软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(11):2499−2517 [doi: 10.13328/j.cnki.jos.004559]

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

面向细粒度源代码变更的缺陷预测方法*

原 子[1], 于莉莉[1,2], 刘 超[1]

[1](北京航空航天大学 计算机学院,北京 100191)

[2](第二炮兵 软件测评中心,北京 100191)

通讯作者: 原子, E-mail: yuanzi@sei.buaa.edu.cn, http://www.sei.buaa.edu.cn

摘 要: 软件在其生命周期中不断地发生变更,以适应需求和环境的变化.为了及时预测每次变更是否引入了缺陷,研究者们提出了面向软件源代码变更的缺陷预测方法.然而现有方法存在以下 3 点不足:(1) 仅实现了较粗粒度(事务级和源文件级变更)的预测;(2) 仅采用向量空间模型表征变更,没有充分挖掘蕴藏在软件库中的程序结构、自然语言语义以及历史等信息;(3) 仅探讨较短时间范围内的预测,未考虑在长时间软件演化过程中由于新需求或人员重组等外界因素所带来的概念漂移问题.针对现有的不足,提出一种面向源代码变更的缺陷预测方法.该方法将细粒度(语句级)变更作为预测对象,从而有效降低了质量保证成本;采用程序静态分析和自然语言语义主题推断相结合的技术深入挖掘软件库,从变更的上下文、内容、时间以及人员 4 个方面构建特征集,从而揭示了变更易于引入缺陷的因素;采用特征熵差值矩阵分析了软件演化过程中概念漂移问题的特点,并通过一种伴随概念回顾的动态窗口学习机制实现了长时间的稳定预测.通过 6 个著名开源软件验证了该方法的有效性.

关键词: 缺陷预测;软件演化;细粒度变更;概念漂移;成本有效性

中图法分类号: TP311

中文引用格式: 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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/4559.htm

Bug Prediction Method for Fine-Grained Source Code Changes

YUAN Zi[1], YU Li-Li[1,2], LIU Chao[1]

[1](School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

[2](Software Testing Center, Second Artillery, Beijing 100191, China)

Corresponding author: YUAN Zi, E-mail: yuanzi@sei.buaa.edu.cn, http://www.sei.buaa.edu.cn

Abstract: Software changes constantly in its lifecycle to adapt to the changing requirements and environments. In order to predict whether each change will introduce any bugs timely, various bug prediction methods for software source code changes have been proposed by researchers. However, there are three deficiencies in existing methods: 1) The prediction granularities are limited at the coarse-grained levels (i.e. transaction or file levels); 2) As vector space model is used to represent software changes, abundant information in software repositories, such as program structure, natural language semantic and history information, can not be mined sufficiently; 3) Only short-time prediction is explored without considering the concept drift caused by new requirements, team restructuring or other external factors during the long time software evolution process. In order to overcome the shortcomings of existing methods, a bug prediction method for source code changes is proposed. It makes prediction for fine-grained (i.e. statement level) changes, which reduces the quality assurance cost effectively. By in-depth mining software repositories with static program analysis and natural language semantic inference technologies, feature sets of changes are constructed in four aspects (i.e. context, content, time, and developer) and key

* 基金项目: 国家自然科学基金(90718018); 国家高技术研究发展计划(863)(2007AA010302)
收稿时间: 2013-10-28; 定稿时间: 2013-12-26

factors that lead to bug injection are revealed. Characteristics of concept drift in software evolution process are analyzed by using matrix of feature entropy difference, and an algorithm of adaptive window with concept reviewing is proposed to achieve stability of long-time prediction. Experiments on six famous open source projects demonstrate effectiveness of the proposed method.

Key words: bug prediction; software evolution; fine-grained change; concept drift; cost effectiveness

随着软件规模和复杂度的不断增加,对于软件开发人员而言,在有限的时间资源制约下,高质量地完成软件变更并非易事.每次变更都存在引入缺陷的风险.被引入的缺陷如果不能被及时发现并修复,可能会给软件系统带来 3 个方面的负面影响:首先,长期寄居于软件系统中的缺陷可能引发新的缺陷和后续一系列错误的变更[1];其次,与修复那些及时被发现的缺陷相比,修复长期寄居的缺陷通常需耗费较高的人力和时间成本[2];最后,如果这些寄居的缺陷在软件发布后被用户发现并报告,会显著降低软件声誉和用户满意度[3].

为了帮助软件人员利用有限的质量保证资源,及时、有效地发现并修复被引入的缺陷,软件缺陷预测领域的研究者提出了面向软件源代码变更的缺陷预测方法[2,4].与传统的面向软件源代码实体(子系统、源文件等)的缺陷预测方法[5−8]不同,该方法将变更作为学习和预测对象,它能够在变更刚刚完成时预测本次变更是否引入了缺陷,从而实现预测的及时性;它能够引导软件人员将更多的质量保证资源分配到引入缺陷的变更所在区域,而不是整个软件实体,从而降低测试和审查的工作量[2].

现有的面向变更的缺陷预测方法通常将事务级别或者文件级别的源代码变更作为预测对象,然而一次事务或文件级别的变更通常包含针对多个语句的修改,即便在该级别预测出了引入缺陷的变更,也需要逐一审查变更中所有发生修改的语句,才能最终定位并修复缺陷.现有方法采用简单的空间向量模型表征变更,忽略了软件库中所包含的程序结构信息、自然语言语义信息以及历史信息,而这些信息对于深入刻画变更特征从而深入探究变更引入缺陷的根源是十分重要的.此外,现有的方法仅探讨较短时间范围内的学习和预测问题,并没有考虑在长时间的软件演化过程中,由于外界环境的变化(如新需求或人员重组等)所引发的概念漂移现象.

针对现有方法的不足,本文提出了一种面向软件源代码变更的缺陷预测方法.该方法将细粒度(语句级别)源代码变更作为学习和预测对象,预测每次语句级别的变更是否引入了缺陷,从而及时引导软件人员将更多的质量保证资源分配到最可能包含缺陷的被修改语句,实现更精确的缺陷定位.与现有的方法相比,该方法可以显著缩小测试和审查范围,从而有效降低定位缺陷所需耗费的质量保证成本.该方法采用程序静态分析和自然语言语义主题推断相结合的技术深入挖掘软件库,从变更上下文、变更内容、变更时间以及变更人员这 4 个方面构建特征集,从而帮助软件人员更好地理解变更易于引入缺陷的因素.针对在长时间软件演化过程中预测细粒度变更所存在的概念漂移问题,该方法首先采用特征熵差值矩阵[9]分析了概念漂移问题的特点(时间邻近实例较优和概念重现),然后针对该特点提出了一种伴随概念回顾的动态窗口学习机制,从而有效地保证了预测性能随时间的稳定性.

本文第 1 节介绍软件变更引入缺陷问题和软件工程中概念漂移问题的相关研究工作.第 2 节首先给出方法中所涉及的基本概念的定义,然后详细介绍方法中用来描述细粒度源代码变更的特征集,最后针对软件演化过程中细粒度源代码变更概念漂移的特点,详细阐述伴随概念回顾的动态窗口学习机制.第 3 节使用 6 个开源软件作为数据集,共开展特征分析、学习机制比较以及成本有效性比较这 3 项实验,从而验证本文方法的有效性.最后总结全文并讨论下一步的研究工作.

1 相关工作

1.1 软件变更引入缺陷问题的研究

缺陷预测和软件演化是在软件工程领域中被广泛研究的两个主题,近 10 年来,分析和预测引入缺陷的软件变更开始受到越来越多的关注.

