软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

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

一种基于深度神经网络的 Android 二进制代码缺陷预测方法*

董 枫[1,2], 张少东[1,2], 李承泽[1,2], 李 祺[1,2], 徐国爱[1,2]

[1](北京邮电大学 网络空间安全学院,北京 100876)

[2](移动互联网安全技术国家工程实验室(深圳),广东 深圳 518000)

通讯作者: 董枫, E-mail: dongfeng@bupt.edu.cn

摘 要: 随着移动互联网的广泛应用，由于移动应用软件缺陷产生的安全问题越来越普遍，移动应用软件特别是 Android 平台软件的缺陷预测研究引起广泛关注。当前，大量基于机器学习的软件缺陷预测模型方法被提出，并且达到较好的预测性能。然而，大多数基于机器学习算法的软件缺陷预测方法面向源代码，不能直接用于处理 Android 二进制缺陷预测，而实际 Android 软件缺陷预测任务中源代码难以获取。此外，现有的机器学习预测模型大多采用传统的浅层结构机器学习算法，如支持向量机、朴素贝叶斯等。由于浅层结构机器学习算法表达特征和缺陷之间复杂关系的能力有限，其在缺陷预测任务的性能达到一定的瓶颈。本文提出一种针对 Android 二进制可执行文件的缺陷预测模型，同时采用具有深层结构的深度神经网络算法进行分类预测，提高了预测性能。首先，我们构建了一个用于 Android 二进制文件缺陷预测的数据集。其次，我们提出了一种创新的 Android 可执行文件缺陷特征提取方法，该方法提取 smali 文件（Andorid 二进制文件的反汇编文件）的符号特征和语义特征来构建缺陷特征向量。最后，我们将缺陷特征向量输入深度神经网络算法来训练和构建缺陷预测模型。我们研发出模型工具原型 DefectDroid，将其应用于大规模 Android smali 文件缺陷预测任务中，并从同项目缺陷预测、跨项目缺陷预测、传统机器学习算法等方面对模型进行性能评估对比。

关键词: 软件缺陷预测;软件安全;Android 二进制文件;机器学习;深度神经网络

中图法分类号: TP311

中文引用格式: 董枫,张少东,李承泽,李祺,徐国爱. 一种基于深度神经网络的 Android 二进制代码缺陷预测方法.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Dong F, Zhang SD, Li CZ, Li Q, Xu GA. A Method of Defect Prediction for Android Binary Files Based on Deep Neural Network. Ruan Jian Xue Bao/Journal of Software, 2017 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

A Method of Defect Prediction for Android Binary Files Based on Deep Neural Network

DONG Feng[1,2], ZHANG Shao-Dong[1,2], LI Cheng-Ze[1,2], LI Qi[1,2], XU Guo-Ai[1,2]

[1](School of CyberSpace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China)

[2](NEL of Security Technology for Mobile Internet (Shenzhen), Shenzhen 518000, China)

Abstract: With the widespread application of mobile internet applications, Android application defect prediction has been paid more and more attention due to the mobile security issues caused by the defect. A large number of machine learning-based software prediction models are proposed and achieve good performance. However, most of them are source-oriented and can not be easily used for Android binary files (apks) defect prediction. Moreover, the traditional machine learning techniques used in these models, such as Support Vector Machine (SVM) and Naive Bayes (NB), have a shallow architecture, which leads to a limited capacity of expressing complex functions between features and defects. In this paper, we propose a practical defect prediction model for Android binary files using deep neural network (DNN). We first

* 基金项目: 国家自然科学基金(61401038); 2016 广东省科学技术厅前沿与关键技术创新项目(2016B010110002)

Foundation item: National Natural Science Foundation of China (61401038); 2016 Frontier and Key Technology Innovation Project of Guangdong Province Science and Technology Department (2016B010110002)

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

build a dataset for apks defect prediction and make it open source on the website. Then we propose a novel approach to generate features that capture both token and semantic features of the defective smali (decompiled files of apks) files in apks. At last, we input the feature vectors into DNN to train and build the defect prediction model in order to achieve accuracy. We implement the model called DefectDroid and apply it to a large number of Android smali files. We compare the prefromance of DefectDroid with within-project defect prediction (WPDP), cross-project defect prediction (CPDP), traditional machine learning algorithms, etc

Key words: software defect prediction, software security, Android binary files, machine learning, deep neural network

软件缺陷(software defect)是开发人员在软件开发过程中由于经验不足、编码不规范等原因引入的缺陷或错误。软件缺陷包括很多类型，例如缓冲区溢出(memory overflow)、运行异常(runtime exception)等[1]，其产生的安全问题可能造成严重的后果，尤其在一些如金融、电力等低容错率行业。软件缺陷预测通过对软件提取缺陷特征，构建缺陷模型来预测和定位软件中可能存在的缺陷，对于解决软件缺陷带来的安全问题具有重要意义。随着移动互联网时代的到来，越来越多的软件从 PC 端迁移到移动端，开发者更容易以忽略软件缺陷为代价来满足移动端软件开发周期短、更新频率高等特点。所以移动端应用软件缺陷相较 PC 端更普遍，产生的危害也更大。针对移动应用软件，特别是 Android 平台应用软件的缺陷预测研究成为当前热点。

，静态缺陷检测方法通过多年的发展，已经形成了两种主要方法：静态代码缺陷扫描(Static Bug-Finding)和缺陷预测(Defect Prediction) [37]，两种方法相对独立发展。

其中，静态代码缺陷扫描方法[31,32,33]主要通过简单的字符串分析或复杂的程序语义分析，将缺陷代码形成缺陷规则，通过启发式规则匹配来检测出潜在的缺陷代码。如基于 FlowDroid[31]等静态分析匹配的缺陷分析方法，通过反编译 APK 得到中间语言，将缺陷形成启发式规则，将规则与中间语言进行匹配，从而发现 apk 中的缺陷。启发式规则方法在源代码审计方面，已经有非常成熟的商业化工具，如 Checkmarx CxSuite[27]，Fortify SCA[35]. 面向二进制的启发式规则匹配缺陷分析方法研究很多，如 FlowDroid，暂时没形成商业化工具。

第二种缺陷预测方法即为本文讨论的缺陷预测模型方法，目前业界主流研究都采用机器学习算法来构建缺陷预测模型。基于机器学习的模型预测法则通过从标记好的软件模块(software module)中提取或生成能表征缺陷的特征，如字符串特征、抽象语法树、控制流图、程序依赖图等的特征，输入到机器学习算法中，通过算法训练，构建缺陷特征和缺陷类别的映射关系模型，来预测未标记的软件模块的缺陷类别[2,6]。

启发式规则匹配法主要依赖缺陷特征以及形成的缺陷规则库来进行匹配来发现缺陷[33]，基于机器学习的模型预测法则通过对缺陷特征进行生成提取，机器学习算法来学习缺陷特征和缺陷类别映射关系知识，来智能化的判断软件中的缺陷[6]。启发式规则匹配法已经在很多文章中研究，并且与基于机器学习的模型法在实现方法和实验上区别较大，故不在本文的讨论范围内，下文讨论的缺陷分析方法默认为基于机器学习的模型法。

当前基于机器学习的软件缺陷预测研究主要集中在两点：

(1) 缺陷特征提取，提取能代表缺陷的特征，作为算法输入构建预测模型。目前常见的特征包括基于软件度量(Software Metrics)的特征[8]，基于代码文本的符号特征(token feature) [1]和基于代码结构如抽象语法树(Abstract Syntax Tree) [3]、程序依赖图(Program Dependence Graph) [7]、控制流图(Control Flow Graph) [10]等的语义特征(semantic feature)。

(2) 机器学习算法的选择，选择合适的算法来提高模型准确率。常见的机器学习算法主要包括支持向量机、朴素贝叶斯、决策树、神经网络等[1,3,4,5,6,7]。

然而，根据作者调研，现有的缺陷预测模型大多数以源代码为输入预测缺陷，无法直接处理 Android 二进制文件的缺陷预测问题。实际软件开发周期过程中，软件开发与缺陷审计往往由两个不同的团队完成。负责缺陷审计安全评估的第三方安全公司或者研究者由于版权或者源代码保护等原因，得到的往往是二进制文件而非源代码。

