



基于分层特征的代码克隆检测方法

张冬梅,陈永乐,杨玉丽

(太原理工大学 信息与计算机学院,山西 晋中 030600)

摘 要:针对现有代码克隆检测方法通常存在标记表示单一而抽象语法树构造复杂的问题,提出一种结合分层特征的代码克隆检测方法。使用双层双向长短期记忆网络提取行级和全局代码层次的深层语义信息,挖掘目标代码的语义特征。引入注意力机制调整重要标记及代码行的影响权重,增强语义形式的代码克隆检测效果,并采用softmax分类器识别克隆代码。实验结果表明,该方法的召回率和精确度分别为91%和97%,相比NICAD、CCIS、CCLearner方法对于复杂语义形式的克隆代码具有更好的检测效果。

关键词:标记转换;分层特征;双向长短期记忆网络;注意力机制;代码克隆检测

开放科学(资源服务)标志码(OSID):



中文引用格式:张冬梅,陈永乐,杨玉丽.基于分层特征的代码克隆检测方法[J].计算机工程,2021,47(10):125-131.

英文引用格式:ZHANG D M, CHEN Y L, YANG Y L. Code clone detection method based on hierarchical feature[J]. Computer Engineering, 2021, 47(10): 125-131.

Code Clone Detection Method Based on Hierarchical Feature

ZHANG Dongmei, CHEN Yongle, YANG Yuli

(College of Information and Computer, Taiyuan University of Technology, Jinzhong, Shanxi 030600, China)

[Abstract] The existing code clone detection methods usually have the problem of single mark representation and complex abstract syntax tree structure. To address the problem, a code clone detection method is proposed based on hierarchical features. The method employs two-layer Bi-directional Long Short-Term Memory (Bi-LSTM) networks to extract deeper semantic information at the line level and global code level respectively. On this basis, the semantic features of the target code are mined. Then the attention mechanism is introduced to adjust the influence weight of important tokens and code lines, and thus enhance the performance of code clone detection for complex semantics. Finally, the softmax classifier is used to determine whether the target code is cloned. Experimental results show that the proposed method displays recall rate of 91% and precision of 97%, providing better performance than the NICAD, CCIS and CCLearner methods in code clone detection for complex semantics.

[Key words] token conversion; hierarchical feature; Bi-directional Long Short-Term Memory (Bi-LSTM) network; attention mechanism; code clone detection

DOI: 10.19678/j.issn.1000-3428.0058958

0 概述

代码克隆是指存在于代码库中两个及以上相同或者相似的源代码片段^[1],当开发人员通过复制、粘贴、修改等方式重用现有代码片段时会产生代码克隆^[2]。现有研究表明一个软件系统的全部代码库中平均有7%~23%是克隆代码^[3-4],例如, Linux中存在22.3%^[5]的代码克隆, JDK中存在29%^[6]的代码克隆。代码克隆在一定程度上可提高开发效率,降低函数

调用的时间成本,但也增加了软件维护的成本,例如:克隆一段含有未知bug的代码,会导致bug传播^[7],维护者需要检查所有克隆代码中是否存在该bug^[8];修改一段代码需要对该段代码中的所有克隆进行一致性修改,若修改不一致则会引入新的bug,降低软件整体质量。目前,研究人员提出4种代码克隆类型^[9]。类型1表示除了空格和注释之外,两个代码片段完全相同的代码对。类型2表示除了变量名、类型名和函数名之外都相同的代码对。类型3

基金项目:山西省重点研发计划(201903D121121);山西省自然科学基金面上项目(201901D211076)。

作者简介:张冬梅(1995—),女,硕士研究生,主研方向为信息安全;陈永乐(通信作者),副教授、博士;杨玉丽,讲师、博士。

收稿日期:2020-07-16 修回日期:2020-09-30 E-mail: xjzdm0000@163.com

表示有若干语句的增删,或使用不同的标识符、文字、类型、空格、布局和注释,但是依然相似的代码对。类型4表示相同功能的异构代码,在文本或者语法上不相似,但在语义上具有相似性。

为维护软件质量、检测和防止新的 bug 及降低开发风险和成本,研究人员提出多种代码克隆检测技术和工具^[10],然而现有检测方法除了 token 转换以外,其他方法都需要将源代码转换为抽象语法树或者程序依赖图等,中间过程复杂且成本高,难以扩展应用范围^[11]。虽然现有 token 方法因检测成本低、过程简单,成为有效的克隆检测方法,但目前基于 token 的克隆检测研究中通常使用字符串匹配方法或各 token 类在代码中出现频次的相似性度量方法,并没有挖掘代码内的语义信息。