一部分研究者采用相关性分析或可视化等手段研究软件变更的各种特征和缺陷引入的关系.Sliwerski 等人[10]定量分析了开源项目的事务级变更,指出变更的规模和时间与引入缺陷的风险存在关联.规模越大,风险越

大.此外,发生在每周五的变更更易于引入缺陷.类似地,Eyolfson 等人[11]也研究了事务级变更时间和缺陷的相关性.通过对两个开源项目 Linux 内核和 PostgreSQL 开展实验发现,每日内不同时段的变更,其引入缺陷的风险不同.从午夜至凌晨 4 点发生的变更,具有较高的缺陷引入倾向性,而从早晨 7 点至正午发生的变更,缺陷引入倾向性较低.此外,他们还讨论了开发人员经验对变更质量的影响.针对操纵变更的开发人员特征与缺陷引入的关系,Rahman 等人[12]展开了更深入的研究.他们发现代码的所有权和开发人员的经验都会对变更的质量造成影响.除了研究开发人员经验之外,一些研究者还进一步探索了开发人员之间的交流信息对变更质量的影响.Abreu 等人[13]通过分析 Eclipse 的邮件列表发现,开发人员之间过于频繁地交流会增加变更引入缺陷的风险.

另一部分研究者致力于构建面向软件源代码变更的缺陷预测方法.Aversano 等人[4]基于文本分类的思想实现了对事务级变更的缺陷预测.他们将每次事务级变更发生前后的软件状态定义为两个快照,利用 TF-IDF 加权技术将每个快照表示成向量.将两个相邻版本加权向量做减法运算,便得到每个事务级变更的向量表示.最后将事务级变更向量作为学习实例,输入分类器得到缺陷预测模型.Kim 等人[2]提出了针对更细粒度层级文件级变更的预测方法.他们的方法同样基于文本分类的思想,与 Aversano 等人[4]的纯文本分类方法不同的是,他们还加入了变更规模以及开发人员姓名等信息.这两种方法采用简单的向量空间模型表征变更,没有充分挖掘软件库中所蕴藏的程序结构信息、自然语言语义信息以及历史信息,因此研究者很难从方法本身分析出导致缺陷引入的重要因素,方法可理解性较弱.此外,这两种方法对于长达几年甚至十几年的项目都只探讨了较短时间范围内(1~3 个月)的预测问题,没有考虑在长时间的软件演化过程中所存在的概念漂移现象.

1.2 概念漂移问题在软件工程领域预测方法中的研究

概念漂移问题已被很多领域的研究者广泛地关注和讨论.Tsymbal 等人[14]对概念漂移问题在各个领域的研究情况进行了系统的调研.他们指出,在现实世界中,概念常常是随时间变化的,因此很多领域都面临着概念漂移问题,如网络入侵探测[15]、宏观经济预测[16]、邮件分类[17]、新闻分类[18]、个性化推荐等[19].在这些领域,应对概念漂移问题的各种学习机制(如实例选择[20]、实例加权[21]、组装器[22]等)得到了广泛的应用.

在软件工程领域,软件外界环境的不稳定(如新需求的加入、团队结构的变化等)常促使软件自身特征及其缺陷的生成规律在其生命周期中不断改变,因此在构建软件工程领域的各种预测方法时也常常面临着概念漂移问题.Ekanayake 等人[23]首次提出缺陷预测领域中的概念漂移问题.他们使用可视化方法展现了面向源文件实体的缺陷预测模型的预测性能随时间的变化情况,从而识别出了软件生命周期中的概念漂移现象.但他们并未给出明确的应对概念漂移问题的措施.Chrupala 等人[24]将多种增量学习模型应用在缺陷报告分流任务中以应对概念漂移的发生,通过实验发现,使用标准的增量模型得到的分流效果要明显优于批量模型.本文在 Ekanayake 等人[23]工作的基础上,既分析了在面向变更的缺陷预测问题中概念漂移的特点,又提出了有效应对概念漂移问题的学习机制,从而推进了缺陷预测领域的概念漂移问题研究.

2 面向细粒度源代码变更的缺陷预测方法

2.1 基本概念

Fluri 等人[25]针对面向对象编程语言提出了一套基于抽象语法树编辑操作的源代码变更分类法.该分类法将变更所操纵的代码实体粒度层级精确至语句级.依据他们的研究成果,本文将细粒度源代码变更定义如下.

定义 1(细粒度源代码变更). 将源文件 d 表示成一棵抽象语法树 $TR=(V_{TR},P_{TR})$, 其中 V_{TR} 表示代码实体节点集,P_{TR} 表示代码实体的父子关系有向边集.对于 $\forall node_i \in V_{TR}$,$l(node_i)$ 和 $val(node_i)$ 分别代表代码实体 $node_i$ 的类型和标识符.对于 $\forall p \in P_{TR}$, 若有 $p(node_i)=node_j$, 则 $node_j$ 是 $node_i$ 的父节点.基于 TR 的编辑操作有 4 类,分别是:

(1) 插入: $INS((l(node_i),val(node_i)),node_j,k)$ 表示将实体 $node_i$ 插入实体 $node_j$ 的第 k 个子节点处.

(2) 删除: $DEL(node_i)$ 表示将实体 $node_i$ 从其父节点 $p(node_i)$的子节点列表中删除.

(3) 移动: $MOV(node_i,node_s,k)$ 表示将实体 $node_i$ 插入实体 $node_s$ 的第 k 个子节点处,并从原父节点 $p(node_i)$ 的子节点列表中删除.

(4) 更新:$UPD(node_i, val(node_i))$ 表示更新实体 $node_i$ 的标识符,使 $val_{new}(node_i) \neq val_{old}(node_i)$.

四大操作类型依据其作用的代码实体被进一步细分为 48 种变更类型(如条件表达式更新、方法参数插入等),每个操作类型都由多个变更类型组成.将基于 TR 的一次编辑操作定义为一次细粒度源代码变更,简称 SCC.

定义 2(修复缺陷的细粒度源代码变更). 通常情况下,当一个缺陷通过变更引入到软件系统之后,它会经历被发现,被报告至缺陷库,被分配以及最终被修复这样一个过程.修复缺陷的细粒度源代码变更指的是为修复存在于软件系统中的缺陷而发生的细粒度源代码变更.

定义 3(引入缺陷的细粒度源代码变更). 导致修复缺陷的细粒度源代码变更发生的细粒度源代码变更.

图 1 给出了 3 个不同类型的细粒度源代码变更,即语句插入、参数插入以及条件表达式更新.其中,语句插入是引入缺陷的细粒度源代码变更,条件表达式更新是修复缺陷的细粒度源代码变更.

定义 4(学习任务). 给定细粒度源代码变更序列 $C_t = SCC_1, SCC_2, ..., SCC_t$, 其中 $SCC_i (i = 1,2,...,t)$ 表示变更实例,由特征集 $F = \{f_1, f_2, ..., f_m\}$ 和标签 $l(SCC_i) \in \{Clean, Buggy\}$ 来描述,这里 *Clean* 和 *Buggy* 分别代表细粒度源代码变更没有引入缺陷和引入了缺陷.本文采用增量学习框架来定义学习任务.在每一个时间步 t,由带标签的实例序列 $C_t = SCC_1, SCC_2, ..., SCC_t$ 得到预测模型 BP_t.当 t+1 时间步一个新的细粒度源代码变更 SCC_{t+1} 到来时,依据其特征集 $F = \{f_1, f_2, ..., f_m\}$ 和当前模型 BP_t 预测其标签,即 $BP_t(SCC_{t+1} \mid f_1, f_2, ..., f_m) = l_p(SCC_{t+1})$. 然后 SCC_{t+1} 的真实标签 l 到达(即软件人员对 BP_t 所预测的结果经过审查核实给出反馈),将 SCC_{t+1} 加入带标签变更实例序列得到 $C_{t+1} = SCC_1, SCC_2, ..., SCC_t, SCC_{t+1}$, 通过学习 SCC_{t+1}, 更新 BP_t 至 BP_{t+1}.

