一种执行轨迹驱动的移动应用功能分类方法

马 超[1),2)] 李俊彤[2),3)] 曹建农[2),3)] 蔡华谦[4)] 吴黎兵[1)] 石小川[1)]

[1)](武汉大学国家网络安全学院 武汉 430072)

[2)](香港理工大学深圳研究院 广东 深圳 518057)

[3)](香港理工大学电子计算学系 香港)

[4)](北京大学信息科学技术学院 北京 100871)

摘 要 程序理解对于诸如遗留系统重构和恶意软件检测等多类场景具有重要作用.移动应用功能分类旨在通过分析目标移动应用的运行时行为来识别其主要功能.由于运行环境的动态性和开发框架的差异性,移动应用行为模式普遍呈现出较高的复杂性,这给移动应用功能分类带来了挑战.本文致力于通过分析移动应用的执行轨迹实现对其功能的自动分类.在形式化定义移动应用功能分类问题的基础上,本文提出了一个系统性的解决方案设计框架 RaT(Run-and-Tell)以指导执行轨迹驱动的移动应用功能分类解决方案的设计.在 RaT 框架的指导下,本文提出了 2 种分别基于执行轨迹统计特征和语义特征的行为表征方法.然后,将所生成的 2 类行为表征与 4 种基于神经网络(即 MLP、FCN、ResNet 及 LSTM)的移动应用功能分类器相结合构造了 8 种移动应用功能分类解决方案.此外,通过利用程序插桩技术,本文采集了来自 Google Play 应用商店 3 类安卓应用类别涵盖 13 种不同功能的 17 个安卓应用程序总计 876 条执行轨迹以构建实验数据集.实验结果表明,采用执行轨迹语义特征行为表征的 RaT 框架解决方案在实验数据集上达到了 73.2%的类间平均分类准确率,其性能明显优于基线方法.

关键词 程序理解;移动应用功能分类;执行轨迹;行为表征;神经网络

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2022.01997

A Trace-Driven Approach to Mobile App Functionality Classification

MA Chao[1),2)] LI Chun-Tung[2),3)] CAO Jian-Nong[2),3)] CAI Hua-Qian[4)] WU Li-Bing[1)] SHI Xiao-Chuan[1)]

[1)](*School of Cyber Science and Engineering, Wuhan University, Wuhan* 430072)

[2)](*Shenzhen Research Institute, The Hong Kong Polytechnic University, Shenzhen, Guangdong* 518057)

[3)](*Department of Computing, The Hong Kong Polytechnic University, Hong Kong*)

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

Abstract Program comprehension assists in many scenarios such as legacy system re-engineering, malware detection, etc. Mobile app functionality classification aims to realize the functionality of a mobile app by analyzing its runtime behavior. Due to the dynamic runtime environment and various development frameworks, the mobile app behavior pattern usually presents great complexity which brings the challenge for its functionality classification. In this paper, we focus on the analysis of execution traces of mobile apps to facilitate the automatic classification of their functionalities. Based on the formulation of mobile app functionality classification, we proposed a systematic framework named RaT (Run-and-Tell) to guide the design of trace-driven mobile app functionality classification. Guided by RaT, we introduced two behavior representation methods based on

收稿日期:2021-03-23;在线发布日期:2021-11-26.本课题得到湖北省重点研发计划(2021BAA039)、广东省重点领域研发计划(2020B010164002)资助.**马 超**,博士,讲师,中国计算机学会(CCF)专业会员,主要研究方向为程序理解、深度学习、时间序列分析.E-mail:whmachao@ieee.org.**李俊彤**,博士,主要研究方向为人机交互、普适计算、时间序列挖掘.**曹建农**,博士,讲座教授,中国计算机学会(CCF)高级会员,主要研究领域为分布式计算、无线网络和移动计算、大数据和机器学习、云计算和边缘计算.**蔡华谦**,博士,特聘副研究员,中国计算机学会(CCF)专业会员,主要研究方向为移动应用、分布式系统.**吴黎兵**,博士,教授,中国计算机学会(CCF)杰出会员,主要研究领域为物联网、网络安全、无线网络、机器学习.**石小川**(通信作者),博士,副教授,中国计算机学会(CCF)专业会员,主要研究方向为大数据分析、强化学习、无线传感器网络、数据挖掘.E-mail:shixiaochuan@whu.edu.cn.

statistical characteristics and semantic features extracted from execution traces, respectively. Afterward, by integrating 2 kinds of behavior representations with 4 types of mobile app functionality classifiers based on neural networks (i.e. MLP, FCN, ResNet, and LSTM), 8 different solutions are implemented for mobile app functionality classification. Furthermore, by leveraging the program instrumentation technique, we collected 876 execution traces of 17 Android apps of 3 categories covering 13 different functionalities from Google Play to build the dataset for evaluation. Experimental results show that, by integrating semantics-based representations, solutions based on the RaT framework achieve 73.2% inter-category classification accuracy on average on the collected dataset, which significantly outperforms the baselines.

Keywords program comprehension; mobile app functionality classification; execution trace; behavior representation; neural network

1 引 言

程序功能分类是指通过对程序进行深入理解进而从中识别程序主要功能的任务.这项任务是程序理解的主要任务之一[1],它有助于完成多种类型的软件工程任务,譬如软件维护、软件演化、遗留系统重构[2]、软件产品线[3]等.然而,面对规模日益庞大的软件系统[4],使程序功能分类自动化、智能化一直是一项极具挑战性的任务.

目前大多数程序功能分类解决方案主要分为基于静态分析和基于动态分析的两大类.基于静态分析的功能分类解决方案指的是通过对目标程序源代码的分析实现对其主要功能的辨识.其主要优点在于仅需对源代码进行扫描即可进行功能分类,从而使得整个功能分类过程的开销可控.但是,基于静态分析的功能分类解决方案必须获得目标程序的源代码,而该前提在某些实际场景(如遗留系统重构、源代码缺失或恶意软件检测等)中往往难以得到满足.因此,为有效应对无法获取源代码场景下的程序功能分类任务,本文拟采用基于动态分析的程序功能分类解决方案.基于动态分析的程序功能分类解决方案在满足目标程序可执行系统可获取的前提下,通过对目标程序执行轨迹的分析进行程序功能分类.由上可见,基于静态分析的程序功能分类解决方案与基于动态分析的程序功能分类解决方案在前置条件上存在明显区别(是否需要源代码),同时又具有较好的互补性,两者的并行探索将有助于推动程序功能分类任务适用场景的不断丰富.

然而,由于运行时环境的动态性和开发框架的差异性,执行轨迹内包含的程序行为模式往往呈现出较高的复杂性.对于基于动态分析的程序功能分类解决方案而言,执行轨迹复杂性带来的挑战主要来自三个方面.首先,执行轨迹包含的方法调用数量多且调用关系复杂,如何以结构化的形式完整保存执行轨迹行为信息是第一个挑战.其次,执行轨迹中的方法调用既包含与功能相关的方法调用又包含与功能无关的方法调用,如何将两者进行有效区分是第二个挑战.最后,在获取执行轨迹中与功能相关的方法调用信息后,如何建立从方法调用信息到功能标签的有效映射是第三个挑战.

为有效应对上述三个挑战,同时考虑到移动应用近年来的快速普及趋势,本文提出了一个通用的移动应用功能分类框架 RaT(Run-and-Tell).首先,本文提出了基于方法调用树的执行轨迹建模方法以应对第一个挑战.其次,通过利用执行轨迹方法调用序列的时序特征或语义特征分别提出了两种不同的执行轨迹行为表征方法,尝试从两个不同的视角应对第二个挑战.最后,通过设计四种不同的深度神经网络分类器,尝试拟合从方法调用模式到功能标签的非线性映射,以期有效提升移动应用功能分类效果.由于 RaT 框架几乎不依赖于特定系统实现,因此可以将其应用于其他类型程序以指导构建执行轨迹驱动的功能分类解决方案.

本文的主要贡献体现在以下三个方面:

(1)以与解决方案解耦合的方式对移动应用功能分类问题进行形式化建模,从而使得本文所设计的解决方案更易于推广至适用于执行轨迹驱动的其他类型软件系统的功能分类解决方案中;

(2)提出了两种分别基于执行轨迹统计特征和语义特征的移动应用行为表征生成方法;

(3)在本文提出的包含 4 层结构通用框架 RaT

(Run-and-Tell)的指导下，设计并实现了集成 2 种执行轨迹行为表征生成方法与 4 类神经网络分类器的 8 个移动应用功能分类解决方案；并利用采集自 17 个随机选择安卓应用程序且涵盖 13 种不同功能的总共 876 条执行轨迹开展了综合性实验. 通过对实验结果的对比分析，不仅验证了 RaT 框架的合理性及所设计解决方案相对现有基线方法的明显性能提升，而且对可能影响移动应用功能分类效果的若干因素进行了探讨.

本文第 2 节对移动应用功能分类问题和执行轨迹建模进行形式化描述；第 3 节详细阐述通用框架 RaT、2 种执行轨迹表征方法及 4 类神经网络分类器的具体设计；第 4 节从实验设置、实验结果对比、影响因素归因等多个方面对所提出 RaT 框架的合理性及解决方案的有效性进行验证及分析；第 5 节对移动应用功能分类的相关工作进行介绍；最后在第 6 节总结本文工作并展望未来的潜在研究方向.

2　问题描述

在“移动应用功能分类”(Mobile App Functionality Classification)术语选择中，本文主要基于两点原因选择术语“功能”(Functionality)而非“特性”(Feature). 首先，在软件工程领域，“特性”一词通常是指能够将某个程序与其他程序区分开来的特色功能. 而为了使本文的目标问题更具有一般性，本文没有对特色功能与通用功能(如登录、注销等)进行区分. 其次，在同时涉及软件工程和机器学习领域背景的情况下，可以有效避免软件工程上下文中的术语“特性”(Feature)与机器学习上下文中的术语“特征”(Feature)发生语义混淆. 移动应用功能分类问题的形式化描述如下所示.

给定：

(1) 执行轨迹集合 $X=\{x_i\}$ 和功能标签集合 $Y=\{y_k\}$，其中每一条执行轨迹 x_i 关联一个功能标签 $y_i\in Y$，即执行轨迹 x_i 采集于目标移动应用执行功能 y_i 期间；

(2) 执行轨迹 $x_i=\{mc_j\}$ 都是为满足功能 y_i 而执行的一个方法调用序列，其调用的每个方法为包含主调方法 mc_j^{caller}、开始时刻 mc_j^{start}、结束时刻 mc_j^{end}、输入参数 mc_j^{in}、返回值 mc_j^{out}、读变量 $mc_j^{var\_read}$ 及写变量 $mc_j^{var\_write}$ 的 7 元组；

(3) 执行轨迹集合 $X=\{x_i\}$ 被划分为两个不相交的子集，训练集 X_{Train} 和测试集 X_{Test}，即 $X=X_{\text{Train}}\cup X_{\text{Test}}$ 且 $X_{\text{Train}}\cap X_{\text{Test}}=\varnothing$.