为解决复杂语义形式的代码克隆检测问题,本文将代码表征为统一标记,利用基于注意力机制的双层(Bi-directional Long Short-Term Memory, Bi-LSTM)网络直接从标记描述的代码中提取出深层语义特征,将基于注意力机制的 Bi-LSTM 网络嵌入 Siamese 结构^[12-13]中,使代码克隆检测问题转化为二分类问题,即检测代码对为克隆对或非克隆对,并在已知克隆对和非克隆对的数据集上训练检测模型。

1 相关工作

1.1 代码克隆检测技术

目前,代码克隆检测主要包括基于文本、标记、树、图、度量这5种方法^[2]:

1) 基于文本的代码克隆检测方法使用基于行的字符串匹配算法^[14-15]。该方法简单快速,主要针对类型1的克隆,但不能检测复杂类型。

2) 基于标记的代码克隆检测方法移除了源码中的空格和注释,将代码转换为统一的标记,利用后缀树比配算法^[3]、最长公共子序列匹配算法^[6]或比对各标记类型出现次数的相似性^[16-17]进行克隆检测。该方法对格式化和重命名更改具有较强的鲁棒性,可以处理类型1、类型2、类型3甚至是类型4的克隆,但由于标记的不同排列产生的代码功能也不尽相同,因此在基于标记的克隆检测方法中应考虑语义信息。

3) 基于树的代码克隆检测方法对目标代码生成一个抽象语法树,利用树匹配算法或将树嵌入空间检测相似的子树^[18-19]。基于树的代码克隆检测方法考虑了代码的语法结构,可容忍语句数量的变化,并检测类型1和类型4的代码克隆。但由于该方法需要大量的时间开销和内存占用,因此无法扩展到大型代码库^[20]。

4) 基于图的代码克隆检测方法为代码生成控制流图(Control Flow Graph, CFG)或程序依赖图(Procedure Dependence Graph, PDG),使用图匹配或图嵌入算法查找相似的代码^[11,21]。但该方法成本较高,不同编程语言使用的转换工具存在较大差异,且扩展性差。

5) 基于度量的代码克隆检测方法收集代码中不同类型的度量值^[3,22]。但由于该方法度量值的不同排列产生的代码语义也不同,因此缺失了大量的代码语法或语义信息。

1.2 Bi-LSTM 模型

循环神经网络(Recurrent Neural Network, RNN)用于处理时间序列问题,特点是带有循环的网络,能够有效利用之前的信息。但 RNN 的记忆和存储能力有限,随着序列间隔的增大, RNN 很难学习到久远的信息,存在梯度消失和梯度爆炸问题。MNIH^[23]等提出并实现了长短时记忆(Long Short-Term Memory, LSTM)网络, LSTM 中每个模块有3个门和1个记忆单元,解决了长期依赖问题,但 LSTM 无法利用序列的下文信息。为解决该问题,研究人员提出 Bi-LSTM。Bi-LSTM 由时序相反的2个 LSTM 构成,且连接同一个输出层,前向 LSTM 负责记忆上文信息,后向 LSTM 负责记忆下文信息,为处理时间序列问题起到了促进作用。

1.3 注意力机制

● 注意力机制是一种模拟人脑注意力机制的模型,通过计算注意力概率分布对输入的关键部分分配较多的注意力,对其他部分分配较少的注意力,且已在图像处理^[24]、自然语言处理^[25]、情感分类^[26]等领域中得到广泛应用。由于基于标记的代码克隆检测方法可以较好地描述代码实现,运行成本低、扩展性好,因此本文选择基于标记的代码克隆检测方法检测代码克隆,并引入基于注意力机制的双层 Bi-LSTM 网络来挖掘代码中更多的上下文语义特征。

2 检测框架

基于分层特征的代码克隆检测框架如图1所示。代码克隆检测过程分为代码表征、代码语义提取、检测分类3个阶段。挖掘代码对中的代码语义信息是构建代码克隆检测模型的关键。为规避不同变量名、函数名的影响,代码表征阶段将代码中的常量、变量、保留字、运算符等代码术语统一转换为标记,建立标记向量化模型,将标记转换为模型可接受的输入形式。语义提取阶段采取 Siamese 结构,建立基于注意力机制的双层 Bi-LSTM 模型提取行级特征和全局代码特征。代码克隆对的检测分类阶段将代

