文章编号:1007-5321(2018)01-0013-11 **DOI**:10.13190/j.jbupt.2017-243

面向 Android 二进制代码的缺陷预测方法

董 枫, 刘天铭, 徐国爱, 郭燕慧, 李承泽
(北京邮电大学 网络空间安全学院, 北京 100876)

摘要: 针对 Android 软件缺陷预测任务中源代码难以获取的问题,提出一种面向 Android 二进制可执行文件的缺陷预测模型,同时采用深度神经网络进行缺陷预测. 首先,通过一种创新的 Android 可执行文件缺陷特征提取方法,提取其符号特征和语义特征来构建缺陷特征向量;其次,用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;最后,将工具原型 DefectDroid 应用于大规模 smali 文件缺陷预测任务中,在同项目缺陷预测、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.

关 键 词: 缺陷预测; 软件安全; Android 二进制文件; 机器学习; 深度神经网络
中图分类号: TP311.5 **文献标志码**: A

Defect Prediction Method for Android Binary Files

DONG Feng, LIU Tian-ming, XU Guo-ai, GUO Yan-hui, LI Cheng-ze
(School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Software defect prediction is an important method in the field of software security. Most of existing defect prediction models 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 have a shallow architecture, which leads to a limited capacity of expressing complex functions between features and defects. The author proposes a practical defect prediction model for Android binary files using deep neural network (DNN). A new approach is proposed to generate features that capture both token and semantic features of the defective smali (decompiled files of apks) files in apks. The feature vectors are input into DNN to train and build the defect prediction model in order to achieve accuracy. The article implements the model called DefectDroid and applies it to a large number of Android smali files. The performance of DefectDroid is compared from three aspects: within-project defect prediction, cross-project defect prediction and traditional machine learning algorithms.

Key words: 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、运行异常等,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

收稿日期: 2017-11-30
基金项目: 国家自然科学基金项目(61401038); 2016 广东省科学技术厅前沿与关键技术创新项目(2016B010110002)
作者简介: 董 枫(1990—), 男, 博士生, E-mail: dongfeng@bupt.edu.cn; 徐国爱(1972—), 男, 教授, 博士生导师.

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. 针对移动平台,特别是 Android 平台应用软件的缺陷预测研究成为当前热点.

基于机器学习的软件缺陷预测研究主要集中在以下 2 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的特征[1]、基于代码文本的符号特征[2]和基于代码结构如抽象语法树(AST, abstract syntax tree)[3]、程序依赖图(PDG, program dependence graph)[4]等的语义特征.

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(SVM, support vector machine)、朴素贝叶斯(NB, naive bayes)、决策树 C4.5(C4.5, decision tree C4.5)、逻辑回归(LR, logistic regression)等[5].

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 2 个不同的团队完成. 负责缺陷审计安全评估的第 3 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,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,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,采用常用的特征子集选择方法中的信息增益法(IG, information gain)[6]从全部的 dalvik 指令集中选择与缺陷相关度高的关键指令集,从而降低特征维度,防止维度灾难. 同时,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,采用深度神经网络(DNN, deep neural network)作为分类器来训练缺陷预测模型. 实现了模型系统 DefectDroid,将其应用于大规模的数据集(50 个 Android 应用软件,92 774 个 smali 文件)中,并且采用 ROC-AUC (the receiver operating characteristic and area under the curve)评价体系[7]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(WPDP, within-project defect prediction)中的准确率达 83.08%,跨项目缺陷预测(CPDP, cross-project defect prediction)中的准确率达 66.36%,对 Android 二进制文件具有缺陷预测能力.

笔者在 Android 应用软件缺陷预测方面的贡献主要包括以下 3 点:

1) 提出针对 Android 二进制文件的缺陷预测模型.

2) 提出了创新的 Android 二进制文件的缺陷特征提取方法.

3) 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.

