软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(9):2713−2728 [doi: 10.13328/j.cnki.jos.005990]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于程序层次树的日志打印位置决策方法*

贾 统[1], 李 影[2], 张齐勋[3], 吴中海[2,3]

[1](北京大学 信息科学技术学院,北京 100871)

[2](北京大学 软件工程国家工程研究中心,北京 100871)

[3](北京大学 软件与微电子学院,北京 102600)

通讯作者: 李影, E-mail: li.ying@pku.edu.cn

摘 要: 基于日志分析的故障诊断是智能运维的关键技术之一,然而该技术存在关键瓶颈——日志的质量.当今,由于程序开发人员缺乏日志打印规范和指导等问题,日志质量欠佳,因此实现日志的自动化打印决策以提升日志质量的需求日益迫切.关注自动化日志打印决策问题,与现有研究工作不同,提出一种基于程序层次树和逆序组合的特征向量生成方法,能够适用于不同编程语言编写的软件系统.此外,还利用迁移学习算法实现跨组件和跨软件系统的日志打印位置决策.在 3 个典型的应用场景——版本升级、组件开发和系统开发及 5 个流行的开源软件系统——OpenStack,Tensorflow,SaltCloud,Hadoop 和 Angel 上的实验表明:所述方法在 Java 系统中的日志打印决策准确率约为 95%,在 Python 系统中的日志打印决策准确率约为 70%.

关键词: 日志打印位置决策;程序层次树;迁移学习

中图法分类号: TP311

中文引用格式: 贾统,李影,张齐勋,吴中海.基于程序层次树的日志打印位置决策方法.软件学报,2021,32(9):2713−2728. http://www.jos.org.cn/1000-9825/5990.htm

英文引用格式: Jia T, Li Y, Zhang QX, Wu ZH. Automatic logging decision through program layered structure tree and transfer learning. Ruan Jian Xue Bao/Journal of Software, 2021,32(9):2713−2728 (in Chinese). http://www.jos.org.cn/1000-9825/5990.htm

Automatic Logging Decision Through Program Layered Structure Tree and Transfer Learning

JIA Tong[1], LI Ying[2], ZHANG Qi-Xun[3], WU Zhong-Hai[2,3]

[1](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](National Engineering Research Center for Software Engineering, Peking University, Beijing 100871, China)

[3](School of Software and Microelectronics, Peking University, Beijing 102600, China)

Abstract: With the development of AIOps, log-based failure diagnosis has become more and more important. However, this technique has a key bottleneck—the quality of logs. Today, the lack of log printing specifications and guidance for programmers is a key factor of poor log quality, thus the need of automatic logging decision so as to improve log quality is becoming urgent. This study focuses on automatic logging decision. Specifically, the aim is to propose a general logging point decision approach. Different from existing works, an automatic feature vector generation method is proposed based on program layered structure tree and reverse composition, which can be applied to software systems written in different programming languages. In addition, this study leverages transfer learning algorithms to achieve cross-component and cross-project logging point decision. The approach is evaluated on five popular open source software systems, namely, OpenStack, Tensorflow, SaltCloud, Hadoop, and Angel, in three typical application scenarios including software upgrading, new component development, and new project development. Results show that the proposed approach performs about 95% accuracy in Java projects and 70% accuracy in Python projects on average.

* 基金项目: 国家重点研发计划(2017YFB1002002)
Foundation item: National Key Research and Development Program of China (2017YFB1002002)
收稿时间: 2019-05-22; 修改时间: 2019-09-24; 采用时间: 2019-12-02

Key words: logging decision; program layered structure tree; transfer learning

随着人工智能技术的发展,Gartner 提出智能运维(AIOps)技术的概念[1],即:通过机器学习等算法分析从多种运维工具和设备收集而来的大规模数据,自动发现并实时响应,以增强 IT 运维能力和自动化程度[2].在 AIOps 技术趋势下,基于系统日志的大规模分布式系统自动化故障诊断技术发展迅速.这些技术通过分析和挖掘系统日志,构建刻画系统正常运行时的请求执行路径的模型,然后自动检测系统运行时的日志序列与模型之间的偏差,以实现精确异常检测和故障诊断.然而,这种基于日志的自动化故障诊断技术存在一个重要的瓶颈,即系统日志的质量.对大规模软件如 Apache,Squid,PostgreSQL 等的系统失效报告分析发现[3],77%的系统失效归结于几种常见错误模式,而这些错误模式中有超过一半(57%)并没有被记录在系统日志中,导致自动化故障诊断模型无从分析与检测,从而需要花费大量人工和时间成本来定位、诊断和修复这些错误.

现今,软件系统开发缺乏统一的日志打印标准和规范,其日志打印决策依赖于程序开发人员的个人理解和调试需求.由于软件开发人员的编程风格、领域知识和需求的不同,程序中日志打印的风格、内容、位置也大相径庭,由此引发分布式系统日志质量的参差不齐.有调研表明:即使是在国际领先的软件企业,也很难找到明确的高质量日志打印规范或指导;即使采用专业的日志打印框架如 Log4j,Self4j 等,程序开发人员仍需根据相对片面的领域知识进行日志打印决策[4].由于大规模分布式软件系统代码量巨大,且结构复杂,人工在系统中添加或修改日志几乎是不可能的.因此,实现日志的自动化打印决策并提高日志质量的需求越来越迫切.

高质量的日志应具备 3 个特征[5].

(1) 日志打印的位置合理,即系统运行时产生的日志能够反映系统运行时状态的变化;

(2) 日志中包含的信息丰富,即日志的文本信息能够帮助系统管理人员理解系统的运行行为;

(3) 日志打印的数量适当.过多的日志会增加系统运行时开销,同时冗余和无用的日志不利于从日志中搜索和辨识故障信息.

本文关注日志打印位置决策问题,为解决该问题,有研究人员提出通过代码分析的方法得到若干日志打印位置的规则[6]或日志打印位置[7];也有研究人员提出机器学习的方法,通过学习日志打印位置附近的代码特征,构建日志打印位置决策模型[4].然而,相关研究工作中存在两个重要问题:(1) 特征提取方法受限于特定编程语言,且日志打印位置决策的区域有限制,例如仅能够对 Exception 模块的日志打印位置进行决策;(2) 现有工作的日志打印位置决策模型无法有效处理不同软件系统在特征空间上的差异,很难实现跨软件系统的日志打印位置决策.因此,如何为由不同编程语言实现的不同软件系统实现自动化与智能化的日志打印位置决策并提高日志质量,仍然是一个亟待解决的难题.

针对该问题,考虑到已多次更新升级的成熟软件往往具有良好的日志打印策略[8],本文提出一种基于程序层次树与迁移学习的日志打印位置决策方法,通过有效迁移成熟软件系统的日志打印知识,来为目标系统决策日志打印位置.具体的,以代码块为单位切分软件系统代码,构建程序层次树提取代码块子向量,并采用子树上提和逆序组合的方式提取代码块的结构特征和上下文特征,以生成代码块特征向量;采用迁移学习的方法构建成熟软件系统与待决策的目标软件系统日志打印程序的共同特征空间,使用机器学习模型学习特征空间中的特征向量分布,以实现跨软件系统的日志打印位置的决策.本文的创新之处在于:(1) 提出一种通用的代码特征向量自动化提取方法,通过构建程序层次树,屏蔽编程语言与程序模块实现细节的异构性,以支持不同编程语言编写的软件系统;(2) 提出一种基于迁移学习的日志打印位置决策模型,利用迁移成分分析算法(transfer component analysis,简称 TCA)和联合分配适配算法(JointDistribution adaptation,简称 JDA)将不同系统特征向量的特征向量映射到新的公共特征空间,进而削减特征差异,以支持跨软件系统的日志打印位置决策.

在由两种编程语言实现的 5 个成熟的软件系统 Apache Hadoop[9],TencentAngel[10],OpenStack[11],SaltCloud[12]和 Tensorflow[13]的实验表明,本文所提出的方法在跨组件、跨相同编程语言的不同软件系统的日志打印位置决策均具有良好的效果.例如,在以 Hadoop 的代码作为训练集决策 Tencent Angle 的日志打印位置实验中,本方法达到了 0.969 的准确率和 0.925 的 $F1$.