码对的语义特征进行拼接作为一个样本的有效特征,并使用 softmax 分类器判断代码对的所属类别,其中,0表示非克隆对,1表示克隆对。

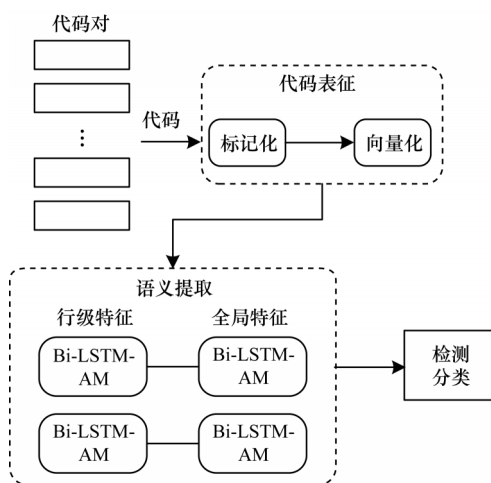


图1 基于分层特征的代码克隆检测框架

Fig.1 Code clone detection framework based on hierarchical features

3 基于分层特征的代码克隆检测

3.1 代码表征

代码克隆检测过程中的代码表征步骤具体如下:

1)转换规则。本文采用如下转换规则将代码术语统一转换为标记:(1)数字常量用NUM替代;(2)字符串用string替代;(3)保留运算符;(4)保留字;(5)标点符号用PUN代替。代码转换实例如图2所示。

```
1. def get_template(template_name, using=None):
2.     chain = []
3.     engines = _engine_list(using)
4.     for engine in engines:
5.         try:
6.             return engine.get_template(template_name)
7.         except TemplateDoesNotExist as e:
```

(a)原始代码

```
1. def string pun string pun string = None pun pun
2. string = pun pun
3. string = string pun string pun
4. for string in string pun
5. try pun
6. return string pun string pun string pun
7. except string as string pun
```

(b)标记代码

图2 代码转换实例

Fig.2 Code conversion example

2)标记向量化。本文采用 word2vec^[27]中实现的 skip-gram 模型来创建标记向量化模型。图3给出了向量化模型的训练过程。

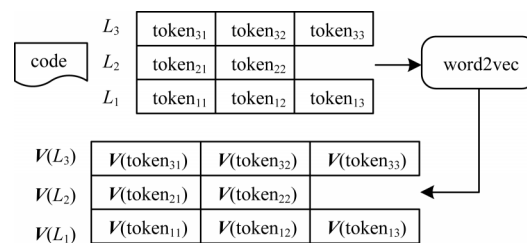


图3 向量化模型的训练过程

Fig.3 Training process of vectorized model

目标代码 code 以 L_m 为单位生成相应的代码流,一行一行地将每个标记输入到模型中进行训练。在图3中, $L_m = (\text{token}_{m1}^1, \text{token}_{m2}^2, \dots, \text{token}_{mt}^t)$, t 表示第 m 行代码的标记个数, token_{mt}^t 表示第 m 行代码中第 t 个标记,对应的向量化表示为 $V(\text{token}_{mt}^t) \in \mathbb{R}^d$, d 为标记嵌入的维度,则 L_m 行的向量化表示为 $V(L_m) = (V(\text{token}_{m1}^1), V(\text{token}_{m2}^2), \dots, V(\text{token}_{mt}^t)) \in \mathbb{R}^{d \times t}$ 。

3.2 语义提取

目标代码由多行语义不同的代码组成,标记和代码行的不同排列产生的代码语义不尽相同。不同于文献[3]将不同标记类型的出现次数作为克隆检测的属性,本文使用双层 Bi-LSTM 直接从标记后的代码表示中提取能够体现前后标记关系、上下行关系的深层语义信息。不同词性的单词对文本分类有不同的作用^[28],不同标记和不同行对代码克隆检测也有不同的作用,因此本文引进注意力机制对重要的标记、行赋予更大的注意力权重,提升代码克隆检测效果。图4给出了基于分层特征的代码克隆检测模型。

给定目标代码对,代码行数分别为 M, N , 假设代码 $\text{code}_i, i \in [1, 2]$ 的第 m 行代码包含 t 个标记,标记记为 token ,该行代码的代码流表示为 $L_{im} = (\text{token}_{im1}^1, \text{token}_{im2}^2, \dots, \text{token}_{imt}^t)$, 当 $i = 1$ 时 $m \in [1, M]$, 当 $i = 2$ 时 $m \in [1, N]$; 代码 $\text{code}_i = (L_{i1}, L_{i2}, \dots, L_{in})$, n 为目标代码的最大行数,当 $i = 1$ 时 $n = M$, 当 $i = 2$ 时 $n = N$ 。