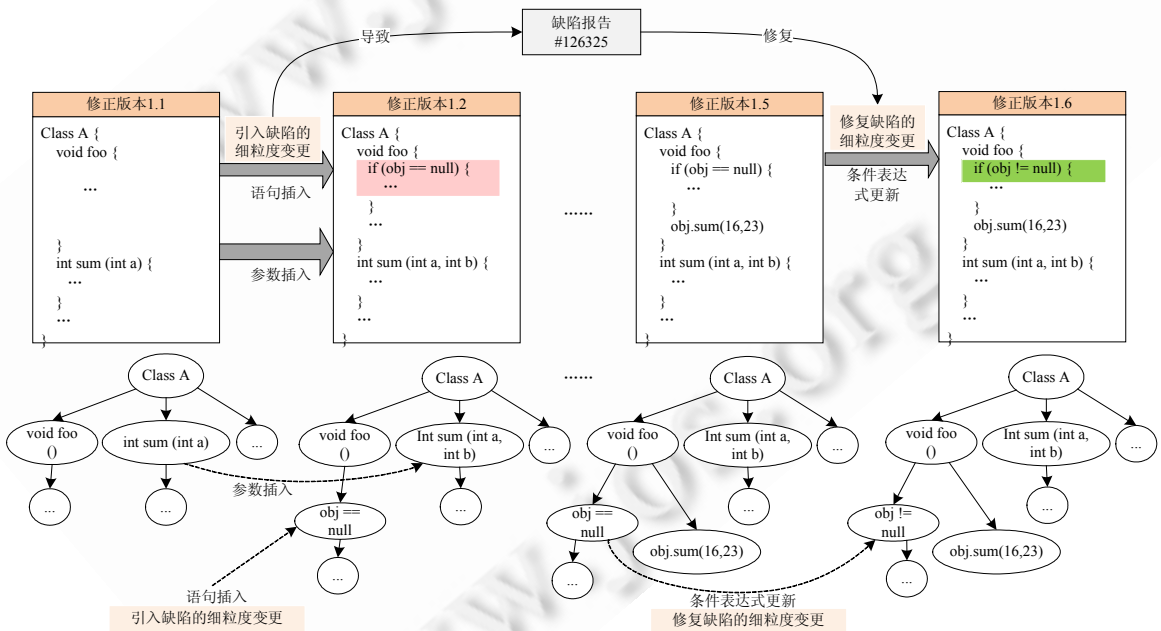


Fig.1 Examples of bug introducing and bug fixing fine-grained source code changes

图 1 引入缺陷和修复缺陷的细粒度源代码变更示例

2.2 特征集

一次细粒度源代码变更可被看作一个事件,描述一个事件往往从其发生的上下文环境、内容、时间以及操纵者 4 个方面入手.本文方法就是基于这 4 个方面构建特征集.为了构建这 4 个方面的特征,需要挖掘软件库 3 个维度的信息,即程序结构信息、自然语言语义信息以及历史信息.特征和软件库信息维度的对应关系如图 2 所示.

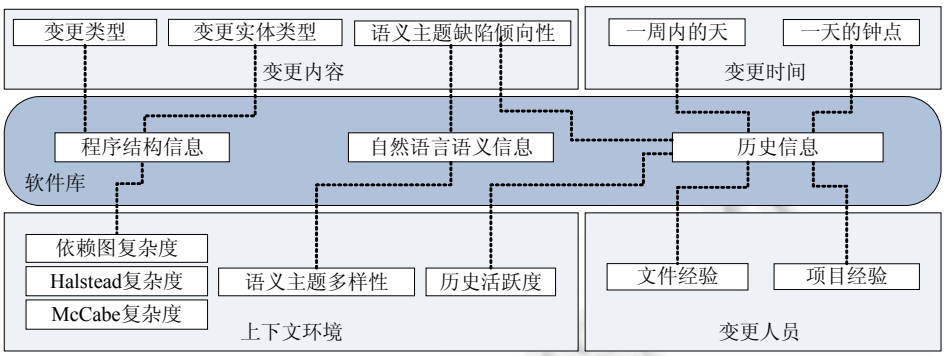


Fig.2 Correspondence between features and information from three dimensions of software repository

图 2 特征和软件库 3 个信息维度对应关系

2.2.1 细粒度源代码变更所处的上下文环境

本文假设,如果一次细粒度源代码变更处在一个较为复杂和活跃的上下文环境中,此次变更就更易于引入缺陷.本文采用细粒度源代码变更所在的源代码文件来代表其所处的上下文环境,关注以下特征:

代码行数:代码行数是用来刻画源代码文件复杂度最简单、最直观的度量.由于其易于获取且具有良好的缺陷预测能力,研究者们常常把它选为构建缺陷预测模型的因素[6,26].

McCabe 复杂度:在 1976 年,McCabe[27]引入了 McCabe 复杂度,用来捕获源代码程序的控制流结构复杂程度.圈复杂度作为其中最著名的度量,用来描述源代码程序执行的线性独立路径数目.研究者认为路径数越多,源文件的逻辑结构越复杂,存在缺陷的概率就越大.因此它作为缺陷预测模型的因素在缺陷预测研究领域得到了广泛应用[6,26].本文将圈复杂度 Cyclomatic 选入特征集来刻画变更上下文环境的控制流结构复杂程度.

Halstead 复杂度:在 1977 年,Halstead[28]引入了 Halstead 复杂度,主要用来度量源代码的词法复杂程度.词法复杂度通常决定了阅读源代码的难易程度,从而影响代码的缺陷倾向性.本文将 Halstead 复杂度的 4 个基本度量和两个引申度量选入了方法特征集,刻画变更上下文环境的词法复杂度.它们分别是:独特算子的数量、独特算域的数量、算子的总数、算域的总数、容量以及工作量.

软件依赖图复杂度:缺陷预测领域的研究者[8]指出,软件实体之间的依赖在很大程度上决定了其操作的正确性.他们将软件系统抽象为软件依赖图,围绕图度量(如度数中心度、中间中心度等)来构建缺陷预测模型.本文依据面向对象软件的特点,将软件抽象为带权多类型边依赖图 $G=(V_G,E_G)$,其中,$V_G=\{v_1,v_2,...,v_N\}$ 表示节点集,即源代码文件组成的集合;$E_G=\{e_1,e_2,...,e_M\}$ 表示边集,即源代码文件之间的关系集合.对于 $\forall e_i\in E_G(i=1,2,...,M)$,$t(e_i)$ 和 $w(e_i)$ 分别代表关系 e_i 的类型和权重.关系类型主要有 4 种,即通过源文件中的类和接口所建立的继承、关联、实现以及依赖关系.由于两个源文件之间可能存在多次关联和依赖,因此使用权重来刻画源文件关系的紧密程度.对 $\forall v_j\in V_G(j=1,2,...,N)$,如果 $S_{v_j}=\{e_{s_1},e_{s_2},...,e_{s_m}\}$ 表示以节点 v_j 为初始节点的边集,而且 $T_{v_j}=\{e_{t_1},e_{t_2},...,e_{t_n}\}$ 表示以节点 v_j 为终止节点的边集,则定义节点 v_j 的加权出度和入度分别为

$$OutDegree\ (v_j)=\sum_{k=1}^{m}w(e_{s_k}),\ InDegree(v_j)=\sum_{l=1}^{n}w(e_{t_l}) \tag{1}$$

本文将这两个加权度数中心度选入特征集,刻画变更上下文环境在软件系统结构中的交互关系复杂度.