针对上述问题，本文设计并实现了一套面向 Android 二进制文件(也称为 apk 文件)的缺陷预测模型。首先，我们提出了一种创新的面向 smali 文件的缺陷特征提取方法。该方法从 smali 文件中提取符号特征和语义特征来共同构建缺陷特征。在提取符号特征与语义特征过程中，我们采用常用的特征子集选择方法(feature

subset selection)中的信息增益法(information gain)[12]从全部的 dalvik 指令集(dalvik opcodes)[13]中选择与缺陷相关度高的关键指令集(critical opcodes)，从而降低特征维度，防止维度灾难。同时，我们在语义特征提取的过程中，根据关键指令集来序列化 smali 文件，从而得到语义信息。最后，我们采用深度神经网络(deep neural network, 简称 DNN)作为分类器来训练缺陷预测模型。我们实现了模型系统 DefectDroid，并将其应用于大规模的数据集（50 个 Android 应用软件，92774 个 smali 文件）中，并且采用 ROC-AUC(the receiver operating characteristic and area under the curve)评价体系[14]对分类器进行性能评估。实验结果表明，DefectDroid 在同项目缺陷预测(within-project defect prediction,简称 WPDP)中准确率达 83.08%，跨项目缺陷预测(cross-project defect prediction,简称 CPDP)中准确率达 66.36%，对 Android 二进制文件具有缺陷预测能力。

总结来说，本文在 Android 应用软件缺陷预测研究领域的贡献主要包括以下三点：

(1) 据目前所知，本文首次提出针对 Android 二进制文件的缺陷预测模型。

(2) 本文提出了一种针对 Android 二进制文件的缺陷特征提取方法。

(3) 我们采用深度神经网络替代传统的机器学习算法作为模型分类器，提高了缺陷预测的性能。

本文组织结构如下。第 1 节对本文涉及的相关工作进行介绍，主要包括缺陷预测模型，Android 二进制文件特征以及深度神经网络。第 2 节描述整个 DefectDroid 的框架以及细节设计，主要包括 smali 特征提取过程。第 3 节对实验前期准备工作进行描述。第 4 节展示实验结果以及对结果的分析。第 5 节总结全文，对模型中可能存在的局限和未来的方向进行初步探讨。

1 引言

1.1 软件缺陷预测

图 1 展示了主流的基于机器学习的软件缺陷预测方法的一般流程。首先，将软件分解为一个一个模块(module)，根据粒度不同，模块可以是粗粒度的包或者文件，也可以是细粒度的函数方法[30]。然后对模块进行缺陷特征提取和标记(label)工作，形成带类标记的特征向量。特征提取方法，根据不同类型度量元的选取，从不同角度提取特征。当前度量元主要分为基于软件代码(code)的度量元以及基于开发过程(process)的度量元[2]。基于代码的度量元主要从代码本身的符号、语义、复杂度等角度去提取特征，而基于开发过程度量元则考虑到开发者的经验、项目历史版本间关系、修改提交等角度的特征。最后将带类标记的特征向量输入到分类算法中，训练算法参数，训练完成后，输入新的未标记特征向量时，分类算法输出此模块的类别。当未标记特征向量来自同一个项目时，预测任务为同项目缺陷预测(within-project defect prediction)，来自不同项目时，则预测任务为跨项目缺陷预测(cross-project defect prediction)[2]。跨项目缺陷预测的意义在于，当需要进行缺陷预测项目已有的训练数据较少时，我们可以通过从不同的项目学习到的缺陷知识，迁移到需要缺陷预测项目来进行缺陷预测。跨项目缺陷预测吸引了大量研究者的关注[4]，如何构建有效的跨项目缺陷预测模型也称为研究热点。主流预测模型为二分类预测模型，即预测为有缺陷倾向性(defect-proneness,简称 DP)模块或无缺陷倾向性(non defect-proneness,简称 NDP)模块。

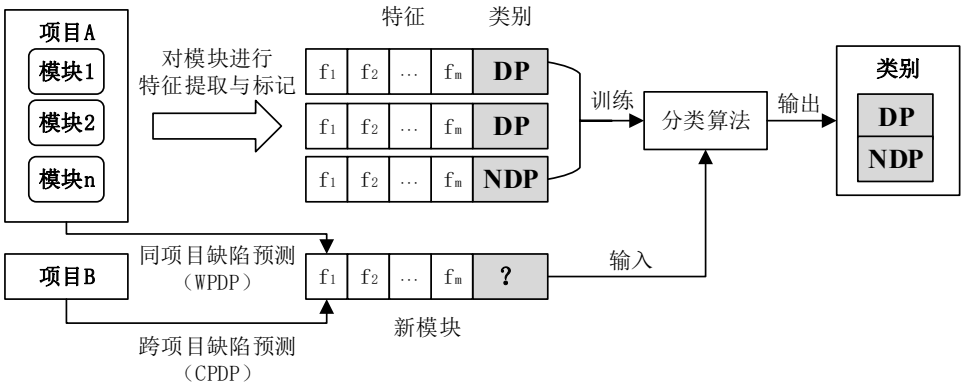

Fig.1 General process of software defect prediction

图 1 软件缺陷预测一般流程

在模块的选择上，大多数模型采用粗粒度的文件例如 java 文件作为预测目标。而 Perl 等人[11]则创新的采用版本管理工具(例如 git, Subversion 等)中的修改提交(commit)作为模块进行建模预测。其结合代码度量元和开发过程度量元从项目的提交中提取特征，并使用支持向量机来训练和预测新提交中的缺陷。以提交为模块的预测模型的优势在于，它每次只需要输入增量提交模块进行预测，而不需要将整个项目输入到模型中，极大的增加了缺陷预测的效率。然而，这种方法只能应用于使用版本管理软件开发的软件项目，不适用于二进制文件。Giger 等人[36]采用细粒度的方法（method-level）作为预测模块进行缺陷预测，取得了一定的效果，但是其计算的开销大的问题，难以应用到实际的软件开发过程中[2]。考虑到计算成本，本文采用常用的粗粒度文件作为预测模块。

在特征提取方面，分为基于代码的度量元和基于开发过程的度量元两大类。本文主要关注基于代码的度量元特征提取。Scandariato 等人[1]采用文本处理的方式，将 Android 源代码中的 java 文件类看作文本，采用词袋模型(bag-of-words)统计文件中相关的词的频数，用符号特征(token feature)生成缺陷特征向量。其分别采用随机森林和朴素贝叶斯算法作为分类器，将模型应用于 20 个 Android 应用软件项目，模型的平均准确率达 80%。Wang 等人[3]则创新的采用深度学习中的特征生成算法(representation-learning algorithm)，深度信念网络(deep belief network)来自动化提取缺陷特征，其通过提取源代码中的抽象语法树(AST)作为语义特征，输入到深度信念网络中，通过深度信念网络的多层迭代训练，输出全新的语义特征，输入到分类算法中，进行训练和预测。其实验结果表明，这种特征提取方法能有效的提高同项目缺陷预测和跨项目缺陷预测的性能。

在分类算法方面，Malhotra 等人[6]提出了一套成熟的 Android 应用软件缺陷预测框架，其采用主流的 18 种传统浅层机器学习算法进行实验，对比分析不同算法之间的分类性能，其实验结果表明，朴素贝叶斯(naïve bayes)、LogitBoost 和多层感知机(multilayer perceptron)有最好的分类性能。

总体来说，当前的软件缺陷预测模型研究在预测性能上已经达到较好的水平，但无法直接用于 Android 二进制文件的缺陷预测任务。因此，本文提出一种面向 Android 二进制文件的缺陷预测方法，该方法以二进制文件反汇编文件，即 smali 文件作为模块，采用基于软件代码的度量元进行特征提取，将代码的符号特征和语义特征相结合，生成缺陷特征向量，输入到以 DNN 作为分类器的预测模型中进行训练和预测，从而达到比传统浅层机器学习算法更好的预测性能。

1.2 Smali文件特征

Android 二进制文件由一个压缩包组成，其中包括 dex 可执行文件、签名、资源文件等组成，其中 dex 文件为源代码编译后生成的 dalvik 虚拟机可执行文件。通常一个 apk 文件中包含一个 dex 文件，通过反编译器可将其反编译为多个反汇编文件，即 smali 文件。每个 smali 文件都由 dalvik 指令集(dalvik opcode)[13]以 smali

语法构成。图 2 为本文的一个典型示例，图右边为源代码文件 main.java，图左边为编译器编译后生成的 main.smali 文件。由图可知，main.smali 文件由一些 dalvik 指令以一定的语法规则组合而成，由 dalvik 虚拟机解释执行。Smali 文件中的助记符“.line”标明了其对应源代码中的行数，例如 main.smali 中“.line 15”对应 main.java 中的第 15 行代码。当我们分析出 smali 文件中的缺陷时，可以准确的定位到源代码的缺陷代码位置。

