) > threshold_low then
            /* compute average
               similarity                    */
            S̄ = Σ_{q_j ∈ C[q]} Sim(d_i, q_j)/|C[q]|;
            if S̄ > threshold_enhanced then
                Sim(d_i, q) = S̄;
                retrieve the link between d_i and q;
            end
        else
            reject the link between d_i and q;
        end
    end
end
```

图3 代码聚类算法伪代码图

代码聚类算法的伪代码如图 3 所示,其中$|C[q]|$表示聚类中代码的数量.我们定义了一个相似度的阈值区间$[\text{threshold}_{Low}, \text{threshold}_{High}]$,将相似度低于 thres-

$threshold_{Low}$的结果作为无效关联进行过滤,将高于 $threshold_{High}$的结果作为有效关联提取出来,而将介于 $threshold_{Low}$和 $threshold_{High}$间的结果进行二次分析,即在阈值区间内选取一个相似度的修正阈值 $threshold_{Enhanced}$. 然后计算出该簇的平均相似度,对于每一个落在阈值区间内的关联关系,如果它所在簇的平均相似度大于修正阈值,则将该关联关系的相似度改为平均相似度,并作为正确结果提取出来;否则进行过滤.

根据不同的项目,阈值区间和修正阈值的选取不同. 一般而言,$threshold_{Low}$表示高查全率下的相似度阈值,也可以表示可接受的最低相似度,而 $threshold_{High}$表示高查准率下的相似度阈值. 由查全率和查准率的定义可知,它们一方的提高是以另一方的下降为代价的,所以 $threshold_{Enhanced}$是对查全率和查准率的折中选择. 例如,已知大部分正确的关联关系其相似度都大于 60 %,但是此时也包含较多的错误关联关系,即查准率较低;而且,已知相似度大于 80 %的关联关系中大部分都是正确的,但此时的查全率低. 也就是说大部分正确的关联关系其相似度都落在区间[60 %,80 %]上,而且该区间内也含有大量错误的关联关系,那么我们可以将阈值区间就设为[60 %,80 %]. 如果需要高查准率,可以在该区间内选取较大的修正阈值,如果需要高查全率则可以选取较小的修正阈值.

将代码聚类的信息整合到LSI模型中,需要修改预处理模块和过滤器的过滤条件,如图1中带纯灰色部分所示.

4.2 代码特征项分类

所谓的代码特征项分类是将每个代码文件进行更细粒度的处理,代码中不同的内容应表现为不同的价值,例如类名如果在一个文档中出现,那么这个类和该文档存在关联关系的概率应该较其它的文档大. 本策略实际上就是将不同的特征项赋予不同的权重因子.

根据对代码中的特征项进行归纳总结,按照其在文档中出现的可能性,本文将代码中的特征项分为三大类:

第一类为类名,考虑到文档中可能会直接出现,例如在API和UML图中,所以可以将类名作为关键字进行查找,对于匹配上的文档,在通过LSI模型计算出来的相似度列表中,将它们关联的相似度适当提高,这里乘以一个关联系数 $t(t \geq 1)$.

第二类为各种注释,包括类注释、方法注释和其它注释. 这些注释中的术语和词汇,有较大可能性出现在文档中,所以应该赋予不同的权重因子,区分其权重.

第三类为一般的特征项,即除了第一和第二类之外的其它特征项,对于这类特征项,按照常规的方法进行处理,即其权重因子设为1.0.

最后将通过 *tf-idf* 方法计算出来的词的权重分别乘以其权重因子,得到新的权重. t 值和权重因子的选取将在5.3节实验中进行讨论.

实现该策略需要加入特征项分类器,以及修改权重的计算方法,如图1中带网格的灰色部分所示.

4.3 引入相似度词典

虽然LSI可以在一定程度上解决词的同义性问题,但需要这些词在文中出现的频率较高. 另外,我们知道程序员在编写代码时习惯使用大量的缩写词,所以有必要引用相似度词典,以显式地解决同义词和缩写词匹配的问题.

