doi: 10.3969/j.issn.0490-6756.2017.04.011

由分类算法学习软件错误定位的关联测度

张志宏，何海江，刘华富
(长沙学院计算机工程与应用数学学院，长沙 410022)

摘　要：基于谱的错误定位(SBFL)技术能找出导致程序出错的可执行代码.测试用例数目和覆盖语句次数可构造 SBFL 的二分型矩阵.利用该矩阵,人们提出许多的 SBFL 关联测度计算公式.然而,这些关联测度往往只适应部分程序集.因此,提出基于分类算法的技术,能学习到程序集特有的关联测度.训练集样本建立在成对的错误语句和正确语句上,其特征由语句对的条件概率相减而成.为证实技术的有效性,在 Siemens 套件、space 和 gzip 三个基准数据集上完成实验.使用 Weka 的 Logistic、SGD、SMO 和 LibLinear 训练出的关联测度,性能都明显优于固定形式的 SBFL 测度.

关键词：分类算法；特征；错误定位；程序谱；关联测度；软件测试
中图分类号：TP31.5　　**文献标识码**：A　　**文章编号**：0490-6756(2017)04-0728-07

Learning association measures of software fault localization with classification algorithms

ZHANG Zhi-Hong, *HE Hai-Jiang*, *LIU Hua-Fu*
(College of Computer Engineering and Applied Mathematics, Changsha University, Changsha 410022, China)

Abstract: Spectrum-based fault localization (SBFL) techniques aim at identifying the executing programs codes that correlate with failure. A dichotomy matrix for SBFL records the bivariate frequency distribution of the test case results and the program element hit numbers. Given the matrix, many SBFL association measures are proposed to compute suspiciousness scores of the program elements. Research shows that any association measure can't be statistically better than other measures when localizing buggy program. Therefore, a technique based the classification algorithm is proposed which can automatically learn the specific measure for a program set. A sample in training dataset is constructed by employing a pair of faulty statement and non-faulty statement ones, and its features are the probability features difference of two statements. It is evaluated with three benchmark datasets: Siemens suite, space and gzip. Experimental result indicates that the learned measures with LibLinear, Logistic, SGD and SMO of Weka outperformed existing SBFL association measures.

Keywords: Classification algorithm; Feature; Fault localization; Program spectra; Association measure; Software testing

收稿日期：2016-11-21
基金项目：国家自然科学基金资助项目(61379117)；湖南省科技计划项目(2015GK3071)；湖南省教育厅科学研究项目(15B026)；长沙市科技计划项目(ZD1601034)；长沙学院人才引进科研项目(SF1404)
作者简介：张志宏(1985－)，男，湖南长沙人，讲师，博士，研究方向为软件测试和图像处理. E-mail:zhzhang@ccsu.edu.cn
通讯作者：何海江. E-mail:haijianghe@sohu.com.

1 引　言

近年来，软件开发、软件测试领域产生了许多的革新技术和成熟工具，显著提升了软件质量. 尽管如此，市场上大量软件仍然包含错误代码. 要追查引起软件发生故障的根源，程序员在调试过程中需要付出艰辛的劳动，而自动化或半自动化定位软件错误代码能减轻工程师们的调试负担.

在该项自动化技术中，研究人员最为关注基于谱的软件错误定位(Spectrum-Based Fault Localization，SBFL)技术[1]. 作为一种轻量方法，SBFL 只需收集每个测试用例的覆盖信息和测试结果. 如表 1 所示，当测试任务完成后，SBFL 统计每条可执行语句的运行特征. 形式上，该特征集可用四元组$<a_{ep},a_{ef},a_{np},a_{nf}>$表示. 对程序的每一条语句，$a_{ep}$表示测试用例覆盖它并且程序运行结果成功的数目；$a_{ef}$则是覆盖它而结果失败的测试用例数目；$a_{np}$是测试用例未覆盖它并且程序运行结果成功的个数；$a_{nf}$则是测试用例未覆盖它而结果失败的个数. SBFL 由四个特征值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. 当然，除语句外，程序切片、谓词、分支、函数、代码片段、类和类的方法等程序实体，都可使用 SBFL 定位其错误.