本文第 1 节介绍日志打印位置决策的相关研究工作.第 2 节对日志打印决策问题进行明确.第 3 节详细介绍本文提出的日志打印位置决策方法.第 4 节给出相关实验,验证提出方法的效果.第 5 节总结本文的工作,并探讨未来的研究工作.

1 相关工作

1.1 基于规则的日志打印位置决策

这类方法通常使用统计分析方法从成熟软件系统中总结日志打印规则,并使用这些规则构建日志打印位置决策模型.Yuan[6]设计了一个 ErrLog 的工具,通过人工分析成熟软件系统的日志打印语句,总结日志打印的规则,并利用这些规则决策新的日志打印位置.该工具的关键问题在于:其仅关注 Error 或 Warn 级别的日志打印位置,并假设这些日志打印位置均位于异常处理程序中.Fu 等人[8]对微软的两个大规模分布式系统日志打印语句进行了分析,总结出其中的若干日志打印策略,然后利用这些策略构建了一个日志打印决策模型.该方法的不足在于:其仅对诸如异常处理,返回值校验等特殊的代码结构进行日志打印位置决策.

1.2 基于机器学习的日志打印位置决策

这类方法使用机器学习算法从成熟软件系统源码中学习日志打印语句周围的代码片段特征,进而构建日志打印位置决策模型.微软的 Fu 等人[8]利用决策树模型构建了一个日志打印位置决策模型,基于该工作,Zhu 等人[4]提出了一个日志打印位置决策工具 LogAdvisor,该工具通过提取系统源码的 3 种类别特征,构建了一个分类模型以决策一段代码是否应该添加日志打印语句.上述两种方法和工具的不足在于,其仅能针对异常处理和返回值校验的代码片段进行日志打印位置决策.另外,由于这两种方法提取的特征依赖于编程语言的语法支持,因此难以拓展到其他语言编写的软件系统中.例如,LogAdvisor 提取的特征中包括 Throw 和 Empty CatchBlock 的数量,但是在脚本语言中,通常少有这种类型的特征.Lal 等人[14]提出了 LogOptPlus 的日志打印位置决策方法,该方法通过提取 18 个统计类型特征和 10 个文本类型特征,利用机器学习算法构建日志打印位置决策模型.该方法的不足在于:其限制日志打印位置决策的区域为“if”和“catch”代码结构,难以拓展.

同基于规则的日志打印位置决策相比,基于机器学习的日志打印位置决策方法具备如下关键优势.

- 首先,日志打印的复杂性导致人工总结日志打印位置难以为继.日志打印决策需要考虑多方面因素,包括故障诊断、性能记录、安全程度和对系统带来的额外负载等,因此难以对不同的软件系统人工总结普适的日志打印策略;
- 基于机器学习的日志打印位置决策假设成熟软件系统中包含良好的日志打印规律,且这些规律能够被机器学习算法捕获[8],从而将该问题转化为一个智能的有监督学习问题;
- 另外,机器学习模型能够伴随训练数据的不同而演化更新,而基于规则的日志打印位置决策方法的更新优化依赖于人工,难以为继.

因此,本文同样基于机器学习算法构建日志打印位置决策模型,旨在提出一种普适的特征提取方法和模型构建方法,突破现有工作中的诸多限制,如编程语言限制、代码结构限制等.另外,本文关注于跨组件、跨软件系统的日志打印位置决策,而现有工作关注于软件系统内部的日志打印位置决策.

1.3 特定任务驱动的日志打印位置决策

这类工作通常利用程序分析及信息理论技术等从软件代码中搜索符合特定任务需求的日志打印位置.Ding 等人[15]提出了一种日志过滤方法,该方法通过监控系统的性能,搜索对系统性能影响最大的日志打印位置,然后过滤这些日志打印位置.Zhao 等人[7,16]利用程序分析技术,通过计算不同日志打印位置带来的信息熵增益,找到能够区分最多程序路径的日志打印位置作为决策结果.我们之前的工作[17]设计了一种面向故障诊断的日志打印位置决策方法,该方法利用信息增益算法从所有日志打印位置中搜索能够最大限度表征故障的日志打印位置.这类方法同基于机器学习的日志打印位置决策方法是相辅相成的关系,同时,运用这两类方法能够有效提升日志打印位置的质量,达到更加智能化的日志打印位置决策.

2 问题定义

日志打印位置决策的目的是从系统源码中选择需要添加日志打印语句的位置(代码行),但是现有的软件系统规模极其庞大,代码量巨大,导致需要添加日志打印语句的位置占所有日志代码行的比例小于 1%,这会导致训练数据极度不平衡,难以构建机器学习模型.为解决该问题,本文首先定义了代码块(CodeBlock)的概念,并使用代码块作为日志打印位置决策的基本单位,即,将该问题转化为对代码块中是否添加日志打印语句进行决策.

代码块是一段不包含任何层次或嵌套结构的代码片段.代码块的提取可以通过两种标识符切割,包括分支或循环的起点和分支或循环的终点.以图 1 为例,该图截取了 Hadoop-YARN 项目中 RMAppManager.java 文件中的一个函数,通过“{”和“}”对该函数进行切分,共获取了 5 个代码块.本质上讲,代码块是一段顺序执行的代码行的集合,以代码块为基本决策单位,能够有效降低日志打印位置决策问题的复杂性,大大降低训练数据的不均衡程度.通常而言,一个代码块内部的日志打印语句通常表征该代码块的功能或执行结果,因此大多数情况下,一个代码块中通常只包含一个日志打印语句,且这个日志打印语句的具体位置不重要.经过对 hadoop 2.8.1 源代码的统计分析发现:在包含日志打印语句的所有代码块中,有 12 812 个代码块仅包含一条日志打印语句(占比 95.1%),658 个代码块包含多于一条日志打印语句(占比 4.9%).以图 1 中代码块 1 的日志打印语句为例,不论该语句位于代码块 1 中的哪一行,该日志打印语句所表达的信息不受影响.另外,代码块作为一个抽象的概念,具备较强的通用性,能够适配不同的程序开发语言编写的软件系统.代码块不同于程序基本块(BasicBlock):基本块的划分以程序功能为依据,包括一个单一入口和出口;代码块则是以程序结构为划分依据,其粒度小于基本块.经过对 hadoop 2.8.1 源代码中所有代码块包含的代码行数进行统计,结果表明,平均每个代码块包含 4.443 行代码.因此,对于日志打印位置决策任务而言,以代码块为粒度决策日志打印位置,能够有效地辅助程序开发人员打印日志.

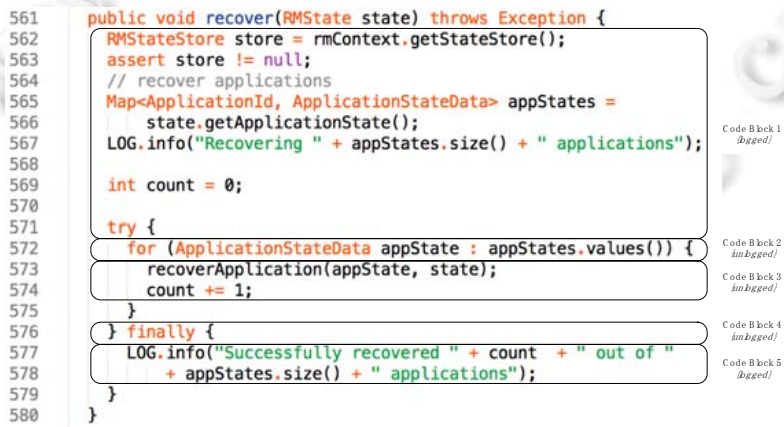


Fig.1 Example of program and code block

图 1 程序片段与代码块示例

以代码块为基础,这里明确本文的研究目标.