1)行级代码层次。对于一个包含 t 个标记的行代码 $L_{im} = (\text{token}_{im1}^1, \text{token}_{im2}^2, \dots, \text{token}_{imt}^t)$ 的向量表示为 $V(L_m) = (V(\text{token}_{im1}^1), V(\text{token}_{im2}^2), \dots, V(\text{token}_{imt}^t))$, 作为 Bi-LSTM 网络的输入,依次输出标记的隐藏状态 h_{im}^j , 如式(1)所示:

$$h_{im}^j = \text{Bi-LSTM}(V(\text{token}_{im}^j)) \quad (1)$$

其中: $i = 1, m \in [1, M]; i = 2, m \in [1, N]; j \in [1, t]$ 。

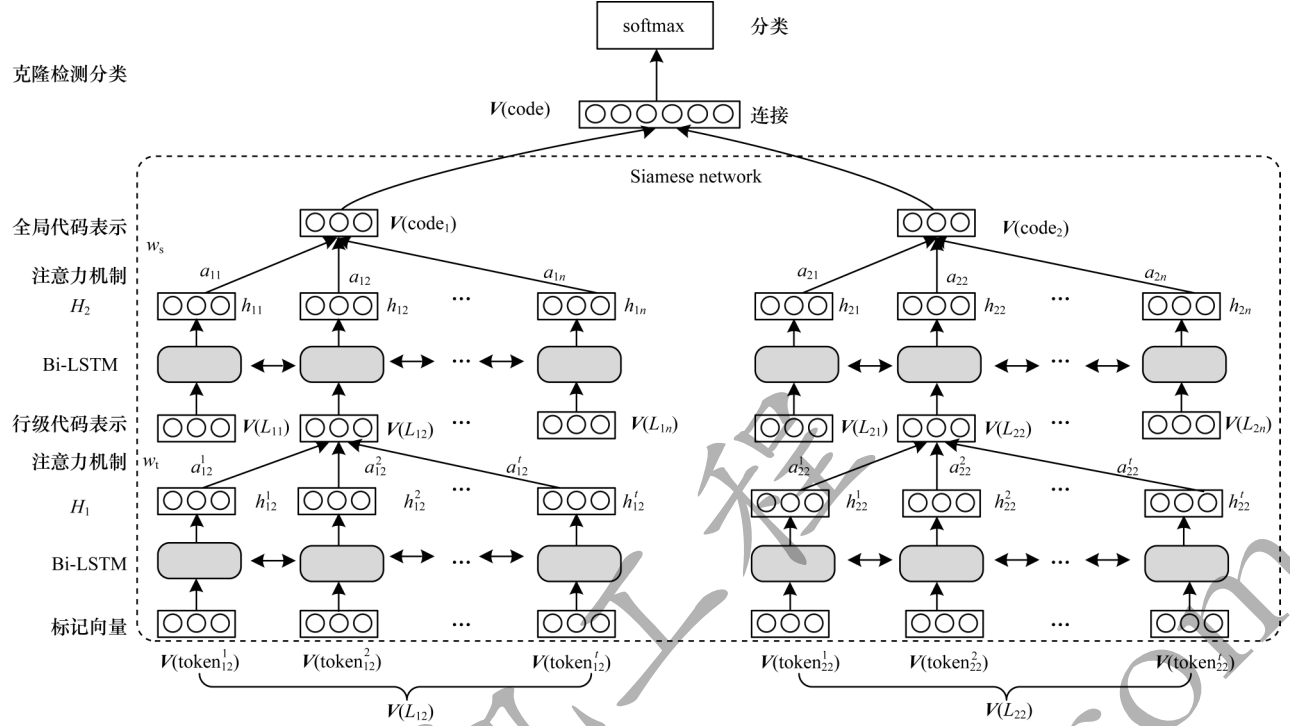


图4 基于分层特征的代码克隆检测模型

Fig.4 Code clone detection model based hierarchical features

由于不同标记对代码的语义表示有不同的作用,因此引入注意力机制,提升特定标记对于行代码的重要程度。通过softmax激活函数得到归一化的注意力权重 α_{im}^j ,将注意力权重与隐藏层表示 h_{im}^j 加权求和得到局部行代码的向量表示,如式(2)~式(4)所示:

$$u_{im}^j = \tanh(w_i h_{im}^j + b_i) \quad (2)$$

$$\alpha_{im}^j = \text{softmax}(u_{im}^j, u_i) \quad (3)$$

$$V(L_{im}) = \sum_{j=1}^t \alpha_{im}^j h_{im}^j \quad (4)$$

其中: w_i 和 b_i 分别为行级注意力层对应神经元的权重和偏置值; u_{im}^j 为 h_{im}^j 的隐藏单元; u_i 为行级上下文向量,用来衡量标记的重要程度,随机初始化并在训练过程中与其他参数共同训练。