表 1　SBFL 的二元型矩阵

Tab. 1　Dichotomy matrix for SBFL

	覆盖该语句	未覆盖语句
测试结果成功	a_{ep}	a_{np}
测试结果失败	a_{ef}	a_{nf}

以二元型矩阵为基础，人们提出许多的 SBFL 方法，包括 Tarantula[2]、Ample[3]、Ochiai[1]、Jaccard[4]、Wang3[5]、NaishOp[6]等，都取得不错的效果. Lucia 等人[7]站在统计视角，将 SBFL 方法的可疑度计算公式归纳为关联测度，并通过大量实验比较了 40 余个测度. 这些关联测度对所有程序集都采用固定的公式，性能有起伏，没有一种测度稳定地超过其他测度. 我们从中受到启发，采用分类算法，对每一个程序集，从旧版本学习特有的关联测度，再应用于新版本的错误定位. 相较这些实证研究，理论分析也有必要. Naish 等人[6]和 Xie 等人[8]在各自的框架下分析了业界多种代表性的 SBFL 方法，两者的分析都限制程序只有一条错误语句. 王克朝等人定义了“失效-错误定位-理解”模型[9]，分析了软件错误定位的关键问题和研究进展.

机器学习[10]、数据挖掘技术[11]在软件工程领域得到广泛应用. 然而，将之运用于错误定位则较少报道. 决策树[12]、线性回归[13]、反向传播神经网络[14]、径向基函数神经网络[15]、马尔科夫链[16]和遗传算法[17]都可作为构造错误定位模型的学习算法. 但是，这些方法要么还需额外的程序信息，要么在线训练速度慢，无法适应大规模软件. 与这些方法不同的是，同样采用机器学习技术，本文技术无需采集四元组外的特征数据，在程序集旧版本学习到关联测度后，就固定下来，当定位程序集新版本的错误时，不用后续的训练工作，自然适应大规模软件.

构造训练集时，我们将错误语句、正确语句两两配对，语句特征值相减，组合成分类算法监督学习集的一个样本. 使用机器学习领域知名的 Weka 平台[18]，选择能输出线性模型的分类算法，从训练集学习到程序集特定的关联测度. 在三个基准数据集上的实验结果证实，本文的方法明显优于固定公式的 SBFL 关联测度.

2 软件错误定位的关联测度

表 1 二元型矩阵的频率模式可转换成如表 2 所示的概率模式. 表中的 $TCs=a_{ep}+a_{ef}+a_{np}+a_{nf}$，也就是测试用例总数目. Lucia 等人[7]将 40 余个关联测度规约为概率表达式.

表 2　关联测度的概率

Tab. 2　Probabilities of association measures

一元概率	二元概率	条件概率
$P(A)=\frac{a_{ef}+a_{ep}}{TCs}$	$P(A,B)=\frac{a_{ef}}{TCs}$	$P(B\mid A)=\frac{P(A,B)}{P(A)}$
$P(\bar{A})=\frac{a_{nf}+a_{np}}{TCs}$	$P(\bar{A},B)=\frac{a_{nf}}{TCs}$	$P(A\mid B)=\frac{P(A,B)}{P(B)}$
$P(B)=\frac{a_{ef}+a_{nf}}{TCs}$	$P(A,\bar{B})=\frac{a_{ep}}{TCs}$	—
$P(\bar{B})=\frac{a_{ep}+a_{np}}{TCs}$	$P(\bar{A},\bar{B})=\frac{a_{np}}{TCs}$	—

Exam 值是业界最为常见的评估测度，当发现第一条错误语句时(程序可能包含多条错误语句)，所需检查代码占总代码数的百分比定义为 *Exam* 值. 一般情况下，*Exam* 值越小，对应的方法越有效. 在文献[7]的实验，若依所有程序集的平均 *Exam* 值，Klosgen、Ochiai、Collective Strength、Φ-Co-

efficient 和 Added Value 优于其他测度;而在多错误程序集上,Added Value、One-Way Support、Collective Strength、Ochiai、Φ-Coefficient、Odd Multiplier 和 Tarantula 优于其他测度. 表 3 列出了这些性能最为优异的关联测度及其概率形式.

