基于线性分类算法的软件错误定位模型

何海江
HE Haijiang

长沙学院 数学与计算机科学系,长沙 410022
Department of Mathematics and Computer Science, Changsha University, Changsha 410022, China

HE Haijiang. Software fault localization model based on linear classification algorithm. Computer Engineering and Applications, 2017, 53(21):42-48.

Abstract: Spectrum-Based Fault Localization (SBFL) techniques aid developers to reduce the debugging effort. As a light-weight promising approach, SBFL only collects the testing result of passed or failed, and the corresponding coverage information. Based on these data, SBFL can then calculate a runtime spectra for each program statement. SBFL approaches apply various functions to map four profile features to a suspiciousness score. However, existing functions don't give good accuracy due to the influence of the fixed parameters. Therefore, a machine learning method is proposed that can automatically construct a suspiciousness function of the specific program set. First, the old versions of a program having fault code are collected. Next, it is mapped from the feature difference in a pair of faulty statement and non-faulty statement to an instance in training dataset. Finally the linear classification algorithm of Weka is applied to learn a mapping function. The function learned from old versions is defined as the fault localization model of the program. To assess the validity of the proposed method, an experiment is performed on three benchmark datasets: Siemens suite, space and gzip. Experimental result demonstrates that the proposed method reduces fault localization cost that exists in SBFL approaches.
Key words: classification algorithm; linear model; fault localization; program spectra; software testing

摘 要:基于谱的错误定位(SBFL)方法能帮助程序员减小软件调试的困难。作为一种轻量方法,SBFL只需收集测试用例的覆盖信息和测试结果,计算程序每条语句的运行特征。众多SBFL方法,将四个运行特征组合成不同的可疑度计算公式。然而,这些公式受固定参数的影响,无法适应不同的程序集。因此,提出一种机器学习方法,能自动确定特定程序集的可疑度计算公式。首先,收集已标注错误语句的程序旧版本;再将错误语句与正确语句的运行特征两两相减,构造为训练集的一个样本;最后基于Weka的分类算法,学习到线性函数,作为该程序的错误定位模型。在Siemens程序包、space和gzip三个基准数据集上,使用Logistic、SGD、SMO和LibLinear学习到的模型,性能都要优于SBFL方法。
关键词:分类算法;线性模型;错误定位;程序谱;软件测试
文献标志码:A **中图分类号:**TP311 doi:10.3778/j.issn.1002-8331.1606-0176

1 引言

软件生命周期的开发阶段和维护阶段都有大量的调试活动。程序员通常利用断点、print语句和断言来调试程序。比较而言,断点方法更为有效,也更受欢迎。执行引起程序故障的测试用例后,程序员先猜测可能出错的语句,设置其为断点,让程序停下来。再利用开发工具,查看内存中的变量值和程序的运行结果。如与预期一致,则判为正确语句,再猜测其他语句;如不符,则判为错误语句。如是反复,直到所有测试用例都通过。当前流行的开发平台,如Visual Studio、MyEclipse等都带有单步执行、变量呈现等调试工具辅助断点方法。在调试过程中,还需要不断分析常量、变量、运算符、函数、语句等之间的依赖关系。如果没有自动化或半自动化的错误定位工具,要想找到错误代码是一件非常困难的

基金项目:湖南省科技计划项目(No.2015GK3071);长沙市科技计划项目(No.K1509011-11)。
作者简介:何海江(1970—),男,副教授,主要从事机器学习、软件测试的研究,E-mail:haijianghe@sohu.com。
收稿日期:2016-06-13 **修回日期:**2016-07-29 **文章编号:**1002-8331(2017)21-0042-07
CNKI**网络优先出版:**2016-11-21, http://www.cnki.net/kcms/detail/11.2127.TP.20161121.2049.086.html