这里的相似度词典用于记录特征项之间的相似度,每个词典项为一个三元组(k_i, k_j, a_{ij}),其中 k_i 和 k_j 为特征项,a_{ij}表示它们之间的相似度,取值为$(0,1]$. 具体实现时,由于只在相似度计算时涉及到词间的相似度,所以使用相似度词典来修改相似度的计算公式. 式(2)可以修正为式(6)[10].

$$Sim_T(Q,D) = \frac{\sum_{i=1}^{m} d_i q_i + \sum_{(k_i,k_j,a_{ij})\in T} a_{ij}(d_i q_j + d_j q_i)}{\sqrt{\sum_{i=1}^{m} d_i^2 \sum_{i=1}^{m} q_i^2}} \quad (6)$$

在上式中,当我们计算代码 Q 和文档 D 的相似度 $Sim_T(Q,D)$时,对于每个特征项,需要在相似度词典中搜索是否含有该特征项的三元组,其中 $d_i * q_i$ 表示在文档向量和代码向量中同时出现的特征项;$d_i * q_j$ 表示对于文档向量中第 i 个特征项,如果它在相似度词典中对应的近义词 q_j 同时在查询向量中出现时,其对相似度计算的贡献. 同样,$d_j * q_i$ 表示对于查询向量中第 i 个特征项,如果它在相似度词典中对应的近似词 d_j 在文档向量中出现时,其对相似度计算的贡献.

这里,相似度辞典 T 在相似度计算前建立,主要通过下面3个方法建立:

第一、如果项目中含有数据词典,则将数据词典添加到相似度词典中,因为项目中的大部分专业词汇都会在数据词典中明确地进行说明,这样就可以很好地解决专业词汇难以查询、含义模糊不明确的问题.

第二、将一些通用的同义词典加入到相似度词典中. 文档中大量的词汇都是常用词,所以有必要加入一个通用的同义词典,来显示地标注出同义词.

第三、将代码中的缩写词进行扩展[8].

实现此策略时只需在计算相似度的时候加入相似度词典即可,如图1中的相似度词典所示.

4.4 基于文档类型的分类搜索

按照文档的描述粒度,本文将文档分为高层概念文档和低层实现文档两种类型. 高层概念文档指需求

文档、概要设计文档等;低层实现文档指详细设计文档、API 说明文档等. 当然,不局限于将文档分为两类,也可以将文档分为多类. 在不同文档类型之间,存在一种基于关联度的偏序关系,即"相邻的"两类文档之间存在较高的关联度.

一般而言,低层文档和代码关联较密切,经常包含代码中的类名、方法名和注释. 同时,低层文档也包含对高层文档的进一步详细描述,例如含有高层文档中提到的概念、方法及其实现,也就是说低层文档作为桥梁连接了高层文档和代码. 因此,按照文档类型的偏序关系,先对低层文档进行关联关系的提取,然后将其结果作为反馈,修正 IR 模型,使查询向量学习和累积低层文档的概念,接着再使用这些新的查询向量对更高一层的文档进行检索,如此循环直到最高层文档为止. 采用这种迭代查询的方法,将有助于充分利用低层文档的"桥梁"作用,以提高代码与高层文档之间关联关系的查全率和查准率.

文献[6,7]指出用户反馈可以极大地提高 IR 模型的检索能力,但是需要大量人工干预. 而这里将每一层文档的处理结果添加到下一次的迭代处理流程中,可以看作是一个自动反馈的过程,所以模型的检索能力将得到进一步的提高.

此外,文献[6]中分析比较了使用用户反馈改进 IR 模型主要有三种方法. 本文根据实际情况选择 Ide dec-hi 方法并进行修改. 其主要思想是,使用结果中所有相关的文档向量和一个不相关度最高的文档向量来修正查询向量,修正查询向量的公式如下:

$$Q_{new} = Q_{old} + \sum_{all\ relevant} D_i - D_{most\ nonrelevant} \qquad (7)$$

由于这里的反馈不是由用户提供的,而是对每层文档处理后自动获取的,所以不能确定所有相关的文档向量. 另一方面,J. Cleland-Huang[4]的实验表明,高相似度的链接几乎都是正确的链接,而低相似度的链接几乎都是错误的链接. 所以这里近似地认为与查询向量相似度最高的一个链接是正确的. 那么式(7)就进一步修改为:

$$Q_{new} = Q_{old} + Q_{most\ relevant} - D_{most\ nonrelevant} \qquad (8)$$

实现时,由于文档的层次结构一般比较明显,所以采用手工的方式进行分类,例如将需求文档、概要设计文档归为高层文档,而将详细设计文档、API 说明文档、测试文档等归为低层文档,如图 1 中带阴影部分所示. 另外,需要将每次处理得到的部分结果反馈到 LSI 子空间中,用于修改查询向量,如图 1 中的粗箭头所示.