令某代码块为 $cb_i \in \{cb_x | 0 \leqslant x \leqslant p\}$,其中,$p$ 代表所有代码块的数量.本文旨在构建一个日志打印位置决策模型 $F(\sim)$,预测代码块 cb_i 的标签类型为 logged 或 unlogged,形式化如下:

$$label_i = F(cb_i).$$

其中,$label_i \in \{\text{logged,unlogged}\}$.

3 方法概述

图 2 给出了本文所述方法的整体流程和框架.

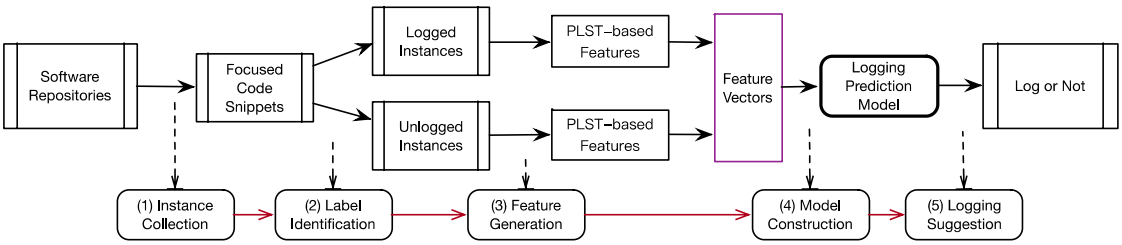


Fig.2 Workflow of our approach

图 2 方法流程

本方法包括实例收集、标签标识、特征向量生成、模型构建和日志打印决策这 5 个关键步骤.实例收集以代码块为单位切分源代码;标签标识即从代码块中搜索日志打印语句,方法为关键词搜索,如 LOG,logging,logger 等.通过实例收集和标签标识,初步得到模型训练数据实例为

$$instance_i=(cb_i,label_i).$$

如果 cb_i 包含日志打印语句,则 $label_i$ 取值为 logged;反之,取值为 unlogged.特征向量生成步骤通过神经语言模型为每个代码块生成一个特征向量,称为子向量,然后组合一段程序中的代码块子向量,最终生成完整的特征向量.第 3.1 节会详细介绍特征生成步骤的方法.模型构建步骤首先通过迁移学习算法屏蔽不同组件或不同软件系统代码特征空间之间的差异,然后使用机器学习算法构建分类模型.第 3.2 节详细介绍模型构建方法.日志打印决策步骤利用已训练的分类模型对新的软件或新的组件中的代码块进行日志打印决策.

3.1 基于程序层次树的特征向量生成

特征向量生成首先利用神经网络语言模型从每个代码块中提取特征向量(下文称为子向量),以提取代码片段的文本特征;然后,针对每个代码块,组合与其位于相同函数中的其他代码块的子向量以生成完整的特征向量,以提取代码片段的上下文和结构特征.本文提出的特征向量生成方法具有 3 个特点.

(1) 通用性,适用于不同的编程语言;

(2) 全覆盖性,能够针对所有的代码块生成特征向量,对代码块或代码片段无假设;

(3) 自动化.与传统特征工程不同,该特征生成方法无需任何人工定义或指导.

下面详细介绍这两个步骤.

3.1.1 基于神经网络语言模型生成代码块子向量

为了提取代码文本特征,需要为每个代码块生成固定长度的特征向量(即子向量):首先,按照空格和符号切分每个代码块,将其转换成单词列表;然后使用神经网络语言模型 PV-DM 学习单词列表,训练模型对其生成向量表示.PV-DM 模型由输入层、投影层、隐藏层和输出层组成:在输入层,使用 1-V 编码对段落中的段落 ID 和前 N 个词语进行编码,其中,N 是用户定义的参数,V 是词汇表的大小;然后,使用共享投影矩阵将输入层投影到具有维度 $(N+1)\times D$ 的投影层 P;输出层则是经典的 SoftMax 分类器.训练过程中,使用随机梯度下降法来训练段向量和字向量,并且通过反向传播来更新梯度.最后,使用训练过的 D 维向量作为每个代码块对应的子向量.

3.1.2 基于程序层次树逆向组合代码块子向量

一个代码块中是否打印日志,不仅受代码块本身内容的影响,还受到其相邻代码块的影响.例如,图 1 中的 Codeblock5 中仅包含日志打印语句,而其相邻的代码块则描述了程序逻辑,为该日志打印语句提供了丰富的上下文信息.此外,分支、循环等复杂的程序结构也对日志决策产生重要影响.直观地说,开发人员更可能在这些复杂的代码结构中添加日志打印语句,以帮助理解程序逻辑和行为.因此,有必要组合相邻代码块的子向量以捕获代码块的上下文特征和结构特征.

受 AST 的启发,本节提出了一种代码语法结构的粗粒度表示方法——程序层次树(program layered syntax tree,简称 PLST)作为组合子向量的基础.与 AST 中每个节点表示变量、关键字或运算符等细粒度结构相比,

PLST 中的节点表示代码块,能够有效降低复杂性并提高特征提取的效率.令待生成特征向量的代码块为目标代码块,则在 PLST 基础上如何提取目标代码块的附近的代码特征,即如何组合目标代码块附近的代码块子向量是一个难题.原因在于:

- 第一,影响日志打印位置的程序范围很难确定.从编程习惯的角度来看,日志打印语句被用来记录刚刚发生的某个事件或行为,因此对于目标代码块而言,最影响日志打印决策的程序片段应该是该代码块之前的程序片段;另外,代码块的日志打印决策也受到程序功能和代码逻辑的影响,而函数是程序中最小的功能逻辑单元,因此本方法将函数内目标代码块及其之前的代码块子向量进行组合,从而完整地表示目标代码块;
- 第二,邻近代码块的上下文信息极其相似,会极大地混淆日志打印决策模型.两个邻近代码块周围的程序文本非常相似,甚至大部分程序文本是相同的,但其数据标签却很有可能是完全不同的.这将极大地混淆决策模型,降低决策效果.针对该问题,本节提出一种逆序组合的方法,采用相对位置代替绝对位置,组合目标代码块前的代码块子向量,从而生成目标代码块的特征向量.

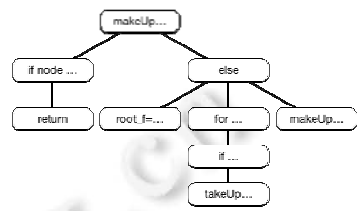
每棵 PLST 代表一个函数,根表示函数名称,节点表示代码块.每一个代码块所处的层由其位于程序中的嵌套层次决定,同一层中的代码块严格遵循其在代码中的先后顺序排布.每个 PLST 通过深度优先遍历可以恢复成原程序代码.为获取 PLST,不同编程语言的代码块由于语法不同而需要少量适应性处理.例如,Java 中的分层标识为“{”,Python 中则为 4 个空格.图 3 展示了程序结构和功能完全相同而编程语言不同的两段代码,这两段代码可以被转换成同一棵 PLST.

```java
public void makeUp(Node node){
    if node == getRoot(node,node_list){
        return;
    }else{
        root_f = getRoot(node,node_list);
        for(int i=0;i<root_f.getChildren().length;i++){
            if(root_f.getChildren()[i] != node){
                takeUp(root_f.getChildren()[i]);
            }
        }
        makeUp(root_f);
    }
}
```

(a) Java 代码示例

```python
def makeUp(node):
    if node == getRoot(node,node_list):
        return
    else:
        root_f = getRoot(node,node_list)
        for child in root_f.getChildren():
            if child != node:
                takeUp(child)
        makeUp(root_f)
```

(b) Python 代码示例



(c) 程序层次树

Fig.3 Example of PLST of different programming languages

图 3 不同编程语言代码的相同程序层次树表示

接下来,利用每个代码块的子向量及其所属的 PLST,提出一种逆序组合的方法为每个代码块生成完整的特征向量.令每个代码块的子向量长度为 l,逆序组合方法首先遍历目标代码块的左兄弟节点,以目标代码块起始,逆序组合其所有左兄弟节点的子向量,设置一个节点上限 m:如果逆序组合的节点数量超过节点上限 m,则删除超过部分的子向量;如果不足 m,以零向量补齐.组合完毕的向量被称为层向量,其长度为 $l\times m$.然后,以相同的方法处理目标代码块的父节点,即逆序组合目标代码块的父节点的左兄弟节点,直到节点上限 m,得到目标代码块父节点的层向量.然后,将该层向量排布在目标节点的层向量之后,设置一个层数上限 n.以此类推,组合每层向量直到层数达到上限 n.如果从目标代码块到 PLST 的根节点的总层数小于 n,则以零向量补齐至完整特征向量长度达到 $l\times m\times n$.

