文章编号:1000-0887(2019)11-1235-11

改进的卷积神经网络源代码相似性度量方法*

谢春丽, 蔺疆旭, 刘小洋, 张文斌, 黄军伟

(江苏师范大学 计算机科学与技术学院, 江苏 徐州 221116)

摘要: 源代码相似性是指不同代码段功能上的相似程度,是软件工程领域一项重要的研究问题.现有的方法主要从文本、结构两方面,利用代码的统计学特征计算相似性,其最大缺点就是无法表达代码的语义特征.为解决此类问题,提出了一种融合统计信息的卷积神经网络(statistics information for code embedding-convolutional neural networks, SICE-CNN)源代码相似性检测方法.该方法首先通过词嵌入对源代码进行信息表示,获取代码的词嵌入向量信息;其次,构建 CNN 训练模型学习源代码文档的嵌入表示;最后,计算源代码对的余弦相似值.实验表明,该方法和一般词嵌入方法相比提高了一定的性能,能较好地检测源代码的语义相似性.

关 键 词: 深度学习; 卷积神经网络; 代码相似性; 词嵌入

中图分类号: TP311 **文献标志码:** A DOI: 10.21656/1000-0887.400221

引 言

源代码相似性度量旨在判断一段代码与其他代码在语句、语义功能上的相似程度,是代码克隆[1-2]、代码剽窃[3-5]、代码推荐[6]、知识产权、信息检索[7]等应用的基础.随着软件规模的不断增大,软件系统中代码重复是不可避免的,这不仅增加了程序代码的容量和运行时间,也给软件维护带来了难题.因此,代码相似性具有重要的研究意义[8].

对源代码相似性的研究起源于 20 世纪 70 年代,早期的源代码相似性方法主要计算源代码中操作符、操作数、代码行数、函数、类型、常量等属性的个数,由于计算方法单一,容易误判,因此准确度较低.后来,研究者们从代码结构方面解析代码内容,此类方法把源代码整个文本视为一系列字符串,或者更进一步,把代码转换为一系列标记,甚至抽取更加抽象的语法树、程序依赖图,采用字符串匹配[9]、后缀树匹配[10]、子图匹配[11]等算法计算相似值.基于结构的相似性计算方法取得了不错的效果,但对代码语法树的解析需要大量的语法研究与规则设计工作, 往往用在较小的数据集上.

从本质上来说,源代码相似性计算的关键步骤是源代码表征问题,对源代码的表征方式决定了信息抽取的程度和粒度,进而影响相似性比较的精度.2012 年,Hindle 等首次提出代码的自然性假设,并证明了源代码具有和自然语言相似的特征[12],这为源代码表征和分析提供了

* **收稿日期:** 2019-07-22; **修订日期:** 2019-09-23
基金项目: 国家自然科学基金(61773185;61877030;61502212); 江苏省高校青蓝工程
作者简介: 谢春丽(1979—),女,副教授,博士(E-mail: xcl_bhb@163.com);
刘小洋(1979—),男,教授,博士(通讯作者. E-mail: liuxiaoyang1979@gmail.com);
张文斌(1976—),男,讲师,硕士(E-mail: zwbwen@163.com).

新的思路.受启于自然语言的处理方法,大量机器学习算法被运用到代码分析中[13-15],但仍然需要手工提取源代码特征,而手工特征很难很好地反应代码的真实情况,无法提取隐藏在源代码中的深层次复杂特征.

为此,本文提出了一种融合 SICE-CNN 相似性的计算方法.该方法首先通过词嵌入对源代码进行信息表示,获取源代码的词嵌入向量信息;然后,基于词向量,构建了 CNN 训练模型学习源代码文档的嵌入向量表示;最后,计算源代码对的余弦相似值.本方法将深度学习技术应用到代码相似性的度量,充分运用了代码中的文本信息,旨在通过挖掘代码深层次的语义信息揭示代码功能.与传统的机器学习方法相比,深度学习通过大量的训练数据自动学习代码的深层特征,经过多层映射和抽象, 组合出更高层次特征[16-19].本文利用深度神经网络在文本处理方面的特长,将文本信息加入源代码文档的表征中,同时结合词嵌入向量和 TF-IDF 统计信息,综合判断代码的功能相似性.

最后,本文对所提出的基于 SICE-CNN 的相似性检测方法进行了实验验证.实验结果表明,该方法优于现有的检测方法,较大幅度地提高了相似性检测的查全率(41.1% = 83.1% - 42%),最终综合提高了语义相似性检测的 F_1 值(19% = 74% - 55%).

本文第 1 节介绍了相关研究的现状,并对此进行总结与分析;第 2 节具体介绍了本文提出的融合统计信息的 CNN 相似性计算框架;第 3 节对所提出的方法进行了验证与评估;第 4 节对论文进行了总结和展望.