1 相关工作

1.1 软件缺陷预测

图 1 所示为基于机器学习的软件缺陷预测方法的一般流程. 首先,将软件分解为模块,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[8];然后对模块进行缺陷特征提取和标记工作,形成带类标记的特征向量;最后将带类标记的特征向量输入分类算法中,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 1 个项目时,预测任务为同项目缺陷预测;来自不同项目时,预测任务为跨项目缺陷预测[9]. 跨项目缺陷预测的意义在于当需要进行的缺陷预测项目已有的训练数据较少时,可以通过从不同的项目学习到缺陷知识,迁移到需要缺陷预测项目来进行缺陷预测. 跨项目缺陷预测受到研究者的广泛关注[10]. 主流预测模型为二分类预测模型,即预测为有缺陷倾向性(DP, defect-proneness)模块和无缺陷倾向性(NDP, non-defect-proneness)模块.

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[11]则创新地采用版本管理工具,如 git、Subversion 等中的修改提交(commit)作为模块进行建模预测. Giger 等[8]采用细粒度的方法(method-level)作为预测模块进行缺陷预测.

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. 主要关注基于代码的度量元特征提取. Scandariato 等[2]采用文本处理的方式,将 Android 源代码中的 java 文件类看作文本,

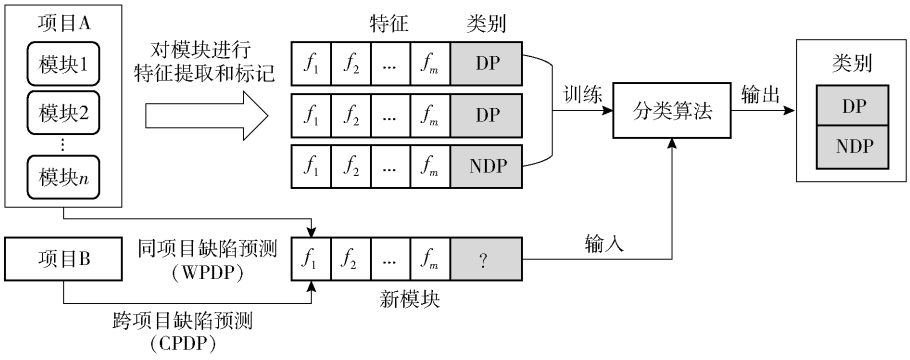

图 1　软件缺陷预测一般流程

采用词袋模型统计文件中相关的词的频数,用符号特征生成缺陷特征向量. Wang 等[3]则创新地采用深度信念网络(deep belief network)来自动化提取缺陷特征. 在分类算法方面,Malhotra 等[5]提出了一套成熟的 Android 应用软件缺陷预测框架.

1.2　Smali 文件特征

Android 二进制文件由 1 个压缩包组成,包括 dex 可执行文件、签名、资源文件等,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 apk 文件中包含 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 指令集以 smali 语法构成. 图 2 所示为典型示例. 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. 当分析出 smali 文件中的缺陷时,可以准确定位到源代码的缺陷代码位置.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    myTextView = (TextView)findViewById(R.id.Text_view);

    if(random() >= 5)
    myTextView.setText("Hi~~");
    }
```

main.java

编译

main.smali