表 3 定义关联测度

Tab. 3 Definitions of association measures

关联测度名	概率形式
Φ-Coefficient(M1)	$\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
Added Value(M15)	$\max(P(B\mid A)-P(B),P(A\mid B)-P(A))$
Collective Strength(M16)	$\frac{P(A,B)+P(\overline{A},\overline{B})}{P(A)P(B)+P(\overline{A})P(\overline{B})}\times\frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{A},\overline{B})}$
Klosgen(M18)	$\sqrt{P(A,B)}\max(P(B\mid A)-P(B),P(A\mid B)-P(A))$
One-Way Support(M27)	$P(B\mid A)\log_2\frac{P(A,B)}{P(A)P(B)}$
Odd Multiplier(M33)	$\frac{P(A,B)P(\overline{B})}{P(B)P(A\overline{B})}$
Tarantula	$\frac{\frac{P(A,B)}{P(B)}}{\frac{P(A,\overline{B})}{P(\overline{B})}+\frac{P(A,B)}{P(B)}}$
Ochiai	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$

3 基于分类算法的关联测度

四元组大概隐含着一种未知的特质,在多维空间计算它们指示的语句可疑度,能帮助程序员快速定位错误. 表 3 的各种关联测度只是简单地组合少数几维特征空间,在所有程序集使用同样的组合形式,便都取得不错的效果. 这也启发我们,采用机器学习技术,或许能学习到与特定程序集契合的关联测度.

要学习程序集的测度,先收集其标注错误代码的 Q 个旧版本 $V_1,V_2,\cdots,V_Q$. 对第 q 个版本来说,$U_{q,OK}$ 表示正确语句的特征集合,$U_{q,Fault}$ 表示错误语句的特征集合,M_q 和 N_q 分别是两个集合的语句条数. 语句的特征可形式化为 $F=(F_1,F_2,\cdots,F_D)$,D 是特征维数. 考虑到方法需要适应大规模软件,只选取线性模型充作关联测度,各个特征的权重 $w=(w_1,w_2,\cdots,w_D)$. 也就是说,有特征为 F 的语句时,依据学习到的测度,其错误可疑度等于 $wF=w_1F_1+w_2F_2+\cdots+w_DF_D$.

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[18]. 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.

LibLinear 是机器学习社区知名的线性分类算法[19],长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 Weka 参数全部采用缺省值. 本文使用 LibLinear 时,解决无约束优化问题(1).

$$\min_w \frac{1}{2}w^Tw+C\sum(\max(1-w^T(F_{q,i}-F_{q,j}),0))^2$$
$$F_{q,i}\in U_{q,Fault},\ F_{q,j}\in U_{q,OK}\ i=1,2,\cdots,N_q,\ j=1,2,\cdots,M_q,\ q=1,2,\cdots,Q \quad (1)$$

其中,T 表示矩阵的转置;C 是惩罚因子. 用以学习到程序集特定的权重 w,则 wF 构成该程序集的关联测度.

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. 本文使用 Logistic 时,解决优化问题(2).

$$\min_w\sum\ln(1+\exp(-w^T(F_{q,i}-F_{q,j})))$$
$$F_{q,i}\in U_{q,Fault},F_{q,j}\in U_{q,OK}\ i=1,2,\cdots,N_q,\ j=1,2,\cdots,M_q,\ q=1,2,\cdots,Q \quad (2)$$

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. 同样地,解决优化问题(2)后,wF 构成该程序集的关联测度.

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[20]、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. 本文方法中,SGD 和 SMO 的无约束优化问题都与 LibLinear 相同,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 都用 Weka 的缺省参数.