1 相关工作

1.1 源代码相似性

代码相似性旨在发现两个代码片段功能上的相似,作为代码推荐或者克隆检测的依据.最早的方法来源于软件复杂性度量,Halstead 提出用程序中不同运算符个数、操作数个数、变量类型等属性表征程序,属性相近的程序具有较高的相似性,这种方法虽然计算简单,但精度较低,且容易受到变量替换等干扰[20].Roy 等随后提出最长公共子序列和模式匹配算法计算源代码之间的相似度,此种方法把源代码当作字符串来处理,计算开销小,独立于编程语言,但是忽略了程序的语法和结构信息[9].Kamiya 等利用解析器把源代码表示成符号序列,并过滤掉无用的字符、空格、注释等,然后采用符号匹配进行相似性计算,这种方法的缺点在于没有考虑代码行的顺序,忽略了代码中的结构信息[1].Baxter 等提出基于抽象语法树的方法[10],Komondoor 等提出基于程序依赖图、控制流图等方法,计算树、图的形状相似性,这类方法能够检测程序代码的结构信息,但计算开销非常大,且仍然不能处理语义信息[21].

以上基于字符串的、符号的、树形和图的四类代码相似性计算方法,在小规模程序上具有不错的效果,但是随着网络开源项目的兴起,网络上功能相似、形式完全不同的开源代码不断增多,早期的方法已经力不从心了,研究者们提出了运用机器学习来分析代码特征的方法,这一思想来源于程序文本的自然性假设.目前度量文本相似性的方法主要有基于关键词的、基于向量空间的以及基于深度学习的 3 种方法.基于关键词的匹配算法主要有 *N*- gram 相似度和 Jaccard 相似度: *N*- gram 方法是用两个文本中长度为 *N* 的共有子串的数量来度量相似性;Jaccard 方法是用两个文本之间词集合的交集和并集的比值作为相似值.基于向量空间的方法首先把文本中的词映射为固定长度的向量,根据词之间的距离来判断文本之间的相似性,常见的有 word2vec 和 TF-IDF 方法[22].由于深度学习在文本处理、计算视觉领域的出色表现[23],研究者也开始采用深度学习解决程序分析[24]、代码剽窃[25]、代码克隆[26]、代码摘要生成、错误定位

和程序修复等问题[27-28].Mou 等从代码的抽象语法树中学习代码符号的词向量,并设计了一个树形结构的卷积层提取代码的结构信息,实现代码分类和搜索应用[29].Nguyen 等构建 DNN 模型从输入的源代码中逐层提取词汇和语法特征,实现代码描述任务[24].White 等提出一个基于自动编码器的深度学习框架,学习代码的词汇级和句法级特征,用于代码克隆检测[30].

以上算法多数是把源代码转换成符号序列、语法树或者程序依赖图,即使是基于词向量的方法,也是通过训练得到标识符对应的词向量,而不是代码词汇的向量表示,这会丢失代码中原本的语义、语法等信息.本文结合深度学习技术,直接学习隐藏在源代码中更加复杂的语义和语法结构特征,从而更接近代码的功能表达,提高了代码相似性计算的精度.

1.2 源代码表征

源代码表征是代码相似性检测的根本问题,代码表征的粒度大小决定了相似性计算的精度和效率.因此,在模型训练之前,首先需要把源代码以向量形式编码,目前比较典型的向量表示方法主要有以下两种.

1.2.1 One-hot 编码

One-hot 编码也叫独热编码,每个状态只有一位有效,是词表征最简单、最直接的一种表征方式.每个词用一个 N 位的向量表示,其中只有一位为 1,其他都为 0. One-hot 编码用可以保证编码之间的相互正交,但是每个词的向量长度要等于词汇表的长度,因此随着编码维度的增大,编码会非常稀疏,而且这种编码的向量不表示任何语义信息,因此相对来说比较适合低维数据.

1.2.2 词嵌入向量

词嵌入是自然语言处理中的一个基本问题,最开始是用统计语言模型预测文本序列问题,但是由于统计语言模型无法处理较长的上下文以及参数空间的爆炸式增长问题,学者提出了分布式表示模型,即用连续的稠密向量去刻画一个词的特征,在这种模型中,具有相似上下文的单词会有相近的向量表示,每个单词不再是独立的,因此词嵌入模型可以较好地表达语义信息.Mikolov 等提出的 word2vec 模型是最经典的词嵌入模型,通过一个浅层神经网络预测每个单词的分布式向量[31].Word2vec 模型同样可用于源代码的表征工作,Ye 等和 Nguyen 等从 API 文档抽取了代码片段,通过该模型训练出一个低维向量空间,在软件搜索以及故障定位任务中,用训练出的向量空间距离表示文档之间的相似度[32-33].Chen 等从 stack overflow 等编程问答社区中抽去了大量的代码语料,通过词向量处理近义词问题,为代码转换等工作提供了基础[34].Hao 等把词嵌入技术应用到抽象语法树等源码的表示上,使相似的代码标识符具有相似的向量[35].

2 源代码相似性计算框架

本节首先给出源代码相似性的形式化定义,然后介绍本文提出的 SICE-CNN 相似性计算框架.