```
# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
    .locals 2
    .param p1, "savedInstanceState"    # Landroid/os/Bundle;
    .prologue
    .line 13
    invoke-super {p0, p1}, Landroid/support/v7/app/AppCompatAct
    .line 14
    const v0, 0x7f040019
    invoke-virtual {p0, v0}, Lcom/example/maomao/helloworldapp/
    .line 15
    const v0, 0x7f0c0050
    invoke-virtual {p0, v0}, Lcom/example/maomao/helloworldapp/
    move-result-object v0
    check-cast v0, Landroid/widget/TextView;
    iput-object v0, p0, Lcom/example/maomao/helloworldapp/MainA
    .line 17
    invoke-direct {p0}, Lcom/example/maomao/helloworldapp/MainA
    move-result v0
    const/4 v1, 0x5
    if-lt v0, v1, :cond_0
    .line 18
    iget-object v0, p0, Lcom/example/maomao/helloworldapp/MainA
    const-string v1, "Hi~~"
    invoke-virtual {v0, v1}, Landroid/widget/TextView;->setText
    .line 19
    :cond_0
    return-void
.end method
```

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. 笔者同时提取 smali 文件的符号特征和语义特征,将二者结合,构成更加全面的缺陷特征向量,输入分类器中进行模型构建和缺陷预测. 详见 2.2 节.

1.3　深度神经网络

深度网络在很多研究中也称为深度学习,由于其深层结构具备表达复杂函数的能力[12],在人工智能领域具有重大意义. 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[13-14].

深度网络根据不同的应用场景,用不同的架构构建深度学习模型,如深度神经网络 DNN、卷积神经网络(CNN, convolutional neural network)、循环神经网络(RNN, recursive neural network)、深度信念网络(DBN, deep belief network)等. 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[15],更好地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,故采用 DNN 来构建监督学习的深度网络模型.

图 3 展示了 DNN 架构的应用,输入层和输出层中间存在很多隐藏层. 其中输入层为提取的 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,通过不断迭代来调整节点间权重参数,从而达到最佳的分类效果. 采用 DNN 构建预测模型需要设置的参数见 4.1 节.

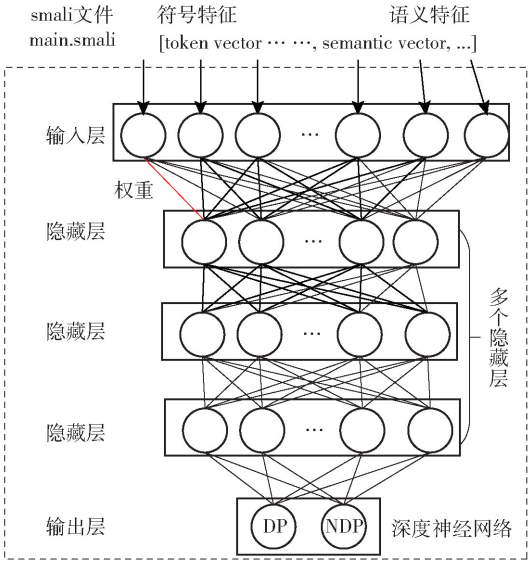

图3 DNN架构(以main.smali输入为例)

2 缺陷预测模型框架

图4所示为面向Android二进制文件缺陷预测模型DefectDroid的整体框架. 首先,将Android apk文件反编译,得到smali文件;随后,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[2],以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

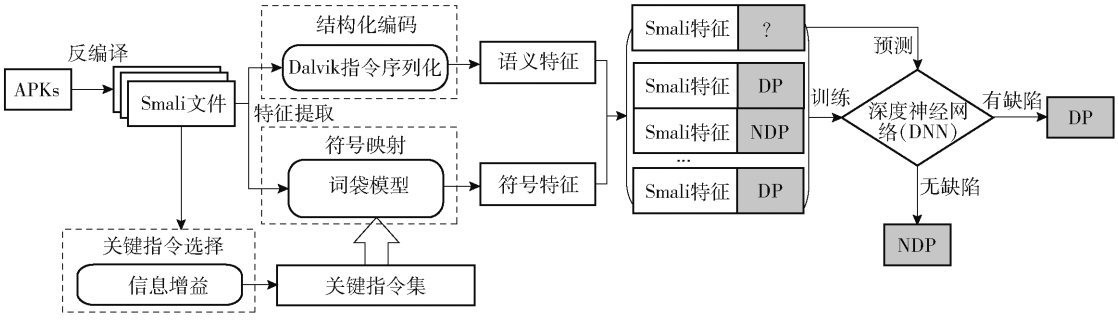

图4 DefectDroid框架

位apk中的缺陷smali文件.

2.1 关键指令集选择

Dalvik虚拟机指令共有245种,并不是所有的指令都与缺陷特征相关,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. 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[6]是特征工程中常用的特征子集选择方法,常用于机器学习模型中的特征选择,以挑选出与分类类别相关的特征,构建出更高效的模型. 其基本思想为:计算每1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,排名靠前的指令即为关键指令集.

信息增益算法在Android二进制文件缺陷预测模型中的应用如下. 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

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

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