Fig 2 A example of main.java and its corresponding compiled file main.smali

图 2 编译示例-main.java 及相应的 main.smali

和源代码一样，smali 代码有其对应的词法规则和语法规则，所以有其相应的符号特征和语义特征。在软件分析中，很多研究常从源代码中提取抽象语法树、程序依赖图、控制流图等一类包含语义的图或树状结构，用于语义特征的存储和提取。Smali 文件符号特征和语义特征的特征已经被应用于 Android 二进制文件代码克隆检测(code clone detection)和恶意软件检测(malware detection)中。例如，Kai Chen 等人[10]提出一种针对 Android 二进制文件的大规模快速代码克隆检测方法，通过提取 smali 文件中的每个方法控制流图来编码 smali 文件的语义特征，来检测代码克隆。Quentin Jerome 等人[15]则直接将 smali 文件中的 dalvik 指令顺序编码作为语义特征来检测 Android 恶意应用软件。不同于简单的指令顺序编码，Niall McLaughlin 等人[16]采用深度学习模型，将指令序列输入到卷积神经网络中，设计了针对 dalvik 指令的特征卷积层进行指令特征的自动化学习，最后编码后进行分类，来进行 Andorid 恶意应用软件检测。

本文同时提取 smali 文件的符号特征和语义特征，符号特征提取采用词袋模型，统计不同 dalvik 指令频数，以一定顺序排列构成符号特征向量。在语义特征提取方面，本文同样采用 dalvik 指令序列编码来生成能代表语义特征的向量。最后，将二者结合构成更加全面的缺陷特征向量，输入到分类器中进行模型构建和缺陷预测。详细见小节 2.2.

1.3 深度神经网络

深度网络[17,18]指一系列由多层节点堆叠构成的神经网络，每层节点的运作模式模拟人脑神经元，节点的输入为每个前置层相连节点的输出，每个输入与相应的连接权重(weight)加权，通过非线性的激活函数(non-linear activation function)得到输出，从而判断信号是否继续在网络中传递。深度网络在很多研究中也称为深度学习(deep learning)，由于其深层结构具备表达复杂函数的能力[18]，在很多人工智能领域作出了重大突破。越来越多的研究者开始将深度网络应用于软件分析领域，进行恶意软件检测和软件缺陷预测工作[16,19,20,21]。

深度网络根据不同的应用场景，有不同的架构(architecture)构建深度学习模型，如深度神经网络 DNN、卷积神经网络(convolutional neural network，简称 CNN)、循环神经网络(recursive neural network，简称 RNN)、深

度信念网络(deep belief network，简称 DBN)等等。根据其使用的意图，深度网络模型可以分为两大类[18]。

(1) 用于无监督学习(unsupervised learning)或特征生成学习(generative learning)的深度网络模型。从大量无类别标记样本中进行无监督学习，从而生成更能表现类别信息的特征，作为分类的输入。

(2) 用于监督学习(supervised learning)如分类(classification)或回归分析的深度网络模型。由大量标记样本进行监督学习，对未知类别样本进行分类预测。

自编码(autoencoder)和 DBN 常用于特征生成的无监督学习深度模型构建，而 DNN 则常用于监督学习深度模型构建，完成分类预测任务。CNN 和 RNN 等架构常用于构建混合的深度学习模型[18]，混合模型往往将无监督学习和监督学习包含在整个模型过程中，无监督学习过程生成特征，监督学习进行分类。由于 DNN 相较于传统的机器学习算法，有更多的中间隐藏层网络，能够表示更复杂的函数[34]，更好的拟合缺陷特征和缺陷类别的映射关系，从而达到更好的预测性能，，故本文采用 DNN 来构建监督学习的深度网络模型，用于软件缺陷预测中。

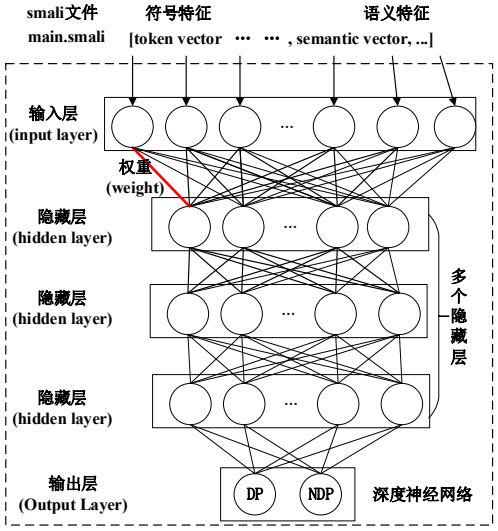

Fig 3 DNN architecture with main.smali

图 3 DNN 架构-以 main.smali 为例

图 3 展示了 DNN 架构在本文中的应用，一个输入层和一个输出层，中间存在很多隐藏层。其中输入层为本文提取的 smali 特征向量，输出层为分类结果。层与层之间的节点神经元全连接，每个连接之间都有权重参数，DNN 在训练的过程中，通过不断的迭代来调整节点间权重参数，从而达到最佳的分类效果。本文中采用 DNN 构建预测模型需要设置的参数包括隐藏层层数(number of hidden layers)、每层神经元节点数(number of neurons in each hidden layer)以及迭代的次数(number of iterations)，详细设置过程见小节 4.1.

2 缺陷预测模型框架

图 4 描述了本文提出并实现的面向 Android 二进制文件的缺陷预测模型(DefectDroid)的整体框架。首先，我们采用 smali/baksmali 工具[22]将 Android apk 文件反编译，得到 smali 文件。随后，我们采用特征子集选择方法中常用的信息增益算法来选取与缺陷特征相关的关键指令集(critical opcodes)用于提取特征，避免数据集特征的维度灾难。在关键指令集的基础上，提取 smali 文件的语义特征和符号特征，构成最后的 smali 特征向量。其中，符号特征的提取采用词袋模型[11]，统计关键指令集的频数构成，语义特征由关键指令集对 smali 文件序列化进行编码得到。Smali 特征向量加上对应的缺陷类别标签后，输入 DNN 进行模型训练。待模型训练

完成后，输入未标记的 smali 文件进行缺陷类别预测，从而帮助定位 apk 中的缺陷 smali 文件。

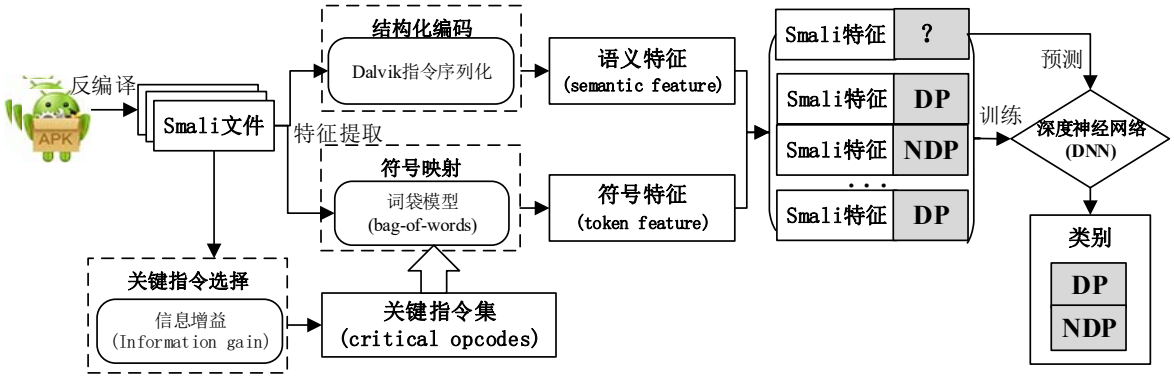

Fig 4 Overview of DefectDroid

图 4 DefectDroid 框架

2.1 关键指令集选择

根据 Android 官方开源资料[13]显示，smali 文件中包含的 dalvik 虚拟机指令总共有 245 种。然而，并不是所有的指令都与缺陷特征相关，而且用全部的 245 种虚拟机指令会导致数据集特征的维度灾难，从而影响模型的性能。本小节采用特征子集选择方法中常用的信息增益算法来筛选与缺陷特征相关度高的关键指令，构成关键指令集，用于符号特征和语义特征的提取。

信息增益算法[12]是特征工程中常用的一种特征子集选择方法，常用于机器学习模型中特征选择，挑选出与分类类别相关的特征，构建出更快更高效的模型。其基本的思想为，计算每一维度特征的信息熵和条件熵，二者的差值则为此维度为整个样本集带来的信息量，也称为信息增益。信息增益体现了特征的重要性，信息增益越大，特征对缺陷越重要。针对所有的 dalvik 指令进行信息增益计算，按照信息增益值排序，排名靠前的维度特征即为我们的关键指令集。