2.1 源代码相似性的定义

问题定义:已知代码片段集合 $C=\{c_1,c_2,\cdots,c_n\}$,通过深度学习获得相似性模型 f,使得 $f(c,C,\theta)\rightarrow T$,其中 $c\in C$,$T=\{t_1,\cdots,t_i,\cdots,t_j,\cdots,t_k\}$,$T$ 为源代码片段集合 C 中和代码片段 c 相似值最大的前 k 个候选集,且当 $i<j$ 时,相似值 $S(t_i)>S(t_j)$,参数 θ 为模型待训练参数.该模型可以用来度量代码片段对的相似值,并给定与代码片段 c 相似的代码集合 T.

针对上述问题,本文提出一种改进的 CNN 代码相似性计算模型,该模型包括 3 个阶段,如

图1所示.首先,对源代码进行词嵌入表征,将源代码中出现的变量、函数名、运算符、保留字、常量等转为向量;其次,利用深度学习强大的特征学习能力提取源代码中隐藏的数据特征,并组合高层抽象特征表征源代码文档;最后,计算代码片段对之间的余弦相似值.

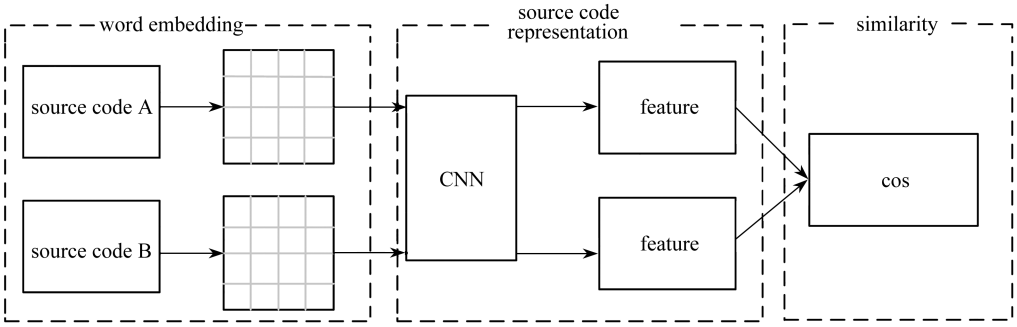

图1 源代码相似性框架

Fig. 1 The framework for source code similarity

2.2 SICE 模型

2.2.1 TF-IDF 值

源代码不同于自然语言文档,源代码具有开放的、无限的语料库.例如,根据标识符的命名规则,变量名个数是无限的.源代码中主要有三类词汇:一是表示数据类型、控制结构、库文件等的保留字;二是用户自定义的变量名、函数名等;三是各种运算符号.对于源代码文档而言,权重的计算方法应该体现出源代码的结构特征.表示结构的特征词在不同的源代码中对程序内容的反映程度不同,其权重的计算方法也应不同,因此应该对于表示控制结构的特征词分别赋予不同的系数,然后乘以特征词的词频,以提高代码表示的效果.TF-IDF 是信息检索领域广泛使用的基于统计的文档表征方法,该方法基于以下假设,假设文档最有意义的词语应该是那些在文档中出现频率高,而在整个文档集合的其他文档中出现频率少的词语.源代码文档 c 被表示为 N 个保留字序列,$c = \{v_1, v_2, \cdots, v_N\}$,保留字$v_i$ 的权值为改进的 TF-IDF 值f_i, $f_i \in F = \{f_1, f_2, \cdots, f_N\}$,$F$ 为源代码中词的权值.