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

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

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文件除以总的smali文件数量,不出现的概率,即没有指令t的smali文件除以总的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，

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

其中：$P(C_{dp}|t)$ 表示特征 t 情况下缺陷 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，$P(C_{ndp}|t)$ 为特征 t 情况下无缺陷 smali 文件的概率，其计算方法为

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

不出现特征 t 的条件熵为

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

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

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

将 IG 算法应用于 3.1 节中构建的 Android 二进制文件缺陷预测数据集中，共 50 个 apk 文件，92 774 个反编译得到的 smali 文件，其中 9 395 个缺陷 smali 文件. 通过 IG 算法，计算每 1 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 30 位指令的 IG 值已经衰减到 0.01，为第 1 位指令的 1.39%；故采用前 30 位指令作为关键指令集. 表 1 所示为按照信息增益算法选取的 30 位关键指令.

表 1　以信息增益值排序的前 30 位 dalvik 指令

序号	指令	IG 值/%
1	.method	0.718 0
2	.end	0.606 6
3	invoke-super	0.456 5
4	invoke-virtual	0.433 3
5	invoke-direct	0.409 0
6	return	0.372 4
7	const	0.370 9
8	iput	0.363 7
9	iget	0.325 0
10	move	0.302 7
11	if-lt	0.293 7
12	check	0.288 8
13	instance	0.261 2
14	or	0.233 1
15	add	0.212 8
16	array	0.177 0
17	if-eqz	0.171 0
18	throw	0.123 5
19	new-array	0.109 6
20	and	0.090 7
21	invoke-static	0.082 8
22	return-object	0.064 5
23	monitor	0.057 8
24	new-instance	0.057 7
25	sparse	0.034 7
26	rem	0.024 0
27	sub	0.012 4
28	div	0.012 1
29	nop	0.011 6
30	cmpl	0.011 4

2.2　smali 特征向量

2.2.1　符号特征

在软件缺陷预测领域，之前的研究[2]将源代码类比于文本，将源代码中的词类与文章中的单词相比，采用文本处理中常用的词袋模型，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 1 个 30 维的符号特征向量 $\boldsymbol{T}[t_1, t_2, \cdots, t_{30}]$. 例如,对图 2 中的 main. smali 文件,提取其符号特征为 $\boldsymbol{T}$[1, 1, 1, 3, 1, 1, 3, 1, 1, 2, 1, 1, 0, 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 1 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,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编码,得到语义特征向量.

图 5 所示为基于 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 $\boldsymbol{S}[s_1, s_2, \cdots,]$. 以图 2 中 main. smali 文件为例,最后得到语义特征向量为 $\boldsymbol{S}$[1, 3, 7, 4, 7, 4, 10, 12, 8, 5, 10, 7, 11, 9, 7, 4, 6, 2].

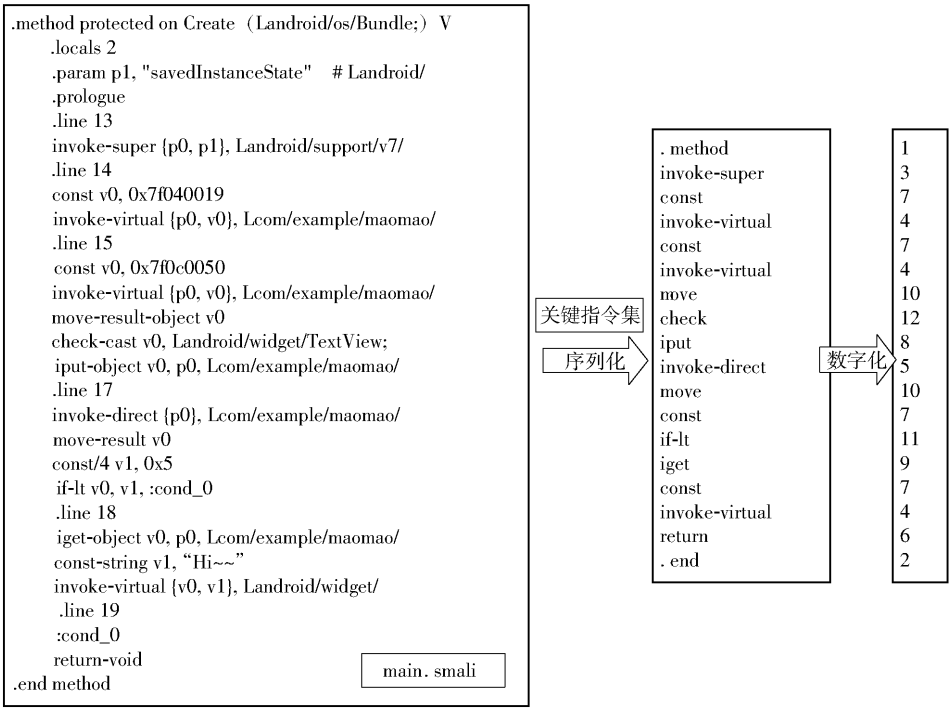

图 5 语义特征提取