信息增益算法在 Android 二进制文件缺陷预测模型中应用如下。模型类别分为缺陷文件 DP 和无缺陷文件 NDP 两类，则 DP 类和 NDP 类出现的概率公式为：

$$P(C_{dp}) = \frac{N_{dp}}{N_{dp} + N_{ndp}} \tag{1}$$

$$P(C_{ndp}) = \frac{N_{ndp}}{N_{dp} + N_{ndp}} \tag{2}$$

其中 N_{dp} 与 N_{ndp} 分别为有缺陷 smali 文件数量和无缺陷 smali 文件数量。则类的信息熵为：

$$H(C) = -P(C_{dp})\log_2 P(C_{dp}) - P(C_{ndp})\log_2 P(C_{ndp}) \tag{3}$$

对某个特征指令 t，需要计算其在所有类别中出现的概率，即出现指令 t 的 smali 文件除以总共 smali 文件数量，不出现的概率，即没有指令 t 的 smali 文件除以总 smali 文件数量，公式为：

$$P(t) = \frac{A + B}{N_{dp} + N_{ndp}} \tag{4}$$

$$P(\bar{t}) = \frac{C + D}{N_{dp} + N_{ndp}} \tag{5}$$

其中 A 为出现指令 t 的有缺陷 smali 文件数量，B 为出现指令 t 的无缺陷 smali 文件数量，C 为不包含指令 t 的有缺陷 smali 文件数量，D 为不包含指令 t 的无缺陷 smali 文件数量。则条件熵为：

$$H(C|T)=P(t)H(C|t)+P(\bar{t})H(C|\bar{t}) \tag{6}$$

其中 T 表示特征 t， $H(C|t)$ 表示出现特征 t 的 smali 文件的条件熵，其计算方法如下：

$$H(C|t)=-P(C_{dp}|t)\log_2 P(C_{dp}|t)-P(C_{ndp}|t)\log_2 P(C_{ndp}|t) \tag{7}$$

其中， $P(C_{dp}|t)$ 表示有特征 t 情况下，是缺陷 smali 文件的概率，其计算方法为有缺陷 smali 文件中出现特征 t 的数量除以总的出现特征 t 的文件数量，

$$P(C_{dp}|t)=\frac{A}{A+B} \tag{8}$$

以此类推 $P(C_{ndp}|t)$ 为有特征 t 情况下，是无缺陷 smali 文件的概率，其计算方法如下：

$$P(C_{ndp}|t)=\frac{B}{A+B} \tag{9}$$

不出现特征 t 的条件熵为：

$$H(C|\bar{t})=-P(C_{dp}|\bar{t})\log_2 P(C_{dp}|\bar{t})-P(C_{ndp}|\bar{t})\log_2 P(C_{ndp}|\bar{t}) \tag{10}$$

其中， $P(C_{dp}|\bar{t})$ 为无特征 t 条件下，有缺陷 smali 文件的概率， $P(C_{ndp}|\bar{t})$ 为无特征 t 条件下，无缺陷 smali 文件概率，其计算方式如下：

$$P(C_{dp}|\bar{t})=\frac{C}{C+D} \tag{11}$$

$$P(C_{ndp}|\bar{t})=\frac{D}{C+D} \tag{12}$$

最后，特征指令 t 的信息增益计算公式为：

$$IG(T)=H(C)-H(C|T) \tag{13}$$

本文中，我们将 IG 算法应用于小节 3.1 中构建的 Android 二进制文件缺陷预测数据集中，总共 50 个 apk 文件，92774 个反编译得到的 smali 文件，其中 9395 个缺陷 smali 文件。通过 IG 算法，计算每一位指令的 IG 值，从小到大排列，数值越大表示其对缺陷预测越重要。从计算结果中发现，第三十位指令 IG 值已经衰减到 0.01，为第一位指令的 1.39%；而第三十一位指令 IG 值为 0.004，是第一位指令的 0.56%，意味着其为数据集中缺陷带来的信息增益不到第一位指令的 1%，重要性较低，故本文采用前 30 位指令作为关键指令集。表 1 为按照信息增益算法选取的 30 位关键指令集。

Table 1 The first 30 dalvik opcode ranked by the IG values

表 1 以信息增益值排序的前 30 位 dalvik 指令

序号	指令	IG 值	序号	指令	IG 值	序号	指令	IG 值
1	.method	0.7180	11	if-lt	0.2937	21	invoke-static	0.0828
2	.end	0.6066	12	check	0.2888	22	return-object	0.0645
3	invoke-super	0.4565	13	instance	0.2612	23	monitor	0.0578
4	invoke-virtual	0.4333	14	or	0.2331	24	new-instance	0.0577
5	invoke-direct	0.4090	15	add	0.2128	25	sparse	0.0347
6	return	0.3724	16	array	0.1770	26	rem	0.0240
7	const	0.3709	17	if-eqz	0.1710	27	sub	0.0124
8	iput	0.3637	18	throw	0.1235	28	div	0.0121

9	iget	0.3250	19	new-array	0.1096	29	nop	0.0116
10	move	0.3027	20	and	0.0907	30	cmpl	0.0114

2.2 smali特征向量

2.2.1 符号特征

在软件缺陷预测领域，部分之前的研究[1]将源代码类比于文本，将源代码中的词类比于文章中的单词，采用文本处理中常用的词袋模型[11]，忽略程序的语法和语义，将软件源代码看作词的集合，对词进行符号化，统计词出现的频数。同样，smali 文件同源文件一样，可以看作是 dalvik 指令集的集合，统计相关指令的频数，提取符号特征。

根据上文得到的关键指令集合，统计每一个 smali 文件中 30 个特征指令出现的频数，按照其信息增益值的顺序生成一个三十维的符号特征向量 T[t_1, t_2,..., t_{30}]。例如，图 2 中的 main.smali 文件，根据方法提取其符号特征为 T[1, 1, 1, 3, 1, 1, 3, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0,...,0]，其中，t_1 值为 1，表示".method"指令在其中只出现了一次，其他为零的值表示相应的指令未出现。

2.2.2 语义特征

除了符号特征外，本文针对 smali 文件中的 dalvik 指令提取语义特征。语义特征提取的核心思想是基于关键指令集中的指令，对 smali 文件按照指令顺序进行匹配，去掉无关指令和助记符，得到基于关键指令集的指令序列，最后，按照关键指令集中指令序号对指令序列数字化，进行编码，得到最后的语义特征向量。

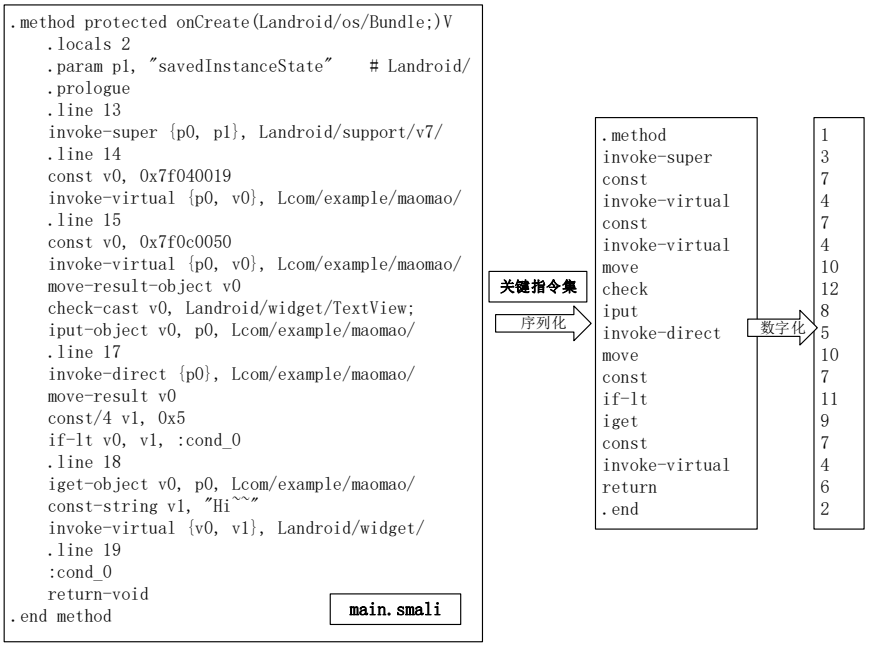

Fig 5 Semantic feature extraction

图 5 语义特征提取

图 5 展示基于 dalvik 指令的 smali 文件语义特征提取过程，首先对输入 smali 文件进行基于关键指令集的匹配，形成指令序列，然后根据关键指令集序号进行数据化，形成语义特征向量 S[s_1, s_2,...,]。以图 2 中 main.smali 文件为例，最后得到语义特征向量为 S[1, 3, 7, 4, 7, 4, 10, 12, 8, 5, 10, 7, 11, 9, 7, 4, 6, 2]。