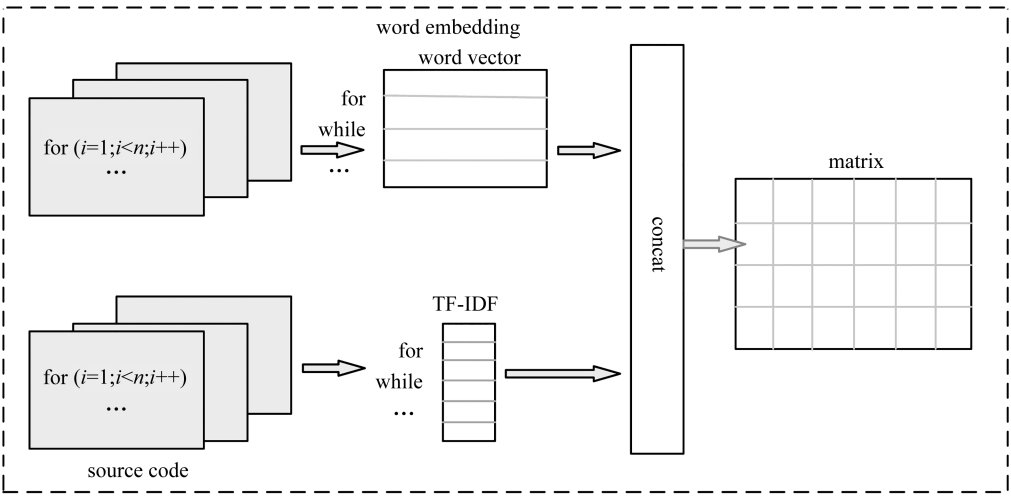

图2 SICE 模型

Fig. 2 The SICE model

2.2.2 SICE 词嵌入

如图 2 所示,本框架首先获取代码段的词向量表示和 TF-IDF 值,结合语义和统计信息表示源代码.模型输入为源代码片段 c, 经过分词之后,源代码文档 c 被表示为 N 个词序列, $c = \{v_1, v_2, \cdots, v_N\}$, 其中 N 为代码 c 中单词个数.词嵌入向量是由一个典型的浅层神经网络词训练得出的,输入为 one-hot 编码的输入上下文 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_k\}$, $\boldsymbol{x}_i \in R^{1\times N}$, 窗口大小为 k, 词汇表大小为 N, 隐藏层有 d 个神经元, 词向量矩阵 $\boldsymbol{W} \in R^{N\times d}$, 输出层是一个 $1 \times N$ 的向量, 每一个值代表着输出一个词的概率, 通过模型训练, 词向量矩阵 $\boldsymbol{W}$ 就是源代码 c 的词序列嵌入向量.

传统方法在用词向量表示文档时,往往将所有词向量平均加权,即认为文档中每个词的贡献是一样的,而对源代码是不同的,每个词的词频采用 TF-IDF = $\{f_1, f_2, \cdots, f_N\}$ 表示.用每个词的 TF-IDF 值点乘对应的词向量,得到初步的代码矩阵 $\boldsymbol{W}^0 = \{f_1 \cdot \boldsymbol{w}_1, \cdots, f_i \cdot \boldsymbol{w}_i, \cdots, f_N \cdot \boldsymbol{w}_N\}$, 其中 $\boldsymbol{w}_i$ 为矩阵 $\boldsymbol{W}$ 的第 i 行向量.

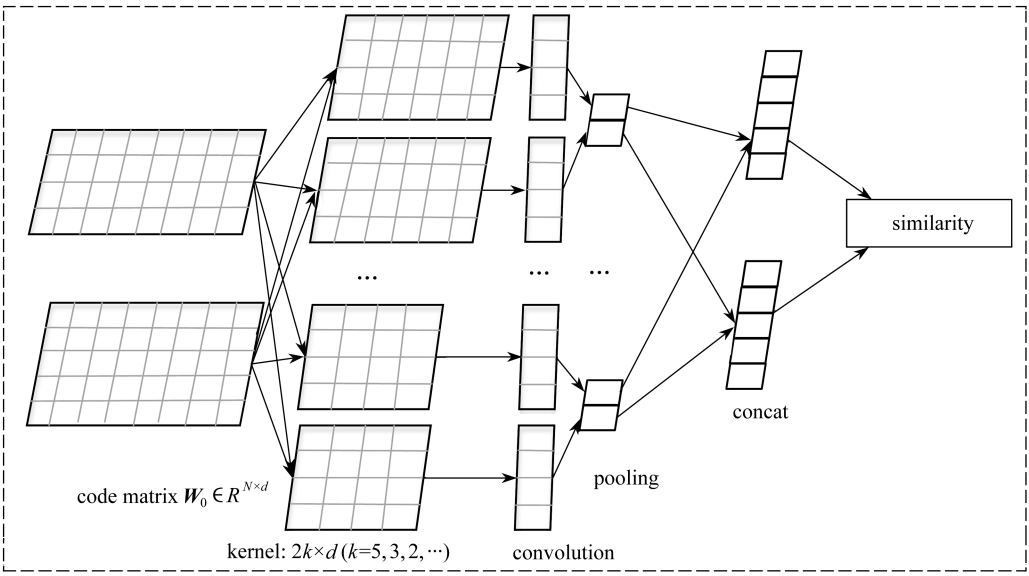

图 3 SICE-CNN 源代码表征

Fig. 3 Source code representation based on SICE-CNN

2.3 基于 SICE-CNN 的源代码表征

在代码表征阶段,已经对源代码进行了词向量化,获得了源代码中词的 TF-IDF 加权矩阵.我们知道,各种类型的词是源代码的基本组成单位.一个最简单也最直接得到源代码表征的方法是将组成代码的所有词的嵌入向量全部加起来,但是这种表示方法会丢失很多信息.本小节将会介绍一种用 CNN 模型对源代码文档表征的方法,如图 3 所示,模型框架包括输入层、卷积层、池化层、连接层和输出层 5 个层次,接下来,本文将一一介绍每层的作用和参数:

(a) 输入层 模型输入为上一阶段获取的一对源代码的初始化词向量矩阵 $\boldsymbol{W}^0$, 不同的源代码可能词向量个数不同,每个词被表示为 d 维向量,源代码对的词向量矩阵分别被表示为 $\boldsymbol{W}^{\mathrm{A}} \in R^{N_1\times d}$, $\boldsymbol{W}^{\mathrm{B}} \in R^{N_2\times d}$, 其中 N_1, N_2 分别为源代码文档 A、B 中的词向量个数.为了后面各层计算能够共享相同的参数信息,在卷积运算之前,将源代码对的初始词向量矩阵 $\boldsymbol{W}^0$ 做 0 填充处理.$N = \max\{N_1, N_2\}$, 填充之后的词向量矩阵为 $\boldsymbol{W}^0 \in R^{N\times d}$, d 为词向量维度.