事情,复杂和大型的软件尤为甚。

为解决错误定位难题,人们提出了许多方法,自动搜索导致程序发生故障的代码,将搜索结果呈现出来,由程序员决定如何修改代码。在这些方法中,基于谱的错误定位(Spectrum-Based Fault Localization,SBFL)[1]最受关注,是近年来的研究热点。SBFL方法计算简单,消耗资源少,只要收集测试过程程序实体的覆盖信息和测试用例执行结果,无需额外的处理,故而适应于大规模软件。SBFL方法的程序实体,可以是语句、分支、代码片段,也可以是谓词、函数、类等。不失一般性,本文以语句作为程序实体。每条语句只用少量运行特征,形式上可用四元组 $<a_{ep}, a_{ef}, a_{np}, a_{nf}>$ 表示。a_{ep} 是覆盖该语句,其程序运行结果为成功的测试用例数;a_{ef} 则是覆盖该语句,其程序运行结果为失败的测试用例数;测试用例执行未覆盖该语句时,a_{np} 是其程序运行结果为成功的测试用例数,a_{nf} 则是其程序运行结果为失败的测试用例数。显然,每条语句的四特征数值之和等于程序的测试用例数目。由四元组计算每条语句的可疑度,按照可疑度从大到小排序,排在前面的语句,包含错误代码的概率较大,程序员依此顺序检查代码。人们提出来许多的SBFL方法,将四个特征组合成不同的语句可疑度计算公式,组合方式可看作一个映射函数,将可执行语句的运行特征映射成一个实数值。不过,这些方法存在明显的缺陷:(1)针对所有程序,采用固定的可疑度公式。而特定的程序,受软件规模、测试套件、编程语言、程序员开发水平等因素的影响,可能出错方式与其他程序迥异。(2)若软件开发过程、维护过程还收集到其他有用信息,很难集成到已有公式中。

为克服SBFL的缺点,提出一种基于分类算法的错误定位方法。针对特定的程序,先收集标注了错误语句的程序旧版本;再将错误语句、正确语句两两配对,特征值相减,组合成分类算法训练集的一个样本;最后使用线性模型分类算法,从训练集学习到该程序独有的映射函数。此后,每当定位这个程序的软件错误时,使用它特有的映射函数逐条计算语句的可疑度。另外,算法很容易扩展,集成其他程序特征。增加描述可执行语句的特征数目,四元组扩充为五元组、六元组等等,算法无需改变。

本文第2章介绍了分类算法和SBFL的相关研究工作。第3章举一个例子描述基于谱的软件错误定位方法。第4章介绍了技术路线及实现策略。第5章列出实验结果,证实新方法的有效性。最后一章给出结论。

2 相关研究工作

本文的工作与两方面研究内容相关。

在软件错误定位方法中,基于程序谱的技术最受关注。Jones等人考虑测试用例覆盖信息、多种程序实体等相关因素,提出Tarantula方法[2],取得了不错的效果。与Tarantula相似,人们开发了一系列技术,不断提高软件错误定位能力。Ample方法被集成到Eclipse插件中,用以分析Java软件的错误代码[3]。Abreu等人[1]使用生物学领域的公式Ochiai计算程序实体的可疑度,效果要优于Tarantula、Jaccard等技术。Lucia等人[4]也通过大量实验证实,无论在单个错误程序,还是在多错误程序中定位错误,Ochiai的能力都很突出。有趣的是,他们使用了将近40种数据挖掘和统计领域的关联测度与Tarantula、Ochiai相比。同时,这些实验数据也揭示,40种方法中任何一种都无法在所有程序中胜出。Wong等人也提出许多SBFL方法,其中较有代表性的是Wong1、Wong2和Wong3[5]。特别是Wong3,根据不同的 a_{ep} 个数,给予不同的可疑度计算,公式本质上是一个分梯度的带权计算方法。丁晖等人[6]则依据信息论,用事件的发生概率代替梯度阈值,构造了更平滑的可疑度计算公式。

除了这些方法外,还有非常多的可疑度计算公式,它们也都是将表征程序实体的四个特征组合成不同形式。这些方法的效果也只能在一个有限的程序集中得到验证。由此可见,理论分析必不可少。Naish等人[7]构造了一段具代表性的选择型代码,代码只包含一条错误语句,并且定义了所有可执行序列的平均测度作为SBFL性能的评价指标。在此框架下,比较了30余种SBFL技术的优劣。最后,经过理论证明,NaishO 和NaishOp在该框架下最优。前述框架限制太多,Xie等人[8]讨论了更一般的情况,只要求程序包含单个错误,并且使用 *Exam* 值为评价指标。经理论证明,在30种SBFL方法中,NaishO 和NaishOp等价,Wong1和另两个可疑度计算公式等价,NaishO和Wong1无法直接比较,但NaishO优于其余25种模型。与这些工作使用固定模型不同的是,采取机器学习技术,从程序集旧版本学习错误定位模型,不同的程序集,能训练出特有的模型。