4 实验结果及讨论

为验证本文方法的有效性,在三个基准数据集上比较它与表 3 八种关联测度的性能. 实验在台式机和笔记本完成,其中笔记本配置 Intel Core i5-4200U 2.3 GHz CPU 和 8 GB 物理内存,操作系统是 Windows Server 2003,运行 Weka 的分类算

法.台式机完成其他实验，配置 Intel Core i3-3220 3.3 GHz CPU 和 4 GB 物理内存，操作系统 Windows Server 2003，调用 MinGW5.1 的 gcc 和 gcov 编译程序和收集覆盖信息.

4.1　实验数据集

Siemens 套件很受欢迎，成为软件错误定位领域的基准数据集. 共 7 个程序集：schedule 和 schedule2 用于优先级调度，print_tokens 和 print_tokens2 都是词法分析器，replace 执行模式匹配及替换，而 tcas 是一个避免高空冲突的高程分隔系统，tot_info 计算给定数据的统计信息测度. 另外，我们还用程序 space 和 gzip 作为实验程序集，比较各种方法的优劣. 这些数据集均从 SIR 软件项目基础资源库（http://sir.unl.edu/portal/index.html）下载.

4.2　特征选择

所有关联测度是表 2 概率项的组合，理所当然，可考虑将这些概率项充作特征. 表 2 的第一列有 4 个概率项，后两个是常数，错误语句和正确语句的概率项相减后，训练样本特征为 0，显然不能作为特征候选. 类似的道理，第一列前两个也排除在外. 中间一列的四个概率项为特征集，也只是四元组的线性组合，很难和表 3 的关联测度竞争. 这促使我们考虑使用最后一列的条件概率项作为特征. 如表 4 所示，条件概率项可扩展到 8 个.

表 4　条件概率的定义

Tab.4　Definitions of condition probabilities

条件概率 1 组	条件概率 2 组
1 $P(A\mid B)=\frac{P(A,B)}{P(B)}$	5 $P(\overline{A}\mid B)=\frac{P(\overline{A},B)}{P(B)}$
2 $P(A\mid \overline{B})=\frac{P(A,\overline{B})}{P(\overline{B})}$	6 $P(\overline{A}\mid \overline{B})=\frac{P(\overline{A},\overline{B})}{P(\overline{B})}$
3 $P(\overline{B}\mid \overline{A})=\frac{P(\overline{A},\overline{B})}{P(\overline{A})}$	7 $P(B\mid \overline{A})=\frac{P(\overline{A},B)}{P(\overline{A})}$
4 $P(B\mid A)=\frac{P(A,B)}{P(A)}$	8 $P(\overline{B}\mid A)=\frac{P(A,\overline{B})}{P(A)}$

表 4 每一行相加后等于 1，而关联测度是线性模型，依据机器学习的原理，只用一列为特征集. 我们考虑使用第一列（也就是项 1 到项 4）作为特征集，命名为特征集 a. 实验过程中，发现项 4 对算法略有干扰，只保留项 1 到项 3，称为特征集 b. 此时，条件概率项的分子中，四元组独缺 a_{nf}，我们在特征集 b 加入 $P(\overline{A},B)$，构成特征集 c.

一般来说，a_{ep} 越大，在成功测试用例的百分比越高，语句包含错误的可能性越小；a_{ef} 越大，在错误测试用例的百分比越高，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. 因此，项 1 和项 2 是最重要的特征.

4.3　实验验证策略

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关联测度后，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. 在训练过程，最优的惩罚因子 C 使得错误分类样本最少，而非 *Exam* 值最小.

当两条语句的错误可疑度相同时，谁先检查，简易可行的策略是 SOS（Statement Order Based）[21]，语句序号在前的先检查，不失为调试过程的简易合理选择. 所有试验结果都依据 SOS 策略.

4.4　*Exam* 值的比较

选择特征集 b 完成实验，表 5 是 8 个关联测度和 Logistic 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. 所有 Siemens 套件的实验数据都是 7 个程序集的平均值，后文不再赘述.