得到 smali 文件的符号特征和语义特征后,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,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 0. 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,提取出数据集中所有 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 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. 所有的实验在系统为 Ubuntu16.04 的服务器上完成. CPU 为 Intel(R) Xeon(R) E5-2620 v3,内存为 64 GB,存储 2 TB.

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[3]. 然而,2 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 漏洞库和 CVE 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[2,5] 中数据集和相关实验代码未公开，无法进行复现和对比工作. 因此，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 1 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，笔者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，需要选择同时存在源代码和二进制文件的应用样本.

3.1.1　应用样本选取

从 Github 上选取符合要求的 Android 应用软件. 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，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 5 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 500 KB. 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，选取的应用样本大小不低于 500 KB.

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. 从项目的贡献者和项目的提交(commit)2 个维度来选取具有普遍性的项目. 仅由 1 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. 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. 选取 10 个不同 Andorid 项目，并从每个项目中选取 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[16].

表 2　数据集选取的 Android 项目

项目名称	版本数量	平均大小/KB	提交数	贡献者	类别
AnkiDroid	575	8 321	8 565	102	教育
BankDroid	55	1 476	1 310	39	金融
BoardGameGeek	44	697	3 242	4	阅读
Chess	24	1 318	340	6	游戏
ConnectBot	281	1 262	1 475	32	网络
Andlytics	25	553	1 478	27	工具
FBreader	404	2 262	9 016	35	阅读
K9Mail	341	3 450	6 654	149	网络
Wikipedia	119	3 948	4 307	43	工具
Yaaic	23	511	1 063	19	网络

注：版本数量为项目发行的版本数，提交数为项目代码更新次数，贡献者为参与项目的开发者数.

3.1.2　缺陷文件标记

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 1 个 smali 文件中包含 1 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.

采用源代码缺陷扫描工具 Checkmarx 结合人工确认来进行缺陷文件标记工作，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. 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.

图 6 所示为标记缺陷 smali 文件的比率分布. 图中横坐标均为不同项目的 5 个版本，纵坐标为每个应用中包含缺陷 smali 文件的比率. 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.

与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，所以由 2 个缺陷规则熟悉的研究生针对缺陷文件进行约 1 周时间的人工确认工作，进一步确保标记数据准确性.

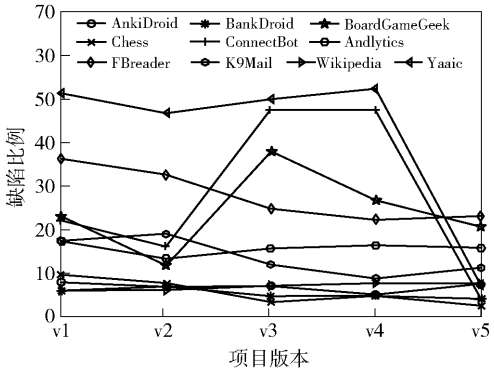

图6 缺陷 Smali 文件比率分布

3.2 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1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.

3.2.1 同项目缺陷预测

构建数据集包括10个 Android 项目,每个 Android 项目包括5个不同版本的应用软件. 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,采用5折交叉验证方法(5-fold cross validation)[2]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,每次取其中1份作为测试数据,其余4份为训练数据,循环执行5次,然后取平均值作为性能评估结果.