基于逆序组合的特征向量生成方法的细节如算法 1 所述,输入是程序层次树和待生成特征向量的目标代码块,输出是目标代码块的特征向量.算法从目标代码块在程序层次树中的层起始,由下至上遍历,直到根节点或达到层数上限 n(第 2 行).对于每一层,以临时目标节点(*temp_tb*)为起始依次遍历其左兄弟节点,临时目标节点被定义为目标节点及其祖先节点,上层的临时目标节点为下层临时目标节点的父亲节点.依照遍历顺序依次排列临时目标节点和其左兄弟节点,直到达到节点上限 m(第 5 行~第 10 行).

在逆序组合的过程中,当某节点存在孩子节点时,如果仅组合该节点对应的子向量是不合理的.原因在于:从代码角度看,其孩子节点也同样位于目标代码块之前,因此会对目标代码块的日志打印决策产生影响.为解决

该问题,提出了一种子树上提的方法来捕获这些孩子节点的特征.如算法 2 所示:子树上提利用深度优先遍历算法,搜索某节点的所有子孙节点(第 2 行),然后将这些代码块依次遍历顺序排列组合成新的代码片段(第 3 行、第 4 行),最后将该代码片段输入已训练的神经语言模型生成新的子向量(第 5 行).在逆序组合过程中,使用该新的子向量替代原子向量.

算法 1. 基于逆序组合的特征向量生成算法.

Input: PLST and target code *block*(*tb*), *n*, *m*;

Output: Feature vector of target code *block*(*fv*).

1. *fv*=[·], *layer*=*init_layer*, *temp_tb*=*tb*
2. **while** *layer*>0 **and** *init_layer*-*layer*<*n* **do**:
3. *fv*.*concatenate*(*temp_tb*.*sub_vector*)
4. *left_num*=0
5. **while** *temp_tb*.*left_cb*!=*null* **and** *left_num*<*m* **do**:
6. *temp_tb*=*temp_tb*.*left_cb*
7. **if** *temp_tb*.*children*!=*null* **then**:
8. *temp_tb*=*SubtreeRaising*(*temp_tb*)
9. *fv*.*concatenate*(*temp_tb*.*sub_vector*)
10. *left_num*++
11. *temp_tb*=*temp_tb*.*father_cb*
12. *layer*––

算法 2. 子树上提算法.

Input: PLST, root code block of a *subtree*(*rcb*), trained PV-DM model;

Output: feature vector of *subtree*(*fv*).

1. *code_snippet*='·'
2. *codeblock_queue*=*DepthFirstSearch*(*rcb*)
3. **while** *codeblock_queue*!=*null* **do**:
4. *code_snippet*.*concatenate*(*codeblock*.*pop*(·))
5. *fv*=*PV-DM*(*code_snippet*)

图 4 是子向量逆序组合过程的一个例子,其中,N_8 是目标代码块.

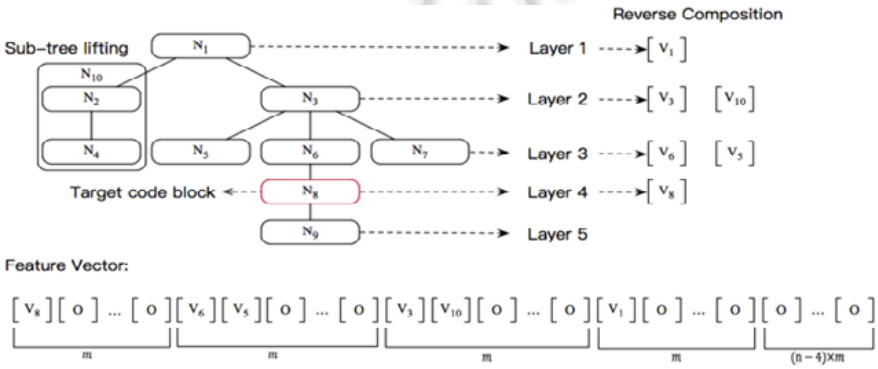


Fig.4 Example of reverse composition

图 4 子向量逆序组合示例

本方法首先逆序组合生成第 4 层到第 1 层的层向量.由于 N_8 是第 4 层中唯一的代码块,排列 N_8 对应的子向量 v_8 并用零填充的其余部分作为第 4 层目标代码块的层向量.对于第 3 层,依次组合 N_8 的父节点 N_6 对应的子

向量 v_6 以及 N_6 的左兄弟节点 N_5 的子向量 v_5.在第 2 层,由于 N_2 有一个子节点 N_4,因此利用子树上提方法将其转换为新节点 N_{10},并依次组成 v_3 和 v_{10}.然后,将从第 4 层到第 1 层的所有向量合成为最终的特征向量.在这个例子中,假设 m 大于 2 并且 n 大于 4,且所有层向量和最终特征向量用零向量填充.

在程序文本中,位置相近的代码块的上下文程序相似甚至多数是相同的,然而这些代码块在日志打印决策上又往往是相反的.因此,如何从相似的程序文本中为不同的代码块提取不同的特征向量,成为通用日志打印位置决策的关键问题.逆序组合方法考虑使用相对距离替代绝对距离,即使两个位置相近的代码块的上下文程序是相似的,他们的对齐也是不同的.考虑图 4 中 N_3 和 N_6 是两个目标代码块,由于 N_6 的第一个左兄弟节点是 N_5,N_3 的第一个左兄弟节点是 N_{10},因此,N_5 和 N_{10} 的子向量对齐.对于目标代码块 N_3,N_{10} 在最终特征向量的 N_3 同层排布,然而对于目标块 N_6,N_{10} 排布在最终完整特征向量中 N_6 的父层向量部分中.

3.2 基于迁移学习的日志打印位置决策模型构建

如图 5 所示,日志打印位置决策模型包括一个特征迁移模型和一个分类模型.

- 特征迁移模型的输入是不同软件或组件的代码块特征向量集合;输出是一个新的特征向量集合,该集合中的特征向量与输入的特征向量一一对应.特征迁移模型可以利用不同的特征空间的特征向量学习到一个共享特征空间,进而将不同软件或组件的代码块特征向量映射到共享特征空间去;
- 分类模型的输入是共享特征空间的代码块特征向量,输出是对该代码块是否添加日志打印语句的决策结果.

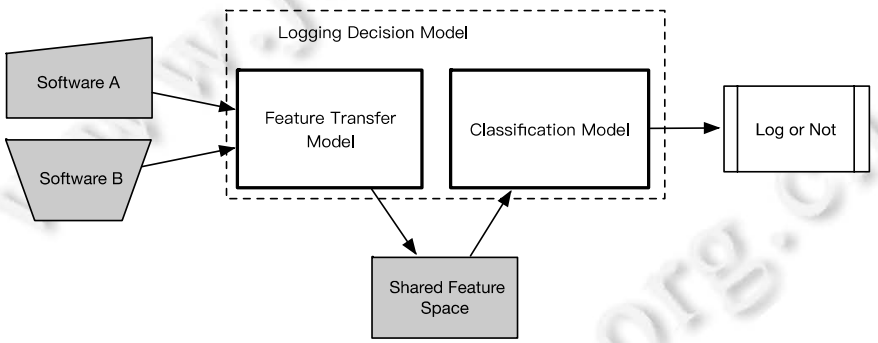


Fig.5 Logging point decision model

图 5 日志打印位置决策模型

下面分别介绍这两个子模型.