假设：

(1) 执行轨迹集合 X 具有以下性质：

① $\exists p,q(|x_p|\neq|x_q|)$，其中 $y_p=y_q$；

② $\exists p,q,r(D(x_p,x_q)\geq D(x_p,x_r))$，其中 $y_p=y_q$ 且 $y_p\neq y_r$，$D(x_p,x_q)$ 表示执行轨迹 x_p 和 x_q 在欧式空间 D 中的距离；

(2) 存在表征函数 $F(x_i)=\boldsymbol{z}_i\in R^N$，使得 $\forall p,q,r(D(z_p,z_q)<D(z_p,z_r))$，其中 $y_p=y_q$ 且 $y_p\neq y_r$，N 为表征向量 $\boldsymbol{z}_i$ 的维度.

目标：

(1) 设计由表征函数 F 和分类器 C 构成的移动应用功能分类解决方案. 首先，学习表征函数 F 拟合映射 $X_{\text{Train}}\rightarrow Z$ 以最大化 $\sum_{y_p\neq y_r}D(z_p,z_r)-\sum_{y_p=y_q}D(z_p,z_q)$；其次，构建分类器 C 拟合映射 $Z\rightarrow Y$ 以最大化移动应用功能分类准确率(如式(1)所示).

$$Acc_C^F=\frac{\sum_{x_i\in X_{\text{Test}}}hit_{C(F(x_i))}}{|X_{\text{Test}}|}\tag{1}$$

其中，$F(x_i)$ 为执行轨迹 x_i 应用行为表征方法 F 构造的表征向量，$C(F(x_i))\in Y$ 则为以 $F(x_i)$ 作为功能分类器 C 的输入预测功能标签. 若预测功能标签与执行轨迹 x_i 真实功能标签一致，则 $hit_{C(F(x_i))}$ 赋值为 1，否则为 0. 对测试集 X_{Test} 中每条执行轨迹的功能标签预测结果求和之后其取值范围为 $[0,|X_{\text{Test}}|]$. 因此，移动应用功能分类准确率 Acc_C^F 的值域为 $[0,1]$，该评价指标越接近 1 说明由执行轨迹行为表征方法 F 与功能分类器 C 构成的移动应用功能分类解决方案效果越好，该评价指标越接近 0 则说明该解决方案效果越差.

执行轨迹集合 X 的第一个性质表明即使功能标签相同，不同的执行轨迹中调用的方法数量也不同. 而现有的大多数分类器仅接受具有固定长度的输入. 因此，如何将复杂多变且大小各异的执行轨迹转换为具有固定长度且保留有利于功能分类信息的表征向量成为本文工作要应对的首要问题. 此外，执行轨迹集合 X 的第二个性质表明，执行轨迹集合 X 中可能存在功能标签相同的执行轨迹 x_p 与 x_q，它们之间的欧式距离反而大于或等于功能标签相异的执行轨迹 x_p 与 x_r 之间的欧式距离，这使得移动应用功能分类任务更加具有挑战性.

3　移动应用功能分类解决方案

3.1　框架概述

为了实现对移动应用功能的准确分类，同时有

效降低手工特征提取开销,本文提出了执行轨迹驱动的 RaT 框架. RaT 框架涉及三种类型的物理设备:移动应用运行设备、执行轨迹采集设备和功能分类设备. 如图 1 所示,RaT 框架的逻辑结构由四层组成,其中每一层都包含一组部署在上述物理设备上的模块. 首先,在移动应用运行设备(如智能手机)上,执行轨迹生成层通过使用程序插桩技术动态监视移动应用 ma_i 上功能 y_j 的执行过程. 接着,在执行轨迹采集设备上,执行轨迹采集层中的执行轨迹采集器将持续记录并持久化存储移动应用 ma_i 上执行功能 y_j 对应产生的执行轨迹 x_j. 然后,功能分类设备上的执行轨迹预处理层负责解析包含不同执行细节信息的执行轨迹 x_j,并将它们转换为某种便于后续处理的统一数据结构(详见 3.2 节),从而实现对移动应用运行时行为的自动表征. 最后,功能分类层通过使用这些具有统一数据结构的执行轨迹,以智能化的方式对移动应用的功能进行分类. 功能分类层包含执行轨迹行为表征生成器(详见 3.3 节)和功能分类器两个模块(详见 3.4 节). 执行轨迹行为表征生成器仅需配置少量超参数即可生成高维表征向量,而基于神经网络的功能分类器则能够自动拟合高维表征向量与功能标签之间的非线性映射关系. 因此,相对于传统手工特征提取过程,基于 RaT 框架的移动应用功能分类解决方案有望在提升性能的同时还能够有效降低特征提取开销.

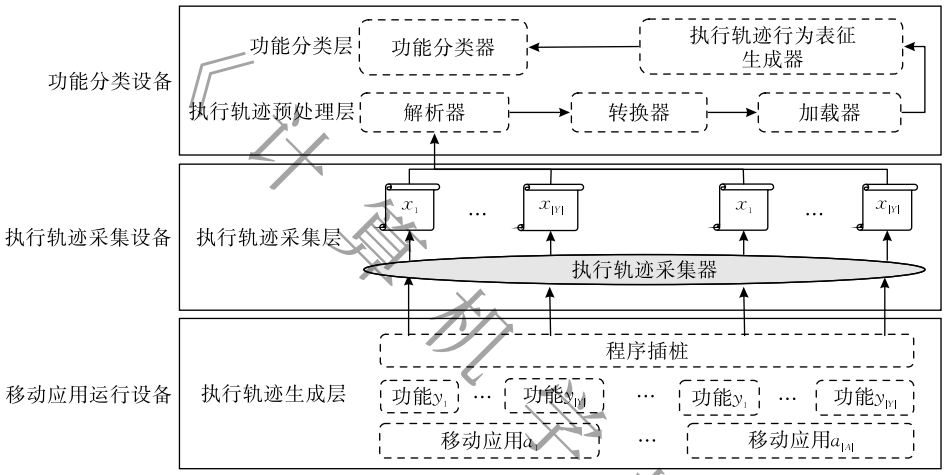


图 1 执行轨迹驱动的移动应用功能分类框架 RaT

3.2 执行轨迹预处理

对执行轨迹进行预处理的主要目的是构建统一的数据结构,以结构化地存储执行特定功能的目标程序的运行时行为信息. 执行轨迹预处理主要包括解析器,转换器和加载器三个部分. 其中,解析器负责从来自于执行轨迹采集层的原始执行轨迹(一般为 Json 格式文件)中提取方法调用信息. 转换器则负责将初始按照调用时序排列的方法调用序列转换为统一的数据结构,以便于后续的处理. 加载器在按统一数据结构对执行轨迹进行重构之后,将其输入到执行轨迹行为表征生成器. 在本文的工作中,执行轨迹预处理层和功能分类层的模块部署在同一台物理服务器(即功能分类设备)上,这使得加载器更容易实现. 解析器和加载器的设计及实现相对简单,而更加关键的任务则是将原始执行轨迹转换为统一数据结构,因此本节的剩余部分将着重说明转换器的设计思路.

如第 2 节对移动应用功能分类问题的定义所述,移动应用的执行轨迹 x_i 包含一系列的方法调用 $\{mc_j\}$. 这一系列方法之间的多层嵌套主调-被调关系通过每个方法 mc_j 中的 mc_j^{caller} 的元素进行记录. 除了方法之间的调用关系之外,执行轨迹还包含每个方法调用的具体执行细节信息,如方法 $\{mc_j\}$ 被调用的开始时间 mc_j^{start}、结束时间 mc_j^{end}、输入参数 mc_j^{in}、输出参数 mc_j^{out}、读变量 $mc_j^{var\_read}$ 及写变量 $mc_j^{var\_write}$. 因此,我们可以将上述方法调用关系及具体执行细节信息作为执行轨迹行为表征的依据. 基于以上分析,本文将执行轨迹建模为树型结构,其中树型结构的每一个节点 $MCT(x_i)_j$ 都代表执行轨迹 x_i 中的一条方法调用 mc_j,每一个父节点是其所有子节点的主调方法(即 $MCT(x_i)_j^{parent}=mc_j^{caller}$). 执行轨迹中的每个方法调用的具体执行细节信息以元组的形式存储在树型结构的每个节点之中(即 $MCT(x_i)_j=mc_j$). 本文将这种树型结构命名为“方法调用树”(Method Call Tree,简记为 MCT). 算法 1

中展示了根据给定的执行轨迹 x_i 构建其对应方法调用树 $MCT(x_i)$ 的伪代码.

算法 1. 方法调用树 Build_MCT.

输入：$x_i=\{mc_j\}$

输出：$MCT(x_i)$

步骤 1. 初始化方法调用树的根节点；

1. $MCT(x_i)_{root}\leftarrow$NULL
2. FOR each $j\in[1,|x_i|]$ DO
3. IF mc_j^{caller} is NULL THEN
4. $MCT(x_i)_{root}\leftarrow mc_j$
5. BREAK
6. END IF
7. END FOR
8. $MCT(x_i)_0^{parent}\leftarrow$NULL
9. $MCT(x_i)_0^{id}\leftarrow 0$
10. $MCT(x_i)_0^{start}\leftarrow MCT(x_i)_{root}^{start}$
11. $MCT(x_i)_0^{end}\leftarrow MCT(x_i)_{root}^{end}$
12. $MCT(x_i)_0^{in}=MCT(x_i)_0^{out}\leftarrow$NULL
13. $MCT(x_i)_0^{var_read}=MCT(x_i)_0^{var_write}\leftarrow$NULL

步骤 2. 构建方法调用树的剩余部分；

14. FOR each $j\in[1,|x_i|]$ DO
15. IF mc_j^{caller} is NULL THEN
16. $MCT(x_i)_j^{parent}\leftarrow 0$
17. ELSE
18. $MCT(x_i)_j^{parent}\leftarrow s$ if mc_j^{caller} is mc_s
19. END IF
20. $MCT(x_i)_j\leftarrow mc_j$
21. END FOR

步骤 3. 输出方法调用树.