得到 smali 文件的符号特征和语义特征后，我们将语义特征向量拼接在符号特征向量后，构成 smali 特征

向量。由于输入 DNN 的向量需要保持维度相同，我们将 smali 数据集中最长特征向量的维度数作为整个 DNN 模型输入的 smali 特征向量的维度数，其他不够长度的位数在后面补零。例如，图 2 中的 main.smali 经过特征提取后，得到的 smali 特征向量为[1, 1, 1, 3, 1, 1, 3, 1, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 7, 4, 7, 4, 10, 12, 8, 5, 10, 7, 11, 9, 7, 4, 6, 2, 0, 0, ..]。

2.3 缺陷预测

通过上述步骤，提取数据集中所有 smali 文件的 smali 特征向量，加上相应的类别标签后，构成数据集，然后输入到 DNN 中进行训练和预测。

3 实验设置

DefectDroid 主要由 smali 特征向量提取和 DNN 分类两部分构成，其中 smali 特征向量提取部分代码使用 Python 语言实现，DNN 分类算法采用由 Google 公司开源的深度学习平台 Tensorflow[24]实现，传统机器学习算法核心代码采用 scikit-learn 机器学习开源平台[23]实现。本文所有的实验在 CPU 为 Intel(R) Xeon(R) E5-2620 v3，内存为 64 GB，存储 2 TB，系统为 Ubuntu16.04 的服务器上完成。

3.1 数据集构建

目前，在软件缺陷预测领域使用较多的数据集为 Promise 数据集和 NASA 数据集[2]，二者的公开极大的推动了缺陷预测的研究，数据集也被业界公认为标准数据集。然而，两个数据集均为源代码数据集，并且不包含 Android 应用软件，故无法作为本文的数据集来构建缺陷预测模型。Android 漏洞库(Android Vulnerability Database) [25]和 CVE(Common Vulnerabilities and Exposures) [26]等公开的缺陷漏洞项目中，也没有 Android 应用软件数据集。面向 Android 软件的研究[1,6]数据集和相关实验代码未公开，因此无法进行复现和对比工作。因此，本文需要构建适用于 Android 二进制文件缺陷预测研究的数据集。

数据集的构建主要分为应用样本的选取和样本反编译后 smali 文件的类别标记两部分。然而，人工标记所有的 smali 文件是一件及其耗时耗力的工作，而 Android 平台应用软件源代码 java 文件和 smali 文件存在一一对应关系（不考虑混淆和加壳等代码保护方法的情况下），当一个 java 文件标记为有缺陷时，其对应的 smali 文件同样存在缺陷。因此，本文借助基于静态代码分析的缺陷检测扫描工具来扫描源代码中的缺陷 java 文件，从而标记出有缺陷的 smali 文件，完成样本标记工作。因此，在应用样本选取过程中，我们需要选择同时存在源代码和二进制文件的的应用样本。源代码仅用于自动化缺陷 smali 文件标记目的，当模型训练完成后，新输入的二进制文件在无需源代码的情况下，可以通过预测模型定位样本中有缺陷的 smali 文件。

3.1.1 应用样本选取

我们从知名的开源网站 Github 上选取符合要求的 Android 应用软件，这样既符合源代码和二进制文件同时存在的标记需求，又方便后续的研究人员获取数据集。数据集中应用软件的选取对模型的有效性有重要的影响力，因此我们在选取过程中结合实验需求和可选应用样本的版本数量、大小、源文件数量、热度等特征，总结了数据选取的标准。

(1) 项目版本数量不小于 20。为了完成同项目缺陷预测实验，每个项目选取五个不同版本的应用；同时为了避免由于版本间隔较小，不同版本软件间代码重叠较多的情况，选取的版本间最小间隔 4 个版本，因此所选取的项目版本数量不小于 20.

(2) 应用大小不低于 500KB。太小的应用包含的源代码较少，其二进制文件中 smali 文件不足，导致模型的输入不够。因此，本文选取的应用样本大小不低于 500KB。

(3) 项目应具有普遍性。所选取的项目需具有一定的热度，并且由多人开发。我们从项目的贡献者(Contributor)和项目的提交(Commit)两个维度来选取具有普遍性的项目。仅由一人开发的软件，不具有普遍性，而一次性提交的项目对比多次提交的项目来说，也不具备普遍性。

(4) 整个数据集应具有代表性。我们尽可能选取不同类别的项目来进行模型训练，从而使模型具有更好

的代表性。

表 2 显示了本文数据集中 Android 项目的选取结果。我们选取了 10 个不同 Andorid 项目，并从每个项目中选取了 5 个不同版本的应用软件作为本文的数据集，详细数据已开源[28]。

Table 2 Selected Android projects

表 2 数据集选取的 Android 项目

项目名称	版本数量	平均大小	提交数	贡献者	类别
AnkiDroid	575	8321	8565	102	教育
BankDroid	55	1476	1310	39	金融
BoardGameGeek	44	697	3242	4	阅读
Chess	24	1318	340	6	游戏
ConnectBot	281	1262	1475	32	网络
Andlytics	25	553	1478	27	工具
FBreader	404	2262	9016	35	阅读
K9Mail	341	3450	6654	149	网络
Wikipedia	119	3948	4307	43	工具
Yaaic	23	511	1063	19	网络

3.1.2 缺陷文件标记

本文以 smali 作为模块单位进行缺陷预测分析，在获取 smali 文件缺陷特征后，需要对其进行缺陷标记，从而构建数据集。当一个 smali 文件中包含一个或多个缺陷时，其被标记为 DP 有缺陷类别，不包含缺陷时，其被标记为 NDP 无缺陷类别。

本文采用源代码缺陷扫描工具 Checkmarx 来进行缺陷文件标记工作。Checkmarx[27]是一款知名的静态代码缺陷扫描工具，其可以扫描定位出源代码中常见的缺陷，例如信息泄露、缓冲区溢出等，并给出每个源代码文件详细的缺陷报告。所有缺陷文件标记工作均在 7.1.6 HF6 版本 Checkmarx Android 规则集上进行，实验数据和标记结果均开源在 github[28]。表 3 展示了 Checkmarx Android 缺陷规则集中部分缺陷规则，可以看到规则中部分是 Android 软件独有的规则，如“Passing Non Encrypted Data Between Activities”,“Improper Verification Of Intent By Broadcast Receiver”等 Android 组件相关的缺陷规则，也有 JAVA 语言通用的缺陷规则如“Integer Overflow”等通用的缺陷规则。通过解析缺陷报告，统计包含缺陷的源代码文件，其对应的 smali 文件即为缺陷文件。

Table 3 Part of Checkmarx Android defect rule set

表 3 Checkmarx Android 缺陷规则集（部分）

规则名称	风险等级	说明
Passing Non Encrypted Data Between Activities	低风险	在 Activities 间传递为加密数据
Improper Verification Of Intent By Broadcast Receiver	中风险	Broadcast Receiver 接收未经验证 Intent
Failure To Implement Least Privilege	低风险	没有实现权限管理
Improper Exception Handling	低风险	错误的异常处理
Improper Resource Shutdown or Release	低风险	资源关闭或释放不恰当
Information Exposure Through an Error Message	低风险	错误日志暴露信息
Side Channel Data Leakage	高风险	侧信道数据泄漏
Integer Overflow	低风险	整数溢出
Insufficient Sensitive Transport Layer	高风险	传输层保护不足
Non Encrypted Data Storage	低风险	数据未加密存储

图 6 为 smali 文件在项目不同版本中数量分布，图 7 为标记缺陷 smali 文件的比率分布。图中横坐标均为不同项目的五个版本，其中图 6 的纵坐标为每个应用中包含的 smali 文件数，图 7 的纵坐标为每个应用中包含缺陷 smali 文件的比率。总共 10 个 Android 应用项目，每个项目 5 个应用，包含 92,774 个 smali 文件，其中 9,395 个 DP 有缺陷 smali 文件，数据标记结果已开源[28]。