2)全局代码层次。将行级代码层次中输出的行向量 $V(L_{im})$ 组成的代码流作为Bi-LSTM网络的输入得到 $(V(L_{i1}), V(L_{i2}), \dots, V(L_{in}))$, n 表示代码最大行数。经过Bi-LSTM网络输出行代码的隐藏状态 h_{im} ,如式(5)所示:

$$h_{im} = \text{Bi-LSTM}(V(L_{im})) \quad (5)$$

其中: $i=1, m \in [1, M]; i=2, m \in [1, N]$ 。

在全局代码层次引入注意力机制来标记对代码克隆检测更重要的代码行。通过softmax激活函数得到归一化的注意力权重 α_{im} ,并与隐藏层输出 h_{im} 加权求和得到全局代码的向量表示,如式(6)~式(8)所示:

$$u_{im} = \tanh(w_s h_{im} + b_s) \quad (6)$$

$$\alpha_{im} = \text{softmax}(u_{im}, u_s) \quad (7)$$

$$V(\text{code}_i) = \sum_{m=1}^n \alpha_{im} h_{im}$$

$$i=1, n=M; i=2, n=N \quad (8)$$

其中: w_s 和 b_s 分别为全局注意力层对应神经元的权重和偏置值; u_{im} 为 h_{im} 的隐藏单元; u_s 为全局上下文向量,用来衡量行的重要程度,随机初始化并在训练过程中与其他参数共同训练。

3.3 克隆检测分类

通过代码对 $(\text{code}_1, \text{code}_2)$ 的向量表示 $(V(\text{code}_1), V(\text{code}_2))$,将两个向量表示连接作为一个样本的有效特征,即 $V(\text{codepair})$ 。例如, $V(\text{code}_1) = (1.2300, -3.3350, \dots, 0.3887)$, $V(\text{code}_2) = (1.01, 0.47, \dots, 2.55)$,则 $V(\text{codepair}) = (1.2300, -3.3350, \dots, 0.3887, 1.0100, 0.4700, \dots, 2.5500)$,其中每个分量是构建分类模型需要的特征属性。最终使用softmax分类器,得到代码对的分类结果,如式(9)所示:

$$y = \text{softmax}(w V(\text{codepair}) + b) \quad (9)$$

其中: w 和 b 分别表示softmax分类器的权重和偏置。

3.4 模型训练

训练采用交叉熵作为优化的损失函数。若 y 为克隆检测对应的真实分类结果, \hat{y} 为模型分类结果,则损失函数如式(10)所示:

$$l = - \sum_i y_i \ln \hat{y}_i \quad (10)$$

4 实验与结果分析

爬取GitHub上的Python开源项目,得到7 000多个Python源文件的100多万行代码,通过提取注释和分词并根据标记转换规则对源代码进行处理构建语料库,利用word2vec工具训练语料库。标记嵌入向量维度为100,行级代码层次的注意力机制的权重与每行包含的标记数相同,全局代码层次的注意力机制的权重与目标代码包含的行数相同。本文采用Bi-LSTM生成标记表示,输出表示为100维,利用tensorflow框架来训练基于注意力机制的双层Bi-LSTM模型,设置Bi-LSTM的隐藏层神经元数目为300, batch size为64,学习率为0.001。本文代码克隆检测问题为二分类问题,即代码对为克隆对或非克隆对,因此采用召回率(Recall)、精确度(Precision)、F1分数3个度量指标来评价实验结果。

4.1 数据集

为训练和测试基于分层特征的代码克隆检测方法,本文采用CCIS^[28]中构建的数据集,该数据集包含6款开源项目中的215个源码片段、通过变异生成的275个克隆片段以及72个噪声片段,数据集信息如表1所示。数据集包括类型1、类型2、类型3的代码克隆对以及非克隆对,通过遍历统计出各类型数量如表2所示。

表1 数据集信息

Table 1 Dataset information

项目	版本	项目行数	基本功能
pandas	0.24.0	384 384	数据结构和数据分析工具
scipy	1.2.0	318 131	科学计算工具
django	2.1.4	333 179	高级Python Web框架
pytorch	1.0.0	244 439	端到端深度学习平台
scikit-learn	0.20.2	230 534	数据挖掘和数据分析工具
keras	2.2.4	66 156	高级神经网络API