22. RETURN $MCT(x_i)$

算法 1 包含三个步骤. 步骤 1 是初始化 MCT 的根节点. 由于一条执行轨迹仅与一种功能相关，因此可将整个执行轨迹抽象为 MCT 的根节点. 算法第 1 到第 7 行遍历执行轨迹中的所有方法调用，从而找出其中的顶层方法调用(即没有显式主调方法的方法调用). 第 8 到第 13 行对根节点的运行信息进行了初始化. 步骤 2(第 14 到第 21 行)利用方法调用中的 mc_j^{caller} 元素抽取出不同方法调用之间的多层嵌套主调-被调关系，然后基于抽取出的调用关系将所有方法调用对应的节点(即 $MCT(x_i)_j$)组织为树型结构(即 $MCT(x_i)$). 步骤 3 返回执行轨迹的统一树型结构 $MCT(x_i)$. 在构建方法调用树 $MCT(x_i)$ 的过程中，执行轨迹中的每条方法调用仅被访问 1 次，因而算法 1 的时间复杂度为 $O(n)$. 本文在 3.3 节提出了两种以方法调用树 $MCT(x_i)$ 为基础的执行轨迹行为表征方法，分别是基于统计特征和基于语义特征的表征方法.

3.3　执行轨迹行为表征

一般而言，执行轨迹的行为表征应满足以下两个条件：(1) 由于绝大多数分类器(如神经网络分类器)的输入必须是固定长度的向量，因而生成的执行轨迹行为表征应是等长的；(2) 考虑到移动应用功能分类问题的特点，执行轨迹的行为表征应该保留其与功能具有关联性的特征，从而帮助提高功能分类器的准确率. 以方法调用树 $MCT(x_i)$ 为基础，本节将介绍执行轨迹行为表征的两种生成方法. 第一种方法从执行轨迹中提取特定变量的原始序列，并采用分布近似技术生成固定长度的表征. 本文将通过该方法生成的执行轨迹行为表征称为基于统计特征的表征. 第二种方法则从执行轨迹中提取与程序功能潜在相关的关键词，并将这些关键词对应的词向量连接起来形成执行轨迹行为表征. 通过该方法生成的执行轨迹行为表征称为基于语义特征的表征.

3.3.1　基于统计特征的执行轨迹行为表征

如算法 2 中所示，生成基于统计特征的执行轨迹行为表征(Statistical Characteristic-based Representation，简记为 SCR)包括三个主要步骤. 步骤 1(算法 2 第 1 行)前序遍历 $MCT(x_i)$ 除根节点以外的节点得到方法调用序列 $MCS(x_i)$. 步骤 2(算法 2 第 2 到第 10 行)，选取特定的变量 v_j，利用函数 $getValue(mc,v_j)$ 计算其原始时间序列. 本文选取了以下三个变量：(1) 每个方法调用的持续时间，以其开始时刻和结束时刻之间的时间间隔表示；(2) 每个方法调用读变量的数量；(3) 每个方法调用写变量的数量. 以上三个变量的原始时间序列计算函数的定义如式(2)、(3)及(4)所示. 步骤 3(算法 2 第 11 到 15 行)，利用函数 $Distr(RAW_{x_i}^{v_j})$(如 Kaplan-Meier 估计[5])估计每个变量的原始时间序列分布，接着应用函数 $Sample(Distr(RAW_{x_i}^{v_j}),d)$(例如 ECDF[6])从估计分布 $Distr(RAW_{x_i}^{v_j})$ 中采样 d 个数据点，然后将执行轨迹 x_i 每个变量的采样数据点水平拼接起来以构建基于统计特征的执行轨迹行为表征 $SCR(x_i)$. 最后，(算法 2 第 16 行)返回所生成的基于统计特征的执行轨迹行为表征 $SCR(x_i)$.

$$getValue(mc,duration)=mc^{end}-mc^{start} \quad (2)$$

$$getValue(mc,var\_read)=|mc^{var\_read}| \quad (3)$$

$$getValue(mc,var\_write)=|mc^{var\_write}| \quad (4)$$

算法 2. 基于统计特征的行为表征Generate_SCR.

输入：$MCT(x_i)$，$V=\{v_j\}$，d

输出：$SCR(x_i)$

步骤 1. 获取除根节点外的方法调用树 $MCT(x_i)$ 前序遍历方法调用序列 $MCS(x_i)$；

1. $MCS(x_i)\leftarrow Preorder(MCT(x_i))\backslash MCT(x_i)_0$

步骤 2. 对于目标变量集合 V 中的每个变量 v_j，依据 $MCS(x_i)$ 抽取该变量对应的原始时间序列 $RAW_{x_i}^{v_j}$，并获得其并集 RAW_{x_i}；

2. $RAW_{x_i}\leftarrow\{\}$
3. $V=\{duration, var\_read, var\_write\}$
4. FOR each $v_j\in V$ DO
5. $\quad RAW_{x_i}^{v_j}\leftarrow\{\}$
6. $\quad$ FOR each $mc\in MCS(x_i)$ DO
7. $\quad\quad RAW_{x_i}^{v_j}\leftarrow RAW_{x_i}^{v_j}\cup\{getValue(mc,v_j)\}$
8. $\quad$ END FOR
9. $RAW_{x_i}\leftarrow RAW_{x_i}\cup\{RAW_{x_i}^{v_j}\}$
10. END FOR

步骤 3. 基于每个变量 v_j 的原始时间序列 $RAW_{x_i}^{v_j}$，利用分布估计技术（*Distr* 函数）和分布采样技术（*Sample* 函数）获得其对应的统计表征 $SCR(x_i)^{v_j}$，并获得其并集 $SCR(x_i)$；

11. $SCR(x_i)\leftarrow\{\}$
12. FOR each $v_j\in V$ DO
13. $\quad SCR(x_i)^{v_j}\leftarrow Sample(Distr(RAW_{x_i}^{v_j}),d)$
14. $\quad SCR(x_i)\leftarrow SCR(x_i)\cup SCR(x_i)^{v_j}$
15. END FOR

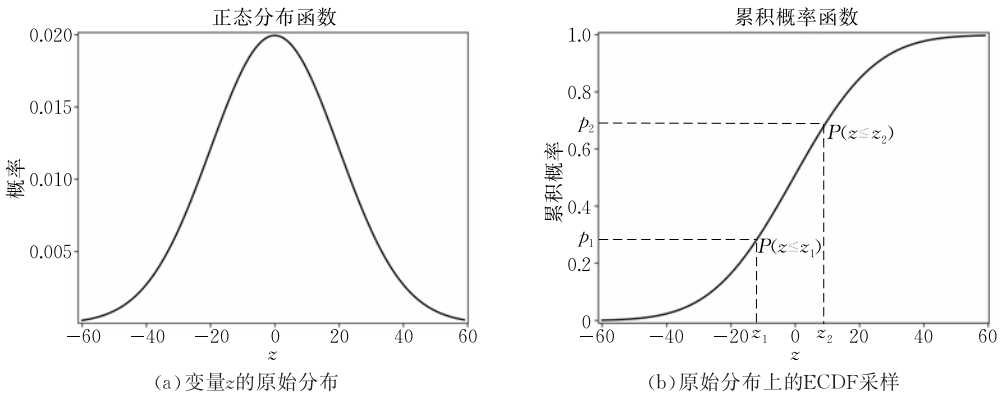
步骤 4. 输出 x_i 基于统计特征的行为表征.

16. RETURN $SCR(x_i)$

如第 2 节移动应用功能分类问题定义中执行轨迹的第一个性质所述，即使是具有相同功能标签的执行轨迹，在方法调用数量和变量访问数量上也可能存在差异性. 因而探索合适的执行轨迹行为表征方法以克服不同执行轨迹之间的差异性是实现程序功能分类的关键要素之一. 一种通行方案是对目标变量的分布进行合理采样，从而有效应对样本差异. 该类工作主要有两种技术路线：基于距离的方法和基于分布近似的方法. 基于距离方法的主要目标是找到一种计算目标变量对应序列距离的方法. 动态时间规整（Dynamic Time Wrapping，简记为 DTW）是其中的代表方法，它能够计算具有不同长度的两个序列之间的距离. 然而，由于耗时较长，DTW 仅适用于基于距离的分类器（如 K-NN 分类器）. 考虑到复杂执行轨迹中方法调用序列不仅长度可变且可能较长，因此本文采用第二种技术路线（即分布近似方法）来生成基于统计特征的执行轨迹行为表征. 常用的基于分布近似的方法包括 BDT[7-8]、SAX[9-10] 以及基于高斯混合模型的方法[11] 等. 然而，真实场景中原始序列并不总是独立同分布的. 为了解决这一问题，本文采取了一种替代方案，即使用等价累计分布函数（ECDF[6]）估计目标变量 v_j 的分布 $D(v_j)$ 的累积概率函数 $P_{D(v_j)}$. 累积概率函数 $P_{D(v_j)}$ 的具体定义如式(5)所示.

$$P_{D(v_j)}(z)=P(v_j\leq z) \tag{5}$$

显而易见，$P_{D(v_j)}$ 是单调递增的，并且其值曲线自然反映了 $P_{D(v_j)}$ 从 0 到 1 的变化过程中变量 v_j 的总体分布情况. 图 2 是一个 ECDF 的具体示例，其中共采样了 2 个数据点 (z_1,z_2).



(a) 变量z的原始分布　　(b) 原始分布上的ECDF采样

图 2　ECDF 采样示例

对于 d 个数据点 $p_i\in R_{[0,1]}$，可以根据式(6)和(7)计算其在分析窗口 i 中的值 z_i. 然后，对每个分析窗口 i 的 d 个采样数据点依照式(8)进行水平拼接，并将其作为变量 v_j 的最终表征返回. 对于执行轨迹 x_i 的每个变量 v_j，重复执行上述过程，通过水平拼接执行轨迹 x_i 所有变量的 $SCR(x_i)_{v_j}$（即算法 2 第 13 行），最终生成基于统计特征的执行轨迹行为表征.

$$D(v_j)=\{p_i\}\subset R^d_{[0,1]},\ p_i<p_{i+1} \tag{6}$$

$$SCR(x_i)^i_{v_j}=\{z_i,\exists k:P^i_{D(v_j)}=p_k\},\ z_i<z_{i+1} \tag{7}$$

$$SCR(x_i)_{v_j}=\bigcup SCR(x_i)^i_{v_j} \tag{8}$$

3.3.2　基于语义特征的执行轨迹行为表征

除了统计特征信息之外，执行轨迹中还包含丰富的语义信息，例如方法调用的函数名和变量名等. 而此类语义信息对于识别目标程序运行过程中执行的功能具有较大帮助. 该想法源自一个合理的假设，程序中函数与变量的名称与其实现的功能具有某种程度的语义关联，这是软件开发人员出于源代码的可读性与可重用性考虑设计的. 因此，本节提出了一种基于语义特征的执行轨迹行为表征方法，利用执行轨迹中丰富的语义信息来生成表征，从而提升功能分类器的准确率.