为了提高软件错误定位效率,人们还做了许多其他工作。测试用例能执行到错误位置,但却不会触发错误,排除这些偶然正确的测试用例能够提高SBFL技术的性能[9]。内存或CPU等资源受限,动态调整程序实体的粒度,Java程序可以先在包或类等粗粒度上定位[10]。总结研究人员的大量工作后,陈翔等人定义了缺陷定位研究框架[11],王克朝等人定义了“失效-错误定位-理解”模型[12],各自从多个方面分析了软件错误定位的影响因素、关键问题和研究进展。

分类算法广泛应用于软件缺陷预测、垃圾邮件过滤、文档分类、生物序列识别、网络攻击检测和图像处理[13]等领域。将分类算法,甚至是机器学习、数据挖掘方法应用于软件错误定位的报道还不多见。

决策树[14]、线性回归[15]和马尔科夫链[16]可作为构造错误定位模型的学习算法。但是,这些方法需要额外的

信息。有的将程序输入条件按类别划分,产生大量的抽象测试用例[14];有的针对特定程序,定义为数不少的谓词[15];有的需要收集程序静态结构,以及与程序故障相关的先验知识[16]。基于神经网络的软件错误定位方法倒是仅仅需要语句覆盖信息和测试用例执行结果。但是与SBFL相比,有两个缺点,无论使用反向传播神经网络[17],还是径向基函数神经网络[18],都存在:(1)学习速度慢,很难在线训练;(2)程序每个版本的语句不同,导致神经网络输入层不同,在一个版本上学到的错误定位模型无法应用到其他版本。Ali等人[19]将单个测试用例作为分类算法的一个样本,每条语句被测试用例覆盖的次数定义为样本特征,测试用例成功与否为样本类别。他们的实验表明,分类算法表现很差,计算采用代价敏感的分类算法处理类别不平衡问题,也无法超过Tarantula。本文方法与Ali等人的工作一样,无需收集额外数据,相比其他机器学习技术,并不增加建模负担。与Ali等人工作不同的是,在语句对上建立训练集样本,自然效果也好得多。

3　基于谱的错误定位方法

SBFL方法只需要收集两类信息:(1)测试用例的执行结果,即成功或失败。(2)程序运行时,语句的覆盖程度,最为常见的是布尔值,即语句被覆盖或语句未被覆盖。以图1为例,可理解SBFL的实现步骤。函数middle有三个整型参数,从中找到中间值,作为函数返回值。该函数共有18行代码,其中15行可执行。测试套件有10个测试用例。依据路径覆盖策略,设计了t1~t6六个测试用例。考虑到程序员编写代码时,遇到>、>=、<和<=容易出错,另设计了t7~t10。程序中存在一处错误,在可执行语句4,正确形式为else if(x<z),<被敲成了>。图1的最后一行表示各测试用例的执行结果,P表成功,F表失败,共有7个成功测试用例,3个失败测试用例。黑圆圈表示执行测试用例时语句被覆盖,空白则表示未运行该语句。以t1为例,当x=5,y=3,z=2测试middle时,语句1、8、9、10和15被覆盖,而其他语句不会运行。有了这些信息,容易统计出可执行语句的四个特征。显然,各语句的 $a_{ep}+a_{ef}+a_{np}+a_{nf}=10$, $a_{ep}+a_{np}=7$, $a_{ef}+a_{nf}=3$。

一个当然的想法是:a_{ep} 越大,语句包含错误的可能性越小;a_{ef} 越大,语句包含错误的可能性越大。相比而言,a_{np} 和 a_{nf} 的大小与错误可能性关联程度小一些。但仍然可以推断:a_{np} 越大,语句包含错误的可能性越小;a_{nf} 越小,语句包含错误的可能性越大。SBFL方法就是基于以上朴素想法而提出,构造一个映射函数,将四个特征映射为一个实数值。Jaccard=$a_{ef}/(a_{ep}+a_{ef}+a_{nf})$ 和 Wong1=a_{ef} 已被证实能有效定位错误语句。依照Wong1,语句1、2、4、15是最可能的错误语句,可疑度为3。依照Jaccard,可疑度0.75的语句4最可能出错。依Jaccard,只需检查1条语句,也就是6.67%的代码就可找到错误。在SBFL研究社区,*Exam* 值是最为常见的评估测度,当发现第一条错误语句时,所需检查的代码百分比定义为 *Exam* 值。其他条件相同,*Exam* 值越小,