表2 数据集类型设置

Table 2 Type setting in the dataset

项目名	项目值
类型1克隆对	116
类型2克隆对	87
类型3克隆对	72
克隆对总计	275
非克隆对总计	72
代码对总计	347

4.2 结果分析

4.2.1 注意力机制的有效性分析

本文在Bi-LSTM中引入注意力机制来赋予标记、行不同的注意力权重,帮助模型更好地进行代码克隆检测。以单行代码为例,“for string in string pun”,其中,“for”作为循环语句的保留字相比标点符号的“pun”对克隆检测具有更大的权重。图5给出了行代码中每个

标记被赋予的注意力权重,可以看出“for”被赋予了更大的权重,而“pun”被赋予的权重较小,说明不同标记对克隆检测所起的作用不同,而注意力机制能够提升特定标记对克隆代码检测的影响。

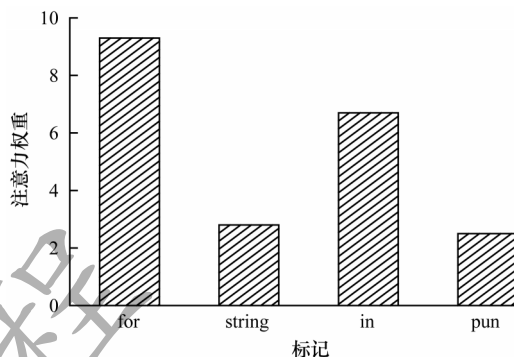


图5 不同标记的注意力权重分布

Fig.5 Attention weight distribution of different tokens

以多行代码对为例,代码1中各行代码具体如下: line1为“def string pun string pun string = None pun”; line2为“string = pun pun”; line3为“string = string pun string pun”; line4为“for string in string pun”; line5为“string”; line6为“return string pun string pun string pun”; line7为“except string as string”; line8为“string pun string pun string pun”。代码2相比代码1在最后一行多出line9为“raise string pun string pun string = string pun”。在克隆检测过程中,前8行作为克隆行应比最后一行非克隆行对检测效果的影响更大,通过得到各行代码的注意力权重并将其可视化,结果如图6所示,可以看出各行代码权重不同,但前8行的权重整体比第9行大,说明注意力机制赋予重要的行代码更大的注意力权重,可以提升特定行对克隆代码检测的影响。

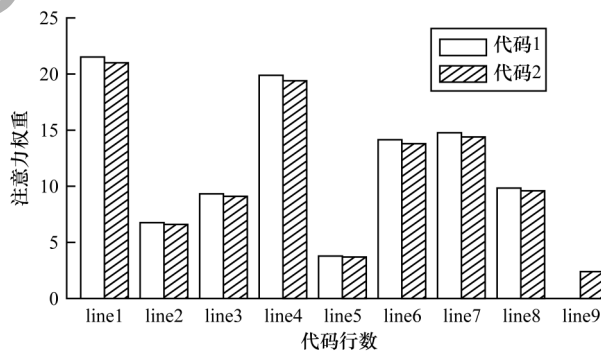


图6 不同代码行的注意力权重分布

Fig.6 Attention weight distribution of different code lines

4.2.2 检测模型效果对比

将本文模型与双层Bi-LSTM、基于注意力机制的双层LSTM(AM-LSTM)、双层LSTM在同一数据集上进行实验对比。为统一比较标准,所有模型的输入向量均由标记转换规则和标记向量化模型生

成,所有网络隐藏层节点数均设置为300。4种模型的检测结果如表3所示,可以看出Bi-LSTM比LSTM体现出更好的性能,因为Bi-LSTM考虑了标记间的前后信息和行间的上下文信息,提取的特征更有效,而本文模型相比传统模型检测效果更佳,主要原因为本文模型结合注意力机制,提升了关键标记对于克隆代码检测的影响,并且获取了更多的上下文代码语义信息。

表3 4种模型的检测结果对比

Table 3 Comparison of detection results of four models

模型	召回率/%	精确度/%	F1分数
本文模型	91	97	0.94
Bi-LSTM模型	90	97	0.93
AM-LSTM模型	90	96	0.93
LSTM模型	89	96	0.92