如 3.2 节所述，在执行轨迹预处理过程中，执行轨迹 x_i 被转换为方法调用树 $MCT(x_i)$，其每个节点都是具有多个属性的方法调用. 为了有效利用这些属性中与程序功能相关的语义信息，需要从执行轨迹的方法调用树 $MCT(x_i)$ 中选择一个关键词集合. 关键词的选择主要遵循两个原则：(1) 具有较高的可读性；(2) 与执行轨迹对应功能具有语义相关性.

算法 3 详细描述了基于语义特征的执行轨迹行为表征(Semantic Representation，简记为 SR)的生成过程，主要包括三个步骤. 步骤 1(算法 3 第 1 到第 10 行)的双重循环通过调用函数 $getRank(KW_{x_i},k)$ 从执行轨迹集合 X 的每条执行轨迹 x_i 中选择 k 个关键词，并根据这些关键词构造词汇表 V. 第 5 行的函数 $WS(mc_j)$ 利用驼峰规则分词从方法调用 mc_j 的重要文本属性(方法名，变量名)中提取原始词素，从而形成候选关键词集合 $KW^{mc_j}_{x_i}$. 第 6 行的函数 $lookUp(KW^{mc_j}_{x_i},Dict)$ 则通过查找预先准备的字典 $Dict$(本文使用的字典包含超过 40 万个英文单词)过滤出 $KW^{mc_j}_{x_i}$ 中可读性较高的关键词. 第 8 行通过调用函数 $getRank(KW_{x_i},k)$ 选择每条执行轨迹 x_i 的前 k 个关键词 $K\_KW_{x_i}$. 第 9 行将 X 中每条执行轨迹 x_i 的前 k 个关键词 $K\_KW_{x_i}$ 的并集构建为词汇表 V.

算法 3.　基于语义特征的行为表征 Generate_SR.

输入：$X=\{x_i\}$，$Dict$，k，d

输出：$SR(x_i)$

步骤 1. 从每条执行轨迹 x_i 中选取 k 个关键词并为执行轨迹集合 X 构建词汇表 V；

1. $V=\{\}$
2. FOR each $x_i\in X$ DO
3. $KW_{x_i}=\{\}$
4. FOR each $mc_j\in MCT(x_i)$ DO
5. $KW^{mc_j}_{x_i}=WS(mc_j)$
6. $KW_{x_i}=KW_{x_i}\cup lookUp(KW^{mc_j}_{x_i},Dict)$
7. END FOR
8. $K\_KW_{x_i}=getRank(KW_{x_i},k)$
9. $V=V\cup K\_KW_{x_i}$
10. END FOR

步骤 2. 利用词汇表 V 为执行轨迹 x_i 生成 d 维的向量表征；

11. $vec\_model=trainModel(V,d)$
12. $SR(x_i)=\{\}$
13. FOR each $kw\in K\_KW_{x_i}$ DO
14. $SR(x_i)=SR(x_i)\cup getVector(kw,vec\_model)$
15. END FOR

步骤 3. 输出 x_i 基于语义特征的行为表征.

16. RETURN $SR(x_i)$

函数 $getRank(KW_{x_i},k)$ 的实现方法有很多种，本文采用了实现简单且应用广泛的 tf-idf[12]. 首先，需要计算关键词 kw 在执行轨迹 x_i 中出现的词频 tf_{kw,x_i}. 其次，需要计算关键词 kw 在所有执行轨迹集合 X 中的逆文档频率 idf_{kw}，其定义如式(9)所示，其中 df_{kw} 为执行轨迹集合 X 中包含关键词 kw 的执行轨迹数量. 最后，如式(10)所示，通过将词频 tf_{kw,x_i} 与逆文档频率 idf_{kw} 相乘，即可为执行轨迹中的每个候选关键词 kw 赋予一个其相对于所在执行轨迹 x_i 的合理权重值 $tf\text{-}idf_{kw,x_i}$. 其后，我们即可依据执行轨迹 x_i 中每个候选关键词的 $tf\text{-}idf_{kw,x_i}$ 值筛选对于区分不同执行轨迹有益的关键词列表 $K\_KW_{x_i}$.

$$idf_{kw}=\log\frac{|X|}{df_{kw}} \tag{9}$$

$$tf\text{-}idf_{kw,x_i}=tf_{kw,x_i}\times idf_{kw} \tag{10}$$

步骤 2(算法 3 第 11 到第 15 行)包括向量模型训练和关键词表征生成两个环节. 第 11 行中函数 $train\_model(V,d)$ 的作用是在词汇表 V 上训练 d 维向量模型，同时保留不同向量之间的语义相似度信息. 从第 13 行开始的循环将关键词 kw 输入训练好的向量模型 vec_model 以生成每条执行轨迹 x_i 对应 $K\_KW_{x_i}$ 中的每个关键词 kw 的 d 维向量表征. 重复进行以上操作后将执行轨迹 x_i 中每个关键词的 d 维向量表征进行水平拼接，即可得到基于语义特征的执行轨迹行为表征 $SR(x_i)$. 步骤 3(算法 3 第 16 行)返回执行轨迹 x_i 的基于语义特征的行为表征 $SR(x_i)$.

鉴于 word2vec 模型[13]在大量实际应用场景中展现出来的稳定性能，本文使用该模型实现函数 *train_model*(V,d). 基于包含所有执行轨迹关键词的词汇表 V，通过最大化如式(11)所示的目标函数，word2vec 模型即可为词汇表中的每个关键词生成多维向量表征. 式(11)中的参数 c 为训练上下文的长度. 如式(12)所示，$p(kw_{t+j}|kw_t)$则采用 Softmax 函数进行定义.

$$\frac{1}{|V|}\sum_{t=1}^{|V|}\sum_{-c\le j\le c,j\ne 0}\log p(kw_{t+j}|kw_t) \tag{11}$$

$$p(kw_{t+j}|kw_t)=\frac{e^{h(kw_t,kw_{t+j})}}{\sum_{t=1}^{|V|}e^{h(kw_t,kw_{t+j})}} \tag{12}$$

3.4 基于神经网络的功能分类器

相对于传统机器学习模型，神经网络体现出更强大的复杂函数拟合能力，在解决非线性问题方面具有巨大潜力. 因此，为了处理执行轨迹行为表征与程序功能之间的复杂映射关系，本文选取 4 种典型神经网络：多层感知机(MLP)、全卷积网络(FCN)、残差网络(ResNet)和长短期记忆网络(LSTM)作为候选功能分类器. 上述 4 种基于神经网络的功能分类器均包含尺寸相同的输入层(与执行轨迹行为表征尺寸相等)和 Softmax 层(与程序功能类别数量相等)，其中输入层用于接收执行轨迹行为表征，Softmax 层则将分类结果映射到具体的程序功能类别标签. 所有的分类器均使用线性修正单元(ReLU 层)作为激活函数. 对于基于多层感知机的功能分类器，通过增加 Dropout 层对部分权重进行随机丢弃以避免模型过拟合. 而对于基于全卷积网络和残差网络的功能分类器，则增加了 Conv1D 层用于执行一维卷积运算. 同时，为了提高上述两类功能分类器的训练速度，本文还采用了批量归一化(Batch Normalization)[14]技术(即 BatchNorm 层). 批量归一化技术将不同批次样本的分布通过归一化操作转换为标准正态分布，以确保每个批次样本分布的一致性，因而可以避免神经网络训练过程中发生梯度消失，从而加速神经网络的训练过程. 此外，还引入全局平均池化(Global Average Pooling)[15]方法(即 GAP 层)以替换最后一层全连接层，从而显著减少模型的参数数量，在避免神经网络过拟合的同时能够进一步加速神经网络的训练过程. 本文实验中所采用的多层感知机(MLP)、全卷积网络(FCN)、残差网络(ResNet)及长短期记忆网络(LSTM)的网络结构和部分超参数设置如图 3 所示.

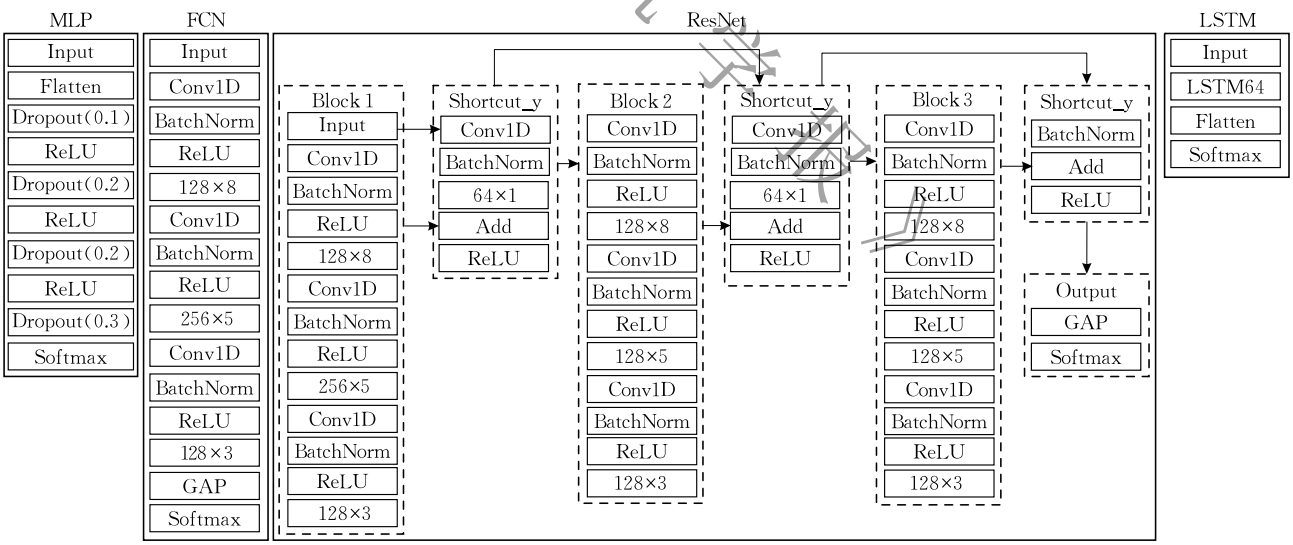


图 3 神经网络功能分类器网络结构及其主要超参数设置

4 实验结果与分析

4.1 实验设置

为了验证本文提出移动应用功能分类解决方案的有效性，首先将目标移动应用(即安卓应用)安装运行在一台物理安卓智能手机上. 然后，利用文献[2]中介绍的程序插桩技术，在执行特定功能期间采集目标安卓应用产生的执行轨迹. 如表 1 所示，本实验从 Google Play 应用商店共采集了来自 3 个安卓应用类别(Calendar、Memo 及 Photography)涵盖 13 种不同功能(详见表 1“应用主要功能”列)的 17 个安卓应用(详见表 1“应用名称”列)总计 876 条执行轨迹. 此外，为了更进一步探究移动应用功能分

类问题的复杂性，实验中人为增加了一个虚拟的应用类别 Mixed. 该应用类别由上述 3 个安卓应用类别的所有功能和执行轨迹组合构成. 为避免训练集与测试集重叠导致的数据泄漏，我们采用留一法，将每个安卓应用类别对应的所有执行轨迹以应用为单元划分为互不重叠测试集和训练集. 其中测试集包含随机选定的某个应用的所有执行轨迹，而训练集则包含同一应用类别下其他应用的所有执行轨迹. 表 1 中展示了本实验所采集的数据集相关详细信息.