3.2.2 跨项目缺陷预测

测试数据和训练数据来自不同项目时,可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. 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,仅挑选部分项目进行本次跨项目缺陷预测. 当进行跨项目预测时,选取源项目中第1个版本应用作为训练数据,将目的项目中的第1个版本作为测试数据.

4 结果分析

设计了3个实验,第1个实验寻找 DNN 在本模型中的最优配置参数,并且验证所提出的缺陷特征是否能有效地进行缺陷预测;第2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3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.

4.1 DNN 最优配置

DNN 算法使用过程中,需要设置3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). 由于隐藏层层数和每层神经元节点数相互影响,所以经常组合在一起进行设置. 设定10个隐藏层层数值,依次为{3, 5, 10, 20, 50, 100, 200, 500, 800, 1 000}. 同时,为了简化参数设置,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, 1 000}. 迭代次数是另1个重要参数,DNN 通过不断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. 设定7个迭代次数数值,依次为{1 000, 2 000, 3 000, 5 000, 10 000, 15 000, 20 000}. 先调节隐藏层层数和每层神经元节点数,迭代次数设定为10 000,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. 同项目缺陷预测的模型性能普遍优于跨项目缺陷预测,因此采用同项目缺陷预测模式进行 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10 000,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,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,计算得到此参数下的 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,得到所有参数组合的 AUC 平均值;

4) 选定所有参数组合中 AUC 平均值最大的组合为参数,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.

实验结果如图7、图8所示. 从图7可以看到,

不同神经元节点数的曲线多数都在 10 和 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，隐藏层层数为 10 时，到达 83.08% 最大值. 因此，选取的参数组合是隐藏层层数为 10，每层节点数为 200. 从图 8 可以看到，错误率随着迭代次数的增加而减少，时间随着迭代的次数的增加，二者在 5 000 次迭代处重合，超过 1 万后，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的效果减弱，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，因此选定的迭代次数为 1 万. 同时可以看到，83.08% 的准确率表明，提取的 smali 缺陷特征能够很好地表征 Android 二进制文件缺陷，模型具备 Android 二进制文件缺陷预测能力.

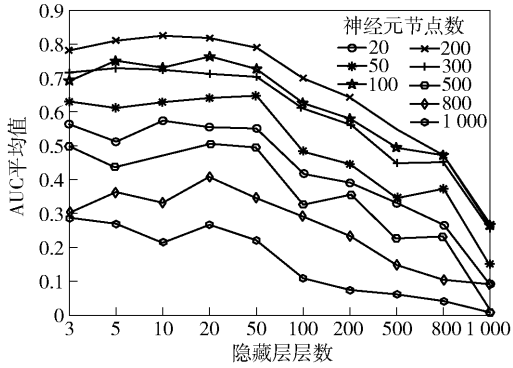

图 7　DNN 层数与神经元数最优配置实验结果

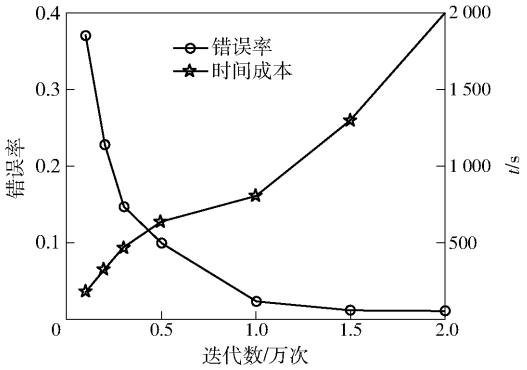

图 8　DNN 迭代次数最优配置实验结果

4.2　同项目缺陷预测模式下算法对比

选取了支持向量机 SVM、朴素贝叶斯 NB、决策树 C4.5、逻辑回归 LR 4 种在软件缺陷预测领域分类性能较好[5]的传统机器学习算法与 DNN 进行对比. 首先在 WPDP 模式下进行对比，其中 DNN 参数配置为 4.1 实验结果中的最优配置，其他 4 种传统机器学习算法则采用 Tantithamthavorn 等研究中缺陷预测模型中的最优配置[17]. 实验步骤如下：

1）采用 5 折交叉验证方法分别对数据集中 10 组项目的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）.

表 3 所示为同项目缺陷预测模式下 DNN 和 4 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%，在笔者提出的缺陷特征条件下，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地用于 Android 二进制缺陷预测.

表 3　WPDP 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	0.825 7	0.775 6	0.842 3	0.823 3	0.778 3