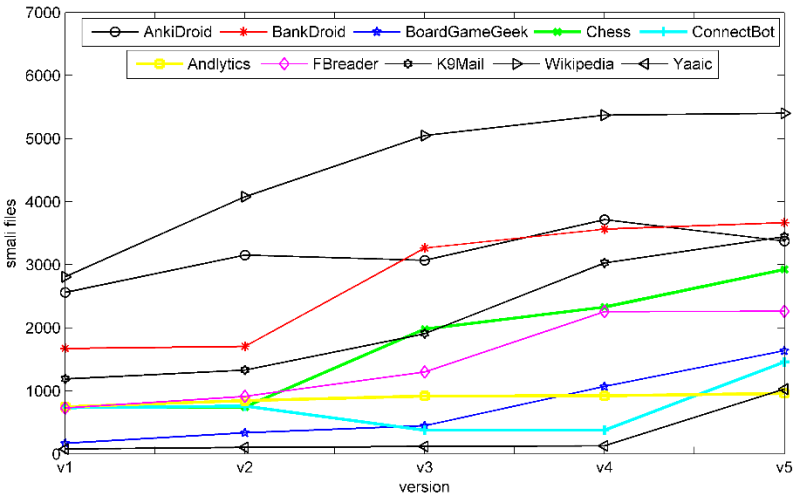

Fig 6 Smali file quantity distribution in selected Android projects

图 6 Smali 文件数量分布

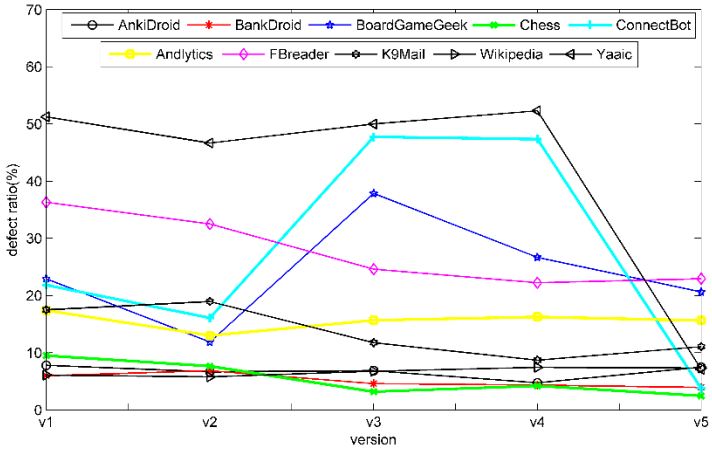

Fig 7 Defect smali file ratio distribution in selected Android projects

图 7 缺陷 Smali 文件比率分布

3.2 评价指标

机器学习算法分类结果的常用评价指标主要有查准率(precision)、查全率(recall)、F-measure 和 ROC-AUC 两种。然而，F-measure 在类不平衡情况下，例如 100 个样本中，正样本 90 个，负样本 10 个，全部判定为正样本时，根据 F-measure 评价体系，查准率为 90%，显然无法体现分类器的真实分类效果。而类不平衡问题是软件缺陷预测数据集中普遍存在的问题[2]，多数情况下，无缺陷模块数量远远超过有缺陷模块，例如本文数据集中，有缺陷 smali 文件占所有 smali 文件比率约为 10%。因此，本文采用 ROC-AUC 评价体系来评估分类器分类性能。

3.3 验证方法

软件缺陷预测模型的预测主要分为两类，即同项目缺陷预测 WPDP 和跨项目缺陷预测 CPDP。同项目缺陷预测模式中训练数据和测试数据均来自同一个 Android 项目，跨项目缺陷预测模式中训练数据和测试数据

来自不同 Android 项目。实际的预测任务中，经常碰到待预测项目缺乏相关的训练数据，此时需要通过从其他项目中训练数据得到相关的缺陷知识，迁移到待预测项目中，来完成预测项目的缺陷预测，因此，跨项目缺陷预测模式具有重要的实践意义。本文也主要从两类模式来构建 Android 二进制缺陷预测模型。

3.3.1 同项目缺陷预测

本文构建数据集包括 10 个 Android 项目，每个 Android 项目 5 个不同版本应用软件。我们做 10 组 WPDP 实验，分别对 10 个 Android 项目实施。每组 WPDP 实验中数据均来自同一项目，分为训练数据和测试数据。同时，为了防止划分训练数据和测试数据而导致分类器过度拟合，本文采用 5 折交叉验证方法(5-fold cross validation) [1]来划分数据以及训练。5 折交叉验证方法将实验数据随机平均分成 5 份，每一次取其中一份为测试数据，其余四份为训练数据，循环执行五次，然后取平均值作为性能评估结果。例如，我们测试项目 AnkiDroid 在同项目缺陷预测中的 AUC 值时，首先，我们将五个版本共 15875 个 smali 文件随机分成 5 份，每份 3175 个 smali 文件，编号 1 至 5。随后，将 1 号数据作为测试数据，其余四份作为训练数据，计算预测 AUC 值，将 2 号数据作为测试数据，其余为训练数据，计算 AUC 值，以此类推。最后，计算 5 个 AUC 平均值作为项目预测评估结果。

3.3.2 跨项目缺陷预测

测试数据和训练数据来自不同项目时，迁移源项目的缺陷知识来预测目标项目中的缺陷知识，为跨项目缺陷预测。本文数据集包含 10 个项目，如果全部进行跨项目缺陷预测，则需要进行 90 组实验，耗时耗力。因此，我们仅挑选部分项目进行本次跨项目缺陷预测。当我们进行跨项目预测时，选取源项目中第一个版本应用作为训练数据，目的项目中的第一个版本作为测试数据。

4 结果分析

本文设计了三个实验，第一个实验寻找 DNN 在本模型中最优配置参数，并且验证本文提出缺陷特征是否能有效的进行缺陷预测；第二个实验从同项目缺陷预测模式中比较 DNN 与传统机器学习算法的分类性能；第三个实验从跨项目缺陷预测模式中比较 DNN 与传统机器学习算法的分类性能。由于 Android 软件相关缺陷预测模型研究[1,6]难以复现，本文将 DNN 算法与缺陷预测表现较好的传统的机器学习算法[6]进行对比。

4.1 DNN最优配置

DNN 算法使用过程中，我们需要设置三个参数，即隐藏层层数(number of hidden layers)、每层神经元节点数(number of neurons in each hidden layer)以及迭代的次数(number of iterations)。隐藏层层数和每层神经元节点数由于相互影响的原因，经常组合在一起进行设置。我们设定 10 个隐藏层层数值，依次为*{3, 5, 10, 20, 50, 100, 200, 500, 800, 1000}*。同时，为了简化参数设置，我们将每层神经元节点数设置为相同，并设置 8 个节点数值，依次为*{20, 50, 100, 200, 300, 500, 800, 1000}*。迭代次数是另一个重要的参数，DNN 通过不断的迭代来调整节点间的权重，从而降低模型的错误率，迭代次数越多，错误率越低，但是消耗的时间越长。我们设定 7 个迭代次数数值，依次为*{1000, 2000, 3000, 5000, 10000, 15000, 20000}*。我们先调节隐藏层层数和每层神经元节点数，迭代次数设定为 10000，由平均 AUC 值确定最优组合，随后调节迭代次数，设定前两个参数为前面最优数值，以时间成本和错误率来确定最优迭代次数。由于同项目缺陷预测的模型性能普遍优于跨项目缺陷预测，因此我们采用同项目缺陷预测模式来进行 DNN 最优配置实验。实验步骤如下：

(1) 设置 DNN 参数，按照上述设置组合值，迭代次数设定为 10000，隐藏层为 3，每层节点数为 20。
(2) 分别对数据集中 10 组 Android 项目进行 WPDP 模式验证方法进行实验，采取 5 折交差验证分割测试数据和训练数据，得到 10 组 AUC 值，计算得到此参数下 AUC。
(3) 根据隐藏层和每层节点数取值组合，由小到大，依次改变参数组合，重复步骤 2 过程，得到所有参数组合 AUC 平均值。
(4) 选定所有参数组合 AUC 平均值最大组合为参数，按照迭代次数取值范围，依次改变迭代次数，计