表 1　移动应用功能分类实验数据集描述

应用类别	应用名称	应用主要功能	执行轨迹
Calendar	com.appgenix.bizcal	添加/删除/编辑事件	36
	com.droidfoundry.calendar	添加/删除/编辑事件	36
	com.joshy21.vera	添加/删除/编辑事件	36
	com.simplemobiletools	添加/删除/编辑事件	36
Memo	com.abhi.newmemo	添加/删除/编辑/共享备忘录	48
	com.honny.memo	添加/删除/编辑/共享备忘录	48
	com.odgen.memo	添加/删除/编辑/共享备忘录	48
	notepad.note.notas	添加/删除/编辑/共享备忘录	48
	org.whiteglow.keepmynotes	添加/删除/编辑/共享备忘录	48
	com.beka.tools.quicknotes	添加/删除/编辑/共享备忘录	48
	org.dayup.gnotes	添加/删除/编辑/共享备忘录	48
Photography	com.almalence.opencam	前/后置摄像头，照/删相，录视频，开闪光	72
	com.cyworld.camera	前/后置摄像头，照/删相，开闪光	60
	com.falstad.megaphotofree	前/后置摄像头，照相，录视频	48
	com.tencent.ttpic	前/后置摄像头，照/删相，录视频，开闪光	72
	com.simplemobiletools.camera	前/后置摄像头，照/删相，录视频，开闪光	72
	net.sourceforge.opencamera	前/后置摄像头，照/删相，录视频，开闪光	72
Mixed	上述所有应用	上述所有功能	876
总计	17	13	1752

如表 2 所示，本文实验的移动应用功能分类解决方案包含两个关键组件：行为表征生成器和程序功能分类器. 本文将 3.3 节中提出的两种表征方法与 3.4 节中设计的四种分类器组合后构建了 8 种不同的移动应用功能分类解决方案. 为保证对比两种表征方法的公平性，本文将两种方法输出表征向量的长度设定为固定值 3000. 选取该值的主要动机在于该值可以整除表 2 中两种行为表征生成器的变量个数(即 $|V|$)或关键词个数(即 k)，从而在解决方案实现时无需对单个变量或关键词维度(即 d)进行向上或向下取整处理. 针对基于统计特征的行为表征(即 SCR)，从执行轨迹中选取 V 的三个变量(即 *duration*、*var_reads* 和 *var_writes*)，并使用分布近似技术(即 ECDF)采样 $3000/|V|$(即 1000)个数据点以表征每个变量. 为了构造基于语义特征的行为表征(即 SR)，从每条执行轨迹中选取 k 个关键词并将每个关键词转换为 $d=3000/k$ 维的向量，然后将所有关键词的特征向量进行水平拼接以得到用于表征整个执行轨迹的总长度为 3000 维的表征向量. 上述行为表征生成器关键参数设置如表 2 第 2 行所示. 此外，本文实验还考察了 6 个不同的 k 值对上述 8 种解决方法性能的影响.

表 2　移动应用功能分类解决方案关键组件主要参数设置

关键组件名称	实现简称	主要参数设置
行为表征生成器	SCR	$V=\{duration, var\_read, var\_write\}$, $d=3000/\|V\|$
	SR	$K=\{10, 20, 30, 40, 50, 60\}$, $d=3000/k(k\in K)$
程序功能分类器	MLP	$Epochs=200$, $Batch\ Size=16$, $Iterations=10$ $Learning\ Rate$: $Ini=1$, $Patience=10$, $Delta=10^{-4}$, $Factor=0.5$, $Min=10^{-1}$
	FCN	$Epochs=200$, $Batch\ Size=16$, $Iterations=10$ $Learning\ Rate$: $Ini=1$, $Patience=10$, $Delta=10^{-4}$, $Factor=0.5$, $Min=10^{-1}$
	ResNet	$Epochs=200$, $Batch\ Size=16$, $Iterations=10$ $Learning\ Rate$: $Ini=1$, $Patience=10$, $Delta=10^{-4}$, $Factor=0.5$, $Min=10^{-1}$
	LSTM	$Epochs=200$, $Batch\ Size=16$, $Iterations=10$ $Learning\ Rate$: $Ini=1$, $Patience=10$, $Delta=10^{-4}$, $Factor=0.5$, $Min=10^{-1}$

即使给定完全相同的训练样本及结构完全一样的神经网络，由于每次训练神经网络初始权重不一致等原因，即使同一神经网络的多次训练测试输出也往往会出现差异. 因此，为避免神经网络分类器输出偏差导致的性能认知误导，在训练样本(即行为表征)完全相同的每组实验中，对每种功能分类器进行

10 次(*Iterations*)独立重复训练,并以 10 次独立重复训练后功能分类器在测试集上的平均分类准确率作为最终评价结果.此外,基于神经网络的功能分类器还有训练环节的迭代次数(*Epochs*)、批次大小(*Batch Size*)及学习率(*Learning Rate*)三个超参数需要进行预先设置.首先,本实验将迭代次数设置为 200,因为过小的迭代次数可能使得神经网络欠拟合,而过大的迭代次数容易导致神经网络过拟合.其次,批次大小是一个 1 到训练集大小之间的正整数,一般为 2 的整数次幂,本实验根据经验将其设置为 16.最后,本实验对神经网络分类器训练阶段学习率的设置采用动态调整的策略.该动态调整策略将神经网络初始化学习率设置为 $Curr\_LR=Ini$,若连续 *Patience* 次迭代训练后神经网络分类器的损失函数值下降程度未达到 *Delta*,则更新学习率为 $Curr\_LR=Curr\_LR\times Factor$(当前学习率必须大于最小预设学习率,即 $Curr\_LR>Min$),直到迭代次数耗尽.值得注意的是,功能分类器在测试集上的分类准确率均是使用其在训练集上损失函数最小化周期时的模型检验所得.移动应用功能分类解决方案中各关键组件的主要参数设置详情如表 2 所示.

给定如表 1 所示的数据集后,本实验将按照以下步骤开展:(1)利用算法 1 对每条执行轨迹进行预处理以构建其对应的方法调用树;(2)基于每条轨迹的方法调用树,分别利用执行轨迹行为表征方法(即算法 2 和算法 3)生成基于统计特征和语义特征的执行轨迹行为表征;(3)将所生成的执行轨迹行为表征划分为训练集和测试集,其中训练集输入功能分类器(即 MLP、FCN、ResNet 或 LSTM)用于训练网络参数,测试集用于检验训练好的神经网络分类器在移动应用功能分类任务上准确率(如式(1)所示).

此外,为体现本文所提出 RaT 框架解决方案相对现有解决方案的性能提升效果,在 4.2 节中我们还对比了同样基于动态分析的 Xin 等人[16]所提出的 FEATUREFINDER 方法.具体而言,在本文数据集的基础上,我们测试了采用 k 近邻(k-NN)和支持向量机(SVM)分类器的 FEATUREFINDER 方法.其中,针对 k 近邻分类器,我们尝试了 $k\in[1,10]$ 的 10 种取值情况(变化步长为 1).而针对支持向量机分类器,我们采用了自动优化的高斯核函数,并尝试了惩罚系数 $C\in[0.1,1.0]$ 的 10 种取值情况(变化步长为 0.1).

本文所有实验均在一台配备一块 8 核 3.7 GHz 主频 CPU、一块 11 GB 显存 NVIDIA GeForce RTX 2080Ti 显卡及 64 GB 2666 MHz 内存的 Windows 10 服务器上完成.所有代码运行环境为 Python 3.6.k 近邻和支持向量机分类器基于 Sklearn 库实现.所有神经网络分类器均基于 Tensorflow 库和 Keras 框架进行实现及利用显卡进行训练加速.

4.2 总体实验结果和对比分析

如表 3 所示,为随机分类器(列“随机分类”)、FEATUREFINDER 方法(列“FEATUREFINDER”)及本文所提方法(列“RaT 框架解决方案”)在表 1 数据集上的移动应用功能分类准确率对比结果.表 3 的第 2 列列举了随机分类的准确率作为效果对照,用于对比说明 RaT 框架解决方案的有效性.随机分类的准确率即为相关应用类别功能总数的倒数.由表 3 可见,RaT 框架解决方案的类间平均分类准确率达到了 0.732.其分类效果不仅显著超过随机分类 0.207 的准确率,而且也明显优于采用 k-NN 分类器(0.486)或 SVM 分类器(0.361)的 FEATUREFINDER 方法.此外,由表 3“RaT 框架解决方案”列可见,分类效果最好的 RaT 框架解决方案均为采用基于语义特征的行为表征(SR).由此说明,本文所提出基于语义特征的行为表征对于提升移动应用功能分类任务性能是有效的.

表 3 移动应用功能分类不同解决方案实验结果

应用类别	随机分类	FEATUREFINDER		RaT 框架解决方案
		k-NN	SVM	
Calendar	0.333	0.667(k=4)	0.389(C=0.8)	**0.889(SR+MLP)**
Memo	0.250	0.479(k=3)	0.438(C=0.5)	**0.854(SR+FCN)**
Photography	0.167	0.475(k=8)	0.405(C=0.7)	**0.674(SR+FCN)**
Mixed	0.077	0.322(k=7)	0.213(C=0.8)	**0.511(SR+ResNet)**
类间平均准确率	0.207	0.486	0.361	**0.732**
标准差	0.110	0.141	**0.101**	0.175

表 4 列举了 RaT 框架指导下设计实现的 8 种解决方案在 4 类安卓移动应用上进行功能分类所达到的类内最高分类准确率和类间平均分类准确率. 其中“Mixed”类涵盖了来自 3 个其他应用类别涉及 13 种不同功能的 17 个安卓移动应用的总计 876 条执行轨迹. 因此，该应用类别相对于其他 3 个应用类别对于移动应用功能分类任务而言更加具有挑战性. 表 4 的第 2 至第 5 列详细列举了 4 种神经网络功能分类器(即 FCN、LSTM、MLP、ResNet)使用两种不同行为表征(基于统计特征的行为表征 SCR 和基于语义特征的行为表征 SR)在 4 种应用类别上的类内最高分类准确率. 由表 4 详细结果可见，采用语义特征行为表征的 RaT 解决方案在全部 4 个应用类别上的功能分类效果都显著优于采用时序特征行为表征的 RaT 解决方案. 表 4 的最后一列则展示了在每个应用类别上的最佳解决方案组合(即行为表征和功能分类器). 从类间平均分类准确率看，SR+ResNet 的解决方案相对其他 7 种解决方案在本实验数据集上具有更好的分类效果，并且采用 SR 行为表征的解决方案显著优于采用 SCR 行为表征的解决方案. 此外，表 4 中“类间平均准确率”和“标准差”部分展示了不同解决方案在不同应用类别上的平均功能分类准确率及其标准差. 从表 4 的“标准差”行中可以看出，虽然采用统计特征(SCR)行为表征的 RaT 解决方案相较于使用语义特征(SR)行为表征的 RaT 解决方案而言其类内平均准确率标准差更小，但其类间平均准确率显著较低. 因此，我们认为语义特征(SR)行为表征相比于统计特征(SCR)行为表征对于提升移动应用功能分类准确率的帮助更大.