BankDroid	0.816 8	0.651 2	0.674 3	0.653 9	0.569 6
BoardGameGeek	0.863 3	0.661 9	0.764 9	0.750 9	0.637 3
Chess	0.818 9	0.726 1	0.802 3	0.762 3	0.684 3
ConnectBot	0.874 1	0.674 8	0.776 4	0.837 4	0.653 8
Andlytics	0.776 9	0.665 2	0.813 4	0.829 8	0.793 6
FBreader	0.929 1	0.878 9	0.839 1	0.790 2	0.735 4
K9Mail	0.668 3	0.521 3	0.632 8	0.530 5	0.529 1
Wikipedia	0.845 7	0.716 9	0.823 9	0.817 8	0.713 9
Yaaic	0.889 7	0.731 5	0.812 9	0.733 7	0.730 5
AUC 平均值	**0.830 8**	0.700 3	0.778 2	0.753 0	0.682 6

注：每个算法结果以 AUC 值量化评估，值越高，算法分类性能越好.

4.3　跨项目缺陷预测模式下算法对比

根据节 3.2.2 节中描述的 CPDP 模式，同样选取上述 4 种传统算法与 DNN 在 CPDP 模式下进行缺陷预测性能对比. 与 WPDP 模式不同的是，CPDP 模式的训练数据和测试数据来自不同的项目，因此，实验方法不同. 在本实验中，采用 CPDP 模式验证方法，用 1 个项目中的第 1 个版本应用作为训练数据，用另 1 个项目的第 1 个版本应用作为测试数据，

挑选10组进行实验,输入到5个分类器中进行训练和预测.

表4所示为跨项目缺陷预测模式下,DNN和4种传统机器学习算法的性能对比实验结果. 首先,从平均AUC值可以看到,在笔者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. 在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. 项目AnkiDroid类别为教育,项目BankDroid类别为金融,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. 因此推测,跨项目缺陷预测的性能与项目的类别有正相关关系.

表4 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	0.467 6	0.532 9	0.639 2	0.537 5	0.526 4
BankDroid	BoardGameGeek	0.618 7	0.619 7	0.620 5	0.523 9	0.512 8
BoardGameGeek	FBreader	0.796 4	0.721 1	0.731 6	0.721 1	0.723 6
Chess	ConnectBot	0.639 2	0.461 9	0.626 7	0.533 5	0.529 1
ConnectBot	K9Mail	0.599 3	0.612 7	0.639 2	0.593 4	0.543 9
Andlytics	FBreader	0.688 1	0.524 6	0.517 7	0.523 4	0.517 3
FBreader	K9Mail	0.674 6	0.523 5	0.535 1	0.552 3	0.468 7
ConnectBot	Yaaic	0.782 3	0.706 9	0.701 3	0.694 2	0.682 1
Wikipedia	Andlytics	0.729 5	0.550 2	0.672 4	0.642 2	0.632 8
Yaaic	AnkiDroid	0.639 8	0.447 6	0.426 5	0.682 3	0.666 3
AUC平均值		**0.663 6**	0.570 1	0.611 0	0.600 4	0.580 3

注:每个算法结果以AUC值量化评估,值越高,算法分类性能越好.

5 结束语

提出了一种针对Android二进制可执行文件的缺陷预测模型DefectDroid,并将其应用于大规模smali文件缺陷预测任务中. 实验结果表明,DefectDroid提取的smali缺陷特征能有效地应用于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可能存在一定的局限性. 在数据集构建方面,50个样本包含的9万多个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[18]. 在软件模块粒度选取方面,笔者采用粗粒度的文件作为预测模块,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. 在下一步工作中,可采用细粒度的方法作为预测模块,弥补现有模型的缺陷.

参考文献:

[1] Prasad M C, Florence L, Arya A. A study on software metrics based software defect prediction using data mining and machine learning techniques [J]. International Journal of Database Theory and Application, 2015, 8(3): 179-190.

[2] Scandariato R, Walden J, Hovsepyan A, et al. Predicting vulnerable software components via text mining[J]. IEEE Transactions on Software Engineering, 2014, 40(10): 993-1006.