(b) 卷积层 卷积核为 $\boldsymbol{f} \in R^{s\times d}$, 共有 X 个卷积核, 每个卷积核在词序列 $\{v_i, v_{i+1}, \cdots,$

$v_{i+s}\}(1 \leqslant i \leqslant N-s+1)$ 进行卷积运算：$p_i = \boldsymbol{W}_i * \boldsymbol{f}$，“$*$”为矩阵卷积运算，$\boldsymbol{W}_i$ 为词序列对应的词嵌入向量 $[\boldsymbol{w}_i, \boldsymbol{w}_{i+1}, \cdots, \boldsymbol{w}_{i+s-1}]$ 构造的矩阵，即初始词向量矩阵 $\boldsymbol{W}^0$ 中的第 i 行到 $i+s-1$ 行，每个卷积运算之后得到 1 个向量 $\boldsymbol{P} = [p_1, \cdots, p_i, \cdots, p_{N-s+1}]' \in R^{d_1 \times 1}$，其中 $d_1 = N-s+1$ 个卷积核经过卷积运算一共得到 X 个 d_1 维向量.

(c) 池化层　池化层的主要作用是为了降维，经过每个卷积运算后，得到一个向量 $\boldsymbol{W}$，模型对卷积后的向量进行最大池化操作，获取向量 $\boldsymbol{P}$ 中的最大值.

(d) 连接层　把 X 个卷积运算，经过池化操作得到的 X 个数值连接成一个向量，利用连接函数把文档的低层特征组合成高层抽象特征.

(e) 输出层　连接层得到每一个源代码文档的高层抽象特征，本文利用余弦相似性度量方法计算源代码对的相似值，如果相似值大于给定的阈值，则认为此源代码对是相似的，否则不相似.

关于 SICE-CNN 的算法，具体代码见附录，函数 get_model 构建并编译 SICE-CNN 模型，其中包含两个相同的 CNN 子网络模型，对两篇文档进行处理最终得到余弦相似度.函数 train_model 负责训练模型，并设置 callback 参数，保存训练中的最佳模型.本算法在实验过程中，采用 PYTHON 语言实现，具体开发环境为 PYTHON3.6、TENSORFLOW 1.4.0 以及 KERAS 2.2.0.

3 实　验

3.1 数据集

数据集来自笔者学校的慕课平台 CourseGrading(http://cstlab.jsnu.edu.cn)，该网站包含大量的学生在线 C++练习题目，类似最大公约数、水仙花数、字符串匹配、插入排序、链表操作、BFS 等问题，其中每道题取部分学生源代码并抽取出代码中的函数，同一道题的源代码认为是语义相似的，不同题目的源代码是不相似的.同时，在选取训练样本代码语料库的时候，会刻意的选择那些通过 OJ(online judge)平台测试的代码，也就是说确保选择的样本是解决问题的正确代码，这一筛选条件的目的主要是为了代码的功能相同.数据集中一共包含 35 道题目，每道题目可能包含不同的函数个数，一共抽取了 904 个函数，817 216 条源代码对，其中 38 460 条相似源代码对，778 756 条不相似源代码对.

3.2 实验过程

本文以上面整理的数据集作为代码语料库，以词嵌入向量作为参照，对比 SICE 以及 SICE-CNN 两种模型计算结果.所有模型代码均基于深度学习框架 Keras 实现，在模型优化阶段，以交叉熵作为损失函数，并选择自适应学习率的 Adam 作为优化算法.训练过程采用留出法，数据集的 80%用作训练，20%作为测试数据.并以如下方法来分别计算两种检测结果与正确标签之间的查准率(P, precision)、查全率(R, recall)以及 F_1 值(F_1-score)数据：

$$P = \frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}} + N_{\mathrm{FP}}},$$

$$R = \frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}} + N_{\mathrm{FN}}},$$

$$F_1 = \frac{2PR}{P+R},$$

其中，N_{TP}(true positive)为预测为 1 且正确预测的样本数，N_{FP}(false positive)为预测为 1 实际为 0 的样本数量，N_{FN}(false negative)为预测为 0 但实际为 1 的样本数量.