	int middle (int x,int y,int z)	t1 5,3,2	t2 3,5,2	t3 2,5,3	t4 2,3,5	t5 3,2,5	t6 5,2,3	t7 2,2,3	t8 2,3,3	t9 3,2,3	t10 3,3,3	$a_{ep}, a_{ef}, a_{np}, a_{nf}$	Wong1	Jaccard
	{													
	int r;													
1	if(y<z)	●	●	●	●	●	●	●	●	●	●	7,3,0,0	3	0.3
2	if(x<y)				●	●	●	●		●		2,3,5,0	3	0.6
3	r=y;				●							1,0,6,3	0	0
4	/*bug*/ else if(x>z)					●	●	●		●		1,3,6,0	3	0.75
5	r=x;						●					0,1,7,2	1	0.33
6	else					●		●		●		1,2,6,1	2	0.5
7	r=z;					●		●		●		1,2,6,1	2	0.5
8	else	●	●	●					●		●	5,0,2,3	0	0
9	if(x>y)	●	●	●					●		●	5,0,2,3	0	0
10	r=y;	●										1,0,6,3	0	0
11	else if(x>z)		●	●					●		●	4,0,3,3	0	0
12	r=x;		●									1,0,6,3	0	0
13	else			●					●		●	3,0,4,3	0	0
14	r=z;			●					●		●	3,0,4,3	0	0
15	return r;	●	●	●	●	●	●	●	●	●	●	7,3,0,0	3	0.3
	} Pass/Fail	P	P	P	P	F	F	F	P	P	P			

图1　错误定位技术举例

对应的方法越有效。Jaccard的 $Exam$ 值=6.67%。而依Wong1方法,四条语句的可疑度都处于最高。最理想的情况,语句4最先检查,这种策略被称为Best,也就是Wong1 Best的 $Exam$ 值=6.67%。另一种策略Worst,最坏的情况,语句4最后才被检查,这样需要检查4条语句,也就是26.67%的代码才能发现错误,Wong1 Worst的 $Exam$ 值=26.67%。

当然,Best和Worst都是假设情况,简易可行的策略是SOS(statement order based)[20]。程序员按照语句的顺序检查代码,依次检查语句1和2后,再检查语句4,此时Wong1 SOS的 $Exam$ 值=20%。Wong1强调 a_{ef} 的作用,故语句1、2、4、15是错误语句的可疑度同为最高,而Jaccard综合考虑 a_{nf}、a_{ep}、a_{ef} 的作用,语句4就变得最为可疑。在这个例子中,Jaccard要优于Wong1。但是一旦测试用例发生变化,又或者换做其他程序,Jaccard和Wong1各有千秋。

4 线性错误定位模型

Weka是一个开源平台,实现了众多的机器学习算法和数据挖掘工具[21]。考虑到错误定位模型需要适应大型软件,只选取那些能输出线性模型的分类算法,共找到四个算法完成实验。

要学习程序的错误定位模型,先收集其标注错误代码的旧版本,所有可执行语句都采集了 a_{ep}、a_{ef}、a_{np} 和 a_{nf}。在这些数据上构建训练集,具体策略如下:

训练集生成算法

输入 程序的旧版本集 $\{V_1, V_2, \cdots, V_Q\}$。

输出 训练集 LC_{train}。

```
Begin
  LC_train = ∅;
  index = 0;//保持样本类别的平衡
  For i=1 to Q
/* U_Fault 是 V_i 错误语句的集合, M 是其语句条数。U_OK 是 V_i 正确语句的集合, N 是其语句条数。F(j) 或 F(k) 是第 j (或 k )条语句的特征, j (或 k )=1,2,…, M(N)。*/
    For j=1 to M
      For k=1 to N
        label=true;  //样本的类别
        Sample= F(j)-F(k);
        If index%2==0
/*保证两种类别的样本个数相同*/
          Sample=Sample *(-1);
          label=false;
        End If
      LC_train = LC_train +(Sample, label);
        index = index+1;
      End For
    End For
  End For
End
```

将训练集输入Weka,经分类算法学习后,优化目标位使得 LC_{train} 的误分类个数最少,便可获得线性模型 w。若新版本程序有特征为 F 的语句,其错误可疑度= $w_1F_1+w_2F_2+\cdots+w_DF_D$,$D$ 是特征维数。

令 $L=M\times N$,训练集 LC_{train} 的样本集合{(*Sample*, *label*)}形式化为 $\{(\boldsymbol{x}_i, y_i)\}, i=1,2,\cdots,L, \boldsymbol{x}_i\in R^n, y_i\in\{-1,+1\}$。LibLinear是机器学习社区知名的线性分类算法,用于解决拥有巨量特征和大规模样本的分类问题。LibLinear的SVMType选择1,其余参数全部采用缺省值,算法解决无约束优化问题1[22]:

$$\min_{\boldsymbol{w}}\frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}+C\sum_{i=1}^{L}(\max(1-y_i\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i,0))^2$$