特征迁移模型采用两种经典的特征迁移算法——迁移成分分析算法(transfer component analysis,简称 TCA)[18]和联合分配适配算法(JointDistribution adaptation,简称 JDA)[19]将源系统特征向量和目标系统特征向量映射到新的特征空间.TCA 算法假设源系统特征向量集合和目标系统特征向量集合分别为 X_S 和 X_T.TCA 利用边缘概率分布来估计源系统特征向量和目标系统特征向量之间的差异.令一组数据集合 X 的边缘概率分布为 $P(X)$,显然,$P(X_S)\neq P(X_T)$.TCA 的目标是获取一个转换函数 ϕ,使得 $P(\phi(X_S))\approx P(\phi(X_T))$.这样,$\phi(X_S)$和$\phi(X_T)$是独立且同分布的.为计算该转换函数,TCA 首先使用最大均值差异(maximum mean discrepancy,简称 MMD)来衡量 $P(X_S)$ 和 $P(X_T)$之间的距离,计算公式如下:

$$dist(X_S,X_T)=\left\|\frac{1}{n_1}\sum_{i=1}^{n_1}\phi(x_{S_i})-\frac{1}{n_2}\sum_{i=1}^{n_2}\phi(x_{T_i})\right\|,$$

其中,n_1 和 n_2 表示训练向量和测试向量的数量.然后,TCA 引入一个计算内核获取满足 $dist(X_S,X_T)$取最小值的最优值.训练完毕后,X_S 和 X_T 分别由$\phi(X_S)$和$\phi(X_T)$取代.

同 TCA 不同,JDA 不但使用边缘概率分布来估计源系统特征向量和目标系统特征向量之间的差异,同时还考虑两者之间的条件概率分布差异.条件概率分布可以被形式化为 $P(Y_S|X_S)$和 $P(Y_T|X_T)$.JDA 的目标是获取一个

转换矩阵 A^T,使得 $P(Y_S|A^T(X_S))\approx P(Y_T|A^T(X_T))$.为计算该转换矩阵,JDA 同样使用 MMD 来衡量 X_S 和 X_T 的距离.但是,由于目标系统特征向量集合没有标签数据,因此 Y_T 是未知的.为生成 Y_T,JDA 使用源系统特征向量集合训练一个简单的分类器(X_S,Y_S),然后对 X_T 进行标注,进而得到仿造的标签 $\hat{Y}_T$,并以 $\hat{Y}_T$ 替代 Y_T.训练完成后,X_S 被 $A^T\phi(X_S)$ 取代,X_T 被 $A^T\phi(X_T)$取代.

分类模型采用 3 种经典的机器学习算法,包括支持向量机(support vector machine,简称 SVM)、k-最近邻(k-nearest neighbor,简称 kNN)和 Logistic 回归(logistic regression,简称 LR).SVM 的基本思路是,在共享特征空间中学习能够决策日志打印的最大边距超平面.kNN 算法的分类决策依据代码块在共享特征空间中最近邻的几个代码块所属的类别,如果与一个代码块相似的若干其他代码块中均包含日志打印语句,则该代码块也应包含日志打印语句.LR 的因变量即是否应添加日志打印语句,自变量是使用经过特征迁移后的代码块共享特征向量,经过回归分析,可以得到特征向量中的每一个维度对因变量的影响权重,进而判定目标系统代码块共享特征向量的因变量值.

4 实验设计与结果验证

4.1 实验设计

为了评估所提出的方法,实验使用 Apache Hadoop[9],Angel[10],OpenStack[11],SaltCloud[12]和 Tensorflow[13]等 5 个流行的大型开源软件系统源代码进行了两组实验.Hadoop 是当今最流行的大数据分析平台之一,Angel 是腾讯开源的分布式机器学习平台,OpenStack 和 SaltCloud 是两个开源云计算平台,Tensorflow 是 Google 开源的深度学习平台.其中,Hadoop 和 Angel 由 Java 语言编写,OpenStack,SaltCloud 和 Tensorflow 由 Python 语言编写.

这两组实验包括多场景评估实验和各阶段评估实验:多场景评估旨在验证本文所提出的跨组件、跨软件系统的日志打印决策方法在不同应用场景下的有效性;各阶段评估旨在分别评估特征向量生成和日志打印位置决策模型的效果.第 4.2 节和第 4.3 节详细介绍这两组实验的实验结果.本节接下来介绍实验过程中的数据处理、基线设置、参数设置、评价指标等.

4.1.1 实验参数设置与数据处理

实验参数的选择,可能会影响实验结果.在特征生成步骤,存在 3 个关键参数需要预设,包括子向量长度 l、子节点数量上限 m 和层数量上限 n.实验中,这 3 个参数的预设值分别为 50,10 和 10,使得最终特征向量的长度为 5 000($l\times m\times n$).为提升训练速度,实验中使用了 PCA 算法对特征向量降维,在保证 95%以上数据方差的基础上,将特征向量降至 200 维.实验过程中使用了若干种不同的参数取值,实验结果对参数不敏感.由于论文长度限制,在本节并未将所有实验结果一一列举.

实验过程中,由于软件代码中只有少量的代码块包含日志打印语句,因此整个训练集和测试集均存在正负样本不均衡的问题.例如,在 Hadoop 代码中共提取 397 442 个代码块,其中仅 13 470 个代码块中包含日志打印语句.为保障分类模型的训练和测试效果,在训练集和测试集中分别采用下采样的方式获取同正样本数量相同的负样本进行训练和测试.例如,在测试过程中,从 Hadoop 不包含日志打印语句的 383 972 个代码块中随机抽样出 13 470 个代码块作为负样本,使用共计 26 940 个代码块作为测试集.在多场景评估实验中的版本升级场景中,由于训练集和测试集分别是软件系统一个组件的两个不同版本,因此训练集和测试集中可能包含相同的样本.为避免数据污染问题,将测试集中与训练集共同包含的样本删除,从而得到最终的测试集.

4.1.2 评价指标

评价指标采用使用经典的机器学习算法评价指标准确率(accuracy)和 $F1$ 值.

- *Accuracy* 指日志打印决策正确的代码块占所有代码块的比例,它反映了日志打印决策模型的整体性能,计算如下:

$$Accuracy=\frac{\#\text{code blocks being calssified correctly}}{\#\text{all code blocks}};$$

- $F1$ 指标精确率(*precision*)和召回率(*recall*)的调和平均值,代表模型对正样本的分类效果.高精确率意味

着多数被分类为需要打印日志的代码块都是正确的,高召回率则意味着多数打印日志的代码块被正确地分类.*F*1 反映了模型能够精确且全面地识别包含日志打印语句的代码块的性能,其计算公式如下:

$$F1=(2\times Precision\times Recall)/(Precision+Recall).$$

4.1.3 相关工作对比

日志打印位置决策任务的相关研究工作主要有 LogAdvisor 和 LogOptPlus,然而这些工作存在若干限制.表 1 列举了本文的方法同这些工作的对比.相关工作均针对特殊类型的代码块和特定的日志级别,同时对编程语言有一定的限制,仅支持 C#,Java 等高级编程语言,无法支持 C 语言以及一些脚本语言.本文所述方法是一种针对所有代码块的通用日志打印位置决策方法,在根本目的上与现有工作有较大区别.如果仅对比特定代码块的日志打印决策效果对本文的方法不公平,如果对比所有代码块的日志打印决策效果则对相关工作不公平,因此实验中没有同相关工作进行决策效果对比.实验选取了 LogAdvisor 中所对比的方法——随机添加日志打印语句(random error logging)作为基线,并随机测试 5 次取平均以减少实验结果的偏差.

Table 1 Comparison on related works

表 1 相关工作对比

相关工作	代码块限制	日志级别限制	编程语言限制
LogAdvisor[4]	exception & return-value-check blocks	ERROR & WARN	C#,Java 等高级编程语言
LogOptPlus[14]	if & catch blocks	NONE	Java
FineLog	无	无	无

4.2 多场景下评估日志打印位置决策

在这个实验中,我们首先定义了 3 个场景,包括版本升级、组件开发和系统开发.根据编程语言的不同,5 个软件系统被分为两组:Python 组和 Java 组,分别在两组实验系统上测试 3 种场景下日志打印位置决策模型的效率.如表 2 所示:在版本升级场景中,选择软件系统的旧版本代码作为训练集,新版本的代码选作测试集,验证和评估所提出的方法在跨版本日志打印位置决策中的效率;在组件开发场景中,选择一个组件的代码作为训练集,同一系统的另一个组件的代码作为测试集,验证和评估所提出的方法在跨组件日志打印位置决策中的效率;在系统开发场景中,选择一个系统的代码进行训练,另一个系统的代码进行测试,验证和评估所提出的方法在跨系统日志打印决策中的效率.