表 4　RaT 框架解决方案实验结果

应用类别	FCN		LSTM		MLP		ResNet		最佳方案
	SCR	SR	SCR	SR	SCR	SR	SCR	SR	
Calendar	0.556	**0.667**	**0.833**	0.778	0.444	**0.889**	0.611	**0.778**	SR+MLP
Memo	0.542	**0.854**	0.562	**0.604**	0.500	**0.625**	0.562	**0.771**	SR+FCN
Photography	0.609	**0.674**	0.550	**0.646**	0.413	**0.630**	0.543	**0.609**	SR+FCN
Mixed	0.322	**0.467**	0.309	**0.372**	0.311	**0.478**	0.300	**0.511**	SR+ResNet
类间平均准确率	0.507	**0.666**	0.564	**0.600**	0.417	**0.656**	0.504	**0.667**	SR+ResNet
标准差	**0.127**	0.158	0.214	**0.169**	**0.079**	0.171	0.139	**0.130**	SCR+MLP

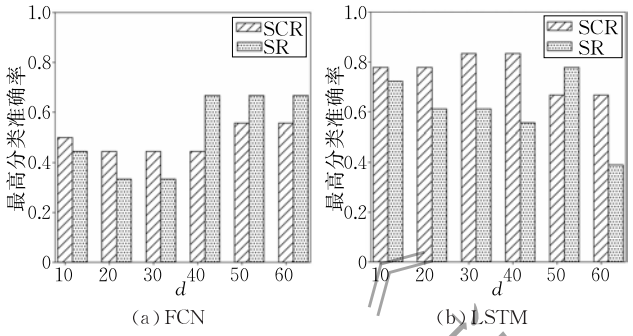
通过对表 4 第 2 至第 5 列的 SCR 和 SR 子列进行比较，可以明显看出基于语义特征的行为表征 SR 相比于基于统计特征的行为表征 SCR 能够更好地提升移动应用功能分类的效果. 我们推测主要有以下两个方面的原因：一方面，面对相同或类似的功能开发需求，不同移动应用的软件开发人员在编码风格或第三方库使用上具有较大的偏好差异. 因而执行相同或类似功能的不同移动应用所产生的执行轨迹在统计特征上也呈现出显著差别，这使得基于统计特征的行为表征(SCR)在移动应用功能分类任务中表现不佳. 另一方面，正如 3.3.2 节中的假设所言，基于语义特征的行为表征(SR)能够提升移动应用功能分类效果的另外一个原因可能在于：为了使得移动应用的源代码更易于理解或重用，软件开发人员会倾向于将函数或变量的名称与待实现的移动应用功能在语义上进行双向关联.

4.3　详细实验结果和影响因素分析

如表 2 所示，为保证对比两种表征方法(即 SCR 和 SR)的公平性，本文将两种表征方法输出表征向量的长度设定为固定值 3000. 其中影响执行轨迹表征向量的主要参数为单个变量或关键词的表征向量维度 d. 因此，本节就该参数对于移动应用功能分类效果的定量影响进行了量化分析.

如图 4 至图 7 所示，本实验对比了两种执行轨迹行为表征方法(即 SCR 和 SR)取不同 d 值时在 4 种应用类别上的分类准确率. 如图 4 所示，在应用类别 Calendar 上，当 $d=40$ 时 SR+MLP 解决方案的平均分类准确率(0.889)最高，达到了随机分类(0.333)的两倍以上，同时也高于所有采用 SCR 行为表征的解决方案最高分类准确率(0.833). 如图 5 所示，相对于行为表征 SCR 而言，行为表征 SR 在 4 个应用类别上均不同程度地提高了所有功能分类器的最高分类准确率，其中 SR+FCN 解决方案在应用类别 Memo 上的分类准确率达到 0.854. 如图 6 所示，对于 Photography 应用类别，大多数情况下 SR 表现都优于 SCR，且最高分类准确率(0.674)是在当 $d=10$ 时 SR+FCN 达到的. 最后，如图 7 所示，在 Mixed 应用类别上，行为表征 SR 使得所有功能分类器的准确率(0.372 至 0.511)达到了随机功能分类器(0.077)的五倍以上，而行为表征 SCR 的

分类效果相对次之(0.3 至 0.322).但同时也应注意到,即使是采用 SR 行为表征的解决方案最高也仅能达到 0.511 的分类准确率.由此可见,相比于其他 3 种应用类别,对包含具有较大差异性移动应用的应用类别进行功能分类的效果仍不够理想.因此,上述实验结果也进一步说明执行轨迹驱动的移动应用功能分类仍是一项复杂且具有挑战性的任务.



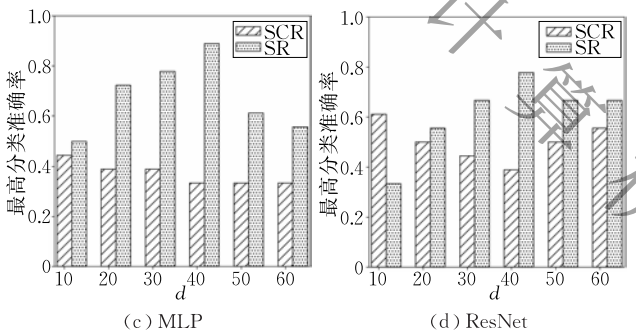
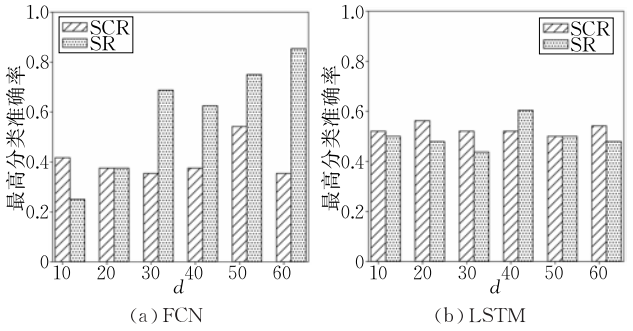


图 4 Calendar 应用类别功能分类效果对比



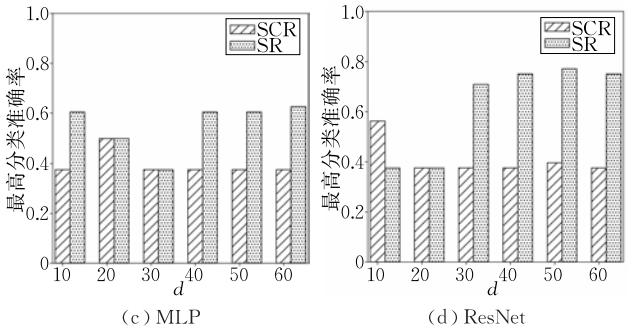
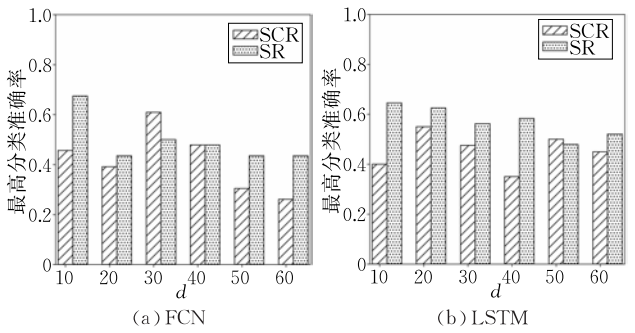


图 5 Memo 应用类别功能分类效果对比



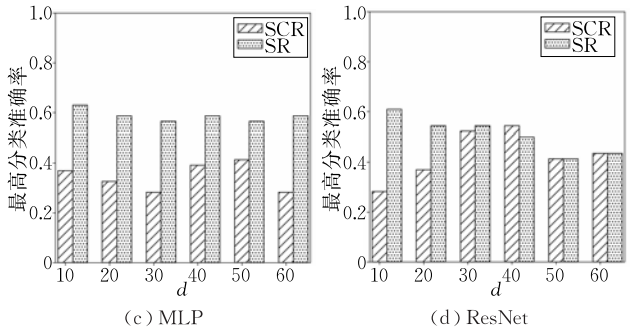
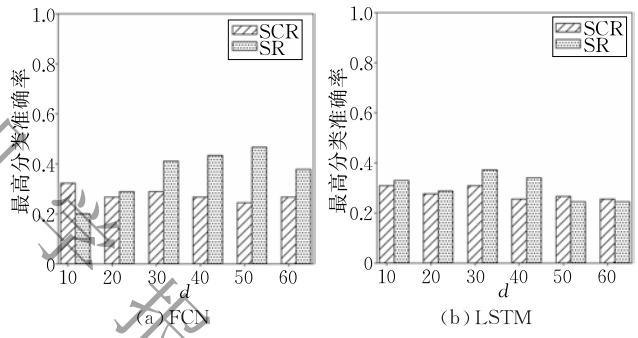


图 6 Photography 应用类别功能分类效果对比



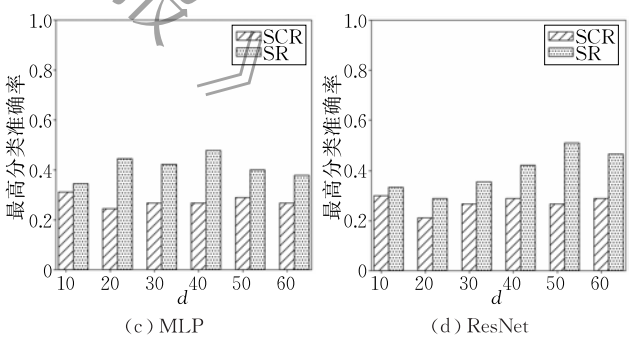


图 7 Mixed 应用类别功能分类效果对比

如算法 3 所述,前 k 个关键词是从执行轨迹中按照预定义策略选择出来的.这些关键词与执行轨迹对应功能之间的语义关联性对于最终生成的行为表征质量具有重要影响.因此,我们在图 8 中将提取自两个具有相同功能(即“添加备忘录”)但不同移动应用 com.abhi.newmemo(图 8 第 1 行)和 com.honny.memo(图 8 第 2 行)执行轨迹的 k 个关键词组织为词云图,以便定性分析该影响因素.由图