5　实验和结果分析

为了验证本文中所提出的 4 种策略的有效性,本节将使用修正后的 LSI 模型对 2 个软件系统进行分析,并将结果与文献[2～4]中的结果进行分析比较.

首先,为了验证改进后的 LSI 模型比常规的信息检索模型在检索能力和检索精度上有所提升,实验 1 将分析文献[2,3]里共同选用的数据集——LEDA 系统 3.4 版. LEDA(Library of Efficient Data Types and Algorithms)是基于 C++ 的自由软件,含有 97000 行代码,219 个类以及 238 页的文档.

其次,文献[4]结合了文档信息对概率模型进行改进,所以,这里将改进后的 LSI 模型与文献[4]中改进后的概率模型进行比较. 实验 2 使用文献[4]中选用的数据集——IBS(Ice Breaker System)系统进行实验,该系统含有 72 个类、18 个包以及 180 个功能需求.

最后,实验 3 的目的在于确定代码特征项分类策略中几个参数的值.

5.1　实验 1

与文献[3]一样,我们对 LEDA 的整个库进行分析,包括有效的代码文件、demo 程序、测试用例和所有的文档. 另外,由于文档数量远小于代码数量,文献[3]中以文档作为检索项来查找相关联的代码,这与以代码为检索项来查找相关联的文档是一样的,但是这样就无法使用策略 4. 为了保证实验的一致性,这里仅使用前三种策略进行后的 LSI 模型.

表 1 展示了该系统的规模以及处理后的情况. 可见,分割后的代码段和文档段分别是 487 和 110,所以最后可以得到 487 ×110 的相似度结果列表. 在进行结果筛选时采用在 3.3 节中提到的两种策略:Cut-Point 法和相似度阈值法.

表 1　LEDA 的情况

LEDA3.4	原始数量	分割后的数量/份
代码	487 份	487
文档	238 页	110
总数		597

使用前三种策略改进后的 LSI 模型对 LEDA 数据集进行实验分析,在最后获得的结果集中抽取前 600 个关联关系作为样本空间,并人工分析这 600 个关联关系的正确性,发现其中包含 150 个正确的关联关系. 本文使用查准率(Recall)和查全率(Precision)[2]作为衡量的指标.

表 2 总结了用 Cut-Point 方法对结果进行筛选后的情况,图 4 是使用表 2 中数据和文献[2,3]中的数据绘制而成的比较图,从图中可以看出:

(1)从查全率来看,改进后的 LSI 模型不如另外两个模型的效果好,这与实验约束不同有关. 文献[2]中假设每个代码最多关联一个文档,只有 88 个正确的关

联,而文献[3]的作者不考虑代码间的继承关系,只有114个正确的关联,也就是说,相对较高的查全率是以损失查准率为代价的.本文没有上述限制,因此在手工分析时获得了150个正确的关联,绝对数量有所提高.

(2)从查准率看来,改进后的LSI模型其查准率明显高于概率模型和常规的LSI模型.概率模型的查准率低反映出其能力的不足.相对常规的LSI模型,改进后的LSI模型在查准率上得到了一定的提高(5%～16%).

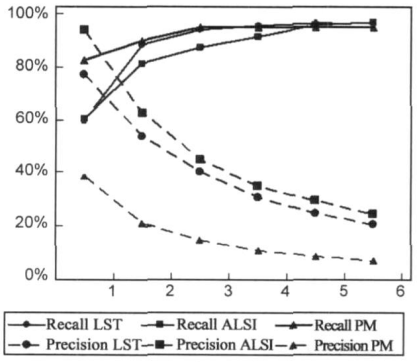

图4 结果比较图

注:横坐标表示C的值,纵坐标表示百分率,带三角的实线和虚线分别表示概率模型的查全率和查准率;带圆形的实线和虚线分别表示常规的LSI模型的查全率和查准准率;带方形的实线和虚线分别表示改进后的LSI模型(ALSI)的查全率和查准率.