其中T表示矩阵的转置,C 是惩罚因子。

Logistic是线性分类的logistic回归,在一个经转换的目标变量上建立线性模型。Logistic回归是非常经典的分类算法,在众多领域都有成熟的应用,其解决优化问题2[19]:

$$\min_{\boldsymbol{w}}\sum_{i=1}^{L}\ln(1+\exp(-y_i\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i))$$

其中ln表示自然对数,exp表示自然指数。算法采用Weka的缺省参数。

SGD(Stochastic Gradient Descent)能学习不同的线性分类模型,包括两类别的支持向量机、两类别的logistic回归等。而SMO(Sequential Minimal Optimization)只能训练支持向量机的分类模型。两者的优化问题都与LibLinear相同[21],但使用不同策略解决优化问题。LibLinear用了坐标下降法和信赖域牛顿方法,SGD使用随机梯度下降方法,SMO则采用序贯最小优化策略。SGD和SMO用Weka的缺省参数。

5 实验结果及分析

本文的方法与九种SBFL技术的实验比较结果。所有实验在一台配置Intel Core i3-3220 3.3 GHz CPU和4 GB物理内存的计算机上完成,操作系统是Windows Server 2003,调用MinGW5.1的gcc和gcov编译程序和收集覆盖信息。

5.1 实验数据集

Siemens套件、程序集space和gzip很受欢迎,成为软件错误定位领域的基准数据集,许多研究者在论文中使用它说明新算法的有效性。Siemens共有7个程序集:print_tokens、print_tokens2、replac、tcas、tot_info、schedule和schedule2。

原replace共有32个缺陷版本,其v27无失败的测试用例,无法比较各种算法,故而将该缺陷版本排除在外,剩下31个缺陷版本。同样的原因,还去掉了schedule2

的一个版本v9。原space有38个缺陷版本，其v1、v2、v32、v34无失败的测试用例；v25和v30的错误代码是指针问题，测试用例执行后，导致程序崩溃，MinGW的gcov无法记录覆盖信息；将这6个版本排除在外，剩下32个缺陷版本。另外，修改了space的v26和v35，在错误的指针语句前，加了判断指针是否为空的语句，避免程序崩溃。SIR软件项目基础资源库为每个程序集提供了多种测试覆盖类型：bigcov、bigrand、universe、cov和rand等，本文选择universe完成所有实验。表1是所有实验程序的基本信息。

5.2 特征选择及实验验证策略

本文的方法，表示可执行语句的特征个数不受限制。为了与SBFL公平比较。使用了与四元组 $<a_{ep}, a_{ef}, a_{np}, a_{nf}>$ 有关的四个特征：a_{ef}/AF，a_{ep}/AP，$a_{nf}/(AF+a_{ep})$，$a_{np}/(a_{np}+a_{nf})$，其中 $AF=a_{ef}+a_{nf}$，$AP=a_{ep}+a_{np}$。显然，这些特征的值在实数区间[0，1]，当错误语句与正确语句特征值相减，训练集样本的特征值处于[－1，+1]。这样就省去了Weka的数据预处理步骤。

使用十折交叉验证的策略完成所有实验。对每个程序集，将其版本平均分为10份，9份作旧版本集构造训练集，1份作新版本测试集。使用四个分类算法在训练集学习到模型后，计算出测试集的错误定位性能。不足10份的程序集，如print_tokens、schedule和schedule2，部分版本重复填入测试集，但保证一个版本不会同时出现在训练集和测试集。所有试验结果都是10折的平均值。

5.3 实验结果对比

Siemens的实验数据是七个对象的平均值。表2、表3和表4是九种常见SBFL方法和Logistic分别在Siemens、space和gzip上的 *Exam* 值比较。表现最好的数据用黑体字标示，表中 *Exam* 值省略了百分符号。Wong1只与 a_{ef} 有关，在那些只有一条错误语句的程序上，依Best策略，Wong1一定优于其他技术。在Siemens和gzip上，Wong1 Best都显著优于其他技术的Best策略。除此之外，依三种策略Logistic都优于其他方法，特别在程序集space表现出显著优势。Ochiai尽管在Siemens和gzip表现不佳，在space上却仅次于Logistic。Lucia等人[4]在比较40种SBFL技术后，发现Ochiai在许多程序上表现优异。另外，NaishO、NaishOp和Wong3在Siemens和gzip上，Kulczynski2[7]在Siemens上，也都有好的性能。相比而言，其他方法无法显示竞争性。