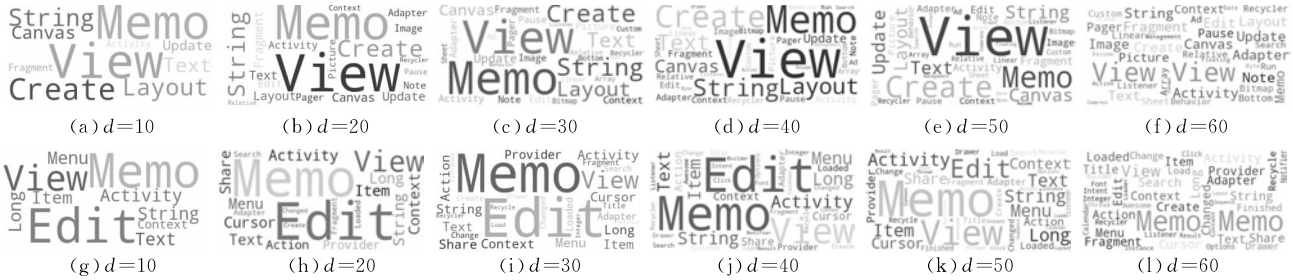


图 8　Memo 应用类别中 com.abhi.newmemo 和 com.honny.memo 移动应用的关键词选取结果

8 可以看出，提取出的关键字不仅有与移动应用功能相关的词素如 Memo、Create、Edit 等，还有一些与移动应用功能在语义上没有明显关联的通用词素如 View、Activity 等. 当 k 取值较小时，与移动应用功能在语义上关联性较高的关键词可能会被漏选；而当 k 取值较大时，则可能将与移动应用功能在语义上关联性不明显的词素提取为关键词，并由此对后续功能分类造成负面影响. 因此，在设计基于语义特征的执行轨迹行为表征生成策略时，如何选择合适的 k 值以尽可能多地提取出与移动应用功能相关的关键词同时尽量过滤掉相对泛化的词素是一个值得仔细思考的问题. 总而言之，关键词选择策略是影响生成基于语义特征的执行轨迹行为表征质量的重要因素之一.

图 9 展示了在 Memo 应用类别上 RaT 框架 8 种移动应用功能分类解决方案的详细训练过程. 图 9(a)、(b)、(c)和(d)是使用基于统计特征的行为表征 SCR 时 4 种功能分类器的训练过程. 从中可以清楚看到，随着分类器训练迭代次数($Epochs$)的增加，4 种分类器在训练集上的损失函数值逐渐收敛至 0，但其在测试集上的损失函数值均在 5 至 10 区间内水平波动. 该现象表明训练集和测试集中执行轨迹的表征在数据分布上差异较大，因而使得采用 SCR 表征的功能分类效果不理想. 图 9(e)、(f)、(g)和(h)是使用基于语义特征的行为表征 SR 时 4 种功能分类器的训练过程. 不难看出随着分类器训练迭代次数($Epochs$)的增加，4 种分类器在测试集上损失函数值的变化趋势呈现出较为明显的差异性. 在经过大约 50 轮训练迭代后，FCN(如图 9(e)所示)和 ResNet(如图 9(h)所示)在测试集上损失函数均值大约降到 1 左右. 而 LSTM(如图 9(f)所示)和 MLP(如图 9(g)所示)在测试集上损失函数则攀升至 7 左右的水平. 以上现象也解释了为何表 4 中 SR+FCN 和 SR+ResNet 这两个解决方案能够达

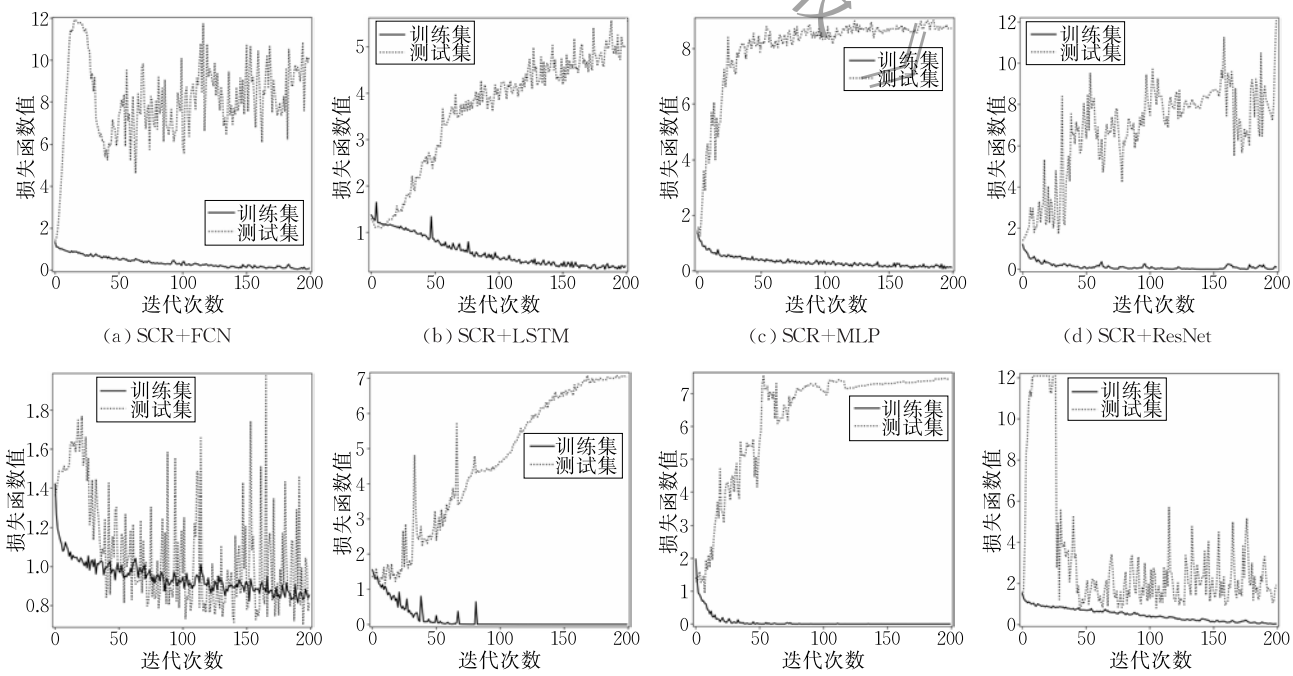


图 9　功能分类器在 Memo 程序类别上的训练过程对比

到更高的分类准确率.此外,该现象还表明,即使基于语义特征的行为表征 SR 能够较好地捕获训练集和测试集中执行轨迹内与移动应用功能相关联的可辨识特征,但是也需要选取合适的神经网络分类器才能达到较好的分类效果.综上所述,依据本文实验结果,基于语义特征行为表征 SR 和神经网络分类器 FCN、ResNet 的移动应用功能分类解决方案能达到较好的效果.

5 相关工作

程序功能分类是指通过对目标程序进行深入理解进而从中识别目标程序主要功能的任务.本节将简要介绍该领域的现有工作以及与其相关的若干研究主题.首先,本文的目标问题与特性定位(Feature Location)问题具有相关性.目前,特性定位是一个较为成熟的研究领域,其任务是识别源代码中与程序特性相对应的代码片段.根据输入数据的不同,现有的方法大致可以分为动态分析、静态分析、文本分析、软件仓库历史信息分析等[17].Wilde 等人[18-19]提出了一种完全动态的特性定位方法,这种方法将程序特性建模为计算单元,通过识别出特定的计算单元以实现对程序特性的定位.该方法的主要缺点在于其假设程序特性和应用场景一一对应.Eisenbarth 等人[20]综合利用动态分析和静态分析方法,提出一种半自动技术,利用给定的一组程序特性来识别和区分一般的或特定的计算单元.Koschke 等人[21]提出了另一种综合性程序特性定位方法,通过细粒度的代码块分析来提升特性定位效果.但是该方法主要缺陷在于当程序特性较多时会导致组合状态空间超线性增长,从而使得计算开销难以承受.测试场景的设计是进行动态分析的关键,Liu 等人[22]提出了一种仅需一条单一场景执行轨迹的程序特性定位方法,但该方法的局限性在于需要源代码支持以从中获取有助于特性定位的辅助信息.尽管程序特性定位与本文研究的程序功能分类问题有一些相似之处,但两者仍有较为明显的区别.首先,程序特性定位与程序功能分类的输入数据不同,前者输入数据必须包含目标程序源代码,而后者输入数据必须包含目标程序执行轨迹.其次,程序特性定位与程序功能分类的目标任务不同,前者旨在搜索与给定程序特性集合相对应的源代码片段,而后者聚焦于如何通过分析目标程序的执行轨迹以准确识别程序运行过程中所执行的特定功能.

本文相关技术还涉及 Mcmillan 等人[23]中提出的通过相关特性搜索目标程序的方法.该方法的关键是对目标程序进行建模,以便于从信息源(如执行轨迹)中有效地提取信息.因此,这种方法又被称为执行轨迹分析及抽象[24].Feng 等人[25]提出了一种层次化的执行轨迹抽象方法,其将具有相似特征的计算单元进行聚类,然后通过识别其中的频繁模式以区分不同粒度的计算单元.然而,类似信息检索中仅靠高频词无法保证检索到真实目标文档的情况,若不同类型计算单元包含某些普遍存在的频繁模式,则该方法存在失效的风险.Alimadadi 等人[26]提出了一种利用生物信息启发的通用方法 SABALAN 从安卓应用执行轨迹中寻找层次化的循环局部模式以辅助程序理解任务.具体而言,SABALAN 实现了一种启发式的局部序列比对算法用于搜索具有类似局部代码执行序列的安卓应用执行轨迹.SABALAN 方法主要存在以下两个方面的不足:一方面,该方法需要领域知识以设计启发式规则.另一方面,由于需要将查询模式与所有执行轨迹序列进行全局匹配,因此当执行轨迹数量较多序列较长时可能导致总体计算开销较大.