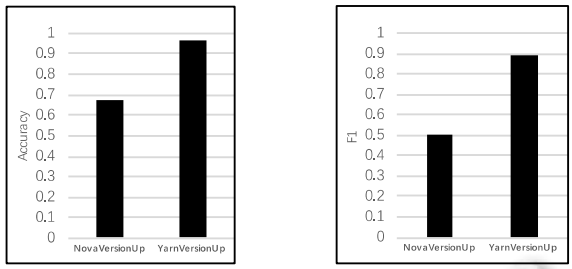
Table 2 Design of multiple scenario experiment

表 2 多场景评估实验设计

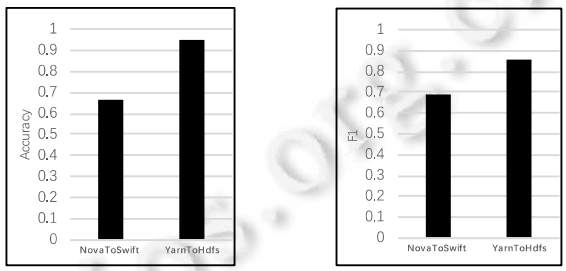
测试 / 训练	版本升级(跨版本决策)	组件开发(跨组件决策)	系统开发(跨系统决策)
Java	Yarn-3.0.0 / Yarn-2.6.5	Hdfs-3.0.0 / Yarn-3.0.0	Angel / Apache Hadoop
Python	Nova-16.0.3 / Nova-14.0.10	Nova-16.0.3 / Nova-16.0.3	TensorFlow / SaltCloud / OpenStack

实验结果如图 6 所示.

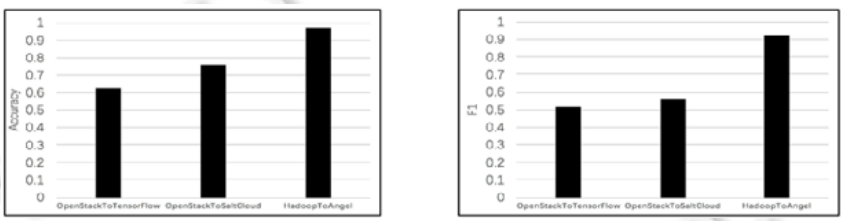
- 在版本升级场景下评估跨版本的日志打印位置决策模型(如图 6(a)所示),Python 组实现了 0.692 的准确率和 0.525 的 *F*1 指标;同时,Java 组达到了 0.965 的准确率和 0.890 的 *F*1;
- 在组件开发场景下评估跨组件的日志打印位置决策模型(如图 6(b)所示),Python 组实验得到了 0.659 的准确率和 0.690 的 *F*1,Java 组实验中达到了 0.960 的准确率和 0.866 的 *F*1;
- 在系统开发场景下评估跨系统的日志打印位置决策模型(如图 6(c)所示),Python 组的实验以 TensorFlow 为测试集时达到 0.663 的准确率和 0.529 的 *F*1,使用 SaltCloud 作为测试集时准确率达到 0.773,*F*1 达到 0.565;在 Java 组实验中达到 0.969 的准确率和 0.925 的 *F*1.同基线(RandomErrorLogging)相比,Python 组提升了近 20%的准确率,Java 组则提升了 40%以上的准确率.



(a) 版本升级(跨版本日志打印位置决策)



(b) 组件开发(跨组件日志打印位置决策)



(c) 系统开发(跨系统日志打印位置决策)

Fig.6 Results of multiple scenario experiment

图 6 多场景评估实验结果

上述实验结果表明:本方法在 3 种典型场景和 5 种流行的软件系统上都取得了良好的表现,在 Java 组中达到 0.95 的准确率和 0.85 的 *F*1,在 Python 组中达到超过 0.65 的准确率和 0.5 的 *F*1.更进一步的,本方法在新系统开发场景下的跨系统日志打印位置决策表现最佳,准确率比另外两类场景高出约 0.1.其原因在于:在新系统开发场景中将整个系统的源代码作为训练集,而在另外两种场景下仅选取一个组件的源代码用于训练,训练集的大小可能影响分类模型的性能.值得注意的是:本方法在 Java 组中取得了更好的效果,远超 Python 组.如图 7 所示,3 个场景中 Java 组的准确性均比 python 组高 20%左右.究其原因:

- Python 的编程风格更加灵活,且支持多种易用功能强大的语法,特征向量生成方法将代码视为纯文本,并侧重于提取文本特征.因此,类似功能的不同实现所生成的特征有较大差异,导致决策模型的效果欠佳;
- 而 Java 语言则存在更好的异常和错误处理机制,日志打印语句往往在这些异常和错误处理代码中出现.这使得特征向量生成更容易捕获包含日志打印语句的代码块和未包含日志打印语句的代码块之间的差异,决策模型表现更良好.

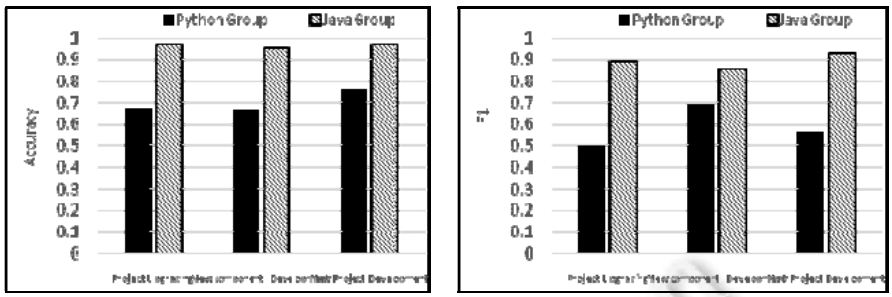


Fig.7 Comparison of results of Python group and Java group

图 7 Python 组和 Java 组实验结果对比

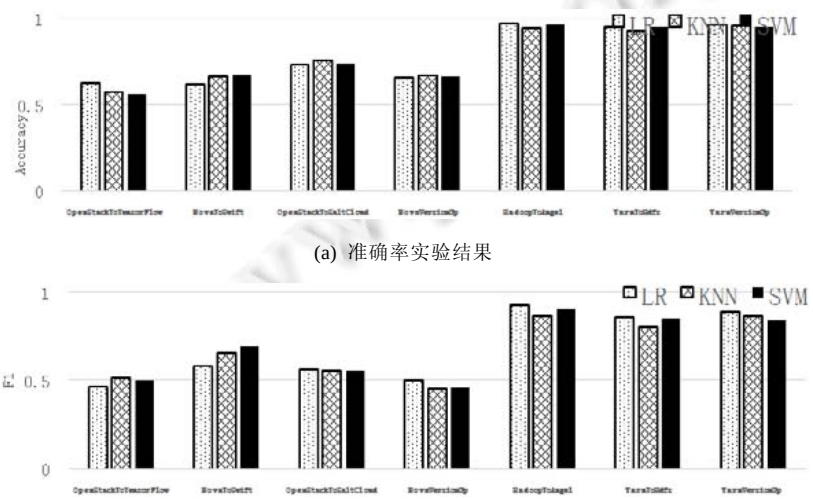
4.3 多阶段评估日志打印位置决策

本实验旨在评估特征向量生成步骤与日志打印决策模型的效果.在特征向量生成评估中,仅利用基础分类模型,包括 Logistic 回归(logistic regression,简称 LR)、*k*-最近邻(*k*-nearest neighbor,简称 *k*NN)和支持向量机(support vector machine,简称 SVM)的分类效果来评估所生成的特征向量的质量.在日志打印决策评估中,则对比分析了加入迁移学习模型后和仅使用基础分类模型的效果.

图 8 为特征向量生成步骤的实验结果.