Weka另外三种算法也都不输给Logistic。调试软件的实际过程中，SOS可用，而Best和Worst不可用，因此后文只比较SOS策略。表5是三种算法与Logistic算

表1 实验程序的基本信息

程序集	缺陷版本数	错误语句数	正确版本程序代码行数	可执行语句数	测试用例数	失败测试用例数
print_tokens	7	1~3	728	194~195	4 130	6~186
print_tokens2	10	1	572	196~200	4 115	33~518
replace	31	1~17	564	241~246	5 542	3~299
schedule	9	1~2	413	148~151	2 650	7~294
schedule2	9	1	375	128~130	2 710	2~65
tcas	41	1~4	174	65~67	1 608	1~131
tot_info	23	1~4	568	121~122	1 052	2~253
space	32	1~8	9 566	3 653~3 660	13 585	7~12 653
gzip	16	1~3	7 364	1 693~1 734	214	1~124

表2 Siemens程序包上的 *Exam* 值比较

	NaishO	NaishOp	Kulczynski2	Wong1	Wong3	Jaccard	Tarantula	Ochiai	SIQ	Logistic
Best	11.52	11.73	12.02	1.87	12.12	19.23	19.85	15.85	15.74	11.39
Worst	18.53	18.17	18.45	53.75	18.55	25.49	26.11	22.28	22.65	17.90
SOS	14.20	14.26	14.54	26.38	14.65	21.70	22.31	18.37	18.67	13.96

表3 程序集space上的 *Exam* 值比较

	NaishO	NaishOp	Kulczynski2	Wong1	Wong3	Jaccard	Tarantula	Ochiai	SIQ	Logistic
Best	3.69	4.14	2.38	3.21	4.11	2.11	3.46	1.40	3.95	1.18
Worst	14.09	4.84	3.08	28.78	4.81	2.80	4.22	2.09	4.63	1.63
SOS	8.85	4.56	2.80	20.97	4.53	2.52	3.89	1.81	4.37	1.41

表4 程序集gzip上的 *Exam* 值比较

	NaishO	NaishOp	Kulczynski2	Wong1	Wong3	Jaccard	Tarantula	Ochiai	SIQ	Logistic
Best	1.84	1.84	3.03	0.10	1.38	3.14	4.48	3.03	3.53	1.58
Worst	5.16	5.16	6.35	14.10	7.09	6.46	12.58	6.35	7.03	5.06
SOS	4.06	4.06	5.24	9.90	4.26	5.35	8.86	5.24	5.91	3.96

表5 三种算法与Logistic算法SOS策略的*Exam*值比较

算法	LibLinear			SMO			SGD		
程序集	Siemens	space	gzip	Siemens	space	gzip	Siemens	space	gzip
Exam±	0.27	−0.01	0	0.15	−0.02	0.85	−0.21	−0.05	0.08

法依SOS策略,在三个数据集上的*Exam*值比较。表中的正值表示对应算法弱于Logistic,负值则强于Logistic。以LibLinear为例,在Siemens上,它比Logistic差,要多检查0.27%的语句,才能找到第一条错误语句;在space上,则略好于Logistic;在gzip上,两者打成平手。综合来看,四种算法中,SGD表现最好,SMO表现最差。

当程序较大,而错误语句的可疑度靠后,程序员检查较多语句后,仍未能定位到错误代码,会产生厌烦情绪。通常,SBFL研究社区会限制检查比例P_{check}(已检查语句数占程序语句总数的比率),计算出$P_{bugVers}$(能发现错误版本数占程序版本总数的比率),来比较各类技术的优劣。图2和图3分别在Siemens和gzip上比较各类方法,P_{check}从1%到20%,步长1%。图中所有技术都依SOS策略。