Xin 等人[16]提出了一种旨在识别安卓应用执行轨迹对应的应用特性的方法 FEATUREFINDER,该方法与本文目标问题紧密相关.首先,FEATUREFINDER 依据用户事件对 5 个安卓应用的原始执行轨迹进行分割,并利用其中 4 个/1 个安卓应用执行轨迹片段分别构建训练集和测试集;其后,FEATUREFINDER 从训练集的执行轨迹片段中提取了 29 个统计特征并尝试利用 10 种机器学习模型对其按照应用特性进行分类;最后,FEATUREFINDER 利用分类效果最好的机器学习模型对测试集中的执行轨迹片段进行聚类操作,然后从聚类的簇中提取关键词并与安卓应用特性标签进行人工比对,从而确认 FEATUREFINDER 方法的有效性.FEATUREFINDER 的主要限制在于两个方面:统计特征的提取需要较多的领域知识;机器学习模型对于统计特征与特性标签之间的复杂映射关系建模能力有限.此外,该工作既没有清晰地定义移动应用功能分类问题,也没有提出可用于指导移动应用功能分类解决方案设计的一般性框架.因此,现有工作存在的诸多不足之处是本文工作的主要动机,即一方面对于移动应用功能分类问题进行形式化定义,另一方面提出一种执行轨迹驱动的移动应用功能分类解决方案框架(即 RaT).

6　总结与展望

针对移动应用功能分类问题，本文首先对其进行了形式化的定义，然后提出了一个包含 4 层结构的通用框架 RaT(Run-and-Tell). 具体而言，本文设计了基于统计特征(即 SCR)和语义特征(即 SR)的执行轨迹行为表征生成方法，并将这 2 类表征与 4 种基于神经网络(即 FCN、LSTM、MLP 及 ResNet)的功能分类器相结合，提出了 8 个移动应用功能分类解决方案. 为了构建实验数据集，针对来自 Google Play 应用商店 3 个应用类别涵盖 13 种不同功能的 17 个安卓应用程序，本文总共采集了 876 条执行轨迹. 本文实验结果表明，RaT 框架解决方案的类间平均分类准确率达到了 73.2%. 其分类效果不仅显著超过随机分类，而且也明显优于现有 FEATUREFINDER 方法. 此外，本文通过对比实验对影响移动应用功能分类解决方案效果的若干要素(如行为表征、神经网络分类器)进行了较为深入的分析及探讨.

尽管就本文实验结果看，RaT 框架指导下的解决方案在解决移动应用功能分类问题方面具有一定的潜力，但其仍面临诸多挑战. 首先，在方法调用或变量名称的语义缺失或被故意隐藏的情况下，本文解决方案可能无法达到预期效果甚至完全失效. 譬如，软件开发过程中为了保护代码不被进行逆向工程而应用代码混淆技术[27]，即利用无意义的符号替换源代码中方法或变量的实际名称，这将给移动应用功能分类解决方案带来新的挑战. 其次，当由于多语言、生僻词、缩写或拼写错误等原因造成执行轨迹中方法调用关键词不在预定义字典中时(Out of Vocabulary，简记为 OoV)[28]，如何改进现有行为表征生成方法以适应该情况将变得更具挑战性. 再次，本文所提出的移动应用功能分类解决方案均是基于神经网络分类器构建，因而其训练开销较大，如何实现快速甚至实时功能分类是未来潜在的研究方向之一. 最后，当面对差异性较大的多应用类别(如 Mixed)时，现有解决方案的效果均不尽如人意. 面对上述挑战，我们一方面需要在未来研究中尝试进一步深入理解执行轨迹与移动应用功能之间的复杂关系，另一方面可以考虑将多种不同类型表征进行有机融合以生成更具有功能辨识力的执行轨迹行为表征，从而进一步提高移动应用功能分类准确率.

参　考　文　献

[1] Rubin J, Chechik M. A survey of feature location techniques //Reinhartz-Berger I, Sturm A, Clark T, et al, eds. Domain Engineering. Berlin, Germany: Springer, 2013: 29-58

[2] Zhang S, Cai H Q, Ma Y, et al. SmartPipe: Towards interoperability of industrial applications via computational reflection. Journal of Computer Science and Technology, 2020, 35(1): 161-178

[3] Kästner C, Dreiling A, Ostermann K. Variability mining: Consistent semi-automatic detection of product-line features. IEEE Transactions on Software Engineering, 2013, 40(1): 67-82

[4] Huang G, Luo C, Wu K, et al. Software-defined infrastructure for decentralized data lifecycle governance: Principled design and open challenges//Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). Dallas, USA, 2019: 1674-1683

[5] Dabrowska, Dorota M. Kaplan-Meier estimate on the plane. Annals of Statistics, 1988, 16(4): 1475-1489

[6] Hammerla N Y, Kirkham R, Andras P, et al. On preserving statistical characteristics of accelerometry data using their empirical cumulative distribution//Proceedings of the 2013 International Symposium on Wearable Computers. New York, USA, 2013: 65-68

[7] Ma C, Shi X, Zhu W, et al. An approach to time series classification using binary distribution tree//Proceedings of the 2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN). Shenzhen, China, 2019: 399-404

[8] Ma C, Shi X, Li W, et al. Edge4TSC: Binary distribution tree-enabled time series classification in edge environment. Sensors, 2020, 20(7): 1-18

[9] Jessica L, Eamonn K, Li W, et al. Experiencing SAX: A novel symbolic representation of time series. Data Mining and Knowledge Discovery, 2007, 15: 107-144

[10] Shieh J, Keogh E. iSAX: indexing and mining terabyte sized time series//Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). New York, USA, 2008: 623-631

[11] Verbeek J J, Vlassis N, Kröse B. Efficient greedy learning of Gaussian mixture models. Neural Computation, 2003, 15(2): 469-485

[12] Salton G, Buckley C. Term-weighting approaches in automatic text retrieval. Information Processing & Management, 1988, 24(5): 513-523

[13] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality//Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2. Red Hook, USA, 2013: 3111-3119

[14] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift// Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37. Lille, France, 2015: 448-456

[15] Lin M, Chen Q, Yan S. Network in network//Proceedings of the 2nd International Conference on Learning Representations (ICLR). Banff, Canada, 2014: 1-10

[16] Xin Q, Behrang F, Fazzini M, et al. Identifying features of android apps from execution traces//Proceedings of the 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILE-Soft). Montreal, Canada, 2019: 35-39

[17] Dit B, Revelle M, Gethers M, et al. Feature location in source code: A taxonomy and survey. Journal of Software: Evolution and Process, 2013, 25(1): 53-95

[18] Wilde N, Gomez J A, Gust T, et al. Locating user functionality in old code//Proceedings of the Conference on Software Maintenance 1992. Orlando, USA, 1992: 200-205

[19] Wilde N, Scully M C. Software reconnaissance: Mapping program features to code. Journal of Software Maintenance: Research and Practice, 1995, 7(1): 49-62

[20] Eisenbarth T, Koschke R, Simon D. Locating features in source code. IEEE Transactions on Software Engineering, 2003, 29(3): 210-224

[21] Koschke R, Quante J. On dynamic feature location// Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE). Long Beach, USA, 2005: 86-95

[22] Liu D, Marcus A, Poshyvanyk D, et al. Feature location via information retrieval based filtering of a single scenario execution trace//Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE). Atlanta, USA, 2007: 234-243

[23] Mcmillan C, Poshyvanyk D, Grechanik M, et al. Portfolio: Searching for relevant functions and their usages in millions of lines of code. ACM Transactions on Software Engineering and Methodology (TOSEM), 2013, 22(4): 1-30

[24] Cornelissen B, Zaidman A, van Deursen A, et al. A systematic survey of program comprehension through dynamic analysis. IEEE Transactions on Software Engineering, 2009, 35(5): 684-702

[25] Feng Y, Dreef K, Jones J A, et al. Hierarchical abstraction of execution traces for program comprehension//Proceedings of the 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC). Gothenburg, Sweden, 2018: 86-96

[26] Alimadadi S, Mesbah A, Pattabiraman K. Inferring hierarchical motifs from execution traces//Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). Gothenburg, Sweden, 2018: 776-787

[27] Collberg C S, Thomborson C. Watermarking, tamper-proofing, and obfuscation-tools for software protection. IEEE Transactions on Software Engineering, 2002, 28(8): 735-746

[28] Zhang R, Mensah S, Kong F, et al. Pairwise link prediction model for out of vocabulary knowledge base entities. ACM Transactions on Information Systems, 2020, 38(4): 1-28

MA Chao, Ph. D., lecturer. His research interests include program comprehension, deep learning, and time series analytics.

LI Chun-Tung, Ph. D. His research interests include computer-human interaction, ubiquitous computing, and time series mining.

CAO Jian-Nong, Ph. D., chair professor. His research interests include distributed computing, wireless network, mobile computing, big data, machine learning, cloud computing and edge computing.

CAI Hua-Qian, Ph. D., associate research fellow. His research interests include mobile application and distributed systems.

WU Li-Bing, Ph. D., professor. His research interests include Internet of Things, network security, wireless network, and machine learning.

SHI Xiao-Chuan, Ph. D., associate professor. His research interests include big data analytics, reinforcement learning, wireless sensor network and data mining.

Background

Program functionality classification refers to the task of categorizing the functionalities of a program that is implemented to meet a specific purpose. Understanding the functionality provided by a program is one of the six fundamental program comprehension processes that facilitate many software engineering tasks including software maintenance, software evolution, and product line engineering. This task is challenging and time-consuming especially in large scale

systems, where the automation of such a task is desired.

Static and dynamic analysis is the most commonly used techniques by far for program comprehension. The former analyzes the structural program dependencies that usually require access to the source code, which is not practical in many real-world applications such as legacy system re-engineering, malware detection etc. In contrast, the dynamic analysis leverages the execution traces captured during the running of a program, which is more reliable and practical. However, due to the dynamic runtime environment and various development frameworks, the execution trace is complicated that has huge differences in size and great variations in pattern. Therefore, automatically categorizing the program functionalities by analyzing its execution traces is a non-trivial task.

In this paper, we have made the following contributions. First, we formulate the program functionality classification problem and model the execution trace in a system-independent manner, which makes our approach being able to be easily generalized to other kinds of programs when trace-driven functionality classification is needed. Second, a systematic framework named RaT (Run-and-Tell) is proposed to guide the solution design of trace-driven program functionality classification. Third, comprehensive experiments are conducted on the 876 execution traces of 17 Android apps collected from Google Play while the analysis is offered to not only validate the effectiveness of the proposed solutions but also explore the impact factors of our proposed solutions to trace-driven programs functionality classification.

This work is partially supported by the Key R&D Program of Hubei Province (No. 2021BAA039) and the Key-Area Research and Development Program of Guangdong Province (No. 2020B010164002).