- 在版本升级(跨版本决策)场景中,Java 组的平均准确率为 0.959,平均 *F*1 为 0.865;在 Python 组中,平均准确度为 0.664,平均 *F*1 为 0.469;
- 在新组件开发(跨组件决策)场景中,Java 组的平均准确率为 0.943,平均 *F*1 为 0.838;同时,Python 组的平均准确率为 0.649,平均 *F*1 为 0.644;
- 在新的系统开发(跨系统决策)场景中,Java 组的平均准确率为 0.960,平均 *F*1 为 0.899;而 Python 组的平均准确率为 0.667,平均 *F*1 为 0.527.

结果表明,特征向量生成方法可以有效地从源代码中提取特征向量.即使使用基本的简单分类模型,Java 组中的实验结果也达到 0.9 以上的精确度和 0.8 以上的 *F*1.



(a) 准确率实验结果

(b) *F*1 实验结果

Fig.8 Experiment result of feature vector generation

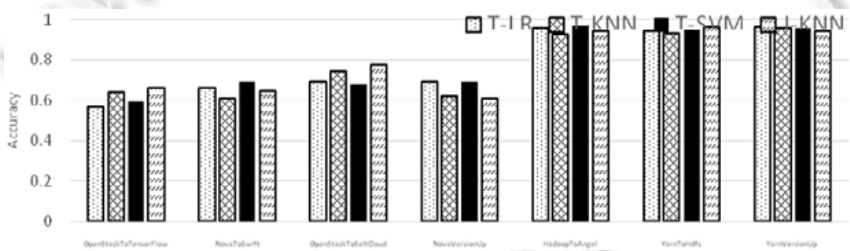
图 8 特征向量生成步骤效果验证实验结果

图 9 展示了日志打印决策步骤的实验结果.

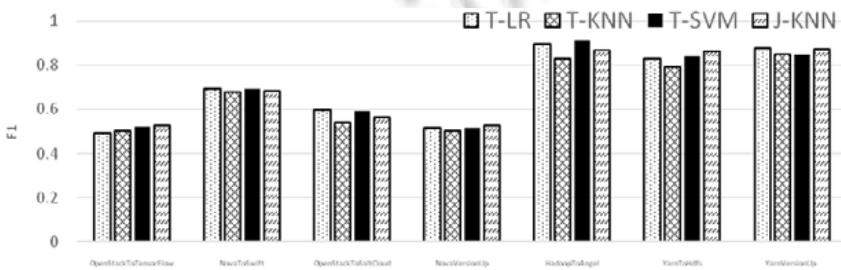
- Python 组:在版本升级(跨版本决策)场景中,TCA 和 SVM 模型组合达到最佳准确率 0.692,而 JDA 和 KNN 模型的组合得到最佳 *F*1 为 0.525;在新组件开发(跨组件决策)场景中,TCA 和 SVM 模型的组合达到最佳准确率为 0.690,TCA 和 LR 模型的组合达到最佳 *F*1 为 0.695;在新的系统开发(跨系统决策)场景中,JDA 和 KNN 模型的组合达到的最佳准确率为 0.663,最佳 *F*1 为 0.529;
- Java 组:TCA 和 LR 模型的组合在版本升级(跨版本决策)场景中达到最佳准确率为 0.962,最佳 *F*1 为 0.878;在新组件开发(跨组件决策)方案中,JDA 和 KNN 的最佳准确率为 0.960,最佳 *F*1 为 0.866;在新的系统开发(跨系统决策)场景中,TCA 和 SVM 的组合达到 0.965 的最佳准确率和 0.912 的最佳 *F*1.

与简单分类模型相比,Python 组实验中迁移学习有效地提高了 2%~9%的准确率和 1%~12%的 *F*1.在版本升级(跨版本决策)场景中,TCA 和 SVM 模型的组合比 SVM 的最优精度提高了 3%,而 JDA 和 KNN 模型组合的 *F*1 比 KNN 模型提高了 7.3%;在新组件开发(跨组件决策)场景中,TCA 和 SVM 模型的组合比 SVM 基线的最高精度提升了 2.3%,TCA 和 LR 模型的 *F*1 比 LR 基线提升 11.1%;在新软件系统开发(跨系统决策)场景中,JDA 和 KNN 模型的组合达到最佳精度和 *F*1,与 KNN 基线相比,它提高了 8.6%的准确率和 1.2%的 *F*1.

在 Java 组中,迁移学习的平均准确率和 *F*1 降低 1%左右.在版本升级场景和新软件系统开发(跨版本和跨系统决策)场景中,TCA 和 LR 模型的组合平均下降约 0.8%的准确率和 1.9%的 *F*1;在新组件开发(跨组件决策)场景中,JDA 的准确率提高了 3.3%,*F*1 提高了 1.1%.Java 组中采用迁移学习后准确率和 *F*1 值下降可能是由于源域和目标域的特征向量之间的分布差异较小,没有明显适合迁移的部分,导致出现负迁移的情况.所谓负迁移即迁移学习在转化和评估过程中的迁移损失大于迁移增益.在 Java 组的实验中,基本模型的结果良好,意味着源域和目标域之间没有明显的分布差异,因此在迁移学习转化过程中带来的损失占据主导.尽管在 Java 组的部分实验中存在性能下降,但迁移学习模型在 Java 组中总体表现仍然出色,能够说明日志打印决策模型的有效性.



(a) 准确率实验结果



(b) *F*1 实验结果

Fig.9 Experiment result of logging point decision models

图 9 日志打印位置决策模型效果验证

请读者查阅表 3 获取详细的实验结果数值,其中,模型列列举了实验中使用的特征迁移模型和分类模型算法,其他列列举了 Java 和 Python 组中不同场景的实验结果.表中黑体部分代表每一个场景的实验中的最优结果.

Table 3 Experiment results

表 3 实验结果汇总

Models			Python group				Java group		
			OpenStack ToTensorFlow	Nova ToSwift	OpenStack ToSaltCloud	Nova VersionUp	Hadoop ToAngel	Yarn ToHdfs	Yarn VersionUp
Random error logging (BaseLine)		Accuracy	0.497	0.500	0.521	0.489	**0.501**	0.511	**0.501**
Non-transfer	Logistic regression	Accuracy	0.624	0.619	0.734	0.659	**0.969**	0.950	**0.965**
		Precision	0.428	0.723	0.520	0.684	0.941	0.805	0.819
		Recall	0.504	0.489	0.609	0.393	0.911	0.915	0.974
		*F*1	0.463	0.584	0.561	0.500	**0.925**	0.856	**0.890**
	*K*NN	Accuracy	0.577	0.661	0.760	0.671	0.947	0.927	0.959
		Precision	0.408	0.736	0.576	0.701	0.857	0.739	0.812
		Recall	0.706	0.592	0.534	0.334	0.867	0.877	0.929
		*F*1	0.517	0.656	0.554	0.452	0.862	0.802	0.867
	SVM	Accuracy	0.563	0.667	0.741	0.662	0.964	0.952	0.952
		Precision	0.396	0.698	0.533	0.748	0.942	0.835	0.807
		Recall	0.685	0.688	0.587	0.328	0.880	0.876	0.872
		*F*1	0.501	0.693	0.559	0.456	0.910	0.855	0.838
TCA	Logistic regression	Accuracy	0.569	0.659	0.692	0.691	0.956	0.941	0.962
		Precision	0.396	0.678	0.470	0.667	0.876	0.781	0.805
		Recall	0.651	0.712	0.813	0.420	0.921	0.886	0.967
		*F*1	0.492	**0.695**	**0.596**	0.516	0.898	0.830	0.878
	*K*NN	Accuracy	0.641	0.611	0.742	0.622	0.927	0.930	0.958
		Precision	0.453	0.619	0.538	0.701	0.802	0.771	0.857
		Recall	0.570	0.749	0.544	0.391	0.865	0.812	0.845
		*F*1	0.505	0.678	0.541	0.502	0.832	0.791	0.851
	SVM	Accuracy	0.593	**0.690**	0.674	**0.692**	0.965	0.947	0.955
		Precision	0.419	0.757	0.455	0.670	0.966	0.824	0.818
		Recall	0.693	0.636	0.845	0.417	0.863	0.858	0.880
		*F*1	0.522	0.691	0.591	0.515	0.912	0.841	0.848
JDA	*K*NN	Accuracy	**0.663**	0.647	**0.773**	0.609	0.940	**0.960**	0.941
		Precision	0.441	0.747	0.581	0.696	0.861	0.878	0.871
		Recall	0.660	0.633	0.550	0.421	0.877	0.854	0.880
		*F*1	**0.529**	0.685	0.565	**0.525**	0.869	**0.866**	0.875

4.4 讨 论

本节讨论实验中涉及的可能影响实验结果的因素.