表 5　三个基准数据集的 *Exam* 值比较

Tab.5　*Exam* comparison for three benchmark datasets

Dataset	M1	M15	M16	M18	M27	M33	Tarantula	Ochiai	Logistic
Siemens	20.48	14.3	20.36	14.25	22.34	22.31	22.31	18.57	14.18
space	1.37	3.97	1.59	4.56	2.93	2.9	2.93	1.16	1.37
gzip	5.29	7.89	5.29	4.06	8.39	6.59	8.86	5.24	4.81

表 6 特征集 b 上 Logistic 与三种分类算法的 *Exam* 值比较

Tab. 6 *Exam* comparison for Logistic and three classification algorithms with feature set b

分类算法	LibLinear			SMO			SGD		
程序集	Siemens	space	gzip	Siemens	space	gzip	Siemens	space	gzip
Exam±	0.11	0.03	−0.01	−0.24	0.02	0.02	−0.44	0.03	0.02

从表 5 可看出,Logistic 在 Siemens 表现最好,在 space 和 gzip 也与最好 *Exam* 值接近. Φ-Coefficient、Strength 和 Ochiai 只在 space 表现好,Klosgen 在 Siemens 和 gzip 都好,而其他关联测度无法表现竞争力. 综合来看,Logistic 的 *Exam* 值明显优于表 3 的所有关联测度. Weka 另外 3 种算法也都不输给 Logistic. 表 6 是 4 种算法依 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 0.11%的语句,才能找到第一条错误语句;在 space 上,略逊于 Logistic;在 gzip 上,则略好于 Logistic. 综合来看,4 种算法中,SGD 表现最好,LibLinear 表现最差. 优化问题相同,只是优化策略不同,结果有较小的偏差.

特征集 a 的实验数据差于特征集 b. 以 Logistic 为例,在 Siemens、space 和 gzip 上,特征集 a 的 *Exam* 值分别高 2.16、0.16 和 0.72. 其他 3 个算法的情况类似. 相比特征集 b,选择特征集 c 后,4 个算法的表现迥异. 如表 7 所示,从算法的角度看,LibLinear 的性能提高了,另外 3 个算法则有变好变差;从程序集的角度看,在 space 上没有变化,另两个则有变好变差. 特别是 Logistic 在 Siemens 和 gzip 的表现,特征集不同导致 *Exam* 值出现显著的差异. 对比 Logistic 和 LibLinear 在 gzip 的表现,发现在第 3 折相差特别大,前者 45.75,后者 0.35,其他各折则完全相同. 在第 3 折,Logistic 学习到的关联测度是:

$$M_{\text{Logistic}} = 107.34 * P(A|B) + 13.28 * P(\overline{B}|\overline{A}) - 1 * P(A|\overline{B}) + 199.4 * P(\overline{A},B) \quad (3)$$

而 LibLinear 学习到的关联测度是:

$$M_{\text{LibLinear}} = 4.47 * P(A|B) + 10.63 * P(\overline{B}|\overline{A}) - 1 * P(A|\overline{B}) + 17.55 * P(\overline{A},B) \quad (4)$$

对 print_tokens、print_tokens2 和 schedule 来说,Logistic 和 LibLinear 的性能差别比较大,而 Siemens 的其他 4 个程序集 *Exam* 值差别则不大.

表 7 特征集 c 四种分类算法的 *Exam* 值比较

Tab. 7 *Exam* comparison for four classification algorithms with feature set c

Dataset	Logistic	LibLinear	SMO	SGD
Siemens	12.02	14.29	15.4	15.77
gzip	8.48	3.94	4.14	3.94
space	1.39	1.38	1.39	1.4

总体来看,特征集的选择,分类算法的不同,都对错误定位性能造成影响.

4.5 $P_{bugVers}$ 的比较

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 1、图 2 和图 3 分别在三个基准数据集上比较分类算法和表现最好的三个关联测度,P_{check} 从 1%到 20%,步长 1%. 图中所有技术都依 SOS 策略,实验基于特征集 b. 如图 1 所示,在 Siemens 上,SGD 表现最好,只在 P_{check}=1%、4%、9%和 11%处略低于 M15 和 M18 外,其余点 SGD 保持最佳,特别是从 13%开始,明显领先于 M15 和 M18. Ochiai 则较大幅度落后于另外三个关联测度.