表2 LEDA实验结果

C	正确链接数	错误链接数	丢失链接数	提取总链接数	查准率	查全率
1	91	6	59	97	93.81%	60.67%
2	122	72	28	194	62.89%	81.33%
3	131	160	19	291	45.02%	87.33%
4	137	251	13	388	35.31%	91.33%
5	144	341	6	485	29.69%	96.00%
6	145	437	5	582	24.91%	96.67%

注:第1列表示选择的 C 值,第2列表示结果中正确的关联关系,第3列表示结果中错误的关联关系,第4列表示没有提取出来的关联关系,第5列表示以 C 为阈值总共提取出来的关联关系,第6列表示查准率,第7列表示查全率.

(3)当 C 的取值为1时,改进后的LSI模型的查准率达到93.81%,只有6个是错误的,说明当相似度较高时,几乎都是正确的关联,这对4.4节中的自动反馈提供了有力的支持.

由此可以看出,改进后的LSI模型确实能够在查全率变动不大的情况下,提高查准率5%～16%.

另外,当采用相似度阈值的方法对结果进行筛选时,以相似度60%为阈值,可以得到查全率在83.33%的情况下,查准率高达49.21%.而文献[2]以相似度60%作为阈值时,其查全率为71.01%,查准率为42.98%.相比之下,改进后的LSI模型无论在查全率和查准率都有大幅提高.

分析其原因在于,第一、由于代码进行了聚类,使得原来没有被发现的关联关系现在被提取出来,即正确的关联关系的基数变大了;第二、LEDA的文档中含有大量类名、类注释和方法注释,策略2提高了这些标识符的权重,使得关联关系更加明显,从而提高了查准率;第三、相似度词典可以更好地解决同义词和缩写词的问题,这对于提高查准率有较大的辅助作用.从该实验中同时可以看出,高相似度的关联关系,几乎都是正确的关联.

5.2 实验2

实验2中采用了基于第4种改进策略的LSI模型.实验对象为文献[4]中使用的测试集IBS系统;比较对象为常规的概率模型和文献[4]中使用文档层次信息改进后的概率模型;实验方法与文献[4]的一样,统计查全率达到90%和95%时的查准率.

本实验中将文档的类型分为2类,第一类是需求文档和概要设计文档,第二类是详细设计文档.而文献[4]同样使用了文档的特性对概率模型进行了改进,提出了名为Hierarchical Enhancement[4]的改进方法,其主要思想是根据需求细化和实现过程将文档内容进行分类,即将属于同一需求的所有子需求及其实现文档归为一类,那么如果有 n 个需求,最后文档就被分为 n 类,对于同一类文档,最后使用条件概率对模型进行修正,其目的是提高需求跟踪的精度.文献[4]的改进方法只需要一次检索,不存在迭代的过程.

实验结果的比较如表3所示,从表中可以看出,在查全率相近的情况下,改进后的LSI模型较常规的概率模型获得15%～19%的查准率提高,较改进后的概率模型也能获得8%左右的查准率提高.虽然文献[4]的改进也可以得到较好的效果,但是在文档预处理时需要大量的人工干预,而且文献[4]的出发点并不是提取文档和代码间的关联关系,而是进行需求跟踪.

表3 IBS的实验结果

检索模型	查全率	查准率
常规概率模型[4]	90.47%	20.43%
	95.01%	16.81%
文献[4]改进的概率模型	90.48%	31.72%
	95.69%	25.65%
本文改进的LSI模型	90.45%	39.55%
	95.33%	32.24%

实验2的结果表明,策略4有助于在相同查全率的情况下提高查准率.原因主要在于,策略4的每次迭代中都整合了上次的检索结果,所以有助于查询词的自

我学习和不断积累,即每次都学习到了该类文档表示的概念,这将大大改进对更上一层文档的检索效果.特别是对于规范的软件文档而言,由于需求是逐步细化的,低层文档是对高层文档的进一步阐述和实现,所以在查询词中融入低层文档的概念,将进一步扩展查询词的内容,使其更接近于高层文档表达的概念.