- 实验选取了有限的 5 个不同语言的不同项目,划分 3 个场景进行实验,希望能够在跨组件、跨软件系统的日志打印位置决策问题上进行初步探索.未来工作中会加大实验规模,以 EcoSystem 为单位验证并优化本文的方法.目前,已经收集了 NPM 生态系统中 21 个项目、NVIDIA 生态系统中 14 个项目、Pytorch 生态系统下 17 个项目、Apache 生态系统下 152 个项目、Chrome 下 13 个项目以及 Eclipse 下 13 个项目,未来我们会在这些项目上对本方法进行实验;
- 在跨软件系统决策场景实验中,从 Hadoop 不包含日志打印语句的 383 972 个代码块中随机抽样出 13 470 个代码块作为负样本,使用共计 26 940 个代码块作为测试集.在版本升级和跨组件决策场景实验中,HDFS 组件 110 396 个代码块中,4 242 个代码块包含日志打印语句,从 106 154 个不包含日志打印语句的代码块中随机采样了 4 242 个代码块作为负样本,使用共计 8 484 个样本作为训练集.对于深度学习模型而言,该量级的训练数据过少,但是对于本文所使用的机器学习模型而言,这个量级的数据是可以训练出有效的分类模型的;
- 本文所述方法是一种针对所有代码块的通用日志打印位置决策方法,在根本目的上与现有工作有较大区别.如果仅对比特定代码块的日志打印决策效果对本文的方法不公平,如果对比所有代码块的日志打印决策效果则对相关工作不公平.未来,我们会修改相关工作(LogAdvisor 和 LogOptPlus)使之支持 Java 语言和 Python 语言,并在特殊代码片段的日志打印决策问题上进行定量的对比实验.

5 总结与展望

本文主要研究软件系统的日志打印位置决策问题,即:给定一段代码,决策与这段代码相关的日志打印位置.具体地,为了适用于不同的编程语言,并进行跨组件和跨软件系统的日志打印位置决策,本文提出一种通用的自动化特征向量提取方法,通过构建程序层次树,屏蔽编程语言与不同程序模块实现细节的异构性,并提出一种基于迁移学习的日志打印位置决策模型,利用特征迁移技术挖掘不同软件系统日志打印程序的共有特征空间,迁移有用信息削减特征差异.在未来的工作中,拟加大实验规模,对本方法进行更加充分的验证.另外,拟从特征生成和模型构建步骤优化现有方法,使之进一步支持跨编程语言不同软件系统的日志打印位置决策,拟进一步研究如何利用深度学习技术(如深度迁移学习等)提升日志打印位置决策效果;与此同时,拟研究日志打印变量和常量的自动化决策方法,以实现完全自动化的日志打印语句撰写.

References:

[1] Prasad P, Rich C. Market guide for AIOps platforms. 2018. https://www.gartner.com/doc/3892967/market-guide-aiops-platforms

[2] Paskin S. What is AIOps. 2018. https://www.bmc.com/blogs/what-is-aiops

[3] Yuan D, Park S, Zhou YY. Characterizing logging practices in open-source software. In: Proc. of the 34th Int'l Conf. on Software Engineering. IEEE, 2012. 102−112.

[4] Zhu, JM, He PJ, Fu Q, *et al*. Learning to log: Helping developers make informed logging decisions. In: Proc. of the 37th Int'l Conf. on Software Engineering. IEEE, 2015. 415−425.

[5] Chen BY, Jiang ZM. Characterizing and detecting anti-patterns in the logging code. In: Proc. of the 39th Int'l Conf. on Software Engineering. IEEE, 2017. 71−81.

[6] Yuan D, Park S, Huang P, *et al*. Be conservative: Enhancing failure diagnosis with proactive logging. In: Proc. of the Usenix Conf. on Operating Systems Design and Implementation, Vol.12. USENIX Association, 2012. 293−306.

[7] Zhao X, Rodrigues K, Luo Y, *et al*. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In: Proc. of the 26th Symp. on Operating Systems Principles. ACM, 2017. 565−581.

[8] Fu Q, Zhu JM, Hu WL, *et al*. Where do developers log? An empirical study on logging practices in industry. In: Proc. of the 36th Int'l Conf. on Software Engineering. ACM, 2014. 24−33.

[9] Hadoop. http://hadoop.apache.org/

[10] Angel. https://github.com/Angel-ML/angel

[11] OpenStack. https://www.openstack.org/

[12] SaltCloud. https://www.saltstack.com/

[13] TensorFlow. https://tensorflow.google.cn/

[14] Lal S, Sardana N, Sureka A. LogOptPlus: Learning to optimize logging in catch and if programming constructs. In: Proc. of the 40th Annual Computer Software and Applications Conf. (COMPSAC). IEEE, 2016. 215−220.

[15] Ding R, Zhou HC, Lou JG, *et al*. Log2: A cost-aware logging mechanism for performance diagnosis. In: Proc. of the Annual Technical Conf. USENIX Association, 2015. 139−150.

[16] Zhao X, Rodrigues K, Luo Y, *et al*. The game of twenty questions: Do you know where to log? In: Proc. of the 16th Workshop on Hot Topics in Operating Systems. ACM, 2017. 125−131.

[17] Jia T, Li Y, Zhang CB, *et al*. Machine deserves better logging: A log enhancement approach for automatic fault diagnosis. In: Proc. of the 29th IEEE Int'l Symp. on Reliability Engineering (ISSRE). IEEE, 2018. 106−111.

[18] Li H, Shang WY, Hassan AE. Which log level should developers choose for a new logging statement? Empirical Software Engineering, 2017,22(4):1684−1716.

[19] Deun KV, Thorrez L, Coccia M, *et al*. Weighted sparse principal component analysis. Chemometrics and Intelligent Laboratory Systems, 2019.

[20] Du M, Li FF, Zheng GN, *et al*. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2017. 1285−1298.

[21] Mi HB, Wang HM, Yin G, *et al*. Performance problems diagnosis in cloud computing systems by mining request trace logs. In: Proc. of the 2012 IEEE Network Operations and Management Symp. IEEE, 2012. 893−899.

[22] Nandi A, Mandal A, Atreja S, *et al*. Anomaly detection using program control flow graph mining from execution logs. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM, 2016. 215−224.

[23] Tak BC, Tao S, Yang L, *et al*. LOGAN: Problem diagnosis in the cloud using log-based reference models. In: Proc. of the IEEE Int'l Conf. on Cloud Engineering. IEEE, 2016. 62−67.

[24] Debnath B, Solaimani M, Gulzar MA, *et al*. LogLens: A real-time log analysis system. In: Proc. of the 38th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS). IEEE, 2018. 1052−1062.

[25] Pan SJ, Tsang IW, Kwok JT, *et al*. Domain adaptation via transfer component analysis. IEEE Trans. on Neural Networks, 2011, 22(2):199−210.

[26] Long MS, Wang JM, Ding GG, *et al*. Transfer feature learning with joint distribution adaptation. In: Proc. of the IEEE Int'l Conf. on Computer Vision. IEEE, 2013. 2200−2207.

贾统(1993－),男,博士,主要研究领域为分布式系统,智能运维.

张齐勋(1979－),男,讲师,CCF 学生会员,主要研究领域为嵌入式软件与系统,大数据系统与分析.

李影(1975－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式计算,可信计算.

吴中海(1968－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为大数据技术,系统安全,嵌入式软件.