4.2.3 检测方法效果对比

为验证本文方法的有效性,将其与CCLearner^[3]、NICAD^[14]、CCIS^[28]方法在相同数据集上进行实验对比,其中,CCLearner是一种基于标记的克隆检测方法,NICAD是一种被广泛认可的克隆检测方法,CCIS是一种基于图的代码克隆检测方法。这3种方法可以检测Python语言中的克隆,且可检测前3种克隆类型。本文使用CCIS^[28]中构建的数据集来比较3种方法的效果。4种检测方法的精确度对比结果如图7所示,可以看出本文方法与NICAD、CCIS、CCLearner方法对6种项目的检测精确度都较高,几乎都可以达到100%,对于平均精确度而言,本文方法与CCLearner相同,CCIS与NICAD相同,本文方法高于CCIS与NICAD。4种检测方法的召回率对比结果如图8所示,可以看出根据平均召回率,CCLearner的代码克隆检测召回率相比NICAD高出7个百分点,相比CCIS高出2个百分点;本文方法的代码克隆检测召回率相比CCLearner高出2个百分点。

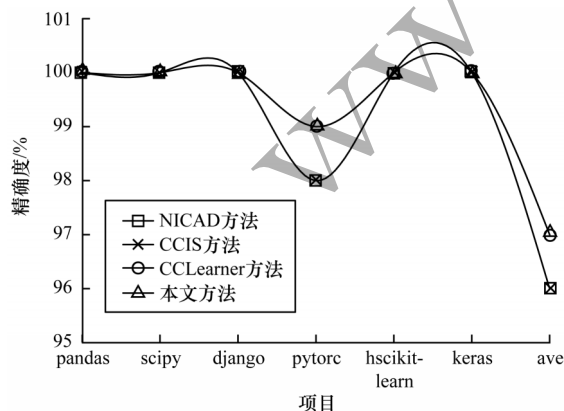


图7 4种方法的精确度对比

Fig.7 Comparison of precision of four methods

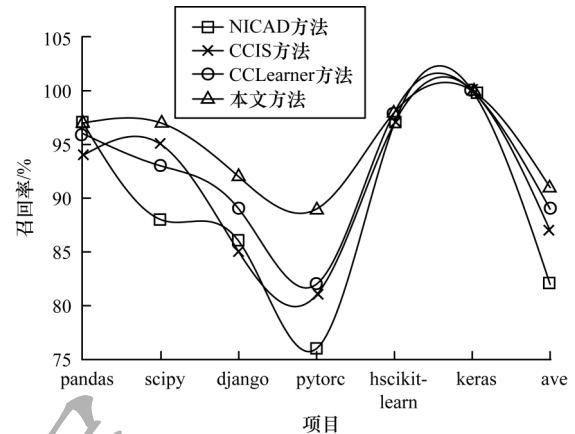


图8 4种方法的召回率对比

Fig.8 Comparison of recall rate of four methods

上述实验结果表明,本文提出的标记转换规则有利于增强代码克隆的类型,可在一定程度上规避不同变量名和函数名对检测效果的影响,同时基于分层特征的代码克隆模型进一步挖掘代码间隐藏的深层信息,提升了克隆检测效果。

5 结束语

本文提出一种基于分层特征的代码克隆检测方法,将代码转换成统一的标记表示,基于Siamese框架建立分层特征代码语义提取模型,提取代码对的行级代码特征和全局代码特征,并引入注意力机制提升克隆检测效果。实验结果表明,该方法的代码克隆检测召回率相比基于标记的代码克隆检测方法高出2个百分点,能更有效地检测克隆代码。后续将针对不同编程语言建立不同的语料库,并利用迁移学习技术设计适用于多编程语言的代码克隆检测方法。

参考文献

- [1] 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 Q Y, LI S P, YAN M, et al. Code clone detection: a literature review[J]. Journal of Software, 2019, 30(4): 962-980. (in Chinese)
- [2] ROY C K, CORDY J R. A survey on software clone detection research: TR 2007-541[R]. Kingston, Canada: Queen's University, 2007: 64-68.
- [3] LI L Q, FENG H, ZHUANG W J, et al. CCLearner: a deep learning-based clone detection approach[C]//Proceedings of 2017 IEEE International Conference on Software Maintenance and Evolution. Washington D. C., USA: IEEE Press, 2017: 249-260.
- [4] ROY C K, CORDY J R. An empirical study of function clones in open source software[C]//Proceedings of the 15th Working Conference on Reverse Engineering. Washington D. C., USA: IEEE Press, 2008: 81-90.
- [5] KIM M, SAZAWAL V, NOTKIN D, et al. An empirical study of code clone genealogies[J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 187-196.