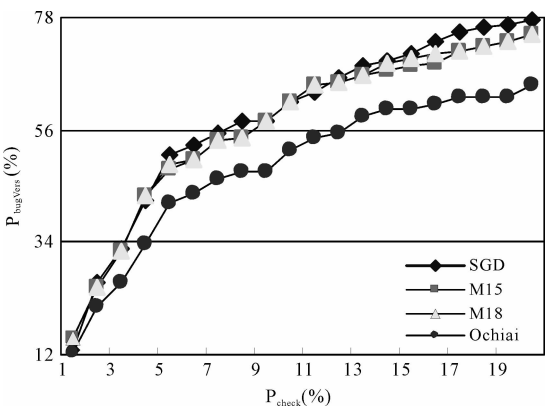

图 1 Siemens 的 $P_{bugVers}$ 比较

Fig. 1 $P_{bugVers}$ comparison on Siemens

如图 2 所示,在 space,P_{check} 从 1%到 8%,Ochiai 最优,其次是 Logistic 和 M16,M1 最差. 从

9%开始，这 4 个关联测度的 $P_{bugVers}$ 几乎完全相同，Logistic 率先在 $P_{check}=11\%$ 处达到 100%，从 12% 开始，四个关联测度的 $P_{bugVers}$ 都等于 100%，说明在 space 上使用关联测度定位软件错误的效果很不错.

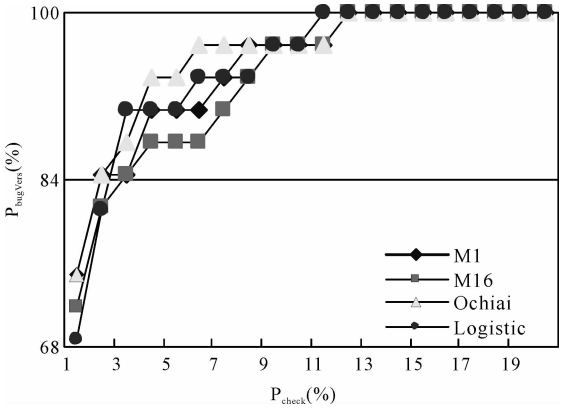

图 2　space 的 $P_{bugVers}$ 比较

Fig. 2　$P_{bugVers}$ comparison on space

如图 3 所示，在 gzip 上，P_{check} 从 1% 到 8%，M18 明显好于另三个方法. 从 19% 开始，LibLinear 和 M18 并驾齐驱，相比 M1 和 Ochiai，其 $P_{bugVers}$ 有超过 6%的优势. 总体来看，以 $P_{bugVers}$ 为评价指标，我们的方法无疑最有竞争力.

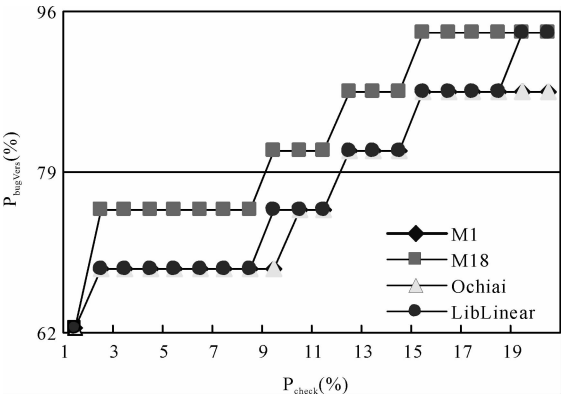

图 3　gzip 的 $P_{bugVers}$ 比较

Fig. 3　$P_{bugVers}$ comparison on gzip

5　结　论

本文提出一种基于分类算法和程序频谱的软件错误定位方法，以三个条件概率为语句特征，学习程序集的特定关联测度. 在三个基准数据集上，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固定形式的关联测度. 分类算法的训练过程，特征集可变化，理所当然，关联测度可加入程序规模、测试用例分布、程序员开发习惯等迥异的各类特征数据. 如果软件开发过程、维护过程还收集到其他有用信息，也能修改关联测度. 特征的选择，分类算法的选择，都对软件错误定位的性能产生偏差. 以后我们将在这两方面展开进一步的研究. 另外，其他分类算法[22]是否同样有效，也值得研究.