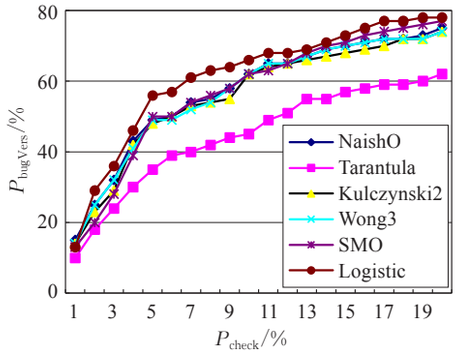

图2 Siemens程序包的$P_{bugVers}$比较

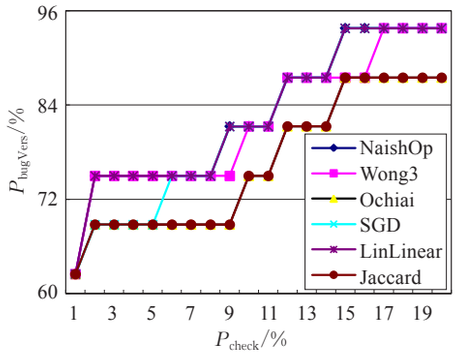

图3 gzip程序集的$P_{bugVers}$比较

如图2所示,在Siemens上,Logistic表现最好,只在$P_{check}=1\%$处略差外,其余点都优于其他方法。Kulczynski2、SMO、NashiO和Wong3差不多,Tarantula则逊色不少。使图清晰起见,未列出另外五种SBFL和两种分类算法的结果。实际上,LibLinear与SMO相似,SGD略好于Logistic,NashiOp与NashiO相同,其余四种SBFL显著落后于SMO。如图3所示,在gzip上,LibLinear和NashiOp表现最好,相比它们,SGD和Wong3略差一点,Jaccard和Ochiai则差了很多。考察未列出的五种SBFL方法和两种分类算法,Logistic和LibLinear完全相同,SMO略差于SGD,NashiO与NashiOp完全相同,其余四种SBFL显著落后于SGD。在space上,检查同样比例的代码,四种分类算法找出错误的版本数目都显著多于九类SBFL方法。总的来看,以$P_{bugVers}$为评价指标,本文的方法无疑最有竞争力。

6 结论

为辅助程序员尽快定位软件的错误代码,人们提出来许多的SBFL方法,利用测试结果,逐条计算语句的错误可疑度。不过,各种SBFL无外乎将四个运行特征组合成不同的映射函数,将可执行语句的运行特征映射成一个实数值。这些映射函数采用固定形式,无法适应程序规模、测试用例分布、程序员开发习惯等迥异的各类情况。如果软件开发过程、维护过程还收集到其他有用信息,如何修改映射函数是一个难题。相比而言,设计了巧妙的训练集生成算法,采用机器学习方法,能从程序旧版本学习到该程序独有的可疑度计算公式,算法也容易集成其他程序特征。

选取了Weka中能输出线性模型的四个分类算法,Logistic、SGD、SMO和LibLinear检验方法的有效性。在Siemens程序包、space和gzip三个基准数据集上,实验结果证实,无论是比较*Exam*值,还是限制P_{check}比较$P_{bugVers}$,它们的性能都要优于SBFL方法。

参考文献:

[1] Abreu R,Zoeteweij P,Golsteijn R,et al.A practical evaluation of spectrum-based fault localization[J].Journal of Systems and Software,2009,82(11):1780-1792.

[2] Jones J A,Harrold M J.Empirical evaluation of the tarantula automatic fault-localization technique[C]//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering.New York:ACM,2005:273-282.

[3] Dallmeier V,Lindig C,Zeller A.Lightweight defect localization for Java[C]//European Conference on Object-oriented Programming,2005,3586:528-550.

[4] Lucia L,Lo D,Jiang Lingxiao,et al.Extended comprehensive study of association measures for fault localization[J].Journal of Software Evolution and Process,2014,26(4):172-219.

[5] Wong W E,Debroy V,Choi B.A family of code coverage-based heuristics for effective fault localization[J].Journal