- [6] KAMIYA T, KUSUMOTO S, INOUE K. CCFinder: a multilingual token-based code clone detection system for large scale source code[J]. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670.
- [7] JUERGENS E, DEISSENBOECK F, HUMMEL B, et al. Do code clones matter? [C]//Proceedings of the 31st International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2009: 485-495.
- [8] MONDAL M, RAHMAN M S, ROY C K, et al. Is cloned code really stable?[J]. Empirical Software Engineering, 2018, 23(2): 693-770.
- [9] 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.
- [10] MOSTAEEN G, SVAJLENKO J, ROY B, et al. CloneCognition: machine learning based code clone validation tool[C]//Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, USA: ACM Press, 2019: 1105-1109.
- [11] KRINKE J. Identifying similar code with program dependence graphs[C]//Proceedings of the 8th Working Conference on Reverse Engineering. Washington D. C., USA: IEEE Press, 2001: 301-309.
- [12] BROMLEY J, BENTZ J W, BOTTOU L, et al. Signature verification using a "Siamese" time delay neural network [J]. International Journal of Pattern Recognition and Artificial Intelligence, 1993, 7(4): 669-688.
- [13] CHOPRA S, HADSELL R, LECUN Y. Learning a similarity metric discriminatively, with application to face verification [C]//Proceedings of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Washington D. C., USA: IEEE Press, 2005: 539-546.
- [14] ROY C K, CORDY J R. NICAD: accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization [C]//Proceedings of the 16th IEEE International Conference on Program Comprehension. Washington D. C., USA: IEEE Press, 2008: 172-181.
- [15] LEE S, JEONG I. SDD: high performance code clone detection system for large scale source code [C]//Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. New York, USA: ACM Press, 2005: 140-141.
- [16] WANG P C, SVAJLENKO J, WU Y Z, et al. CCAaligner: a token based large-gap clone detector [C]//Proceedings of the 40th International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2018: 1066-1077.
- [17] SEMURA Y, YOSHIDA N, CHOI E, et al. CCFinderSW: clone detection tool with flexible multilingual tokenization [C]//Proceedings of the 24th Asia-Pacific Software Engineering Conference. Washington D. C., USA: IEEE Press, 2017: 654-659.
- [18] WEI H H, LI M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code [C]//Proceedings of the 26th International Joint Conference on Artificial Intelligence. Washington D. C., USA: IEEE Press, 2017: 3034-3040.
- [19] 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. Washington D. C., USA: IEEE Press, 2016: 87-98.
- [20] SAJNANI H, SAINI V, SVAJLENKO J, et al. SourcererCC: scaling code clone detection to big-code [C]//Proceedings of the 38th International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2016: 1157-1168.
- [21] WANG M, WANG P C, XU Y. CCSHarp: an efficient three-phase code clone detector using modified PDGs [C]//Proceedings of the 24th Asia-Pacific Software Engineering Conference. Washington D. C., USA: IEEE Press, 2017: 100-109.
- [22] HOCHREITER S, SCHMIDHUBER J. Long short-term memory [J]. Neural Computation, 1997, 9(8): 1735-1780.
- [23] MNH V, HEES N, GRAVES A, et al. Recurrent models of visual attention [C]//Proceedings of the 27th International Conference on Neural Information Processing Systems. New York, USA: ACM Press, 2014: 2204-2212.
- [24] YANG Z C, YANG D Y, DYER C, et al. Hierarchical attention networks for document classification [EB/OL]. [2020-06-11]. <https://arxiv.org/abs/1707.00896>.
- [25] WANG Y Q, HUANG M L, ZHU X Y, et al. Attention-based LSTM for aspect-level sentiment classification [C]//Proceedings of 2016 Conference on Empirical Methods in Natural Language Processing. Philadelphia, USA: Association for Computational Linguistics, 2016: 606-615.
- [26] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space [EB/OL]. [2020-06-11]. <https://arxiv.org/pdf/1301.3781.pdf>.
- [27] 路永和, 王鸿滨. 文本分类中受词性影响的特征权重计算方法 [J]. 现代图书情报技术, 2015(4): 18-25.
- LU Y H, WANG H B. Feature weighting method affected by part of speech in text classification [J]. New Technology of Library and Information Service, 2015(4): 18-25. (in Chinese)
- [28] WANG Y F, LIU D S, HOU M. Clone code detection based on image similarity [J]. Journal of Computer Applications, 2019, 39(7): 2074-2080. (in Chinese)
- 王亚芳, 刘东升, 侯敏. 基于图像相似度检测代码克隆 [J]. 计算机应用, 2019, 39(7): 2074-2080.