3.3 实验结果

将词嵌入、SICE、SICE-CNN 三种方法在我们的数据集上各运行 10 遍,取其平均检测结果,结果如表 1.其中,第 1 列是相似性计算方法,第 2 列~第 4 列分别为相应方法测试结果的 N_{TP},N_{FN},N_{FP} 的平均值,第 5 列 ~ 第 7 列为各方法的查准率、查全率和 F_1 值.根据表 1,可以看出:SICE 方法和融合 SICE 的 CNN 方法的表现总体上优于单独使用词嵌入向量方法,平均 F_1 值分别提高了 9%和 19%.其中,SICE、SICE-CNN 方法在查全率得的优势比较明显,分别提高了 18%和 41%,但两种方法的查准率都有所降低.同时,为了测试每种方法的最好性能,对相似性阈值做了测试,阈值从 0 开始按照步长 0.1 逐步增加到 1.测试结果如图 4 所示,图中的横坐标表示阈值的变化,从图中可以看出,前两种方法当阈值约为 0.9 左右时,F_1 性能达到最好,第三种方法,阈值约为 0.6 时,性能最佳.

表 1 相似性计算结果

Table 1 Similarity results

	N_{TP}	N_{FN}	N_{FP}	P/%	R/%	F_1/%
word embedding	30 328	8 132	42 022	79	42	55
SICE	26 610	11 850	17 842	69	60	64
SICE-CNN	25 662	12 798	5 202	66.7	83.1	74

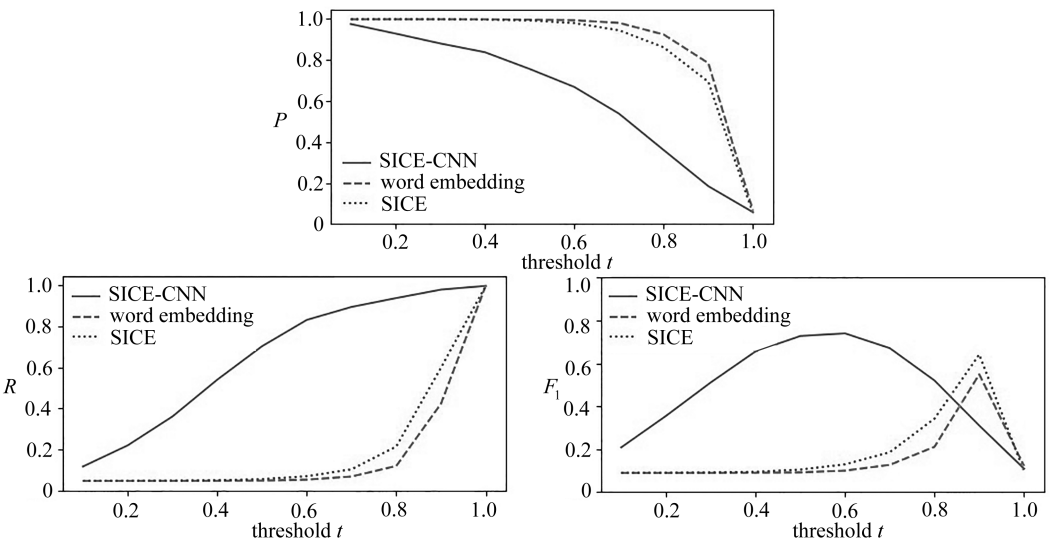

图 4 性能随阈值参数变化情况

Fig. 4 Performance changes with the threshold parameter ([0.1,1])

4 总结和未来工作

在本文中,将 TF-IDF 统计信息融合到词嵌入以及 CNN 的方法中.通过分析大量源代码的文本信息和语义信息,深度神经网络模型通过反复迭代提取代码功能相似的特征,最终生成分类器模型.总体上,本文所提出的方法性能优于一般的词向量方法,而且受阈值变化影响较为缓和,但在精准度上还有待提高.

本文方法虽然将源代码中词的统计信息融入到模型中,但是没有考虑保留字、用户自定义和操作符等不同类型,这可能也是造成精度不高的原因.因此,接下来,笔者将进一步细化词类型,并增大数据集验证实验效果.

附录 SICE-CNN 源代码相似性计算

```
Input: two matrices forsource code
Parameters: max_length, emded_dim, rate, filter_num
Output: similarity
def get_model( )://定义 SICE-CNN 结构
left_seq = Input( )//定义输入层
right_seq = Input( )
input_seq = Input( )
for fsize in filter_sizes: //定义卷积层
conv = Conv1D( )( input_seq)
maxpool = MaxPooling1D( pool_size)( conv) //定义 pooling 层
merge = Concatenate( axis=1)( all_convs)//定义 merge 层
output = Dropout( rate=0.2)( merge)//定义输出层
cnn_model = Model( input_seq, output)//构建模型
cnn_left = cnn_model( left_seq)
cnn_right = cnn_model( right_seq)
prediction = Dot( axes=1, normalize=True)( [ cnn_left, cnn_right] ) //余弦相似度作为预测结果
siamese_model = Model( inputs=[ left_seq, right_seq] , outputs=prediction)
return siamese_model
    def train_model( siamese_model, train_doc, valid_doc, epoch=10, batch_size=128)
siamese_model=get_model( )
training_generator = DataGenerator( train_doc)//生成训练数据
valid_generator = DataGenerator( valid_doc)//生成验证数据
json_string = siamese_model.to_json( )//保存 CNN 网络的结构
//按批送入数据进行训练
siamese_model.fit_generator( training_generator,epochs, valid_generator)
if __name__ == "__main__":
train_data,test_data,valid_data=load_data( )//分割并装载训练、测试、验证数据
model=get_model( ) //获取 model
model.load_weight( weight_path)//使用预训练的 TF-IDF 加权词向量
train_model( model, train_path, valid_path, max_length, epoch=20, batch_size=32)//训练模型
predict_data( test_path, arch_path, weight_path, res_loss_path, max_length)//预测结果
```