语义主题多样性:近几年来,研究者发现源代码中的自然语言(如注释、软件实体名)包含着丰富的语义主题信息,这些语义主题信息反映了软件系统的领域功能和商业逻辑.挖掘出这些语义主题信息可以为故障定位、程序理解等软件维护活动提供帮助[29,30].本文假设,如果一个源代码文件具有较为广泛的语义主题,则说明该文件涉及的领域功能繁杂,商业逻辑复杂度较高,那么对其进行更改引入缺陷的概率就比较大.依据此假设,本文提出了语义主题多样性度量,并选入特征集.为了定义源代码文件的主题多样性度量,需要先描述其主题发现过程.主题发现模型包括潜在语义索引(LSI)、概率潜在语义索引(pLSI)和潜在狄利克雷分配(LDA)模型等[31].Blei 等人[31]指出,LDA 模型在文档建模和协同过滤中的表现优于其他模型.本文针对变更上下文环境的特点改进了

LDA 模型,使其适用于演化中的软件主题发现.与直接从静态软件系统中发现主题不同,由于变更上下文环境随时间不断变化,每个源文件常有多个修正版本,因此主题发现需要基于历史中的所有文件版本来进行.Thomas 等人[32]指出,在软件演化过程中,源文件两个相邻版本常具有很高的文本相似度,直接运用主题发现模型会导致推断结果不准确.于是他们提出了针对差异文件的主题发现方法 DiffLDA.但是该方法建立在模型学习前源文件所有版本都可见的假设之上.而在本文的增量预测场景中,每一个时间步只有历史版本和当前版本的源代码是可见的,未来版本并不可见.因此,本文采用增量 DiffLDA 推断来完成主题发现过程,具体描述如下:

假设细粒度变更 $SCC_i(i=1,2,...,t)$ 发生在软件系统第 u 个版本 V_u 的第 v 个源文件 $d_{v,u}$ 中,将其对应的软件历史表示为 $H=V_1,V_2,...,V_u$.将源文件 $d_{v,u}$ 与其前一个版本 $d_{v,u-1}$ 进行文本比较得到 $\{\delta^a_{v,u},\delta^d_{v,u}\}$,其中,$\delta^a_{v,u}$ 代表 $d_{v,u}$ 相对于 $d_{v,u-1}$ 增加的文本片断,$\delta^a_{v,u}$ 代表 $d_{v,u}$ 相对于 $d_{v,u-1}$ 删除的文本片断.$\delta^a_{v,u},x\in\{a,d\}$ 简称差异文件.差异文件通过一个词序列来表示,即 $\delta^a_{v,u}=w_1,w_2,...,w_{|\delta^x_{v,u}|}$,其中,$|\delta^x_{v,u}|$ 表示差异文件 $\delta^x_{v,u}$ 的长度,$w_k(k=1,2,...,|\delta^x_{v,u}|)$ 表示差异文件 $\delta^x_{v,u}$ 中的词.定义 $D_u=\{\delta^x_{j,u}\mid x\in\{a,d\},j=1,2,...,|V_u|\}$ 为 V_u 版本所有源代码文件 $d_{j,u}(j=1,2,...,|V_u|)$ 所对应的差异文件组成的集合,对应词序列 $W_u=\delta^x_{1,u},\delta^x_{2,u},...,\delta^x_{|V_u|,u},x\in\{a,d\}$.进一步定义 $D_{1,...,u}=\bigcup_{k=1}^{u}D_k$ 为版本 V_1~版本 V_u 的所有差异文件组成的集合,对应词序列 $W_{1,...,u}=W_1,W_2,...,W_u$.假设软件系统的主题为 $T=\{t_1,t_2,...,t_{|T|}\}$,$z_{1,...,s}$ 代表 $D_{1,...,u-1}$ 的词序列在版本 V_{u-1} 时的主题分配情况.其中,$s=\sum_{q=1}^{u-1}\sum_{p=1}^{|V_q|}\sum_{x\in\{a,d\}}|\delta^x_{p,q}|$.将 $z_{1,...,s}$ 和 W_u 作为输入,采用增量 LDA Gibbs 抽样算法进行推断[33](具体过程见算法 1),得到 $D_{1,...,u}$ 的词序列在版本 V_u 时刻的主题分配情况 $z_{1,...,s'}$,其中,$s'=\sum_{q=1}^{u}\sum_{p=1}^{|V_q|}\sum_{x\in\{a,d\}}|\delta^x_{p,q}|$.依据 LDA Gibbs 抽样原理,按照下式估算 $D_{1,...,u}$ 中的每个差异文件 $\delta^x_{p,q},x\in\{a,d\},p\in\{1,2,...,|V_q|\},q\in\{1,2,...,u\}$ 的主题分布情况 $\theta_{\delta^x_{p,q}}=\left\langle\theta^1_{\delta^x_{p,q}},\theta^2_{\delta^x_{p,q}},...,\theta^{|T|}_{\delta^x_{p,q}}\right\rangle$:

$$\theta^k_{\delta^x_{p,q}}=\frac{n_k^{(\delta^x_{p,q})}+\alpha}{n_{\cdot}^{(\delta^x_{p,q})}+|T|\alpha},x\in\{a,d\},k\in\{1,2,...,|T|\},p\in\{1,2,...,|V_q|\},q\in\{1,2,...,u\}\tag{2}$$

其中,$n_k^{(\delta^x_{p,q})}$ 表示 $\delta^x_{p,q}$ 中分配给主题 k 的词数;$n_{\cdot}^{(\delta^x_{p,q})}$ 表示 $\delta^x_{p,q}$ 的长度;α是 $\theta^k_{\delta^x_{p,q}}$ 的狄利克雷先验,为模型的参数.

算法 1. 增量 LDA Gibbs 抽样.

输入:版本 V_1 至 V_{u-1} 对应词序列 $W_{1,...,u-1}$,版本 V_{u-1} 时的主题分配情况 $z_{1,...,s}$,版本 V_u 的差异文件集 D_u 对应的词序列 W_u.

输出:版本 V_u 时的主题分配情况 $z_{1,...,s}$.

1: $W=W_{1,...,u-1}$
2: **for** $l=1,...,|W_u|$ **do**
3: w=W_u 序列的第 l 个元素
4: W=W,w
5: 根据 $P(z_{s+l}\mid z_{1,...,s+l-1},W)$ 抽样 z_{s+l}
6: $z_{1,...,s+l}=z_{1,...,s+l-1},z_{s+l}$
7: //$R(l)$是以 $s+l$ 为上界的词索引随机生成函数
8: **for** r in $R(l)$ **do**
9: 根据 $P(z_r\mid z_{1,...,s+l\backslash r},W)$ 抽样 z_r
10: 用 z_r 更新 $z_{1,...,s+l}$ 序列的第 r 个元素
11: **end for**
12: **end for**
13: **return** $z_{1,...,s}$.

经过进一步计算得到细粒度变更 SCC_i 所在的源代码文件 $d_{v,u}$ 的主题分布情况 $\theta_{d_{v,u}}$.

具体计算公式如下所示:

$$\theta_{d_{v,u}}=\frac{\sum_{q=1}^{u}\theta_{\delta_{v,q}^{a}}\left|\delta_{v,q}^{a}\right|-\sum_{q=1}^{u}\theta_{\delta_{v,q}^{d}}\left|\delta_{v,q}^{d}\right|}{\sum_{q=1}^{u}\left|\delta_{v,q}^{a}\right|-\sum_{q=1}^{u}\left|\delta_{v,q}^{d}\right|} \tag{3}$$

语义主题多样性度量主要用来刻画源代码文件包含主题的广泛程度.

在信息检索领域,Yan 等人[34]在研究科技文献被引用数量及其包含主题内容广泛程度之间关系时,采用信息熵来度量电子文献主题的广泛程度,作为文献引用数量预测模型的特征得到了较好的预测效果.本文也采用信息熵来度量源代码文件包含主题的广泛程度.

按照下式,计算得到细粒度源代码变更 SCC_i 所在的源代码文件 $d_{v,u}$ 的主题多样性度量:

$$Versatility(d_{v,u})=\sum_{k=1}^{|T|}-\theta_{d_{v,u}}^{k}\cdot\log\theta_{d_{v,u}}^{k} \tag{4}$$

历史活跃度:目前缺陷预测领域的研究者发现,从软件库中提取历史度量可以明显改进缺陷预测性能[35].本文假设,如果一个源代码文件在历史上越活跃,那么对这个源代码文件进行变更,引入缺陷的风险就会越大.本文将 3 个刻画源代码文件历史活跃性的度量纳入方法特征集,即:

(1) 源代码文件在历史上被修改的次数;

(2) 源代码文件在历史上修复缺陷的次数;

(3) 在历史上修改过源代码文件的开发人员的个数.