算错误率和时间成本。

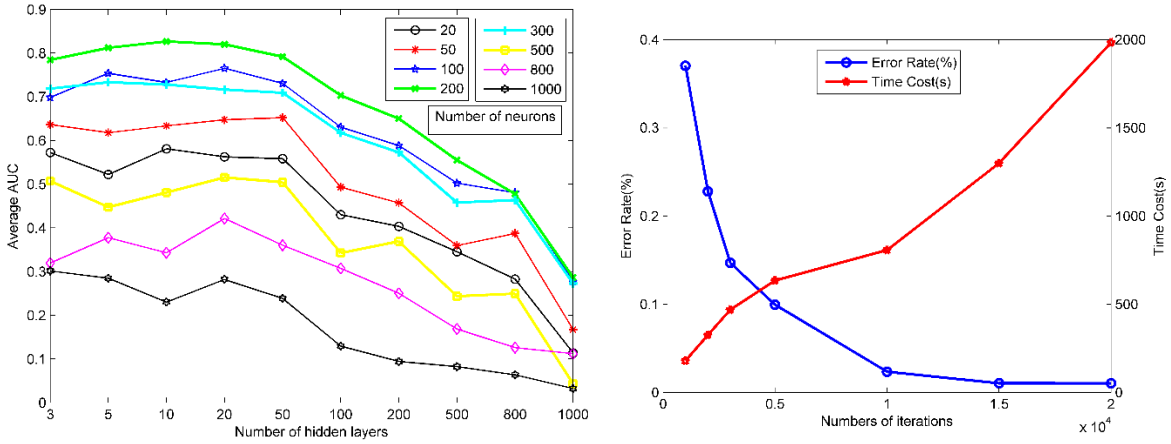

Fig 8 The experimental results of different parameters of DNN

图 8 DNN 最优配置实验结果

图 8 展示了实验结果。从左边图可以看到，不同神经元节点数的曲线多数都在 10 与 20 两个参数的隐藏层层数达到最大的 AUC 值，当隐藏层层数超过 20 后，大部分曲线都呈下降趋势。而节点数为 200 的曲线超越了其他曲线，隐藏层层数 10 时，到达 83.08%最大值。因此，选取的参数组合为隐藏层层数为 10 层，每层节点数为 200。从右图可以看到，错误率随着迭代次数增加而减少，时间随着迭代的次数增加，二者在 5000 附件重合，超过 10000 后，错误率曲线斜率减小，说明迭代次数增加带来的错误率减小效果减弱，而时间成本曲线斜率增加，说明迭代次数增加带来的时间成本增加效果更强，因此我们选定的迭代次数为 10000。同时我们可以看到，83.08%的准确率表明我们提取的 smali 缺陷特征能够很好的表征 Android 二进制文件缺陷，我们的模型具备 Android 二进制文件缺陷预测能力。

4.2 WPDP模式下算法对比

我们选取了四种在软件缺陷预测领域分类性能较好[6]的传统机器学习算法来与 DNN 进行对比，分别为支持向量机(Support Vector Machine,简称 SVM)、朴素贝叶斯(Naïve bayes,简称 NB)、决策树 C4.5(Decision Tree C4.5,简称 C4.5)、逻辑回归(Logistic Regression)。我们首先在 WPDP 模式下将 DNN 算法和四种传统算法进行对比，其中 DNN 参数配置为 4.1 实验结果中的最优配置，其他四种传统机器学习算法则采用 Tantithamthavorn 等人研究中缺陷预测模型中最优配置[6,29]。实验步骤如下：

(1) 采用 5 折交差验证方法分别对数据集中 10 组项目数据进行实验数据和训练数据分割。

(2) 分别将 10 组项目数据输入到 DNN 分类器，计算各组分类 AUC 值，并计算平均值。

(3) 将分类器依次换为 SVM、NB、C4.5 和 LR，重复步骤 1 和 2.

Table 4 Comparison of performance between DNN and traditional ML algorithms in WPDP

表 4 同项目缺陷预测模式下 DNN 与传统机器学习算法性能对比

项目名称	DNN	SVM	NB	LR	C4.5
AnkiDroid	0.8257	0.7756	0.8423	0.8233	0.7783
BankDroid	0.8168	0.6512	0.6743	0.6539	0.5696
BoardGameGeek	0.8633	0.6619	0.7649	0.7509	0.6373
Chess	0.8189	0.7261	0.8023	0.7623	0.6843
ConnectBot	0.8741	0.6748	0.7764	0.8374	0.6538
Andlytics	0.7769	0.6652	0.8134	0.8298	0.7936
FBreader	0.9291	0.8789	0.8391	0.7902	0.7354
K9Mail	0.6683	0.5213	0.6328	0.5305	0.5291

Wikipedia	0.8457	0.7169	0.8239	0.8178	0.7139
Yaaic	0.8897	0.7315	0.8129	0.7337	0.7305
平均值	**0.8308**	0.7003	0.7782	0.7530	0.6826

表 4 为同项目缺陷预测模式下，DNN 和四种传统分类算法的预测性能对比结果。从平均 AUC 值可以看到，DNN 达到最大 83.08%，在本文提出缺陷特征条件下，缺陷预测性能最优。传统分类算法中，NB 和 LR 性能较好，分别为 77.82%和 75.30%，而 SVM 和决策树 C4.5 算法较差，分别为 70.03%和 68.26%。DNN 比表现最好的 NB 提高了 6%，比表现最差的 C4.5 提高了 15%，在缺陷预测方面的性能比传统算法好。从总体预测结果来看，本模型提出的缺陷特征能够较好的用于 Android 二进制缺陷预测。

4.3 CPDP模式下算法对比

根据小节 3.3.2 中描述 CPDP 模式，我们同样选取上述四种传统算法与 DNN 在 CPDP 模式下进行缺陷预测性能对比。同 WPDP 模式不一样的是，CPDP 模式的训练数据和测试数据来自不同的项目中，因此，实验方法不同。在本实验中，我们采用 CPDP 模式验证方法，用一个项目中的第一个版本应用作为训练数据，用另一个项目的第一个版本应用作为测试数据，挑选了 10 组进行实验，输入到 5 个分类器中进行训练和预测。

Table 5 Comparison of performance between DNN and traditional ML algorithms in CPDP

表 5 跨项目缺陷预测模式下 DNN 与传统机器学习算法性能对比

源项目	目的项目	DNN	SVM	NB	LR	C4.5
AnkiDroid	BankDroid	0.4676	0.5329	0.6392	0.5375	0.5264
BankDroid	BoardGameGeek	0.6187	0.6197	0.6205	0.5239	0.5128
BoardGameGeek	FBreader	**0.7964**	0.7211	0.7316	0.7211	0.7236
Chess	ConnectBot	0.6392	0.4619	0.6267	0.5335	0.5291
ConnectBot	K9Mail	0.5993	0.6127	0.6392	0.5934	0.5439
Andlytics	FBreader	0.6881	0.5246	0.5177	0.5234	0.5173
FBreader	K9Mail	0.6746	0.5235	0.5351	0.5523	0.4687
ConnectBot	Yaaic	0.7823	0.7069	0.7013	0.6942	0.6821
Wikipedia	Andlytics	0.7295	0.5502	0.6724	0.6422	0.6328
Yaaic	AnkiDroid	0.6398	0.4476	0.4265	0.6823	0.6663
Average		**0.6636**	0.5701	0.6110	0.6004	0.5803

表 5 为跨项目缺陷预测模式下，DNN 和四种传统机器学习算法的性能对比实验结果。首先，我们从平均 AUC 值可以看到，在本文提出的缺陷特征条件下，DNN 以 66.36%超过其他传统机器学习算法，达到最好的预测性能。传统机器学习算法中，依然是 NB 和 LR 算法表现较好，C4.5 次之，SVM 表现最差。从不同组的预测数据可以看到，AUC 值普遍较低，如项目 AnkiDroid 预测项目 BankDroid，其 DNN 为最低值 46.76%，而当用项目 BoardGameGeek 预测项目 FBreader 时，其 DNN 值高达 79.64%，接近同项目缺陷预测模式的预测性能。我们发现项目 AnkiDroid 类别为教育，项目 BankDroid 类别为金融，二者类别相差较大，教育和金融领域业务差别大，代码功能区别大，而项目 BoardGameGeek 和项目 FBreader 类别同为阅读，项目 BoardGameGeek 中训练得到的缺陷知识与同为阅读应用的 FBreader 中潜在的缺陷知识相近，所以预测性能较好。因此，我们推测跨项目缺陷预测的性能与项目的类别有正相关关系。

5 讨论

本章节讨论本文所提出的面向 Android 二进制文件缺陷预测模型 DefectDroid 可能存在的局限性和未来可能的改进。

首先，在数据集构建方面，我们可能存在两个局限性。第一，我们选的 10 个 Android 项目，50 个样本包含的 90000 多个 smali 文件无法覆盖所有类型的缺陷，模型学习的缺陷知识不全面，但目前的预测性能在可接受范围内，后续可以增加数据集的项目数以增加覆盖率。第二，数据标记过程中，使用 Checkmarx 工具进行标记，工具的误报和漏报问题，会影响模型的预测性能。和所有的静态分析工具一样，Checkmarx 的标记结果存在一定的误报和漏报，但人工标记或人工审核全部结果耗时耗力。从实验结果来看，忽略 Checkmarx 标记

结果中的误报和漏报产生的性能影响远远小于人工标记或审核结果的成本，并且我们将 Checkmarx 的标记结果开源[28]，后续研究者可以一起改进数据集和标记结果。