[3] Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction[C]//Proceedings of the 38th International Conference on Software Engineering. Austin: ACM, 2016: 297-308.

[4] Nguyen V H, Le M S T. Predicting vulnerable software components with dependency graphs[C]//International Workshop on Security Measurements and Metrics. [S.

l.]: ACM, 2010: 3.

[5] Malhotra R. An empirical framework for defect prediction using machine learning techniques with Android software [J]. Applied Soft Computing, 2016, 40(10): 993-1006.

[6] Liu H, Motoda H. Feature selection for knowledge discovery and data mining [M]. [S. l.]: Kluwer Academic Publishers, 1998: 1-10.

[7] Lessmann S, Baesens B, Mues C. Benchmarking classification models for software defect prediction: a proposed framework and novel findings [J]. IEEE Transactions on Software Engineering, 2008, 34(4): 485-496.

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

[9] 陈翔, 顾庆, 刘望舒, 等. 静态软件缺陷预测方法研究[J]. 软件学报, 2016, 27(1): 1-25.
Chen Xiang, Gu Qing, Liu Wangshu, et al. Survey of static software defect prediction[J]. Ruan Jian Xue Bao/ Journal of Software, 2016, 27(1): 1-25.

[10] Zhang F, Zheng Q, Zou Y, et al. Cross-project defect prediction using a connectivity-based unsupervised classifier[C]// Proceedings of the 38th International Conference on Software Engineering. Austin: ACM, 2016: 309-320.

[11] Perl H, Dechand S, Smith M. VCCFinder: finding potential vulnerabilities in open-source projects to assist code audits[C]// ACM Sigsac Conference on Computer and Communications Security (CCS'15). Denver: ACM, 2015: 426-437.

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

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

[14] Mclaughlin N, Jesus M D R, Kang B J, et al. Deep android malware detection[C]// ACM on Conference on Data and Application Security and Privacy. [S. l.]: ACM, 2017: 301-308.

[15] Bengio Y. Learning deep architectures for AI [J]. Foundations & Trends® in Machine Learning, 2009, 2(1): 1-127.

[16] Dong F. DefectDroid[EB/OL]. (2017-10-01)[2017-11-05]. https: // github. com/breezedong/DefectDroid. git

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

[18] 杨朝红, 宫云战, 肖庆, 等. 基于软件缺陷模型的测试系统[J]. 北京邮电大学学报, 2008, 31(5): 1-4.
Yang Zhaohong, Gong Yunzhan, Xiao Qing, et al. A defect model based testing system[J]. Journal of Beijing University of Posts and Telecommunications, 2008, 31(5): 1-4.