2.2.2 细粒度源代码变更内容

本文假设,如果细粒度源代码变更内容本身具有一定的语法或语义难度,那么其更易于引入缺陷.本文从变更类型、变更所操纵的实体类型以及变更所涉及的语义主题缺陷倾向性这 3 个方面来刻画变更内容.

变更类型:Pan 等人[36]通过分析缺陷修复模式发现,一些类型的变更(如条件表达式变更),由于实现的抽象语法树操作逻辑较为复杂,更易于引入缺陷.本文将 Fluri 等人[25]提出的 48 种细粒度源代码变更类型(如条件表达式变更、方法接口变更、返回类型变更等)选入特征集,用以刻画细粒度变更本身的语法树操作特点.

变更实体类型:除了反映抽象语法树操作特点的变更类型之外,变更所操作的语法树节点类型(即实体类型)也对操作本身的难易程度产生影响.如 Pan 等人[36]所提到的,对 if 条件语句进行的修改很容易引发缺陷.本文将 Fluri 等人[25]定义的 104 种实体类型也作为特征选入特征集.

语义主题缺陷倾向性:缺陷预测领域的研究者们发现[37],负责不同领域功能的源代码片断,其存在缺陷的概率也不同.例如,负责实现 I/O 操作等一些简单语义功能的源代码,其发生缺陷的概率较低;而负责实现编译器等一些复杂语义功能的源代码,其发生缺陷的概率较高.本文假设,如果细粒度源代码变更内容涉及多个易于发生缺陷的领域功能,则本次变更引入缺陷的概率就会较大.这里的领域功能通过语义主题来表示.根据这个假设,本文提出了语义主题缺陷倾向性度量,并作为特征选入特征集.

第 2.2.1 节引入了差异文件的概念,为细粒度源代码变更 $SCC_i(i=1,2,...,t)$ 发生之前所有软件版本 $\{V_1,V_2,...V_{u-1}\}$ 中的每个差异文件 $\delta_{p,q}^{x},x\in\{a,d\},p\in\{1,2,...,|V_q|\},q\in\{1,2,...,u-1\}$ 定义一个缺陷感染率度量,该度量用来描述每个差异文件有多大比例的代码量存在问题,从而会在后续被修复缺陷的变更所更改.形式化表示为

$$IR\left(\delta_{p,q}^{x}\right)=\frac{\left|DW_{\delta_{p,q}^{x}}\right|}{\left|\delta_{p,q}^{x}\right|},x\in\{a,d\},q\in\{1,2,...,u-1\},p\in\{1,2,...,|V_q|\} \tag{5}$$

其中,$DW_{\delta_{p,q}^{x}}$ 表示由于修复缺陷而发生修改的代码片断中的词集合,$\left|DW_{\delta_{p,q}^{x}}\right|$ 代表该集合中元素的个数.

假设 SCC_i 涉及源代码文件的 n 个词,表示为词的集合 $W_{SCC_i}^{r}=\left\{w_{SCC_i}^{1},w_{SCC_i}^{2},...,w_{SCC_i}^{n}\right\}$,根据第 2.2.1 节所描述的主题推断过程,每个词被分配给一个主题.与细粒度源代码变更 SCC_i 相关的主题被表示成集合 $T_r=\{t_{r_1},t_{r_2},...,t_{r_m}\}$.对于 SCC_i 每一个相关主题 $t_{r_k},k=1,2,...,m$,依据 SCC_i 发生之前该主题的缺陷倾向性来推断 SCC_i 的语义主题缺陷倾向性.首先定义主题 $t_{r_k},k=1,2,...,m$ 在 SCC_i 发生之前的缺陷倾向性度量,表示如下:

$$TBP(t_{r_k})=\frac{\sum_{q=1}^{u-1}\sum_{p=1}^{|V_q|}\theta_{\delta_{p,q}^x}^{r_k}\cdot IR(\delta_{p,q}^x)}{\sum_{q=1}^{u-1}\sum_{p=1}^{|V_q|}1(\delta_{p,q}^x\neq\varnothing)} \tag{6}$$

其中,$1(\delta_{p,q}^x\neq\varnothing)$ 代表差异文件 $\delta_{p,q}^x$ 不为空时,函数值为 1,反之则值为 0.

最后依据 SCC_i 每个相关主题的缺陷倾向性,汇总得到 SCC_i 的语义主题缺陷倾向性度量,即

$$TBP(SCC_i)=\sum_{k=1}^{m}TBP(t_{r_k}) \tag{7}$$

语义主题缺陷倾向性度量捕获了蕴藏在源代码文件中的非结构化语义信息.这些信息是无法通过分析抽象语法树获取的,因此它是对程序结构化信息的一种重要补充.

2.2.3 细粒度源代码变更发生的时间

有研究者指出[10,11],由于开发者的工作效率和质量呈现一定的时间周期性,因此变更时间与变更质量存在一定的关系.一周内的某个工作日或者每日内的某个时段发生的变更,可能更容易导致缺陷.根据已有的研究,本文定义两个时间周期相关的度量,即一周的天和一天的钟点,刻画变更发生的时间周期特点,并选入特征集.

2.2.4 操纵细粒度源代码变更的开发人员经验

近几年来,在缺陷预测领域,越来越多的研究者开始关注人的因素[11−13].细粒度源代码变更是由开发人员完成的,因此开发人员对项目和模块的熟悉程度直接决定了变更的质量.本文假设,如果实现细粒度源代码变更的开发人员缺乏相应的项目和模块开发经验,该变更就易于引入缺陷.本文定义了两种刻画开发人员经验程度的度量,一种用来描述开发人员对整个项目的熟悉程度,另一种描述其对变更所在的文件的熟悉程度.即:

文件经验:开发人员在软件历史中修改该源文件的次数.

项目经验:开发人员在软件历史中修改整个项目的次数.

2.3 学习机制

在软件演化过程中,新需求不断加入,开发团队人员不断更迭,这些外界环境的变化促使软件自身特征和其缺陷的生成规律也在不断改变,因此在构建缺陷预测模型时面临着概念漂移问题.概念漂移是导致预测模型准确率下降的重要因素.Ekanayake 等人[23]通过观察缺陷预测模型随时间预测性能的变化情况发现,时间邻近的实例与时间相距较远的实例相比具有更好的预测性能.Hindle 等人[38]通过对软件版本库进行傅里叶分析发现,变更事件具有可重现性特征.基于前人的工作,本文假设在对源代码变更实例学习过程中遇到的概念漂移问题具有时间邻近的实例较优和历史概念重现两大特点.对于具有时间邻近实例较优特点的概念漂移问题,通常采用滑动窗口机制[39].然而,由于滑动窗口大小的确定较为困难(过大的窗口影响概念漂移的反应速度,而过小的窗口降低模型的可信度),因此研究者们提出了带探测器的自适应滑动窗口调整法[40].该方法不需要预设窗口大小,它通过探测机制来识别概念漂移的发生,然后自动地收缩窗口来应对漂移.但是这种方法在自动收缩窗口后,会出现一段时间内由于窗口实例数目太少而导致模型预测不可信的问题.由于变更具有重现性特点,因此可以采用历史中相同或相似的概念对当前窗口中较少的实例进行补充,从而解决模型不可信的问题.