参考文献：

[1]　Abreu R, Zoeteweij P, Golsteijn R, *et al*. A practical evaluation of spectrum-based fault localization [J]. J Syst Software, 2009, 82: 1780.

[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.

[3]　Dallmeier V, Lindig C, Zeller A. Lightweight defect localization for Java [J]. Lect Notes Comput Sci, 2005, 3586: 528.

[4]　Chen M Y, Kiciman E, Fratkin E, *et al*. Pinpoint: problem determination in large, dynamic Internet services [C] // Proceedings of the 2002 International Conference on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2002.

[5]　Wong W E, Debroy V, Choi B. A family of code coverage-based heuristics for effective fault localization [J]. J Syst Software, 2010, 83: 188.

[6]　Naish L, Lee H J, Ramamohanarao K. A model for spectra-based software diagnosis [J]. ACM Trans Softw Eng Meth, 2011, 20: 563.

[7]　Lucia, Lo D, Jiang L X, *et al*. Extended comprehensive study of association measures for fault localization [J]. J Softw-Evol Proc, 2014, 26:172.

[8]　Xie X Y, Chen T Y, Kuo F, *et al*. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization [J]. ACM Trans Softw Eng Meth, 2013, 22: 31.

[9]　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，2015，38：2262.

[10]　Calumby R T, Goncalves M A, Torres R S. On interactive learning-to-rank for IR: overview, recent advances, challenges, and directions [J]. Neurocomputing, 2016, 208: 3.

[11]　梁珺，刘云. 基于析取规则对不确定数据挖掘的优化研究[J]. 四川大学学报：自然科学版，2016，53：788.

[12] Briand L C, Labiche Y, Liu X T. Using machine learning to support debugging with tarantula[C] // Proceedings of the 18th IEEE International Symposium on Software Reliability. Piscataway, NJ: IEEE, 2007.

[13] Gore R, Reynolds P F J. Reducing confounding bias in predicate-level statistical debugging metrics[C] // International Conference on Software Engineering. Piscataway, NJ: IEEE, 2012.

[14] Wong W E, Qi Y. BP neural network-based effective fault localization [J]. Int J Softw Eng Know, 2009, 19: 573.

[15] Wong W E, Debroy V, Golden R, *et al*. Effective software fault localization using an RBF neural network [J]. IEEE Trans Reliab, 2012, 61: 149.

[16] Zhang S, Zhang C L. Software bug localization with Markov logic [C] // Proceedings of the 36th International Conference on Software Engineering NIER track. Piscataway, NJ: ACM, 2014.

[17] 王赞, 樊向宇, 邹雨果, 等. 一种基于遗传算法的多缺陷定位方法[J]. 软件学报, 2016, 27: 879.

[18] Hall M, Frank E, Holmes G, *et al*. The WEKA data mining software: an update [J]. SIGKDD Explorat, 2009, 11: 10.

[19] Fan R E, Chang K W, Hsieh C J, *et al*. LIBLINEAR: A library for large linear classification [J]. J Mach Learn Res, 2008, 9: 1871.

[20] 王岩, 张波, 薛博. 基于 FOA-SVM 的中文文本分类方法研究[J]. 四川大学学报: 自然科学版, 2016, 53: 759.

[21] Xu X F, Debroy V, Wong W E, *et al*. Ties within fault localization rankings: exposing and addressing the problem [J]. Int J Softw Eng Know, 2012, 21: 803.

[22] 王静, 杜勇, 赵忠华. 神经网络算法在特色农产品品质分类中的应用[J]. 四川大学学报: 自然科学版, 2016, 53: 805.