其次，在特征提取的过程中，我们没有考虑代码混淆或者软件加壳等因素。我们在提取缺陷特征过程中，提取 dalvik 指令集的符号特征和语义特征，当一个 Android 二进制文件被代码混淆后，其符号特征和语义特征会随着混淆发生变化，产生的影响还需要进一步验证；另外，当一个 Android 二进制文件被加壳后，其文件中的 dex 文件往往被加密，无法反编译出 smali 文件，无法使用本模型进行预测分析。这种情况可以进行相关的脱壳处理或者其他方法处理后，使用模型进行预测分析。

最后，在软件模块粒度选取方面，本文采用粗粒度的文件作为预测模块，虽然对比 Giger 等人[36]采用细粒度的方法（method-level）作为预测模块消耗更小的计算成本，能低成本更快的分析建立模型，帮助分析人员定位到软件中的缺陷文件，但是由于模型特征以文件为单位，没有考虑文件间缺陷关联关系，文件粒度预测模型暂时无法预测跨文件类型缺陷。例如，跨文件的信息泄露类型缺陷，信息泄露源头和泄露终点在不同文件中，模型未将信息泄露源头文件和泄露终点文件关联，故暂时无法预测此类缺陷。在下一步工作中，我们计划采用细粒度的方法作为预测模块，来补充现有模型的局限。

6 总结与展望

本文提出了一种针对 Android 二进制可执行文件的缺陷预测模型，DefectDroid，并使用 Python 语言将其实现。首先，我们构建了一个包含 10 个 Android 项目，50 个样本，90000 多个 smali 文件的 Android 二进制文件缺陷预测数据集。其次，我们提出针对 smali 文件缺陷的特征提取方法。采用特征自己选取方法中常用的信息增益算法从全部的 dalvik 指令集中选取缺陷相关的特征子集，在此基础上，对 smali 文件提取符号特征和语义特征来构成缺陷特征向量。最后，我们将缺陷特征向量输入到 DNN 为分类器的模型中进行训练和预测，从而获得比传统机器学习算法更好的预测性能。

我们将 DefectDroid 应用于大规模 Android smali 文件缺陷预测任务中，并从同项目缺陷预测、跨项目缺陷预测、传统机器学习算法等方面对模型进行性能评估对比。实现结果表明，DefectDroid 提取的 smali 缺陷特征能有效的应用于 Android 二进制缺陷预测中，同时，在 DefectDroid 提取 smali 缺陷特征的条件下，DNN 分类器的缺陷预测性能在同项目缺陷预测和跨项目缺陷预测模式中均比传统机器学习算法性能优越。DefectDroid 能够帮助定位 Android 二进制文件中的缺陷 smali 文件。

未来，我们将持续的扩展我们的数据集[28]从而让其更具有代表性，并且在标记结果方面进行进一步的确认修复工作，从而提高数据集的准确性。我们还将在代码混淆对缺陷特征提取的影响做进一步的实验和验证。此外，我们还将细化预测模块的粒度，采用细粒度的方法或函数作为预测模块来构建模型，优化计算成本使预测更加高效实用。

References:

[1] Scandariato R, Walden J, Hovsepyan A, Joosen W. Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering, 2014.40(10):993-1006

[2] Chen X, Gu Q, Liu WS, Liu SL, Ni C. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016,27(1):1−25 (in Chinese). http://www.jos.org.cn/1000-9825/4923.htm

[3] Wang S, Liu TY, Tan L. Automatically learning semantic features for defect prediction, in: Proceedings of the 38th International Conference on Software Engineering (ICSE ’16), ACM, 2016,pp.297-308.

[4] Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier, in: Proceedings of the 38th International Conference on Software Engineering (ICSE ’16), ACM, 2016, pp.309-320

[5] Hall T, Beecham S, Bowes D. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. IEEE Transactions on Software Engineering, 2012, 38(6):1276-1304.

[6] Malhotra R. An empirical framework for defect prediction using machine learning techniques with Android software. Applied Soft Computing, 2016.40(10):993-1006

[7] Nguyen VH, Tran LMS. Predicting vulnerable software components with dependency graphs, in: International Workshop on Security Measurements and Metrics, ACM, 2010, pp.3

[8] Prasad MC, Florence L, Arya A. A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques. International Journal of Database Theory and Application, 2015.8(3):179-190

[9] Yuan Z, Yu LL, Liu C. Bug prediction method for fine-grained source code changes. Ruan Jian Xue Bao/Journal of Software, 2014, 25(11):2499−2517 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4559.htm

[10] Chen K, Liu P, Zhang Y. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets, in: International Conference on Software Engineering (ICSE '14). ACM, 2014:175-186.

[11] Perl H, Dechand S, Smith M. VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits, in: ACM Sigsac Conference on Computer and Communications Security (CCS '15). ACM, 2015:426-437.

[12] Liu H, Motoda H. Feature Selection for Knowledge Discovery and Data Mining. Kluwer Academic Publishers, 1998.

[13] Android Open Source Project, Dalvik bytecode: https://source.android.com/devices/tech/dalvik/dalvik-bytecode.html#instructions

[14] Lessmann S, Baesens B, Mues C. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, 2008, 34(4):485-496.

[15] Jerome Q, Allix K, State R. Using opcode-sequences to detect malicious Android applications, in: IEEE International Conference on Communications. IEEE, 2014:914-919.

[16] Mclaughlin N, Jesus M D R, Kang B J, et al. Deep Android Malware Detection, in: ACM on Conference on Data and Application Security and Privacy. ACM, 2017:301-308.

[17] Lecun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553):436-444.

[18] Deng L, Yu D. Deep Learning: Methods and Applications. Now Publishers Inc. 2014.

[19] David O E, Netanyahu N S. DeepSign: Deep learning for automatic malware signature generation and classification, in: International Joint Conference on Neural Networks. IEEE, 2015:1-8.

[20] Mou L, Li G, Jin Z, et al. TBCNN: A Tree-Based Convolutional Neural Network for Programming Language Processing. Eprint Arxiv, 2014.

[21] Yuan Z, Lu Y, Wang Z, et al. Droid-Sec: deep learning in android malware detection. Acm Sigcomm Computer Communication Review, 2014, 44(4):371-372.

[22] JesusFreke, smali/baksmali, https://github.com/JesusFreke/smali/wiki

[23] Scikit-learn, http://scikit-learn.org

[24] Google tensorflow, https://www.tensorflow.org/

[25] CCERT, Android Vulnerabilities Database, http://android.scap.org.cn/

[26] MITRE, Common Vulnerabilities and Exposures, http://cve.mitre.org/

[27] Checkmarx, https://www.checkmarx.com/

[28] DefectDroid, https://github.com/breezedong/DefectDroid.git

[29] Tantithamthavorn C, Mcintosh S, Hassan A E. Automated parameter optimization of classification techniques for defect prediction models，in: International Conference on Software Engineering (ICSE '16). ACM, 2016:321-332.

[30] Tantithamthavorn C. Towards a better understanding of the impact of experimental components on defect prediction modelling, in: International Conference on Software Engineering Companion. ACM, 2016:867-870.

[31] Arzt S, Rasthofer S, Fritz C, et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps, in: ACM Sigplan Conference on Programming Language Design and Implementation. ACM, 2014:259-269.

[32] LU, L., LI, Z., WU, Z., LEE, W., AND JIANG, G. Chex: statically vetting android apps for component hijacking vulnerabilities, in: Proceedings of the 2012 ACM conference on Computer and communications security (2012), ACM, pp. 229–240.

[33] Jiang Y Z X, Xuxian Z. Detecting passive content leaks and pollution in android applications, in: Proceedings of the 20th Network and Distributed System Security Symposium (NDSS). 2013.

[34] Bengio Y. Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2009, 2(1): 1-127.

[35] HP Fortify SCA, https://software.microfocus.com/pt-pt/software/sca

[36] Giger E, D'Ambros M, Pinzger M, et al. Method-level bug prediction, in: Acm-Ieee International Symposium on Empirical Software Engineering and Measurement. IEEE, 2013:171-180.

[37] Rahman F, Khatri S, Barr E T, et al. Comparing static bug finders and statistical prediction, in: International Conference on Software Engineering (ICSE '14). ACM, 2014:424-434.

附中文参考文献:

[2] 陈翔,顾庆,刘望舒,刘树龙,倪超.静态软件缺陷预测方法研究.软件学报,2016,27(1):1−25. http://www.jos.org.cn/1000-9825/4923.htm

[9] 原子,于莉莉,刘超.面向细粒度源代码变更的缺陷预测方法.软件学报,2014,25(11):2499−2517. http://www.jos.org.cn/1000-9825/4559.htm.