本文针对上面所假设的源代码变更概念漂移特点,提出了伴随概念回顾的动态窗口学习算法,简称 AWCR (adaptive window with concept reviewing),该算法通过动态维护 1 个基本分类器、1 个历史概念库、2 个缓冲区(训练实例缓冲区和新概念缓冲区)、3 种漂移状态(稳定、警告以及失控),有效地应对在对细粒度源代码变更进行学习时遇到的概念漂移问题.每当新实例到达时,先通过漂移探测算法判定其所处漂移状态,如果状态为警告,则开始缓存新概念实例至新概念缓冲区,同时更新训练实例缓冲区来调整基本分类器;如果状态从警告转为失控,则采用 KNN 非参数统计检验方法[41]从概念库中查询与新概念相似的概念,并将历史相似概念所对应的实例补充至新概念缓冲区,同时使用新概念缓冲区中的实例重建基本分类器,如果当前新概念从未在历史中出现,则存储新概念至概念库,最后刷新训练实例缓冲区;如果状态为稳定,则仅更新训练实例缓冲区调整基本分类器.

本文选用著名的 DDM 方法[40]来探测概念漂移状态.该方法假设对每个实例的预测结果正确与否是来自伯

努利实验序列的随机变量,对于序列中的第 n 次预测,假设截止到 n 之前所有错误预测的个数为 $error_n$,则其预测结果错误的概率为 $p \leftarrow error_n/n$,标准差为 $s \leftarrow \sqrt{p(1-p)/n}$. 如果随着 n 的增加,p 显著增长,则预示着概念漂移的发生.记录 $p_{\min}$ 和 $s_{\min}$ 为到目前为止最小的 p 和 s 的值.依据当前 p 和 s 以及 $p_{\min}$ 和 $s_{\min}$ 的值来判断概念所处的状态.由于当 n 足够大时,随机变量近似正态分布,因此当 $p+s>p_{\min}+2s_{\min}$ 时,有 95%的置信度认为概念漂移发生,因此设定状态为警告.当 $p+s>p_{\min}+3s_{\min}$ 时,有 99%的置信度认为概念漂移发生,因此设定状态为失控.

算法 2 详细描述了 AWCR 算法的整个工作机理.其中,R 代表概念库;C_{train} 和 C_{new} 分别代表训练实例缓冲区和新概念缓冲区;*in*,*waring* 以及 *out* 分别代表稳定、警告以及失控 3 种状态.

算法 2. 概念回顾的动态窗口学习算法(AWCR).

输入:变更实例 SCC_t,缓冲区容量 v,概念库容量 V,前一个状态 *ps*,概念库 R.

输出:预测结果 l_p.

```
1:  变量初始化
2:  if t<v then
3:      将 SCC_t 放入 C_train
4:  else if t=v then
5:  用 C_train 初始化基本分类器 CL
6:  else
7:      l_p←用 CL 预测 SCC_t
8:      l←获取 SCC_t 的真实标签
9:      s←采用 DDM 探测方法获取当前状态
10:     if s=waring then
11:         if ps=in then
12:             清空 C_new
13:         end if
14:         用 SCC_t 更新 C_new,如果超出容量 v,则移除最旧实例
15:         ps←waring
16:     end if
17:     if s=out then
18: if ps=waring then
19:             e←采用 KNN 非参数检验方法查询 R 中与 C_new 相似概念
20:             if e=false then
21:                 将 C_new 作为新概念存储至 R,如果 R 达到容量 V,则移除最旧概念
22:             else
23:                 用 R 中查询到的相似概念对应实例补充 C_new
24:             end if
25:             用 C_new 刷新 C_train,并清空 C_new
26:         end if
27:         ps←out
28:     end if
29:     if s=in then
30:         if ps=waring then
31:             清空 C_new
32:         else if ps=none then
```

33: 将 C_{train} 作为新概念存储至 R
34: **end if**
35: $ps \leftarrow in$
36: 将 SCC_t 加入 R 中对应的概念实例集
37: **end if**
38: 用 SCC_t 更新 C_{train},如果超出容量 v,则移除最旧实例
39: 用 C_{train} 训练 CL
40: **end if**
41: **return** l_p

3 实验及结果分析

本节选取 6 个著名开源项目开展实验,包括 3 部分:首先对细粒度源代码变更进行特征分析,使用卡方检验评估每个特征对分类的重要程度,并分析特征重要程度随时间的变化;其次使用特征熵差值计算法定量分析概念漂移问题的特点,并在分类性能方面将本文提出的学习机制和几种较为流行的应对概念漂移的学习机制进行对比,以验证本文学习机制的有效性;最后在降低人力成本方面,将本文提出的细粒度缺陷预测方法和已有的面向事务级和文件级源代码变更的缺陷预测方法进行比较,以验证本文提出方法的成本有效性.

3.1 数据集

为了评估面向细粒度源代码变更的缺陷预测方法的有效性,本文收集并分析了 6 个开源项目的软件库,表 1 列出了每个项目的基本信息.为了验证方法的通用性,本文选择的开源项目分属不同的应用领域.

Hassan 等人[42]指出,项目初始阶段(1~500 个修正版本)的变更,其模式并不稳定,选用该区段作为数据集通常无法揭示软件演化过程中有价值的规律.因此,本文选择第 500 个修正版本作为起始点.由于 Argouml 项目在前 2 500 个修正版本中没有出现引入缺陷的变更,因此其起始点设为 2 500.

Table 1 Description of datasets

表 1 数据集描述

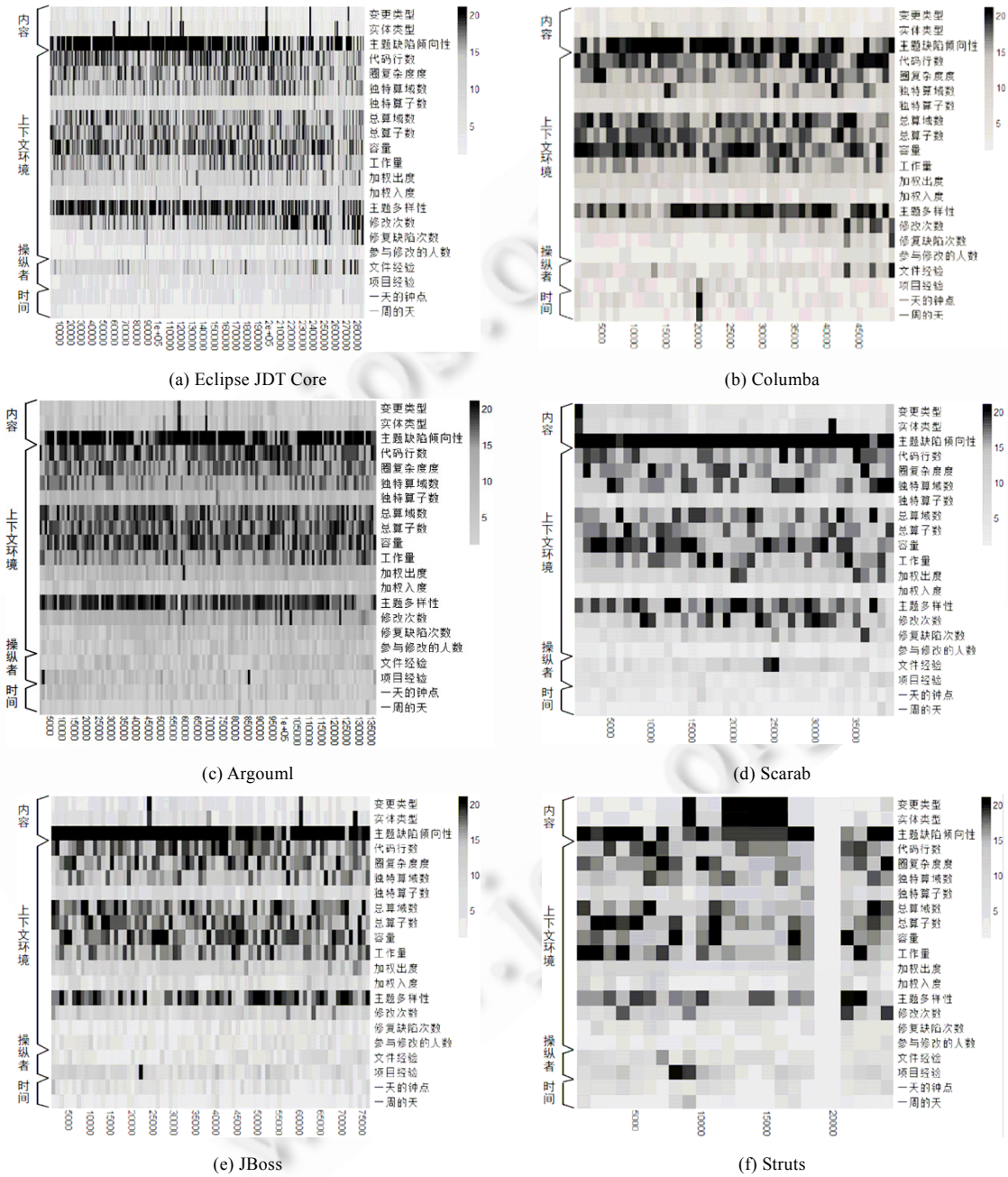
项目名称	修正版本	时间区段	SCC 数目	Buggy SCC 百分比	类型
Eclipse JDT Core	500~21000	9/2001~04/2010	284 173	16	Java IDE
Columba	500~4200	02/2003~05/2006	49 152	8	邮件客户端
Argouml	2500~16000	01/2001~09/2006	135 786	9	UML 建模工具
Scarab	500~9600	05/2001~11/2004	38 742	21	缺陷追踪系统
JBoss	500~8200	08/2000~10/2004	75 978	8	J2EE 应用服务器
Struts	500~4200	01/2001~09/2003	23 550	11	MVC 框架