5.3 实验 3

代码特征项分类策略中针对不同类型的特征项赋予不同的权重因子,包括使用第一类特征项进行直接检索时需要提高的相似度幅度(设为 t 倍)和第二类特征项的 3 个权重因子(类注释、方法注释和其它注释).本文是第一个提出代码特征项分类的,所以这些参数的设定还没有可以参考的资料,与 LSI 子空间维数 k 的确定一样,应该根据不同的项目进行设置,下面用实验来统计其变化规律,并给出设置的指导原则.实验对象是实验一中使用到的 LEDA 系统,实验方法是统计不同 t 值和权重因子下,检索的查全率和查准率.

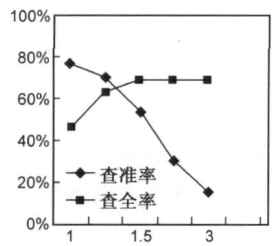

图5　不同t值下的LEDA的结果图

首先确定 t 的值,设置 t 的范围为[1.0,2.0],步长是 0.2,实验结果如图 5 所示.从图中可以看出,当 t = 1.2 时,查全率为 63.33 %,查准率为 69.85 %,这是比较理想的情况,之后查准率就出现了急剧的下降,特别当提 t > 2 后,查全率就保持不变.可以看出,在 LSI 模型中加入关键字匹配子系统可以提高检索的精度,但是,t 值的设置不宜太大,否则会扰乱 LSI 模型的计算体系.

接着,将第二类特征项分别赋予不同的权重因子值,其取值范围是[1.0,3.5],步长是 0.5.当单独提高某一种注释的权重因子时,实验表明查准率和查全率都呈现出抛物线的形状,即权重因子只在某个取值上使之达到最大值,之后就开始下降.分析其原因,在于过大地提高其权重因子,会破坏 *tf-idf* 的权重计算原理,因此效果会不断恶化.即权重因子的设置不是越大越好,而是中间某个值达到最优.

当同时修改多种注释的权重因子时,将类注释、方法注释和其它注释的权重因子分别设为 2.0、1.5 和 1.5 时取得最好效果,能提高查全率 3 %,同时提高查准率 4 %.

综上所述,t 值和权重因子的值并不是越大越好,而是在中间某个值达到最优,而且类注释的权重因子 ≥方法注释的权重因子 ≥其它注释的权重因子.

6 总结和展望

本文从信息检索的角度出发,在常规的 LSI 检索模型的基础上,结合了软件工程中文档的和代码的特点,提出了 4 种改进策略用于提取文档和代码间的关联关系,并且实现了一个改进后的原型系统.最后通过对 2 个系统的测试,并与前人的实验结果进行比较,得出的结论是:与常规的 IR 模型相比,前 3 种改进策略可以提高检索的查准率 5 %～16 %,第 4 种改进策略可以提高 15 %左右.

目前,相似度词典的构造和软件文档的分类仍然需要人工干预,没有完全实现自动化,下一步工作将研究如何更好地实现自动化;而且需要进一步对大型开源软件进行测试,例如 Eclipse 和 Lucene;另外,已有方法只能处理文档和代码都使用相同语言编写的情况,所以研究如何处理中文文档将是下一步的工作重点.

参考文献:

[1] Marcus A, Maletic J I, Sergeyev A. Recovery of traceability links between software documentation and source code[J]. International Journal of Software Engineering and Knowledge Engineering(IJSEKE), 2005, 15(5): 811 - 836.

[2] Antoniol G, Canfora G, Casazza G, et al. Recovering traceability links between code and documentation[J]. IEEE Transactions on Software Engineering, 2002, 28(10): 970 - 983.

[3] Marcus A, Maletic J I. Recovering documentation to source code traceability links using latent semantic indexing[A]. Proceedings 25th International Conference on Software Engineering (ICSE '03)[C]. Portland, OR, USA, 2003. 125 - 135.