of Systems and Software,2010,83(2):188-208.
[6] 丁晖,陈林,钱巨,等.一种基于信息量的缺陷定位方法[J].软件学报,2013,24(7):1484-1494.
[7] Naish L,Lee H J,Ramamohanarao K.A model for spectra-based software diagnosis[J].ACM Transactions on Software Engineering and Methodology,2011,20(3):1-32.
[8] Xie Xiaoyuan,Chen T Y,Kuo F,et al.A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization[J].ACM Transactions on Software Engineering and Methodology,2013,22(4):31.
[9] Miao Yi,Chen Zhenyu,Li Sihan,et al.A clustering-based strategy to identify coincidental correctness in fault localization[J].International Journal of Software Engineering and Knowledge Engineering,2013,23(5):721-741.
[10] Perez A,Abreu R,Riboira A.A dynamic code coverage approach to maximize fault localization efficiency[J].Journal of Systems and Software,2014,90:18-28.
[11] 陈翔,鞠小林,文万志,等.基于程序频谱的动态缺陷定位方法研究[J].软件学报,2015,26(2):390-412.
[12] 王克朝,王甜甜,苏小红,等.软件错误自动定位关键科学问题及研究进展[J].计算机学报,2015,38(11):2262-2278.
[13] 罗三定,彭琼,李婷.瓷砖图像的纹理特征分类研究[J].计算机工程与应用,2016,52(8):196-200.
[14] Briand L C,Labiche Y,Liu Xuetao.Using machine learning to support debugging with tarantula[C]//Proceedings of the 18th IEEE International Symposium on Software Reliability.Piscataway,NJ:IEEE,2007:137-146.
[15] Gore R,Reynolds P F.Reducing confounding bias in predicate-level statistical debugging metrics[C]//Proceedings of the 2012 International Conference on Software Engineering.Piscataway,NJ:IEEE,2012:463-473.
[16] Zhang Sai,Zhang C.Softwre bug localization with Markov logic[C]//Proceedings of the 36th International Conference on Software Engineering NIER Track.Piscataway,NJ:IEEE,2014.
[17] Wong W E,Qi Yu.BP neural network-based effective fault localization[J].International Journal of Software Engineering and Knowledge Engineering,2009,19(4):573-597.
[18] Wong W E,Debroy V,Golden R,et al.Effective software fault localization using an RBF neural network[J].IEEE Transactions on Reliability,2012,61(1):149-169.
[19] Ali S,Andrews J A,Dhandapani T,et al.Evaluating the accuracy of fault localization techniques[C]//IEEE/ACM International Conference on Automated Software Engineering.Piscataway,NJ:IEEE,2009:76-87.
[20] Xu Xiaofeng,Debroy V,Wong W E,et al.Ties within fault localization rankings:Exposing and addressing the problem[J].International Journal of Software Engineering and Knowledge Engineering,2011,21(6):803-827.
[21] Hall M,Frank E,Holmes G,et al.The WEKA data mining software:An update[J].SIGKDD Explorations,2009,11(1).
[22] Fan R E,Chang K W,Hsieh C J,et al.LIBLINEAR:A library for large linear classification[J].Journal of Machine Learning Research,2008,9(9):1871-1874.

(上接36页)

[4] Fan Jianping,Shen Yi,Yang Chunlei,et al.Structured max-margin learning for inter-related classifier training and multilabel image annotation[J].IEEE Transactions on Image Processing,2011,20(3):837-854.
[5] 毛罕平,张艳诚,胡波.基于模糊C均值聚类的作物病害叶片图像分割方法研究[J].农业工程学报,2008,24(9):136-140.
[6] Omrani E,Khoshnevisan B,Shamshirband S,et al.Potential of radial basis function based support vector regression for apple disease detection[J].Measurement,2014,55:512-519.
[7] Dubey S R,Jalal A S.Fusing color and texture cues to identify the fruit diseases using images[J].International Journal of Computer Vision and Image Processing,2014,4(2):52-67.
[8] 秦丰,刘东霞,孙炳达,等.基于图像处理技术的四种苜蓿叶部病害的识别[J].中国农业大学学报,2016,21(10):65-75.
[9] Chen E K,Yang X K,Zha H Y,et al.Learning object classes from image thumbnails through deep neural networks[C]//International Conference on Acoustics,Speech and Signal Processing.Las Vegas,Nevada,USA:IEEE,2008:829-832.
[10] Collobert R,Sinz F,Weston J,et al.Large scale transductive SVMs[J].Journal of Machine Learning Research,2006,7:1687-1712.
[11] Friedman J H,Hastie T,Tibshirani R.Additive logistic statistical view of boosting[J].Annals of Statistics,2000,28:337-407.
[12] Mitchell T M.Machine learning[M].Columbus OH,USA:McGraw-Hill Science/Engineering/Math,1997.
[13] Bengio Y,Lamblin P,Popovici D,et al.Greedy layer-wise training of deep networks[C]//International Conference on Neural Information Processing Systems.Canada:NIPS Foundation,2006:153-160.
[14] Weston J,Ratle F,Collobert R.Deep learning via semi-supervised embedding[C]//International Conference on Machine Learning.Helsinki,Finland:ACM,2008:1168-1175.
[15] Girshick R,Donahue J,Darrell T,et al.Region-based convolutional networks for accurate object detection and segmentation[J].IEEE Transactions on Pattern Analysis & Machine Intelligence,2016,38(1):142-158.