3.2 特征分析实验

本文通过分析软件库中 3 个维度的信息(程序结构、自然语言语义以及历史),从 4 个方面(上下文、内容、时间以及人员)刻画细粒度源代码变更,每个方面都包含多个特征.每个特征对缺陷预测效果的贡献度和其贡献度随时间的波动性能够帮助软件开发者更好地理解软件生命周期中缺陷引入的规律,从而采取有效措施来保证软件质量.卡内基梅隆大学的 Yang 教授[43]指出,卡方检验和信息增益是衡量特征对分类效果贡献度较好的两种方法,且卡方检验在多次实验中性能更优.另外,Kim 等人[2]在研究面向文件级源代码变更的缺陷预测方法时也选用卡方检验来比较不同特征对缺陷预测效果贡献程度.借鉴前人的工作,本文采用卡方检验来比较各个特征对缺陷预测的重要程度以及其重要程度随时间的波动性.

由于开源软件活动周期平均为 1 个月,每月平均发生约 1 000 次细粒度源代码变更,因此选择 1 000 为观察周期记录特征重要程度排序情况.本文使用热图对软件演化过程中的特征重要程度排序情况进行可视化表示(如图 3 所示).特征集中共 21 个特征,排序按照由小到大的原则(从 1 到 21),序号越大,说明特征重要程度越高.如果特征在相邻两个观察周期序号变化大于 10,则定义为一次显著波动.

特征对缺陷预测效果的贡献度:从图 3 可以观察出,一些特征在所有项目中都表现突出,它们对提高缺陷预测性能起到至关重要的作用;一些特征在所有项目中都表现较差,它们对提高缺陷预测性能贡献度很低;另有一些特征在不同项目中的表现差异较大.



(a) Eclipse JDT Core (b) Columba

(c) Argouml (d) Scarab

(e) JBoss (f) Struts

Fig.3 Chi-Square measure ranks of features

图 3 特征的卡方度量排序

变更内容维度的语义主题缺陷倾向性、变更所在上下文环境维度的语义主题多样性这两个特征在所有项目中都表现突出,它们对提高缺陷预测性能贡献最大.语义主题缺陷倾向性特征的重要性说明对隶属于不同领

域功能的代码片断的修改,引入缺陷的风险是不同的.在历史上具有较高变更风险的领域功能在未来的修改中更可能引发缺陷.这个现象能够指导软件人员将更多的质量保证资源投入到历史上具有较高变更风险的领域功能中,并通过加强对相应领域功能的交流培训以及改进文档质量等手段尽可能地降低修改此功能时引入缺陷的风险.语义主题多样性特征的重要性说明对领域功能繁杂的源文件的更改更易于引入缺陷.这个现象说明了良好的设计模式和及时的代码重构对于提高软件系统的质量是至关重要的.因此,当开发人员发现系统中存在功能较为繁冗的源文件时,需要尽早对其进行模块化处理,从而降低源文件的修改难度,避免缺陷的反复引入.

变更时间维度中一周的天这个特征在所有项目中都表现较差,它对提高缺陷预测性能贡献很低.这主要是由于开源软件的开发和管理模式较为松散,开发人员工作时间和效率没有受到严格约束,因此无法准确捕捉其行为的周期性规律.下一步计划扩充数据源,观察变更时间特征在更严格和集中的管理模式下的商业项目中的表现.

开发者经验在不同项目中表现不同.Eclipse JDT Core,Columba 以及 Scarab 这 3 个项目的源文件经验对缺陷预测的贡献大于项目经验,这说明在这几个项目中开发人员的专业化经验更为重要,因此需要着重培养开发人员的模块专业化水平,并将核心模块分配给最熟悉该模块的开发者而不是让多个经验不同的开发者合作开发.在 JBoss 和 Struts 中,项目经验更为重要,在这两个项目中对系统整体的熟悉程度影响着变更质量,因此需要加强对开发新手的系统整体架构培训力度以及开发人员之间的交流,从而加强开发人员对整个系统的理解.

特征贡献度随时间的波动性:Eclipse JDT Core 项目的特征重要程度随时间的波动性在所有项目中表现最为明显.其中,变更内容中的语义主题缺陷倾向性特征波动性最强,达 37 次.另外共有 7 个特征的明显波动次数超过 10 次.这是由于该项目历时较长,规模较大,因此影响开发环境的因素也相对复杂.在其他项目中,变更内容中的变更实体类型特征波动性较大.

3.3 学习机制性能比较实验

3.3.1 概念漂移问题

本实验采用定量分析和可视化手段描述面向细粒度源代码变更的缺陷预测方法中概念漂移问题的特点.Abdulsalam[9]提出了时间窗口特征熵差值计算法.它通过度量不同时段特征的不同取值在不同类中分布的变化程度定量描述概念漂移特征.本实验中,特征 $f_i(i=1,2,...,m)$ 针对两时间窗口 s 和 t 的特征熵差值形式化表示为