[4] Cleland-huang J, Settimi R, Chuan D, et al. Utilizing supporting evidence to improve dynamic requirements traceability[A]. Proceedings 13th IEEE International Conference on Requirements Engineering (RE '05)[C], Paris, France, 2005. 135 - 144.

[5] 钱剑飞,陈华,陈奇. 一种代码与中文文档关联信息的自动提取方法[J]. 浙江大学学报(工学版), 2004, 38(11): 1417 - 1421.
Qian Jian-fei, Chen Hua, Chen Qi. Automatic retrieval method for tracing links between code and chinese documents[J]. Journal of Zhejiang University (Engineering Science), 2004, 38(11): 1417 - 1421. (in Chinese)

[6] Salton G, Buckley C. Retrieval performance by relevance feedback[J]. Journal of the American Society for Information Science, 1990, 41(4): 288 - 297.

[7] De Lucia A, Oliveto R, Sgueglia P. Incremental approach and user feedbacks: A silver bullet for traceability recovery[A]. Proceedings 22nd IEEE International Conference on Software Maintenance(ICSM '06)[C]. Philadelphia, USA, 2006. 299 - 309.

[8] 邰晓英. 信息检索技术导论[M]. 北京:科技出版社, 2006.

[9] P C H, Raghavan P. Latent semantic indexing: A probabilistic

analysis[J]. Journal of Computer and System Sciences, 2000 (61) :217 - 235.

[10] Hayes J H, Dekhtyar A, Osborne J. Improving requirements tracing via information retrieval[A]. Proceedings 11th IEEE International on Requirements Engineering Conference[C]. Monterey, CA, USA, 2003. 138 - 147.

[11] 王映辉,王立福,张世琨,王琼芳. 一种软件需求变化追踪方法[J]. 电子学报,2006,34(8) :1428 - 1432.
Wang Ying-hui, Wang Li-fu, Zhang Shi-kun, Wang Qiong-fang. A tracing approach of software requirement change[J]. Acta Electronics Sinica, 2006, 34(8) :1428 - 1432. (in Chinese)

[12] Yu C T, Luk W S, Cheung T Y. A statistical model for relevance feedback in information retrieval[J]. Journal of the Association for Computing Machinery, 1976, 23(2) :273 - 286.

[13] Hayes J H, Dekhtyar A, Sundaram S K. Advancing candidate link generation for requirements tracing: The study of methods[J]. IEEE Transactions on Software Engineering, 2006, 32 (1) :4 - 19.

作者简介:

赖冠辉 男,1983 年生于广东省东莞市,现为北京航空航天大学软件工程研究所硕士生,主要研究领域为软件工程与信息检索.
Email: garylai@sei.baa.edu.cn

王晓博 男,1982 年生于河南平顶山,现为北京航空航天大学软件工程研究所博士生,主要研究领域为程序分析、软件可视化与数据挖掘等.
Email: boby@sei.buaa.edu.cn

(上接第 35 页)

参考文献:

[1] Abry P, Veitch D. Wavelet analysis of long-range dependent traffic[J]. IEEE Trans Information Theory, 1998, 44(1) :2 - 15.

[2] Leland W, Taqqu M, Willinger W. On the self-similar nature of Ethernet traffic (extended version) [J]. IEEE ACM Transactions on Networking, 1994, 2(1) :1 - 15.

[3] Ge X H, Zhu G X, Zhu Y T. On the testing for alpha-stable distributions of ethernet network traffic[J]. The Journa1 of Electronics, 2003, 1(1) :309 - 312.

[4] Sikdar B, Vastolak S. The effect of TCP on the self-similarity of network traffic[J]. Computer Communications and Networks, 2003, 1(1) :370 - 373.

[5] Figueiredo D R, Liu B, Misra V. On the autocorrelation structure of TCP traffic[J]. Computer Networks, 2002, 40(3) :339 - 361.

[6] Park K, Kim G, Ccrovellam. On the relationship between file sizes transport protocols and self-similarity network traffic[A]. Proceeding of ICNP[C]. Columbus: IEEE Press, 1996. 171 - 180.