参考文献(References):

[1] KAMIYA T, KUSUMOTO S, INOUE K. CCFinder: a multilinguistic token-based code clone detection system for large scale source code[J]. *IEEE Transactions on Software Engineering*, 2002, **28**(7): 654-670.

[2] BELLON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools[J]. *IEEE Transactions on Software Engineering*, 2007, **33**(9): 577-591.

[3] LIU C, CHEN C, HAN J,et al. GPLAG: detection of software plagiarism by program dependence graph analysis[C]//*Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Philadelphia, PA, USA, 2006.

[4] COSMA G, JOY M. Towards a definition of source-code plagiarism[J]. *IEEE Transactions on Education*, 2008, **51**(2): 195-200.

[5] COSMA G, JOY M. An approach to source-code plagiarism detection and investigation using latent semantic analysis[J]. *IEEE Transactions on Computers*, 2012, **61**(3): 379-394.

[6] MENS K, LOZANO A. *Source Code-Based Recommendation Systems: Recommendation Systems in Software Engineering*[M]. Springer, 2014: 93-130.

[7] MCMILLAN C, POSHYVANYK D, GRECHANIK M, et al. Portfolio: searching for relevant functions and their usages in millions of lines of code[J]. *ACM Transactions on Software Engineering and Methodology*, 2013, **22**(4): 1-30. DOI: 10.1145/2522920.2522930.

[8] RAGKHITWETSAGUL C, KRINKE J, CLARK D. A comparison of code similarity analysers[J]. *Empirical Software Engineering*, 2017, **23**: 2464-2517.

[9] ROY C K, CORDY J R. NICAD: accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C]//*Proceedings of IEEE International Conference on Program Comprehension*. 2008: 172-181.

[10] BAXTER I D, YAHIN A, MOURA L, et al. Clone detection using abstract syntax trees[C]// *Proceedings of the Conference on Reverse Engineering*. Benevento, Italy, 2006: 368-377.

[11] CHAE D K, HA J, KIM S W, et al. Software plagiarism detection: a graph-based approach [C]//*Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management*. ACM, 2013: 1577-1580.

[12] HINDLE A, BARR E T, SU Z. On the naturalness of software[C]//*2012 34th International Conference on Software Engineering* (*ICSE*). Zurich, Switzerland, 2012: 837-847.

[13] KARAIVANOV S, RAYCHEV V, VECHEV M T. Phrase-based statisticaltranslation of programming languages[C]//*Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. Portland, Oregon, USA, 2014: 173-184.

[14] RAYCHEV V, VECHEV M, YAHAV E. Code completion with statistical language models[C]// *Proceedings of the 35th ACM Sigplan Conference on Programming Language Design and Implementation*. Edinburgh, United Kingdom, 2014: 419-428.

[15] NGUYEN A T, NGUYEN T T, NGUYEN T N. Divide-and-conquer approach for multi-phase statistical migration for source code(T)[C]//*Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. Lincoln, NE, USA, 2016: 585-596.

[16] 张峰逸, 彭鑫, 陈驰, 等. 基于深度学习的代码分析研究综述[J]. 计算机应用与软件, 2018, **35**(6): 9-17.(ZHANG Fengyi, PENG Xin, CHEN Chi, et al. Research on code analysis based on deep learning[J]. *Computer Applications and Software*, 2018, **35**(6): 9-17.(in Chinese))

[17] 陈秋远, 李善平, 鄢萌, 等. 代码克隆检测研究进展[J]. 软件学报, 2019, **30**(4): 962-980. (CHEN Qiuyuan, LI Shanping, YAN Meng, et al. Code clone detection: a literature review [J]. *Journal of Software*, 2019, **30**(4): 962-980.(in Chinese))

[18] TUFANO M, WATSON C, GABRIELE B, et al. Deep learning similarities from different representations of source code[C]//*Proceedings of the 15th International Conference on Mining Software Repositories*. New York, USA, 2018: 542-553.

[19] HELLENDOORN V J , DEVANBU P. Are deep neural networks the best choice for modeling

source code? [C]//*Proceedings of the 11th Joint Meeting*. Paderborn, Germany, 2017: 763-773.

[20] HALSTEAD M H. *Elements of Software Science*[M]. New York: Elsevier North-Holland, 1977.

[21] KOMONDOOR R, HORWITZ S. Using slicing to identify duplication in source code[C]//*Proceedings of International Symposium on Static Analysis*. Berlin, Heidelberg, 2001.

[22] ARROYO-FERNÁNDEZ I, MÉNDEZ-CRUZ C F, SIERRA G, et al. Unsupervised sentence representations as word information series: revisiting TF-IDF[J]. *Computer Speech & Language*, 2019, **56**: 107-129.

[23] 何绪飞, 艾剑良, 宋智桃. 多元数据融合在无人机结构-健康监测中的应用[J]. 应用数学和力学, 2018, **39**(4): 395-402.(HE Xufei, AI Jianliang, SONG Zhitao. Multi-source data fusion for health monitoring of unmanned aerial vehicle structures[J]. *Applied Mathematics and Mechanics*, 2018, **39**(4): 395-402.(in Chinese))

[24] NGUYEN A T, NGUYEN T D, PHAN H D,et al. A deep neural network language model with contexts for source code[C]//*Proceedings of IEEE International Conference on Software Analysis*. Campobasso, Italy, 2018: 323-334.

[25] OTTENSTEIN K J. An algorithmic approach to the detection and prevention of plagiarism[J]. *ACM SIGCSE Bulletin*, 1976, **8**(4): 30-41.

[26] WHITE M, TUFANO M, VENDOME C,et al. Deep learning code fragments for code clone detection[C]//*Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (*ASE*). Singapore, 2016: 87-98.

[27] LAM A N, NGUYEN A T, NGUYEN H A,et al. Combining deep learning with information retrieval to localize buggy files for bug reports[C]//*Proceedings of 2015 30th IEEE/ACM International Conference on Automated Software Engineering*(*ASE*). Lincoln, NE, USA, 2015: 476-481.

[28] HUO X, THUNG F, LI M. Deep transfer bug localization[J]. *IEEE Transactions on Software Engineering*, 2019. DOI: 10.1109/TSE.2019.2920771.

[29] MOU L, LI G, JIN Z, et al. *TBCNN: a Tree-Based Convolutional Neural Network for Programming Language Processing*[M]. Eprint Arxiv, 2014.

[30] WHITE M, TUFANO M, MARTÍNEZ M,et al. Sorting and transforming program repair ingredients via deep learning code similarities[C]//*Proceedings of 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering* (*SANER*). Hangzhou, China, 2019: 479-490.

[31] MIKOLOV T, SUTSKEVER I, KAI C, et al. Distributed representations of words and phrases and their compositionality[J]. *Advances in Neural Information Processing Systems*, 2013, **26**: 3111-3119.

[32] YE X, SHEN H, MA X, et al. From word embeddings to document similarities for improved information retrieval in software engineering[C]//*Proceeding of IEEE/ACM International Conference on Software Engineering*. 2016.

[33] NGUYEN T D, NGUYEN A T, PHAN H D, et al. Exploring API embedding for api usages and applications[C]//*Proceedingof IEEE/ACM International Conference on Software Engineering*. Buenos Aires, Argentina, 2017: 438-449.

[34] CHEN C, XING Z, WANG X. Unsupervised software-specific morphological forms inference from informal discussions[C]//*Proceeding of IEEE/ACM International Conference on Software Engineering*. Buenos Aires, Argentina, 2017: 450-461.

[35] HAO P, MOU L, GE L, et al. Building program vector representations for deep learning[C]//*Proceeding of International Conference on Knowledge Science*. 2015: 547-553.

A Source Code Similarity Approach Based on Improved Convolutional Neural Networks

XIE Chunli, LIN Jiangxu, LIU Xiaoyang, ZHANG Wenbin, HUANG Junwei

(*School of Computer Science & Technology, Jiangsu Normal University, Xuzhou, Jiangsu* 221116, *P.R.China*)

Abstract: The source code similarity refers to the functional similarity of different code segments, which touches off important research in the field of software engineering. The existing methods mainly extracted texts and structure features manually from source codes to calculate the similarity based on the statistical information in disregard of the semantic characteristics of codes. To solve this problem, a source code similarity detection method based on the CNN was proposed. First, the source code was represented through word embedding to obtain the vector information of word embedding. Second, the CNN training model was constructed to learn the embedded representation of source code documents. Finally, the cosine similarity value of source code pairs was calculated. Experiments show that, the proposed method can certainly improve the performance with respect to the semantic similarity of source codes.

Key words: deep learning; convolutional neural network; code similarity; word embedding

Foundation item: The National Natural Science Foundation of China (61773185; 61877030; 61502212)

引用本文/Cite this paper:

谢春丽, 蔺疆旭, 刘小洋, 张文斌, 黄军伟. 改进的卷积神经网络源代码相似性度量方法[J]. 应用数学和力学, 2019, **40**(11): 1235-1245.

XIE Chunli, LIN Jiangxu, LIU Xiaoyang, ZHANG Wenbin, HUANG Junwei. A source code similarity approach based on improved convolutional neural networks[J]. *Applied Mathematics and Mechanics*, 2019, **40**(11): 1235-1245.