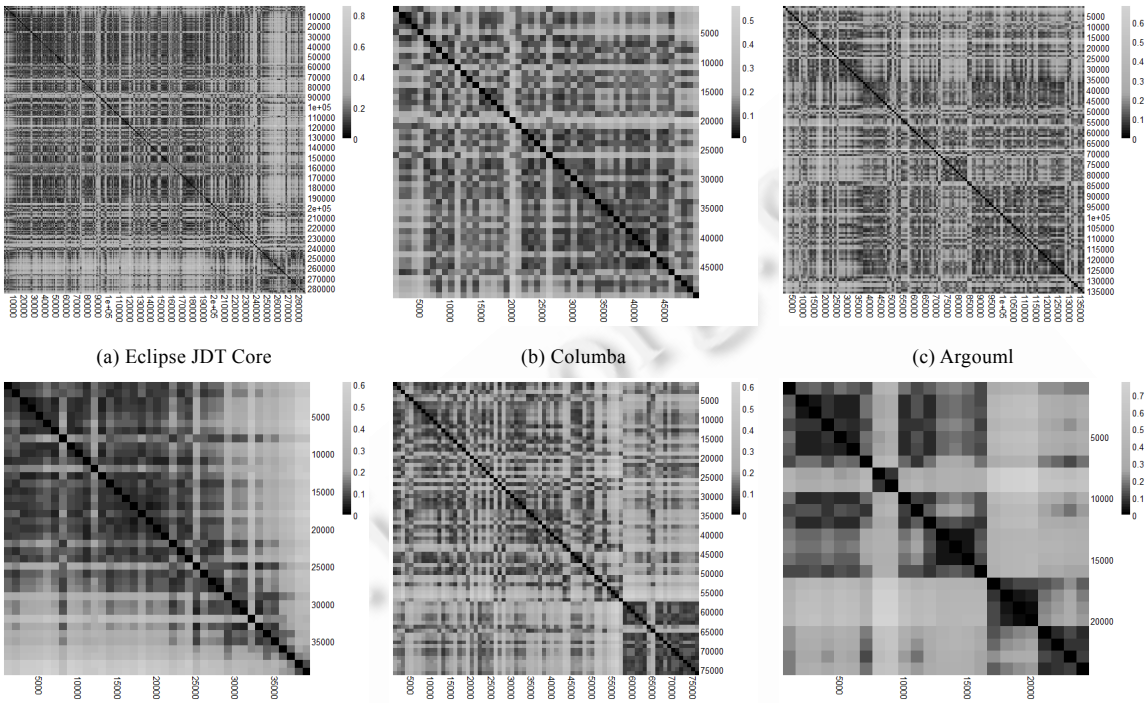
$$H_i=\sum_{l\in\{Clean,Buggy\}}\sum_{k=1}^{K}(u_{kl_s}\log u_{kl_s}-u_{kl_t}\log u_{kl_t}) \tag{8}$$

其中,l 代表细粒度源代码变更的类标签,K 代表特征 f_i 的值区间的个数,u_{kl_s} 和 u_{kl_t} 代表类标签为 l,特征f_i的值处于第 k 个值区间的实例个数占整个时间窗口中实例个数的比值.对所有特征,有

$$H=\frac{1}{m}\sum_{i=1}^{m}H_i \tag{9}$$

本实验中将时间窗口大小依然设置为 1 000.将项目软件历史中所有时间窗口两两求得特征熵差值,得到特征熵差值对称矩阵.本实验采用热图将对称矩阵可视化,从而展现其概念漂移特点,如图 4 所示.图 4 中颜色从深到浅代表特征熵差值从小到大.特征熵差值越小,说明两个时间窗口中的概念越相近.从图 4 中可以看出,特征熵差值对称矩阵对角线附近区域颜色较深,说明概念呈现时间近邻性特征,即时间越相近,概念差异越小.时间近邻性特征在 Scarab 项目中最为明显.呈现时间近邻性特点的概念,可以使用窗口机制对其进行学习.除了时间近邻性以外,从图 4 中还可以看出明显的概念重现性特征.依据 Abdulsalam 的研究工作[9],定义特征熵差值阈值为 0.1.即如果两个时间窗口的特征熵差值小于 0.1,则认为概念重现发生.例如,在项目 Columba 中,第 1 个时间窗口(0~1 000 时间步)中的概念分别在第 5 个(4 000~5 000 时间步),第 11 个(10 000~11 000 时间步)以及第 26 个(25 000~26 000 时间步)窗口发生重现.如果概念具有重现性特点,则可以采用回顾学习机制.另有一些窗口概念和时间邻近的窗口概念并不相似,而且在整个历史中并未发生概念重现.例如 Eclipse JDT Core 项目中的第 238 个时间窗口(237 000~ 238 000 时间步)中的概念在整个历史中没有找到相近概念.对这种情况下的概念进行学习和预测,其预测性能只能通过累积该窗口中的实例来改善.本实验通过特征熵差值对称矩阵验证了第 2.3 节

的假设,即概念漂移问题在面向变更的缺陷预测问题中具有时间邻近的实例较优和历史概念重现两大特点.



(a) Eclipse JDT Core (b) Columba (c) Argouml

(d) Scarab (e) JBoss (f) Struts

Fig.4 Symmetric matrix of feature entropy difference

图 4 特征熵差值对称矩阵

3.3.2 学习机制性能比较

针对细粒度源代码变更缺陷预测模型学习中的概念漂移特点(即时间邻近的实例较优和概念重现性),本文提出了伴随概念回顾的动态窗口学习机制(AWCR).为了检验该算法的性能,本实验选择了当前主要的几种应对概念漂移问题的学习机制与其比较,包括动态窗口(adaptive window)算法[40]、概念回顾(RCD)算法[44]、组装器 OCBoost 算法[22]以及霍夫丁动态树(Hoeffding adaptive tree)算法[45].其中,动态窗口算法在探测到概念漂移后通过收缩窗口,丢弃旧概念实例来应对概念漂移;概念回顾算法通过搜索历史相似概念对应的分类器并丢弃当前窗口实例来应对概念漂移;组装器 OCBoost 算法[22]依据分类错误情况动态调整实例权重和单分类器权重,从而应对概念漂移;霍夫丁动态树算法通过构建新子树,替换旧子树来应对概念漂移.此外,无应对概念漂移能力的传统霍夫丁树(Hoeffding tree)算法[46]也被纳入了比较范围.

缺陷预测属于软件库挖掘研究领域的一个重要分支,它通过获取、处理和分析软件库中的各类信息构建预测模型.由于信息不完整、不准确以及信息获取和处理手段不完善,数据集噪声是不可避免的.除了概念漂移之外,噪声是导致模型预测可信度和准确率下降的另一个重要因素.Melville 等人[47]指出,组装器的抗噪能力明显优于单分类器.其中,随机森林分类器对于含有噪声的数据具有最为显著的分类效果[48].因此,在 AWCR 算法中应优先选择随机森林作为其基本分类器.为了验证随机森林分类器在 AWCR 学习机制中的优越性,本实验将随机森林和决策树分别作为 AWCR 的基本分类器(简写为 AWCR-RF 和 AWCR-J48),比较其分类性能.

本实验是基于 Waikato 大学著名的大规模数据流挖掘平台(MOA)[49]展开的,参与比较的算法中的参数都与 MOA 平台默认参数设置保持一致.本文算法的两个重要参数缓冲区容量和概念库容量分别设置为 1 000(参照开源项目平均开发周期)和 15(参照 MOA 组装器算法标准设置).

由于细粒度源代码变更数据集存在样本类别分布不均衡问题(引入缺陷的细粒度变更数目远小于非引入

缺陷的细粒度变更数目),因此在评估预测性能时,一些传统的评价标准不再适用(如准确度).MOA 平台通常采用 Kappa 统计指标评估各种针对流数据的分类算法.该方法不受类分布的影响,适用于在类不均衡数据集上评估预测方法.Kappa 统计是 1960 年 Cohen 等人[50]提出的,它最先用来作为评价判断一致性程度的指标.后来被机器学习领域的研究者用来作为评价分类器分类结果与随机分类差异度的指标.除了适用于不均衡数据集之外,该评估方法的另一优势在于,仅通过 1 个指标就可以刻画出当前算法性能与随机分类算法(通常作为分类基线)的优劣,具有很好的实用性.Kappa 统计指标的取值范围是[–1,1],当取值为 1 时,说明分类标签和实际标签完全一致,分类结果达到最优;当取值为 0 时,说明当前算法与随机分类效果一样;当取值为–1 时,说明当前算法最差.一般来说,Kappa 统计指标越接近于 1 越优,大于 0 说明当前算法优于随机分类;大于 0.4 小于 0.75 说明当前算法具有较好的分类效果;大于 0.75 说明当前算法已取得很好的分类效果[51].

实验中的所有算法使用交织的测试后训练(interleaved test-then-train)流数据评估策略和 Kappa 统计评估指标.图 5 给出了各算法的累积 Kappa 统计值随时间的变化.

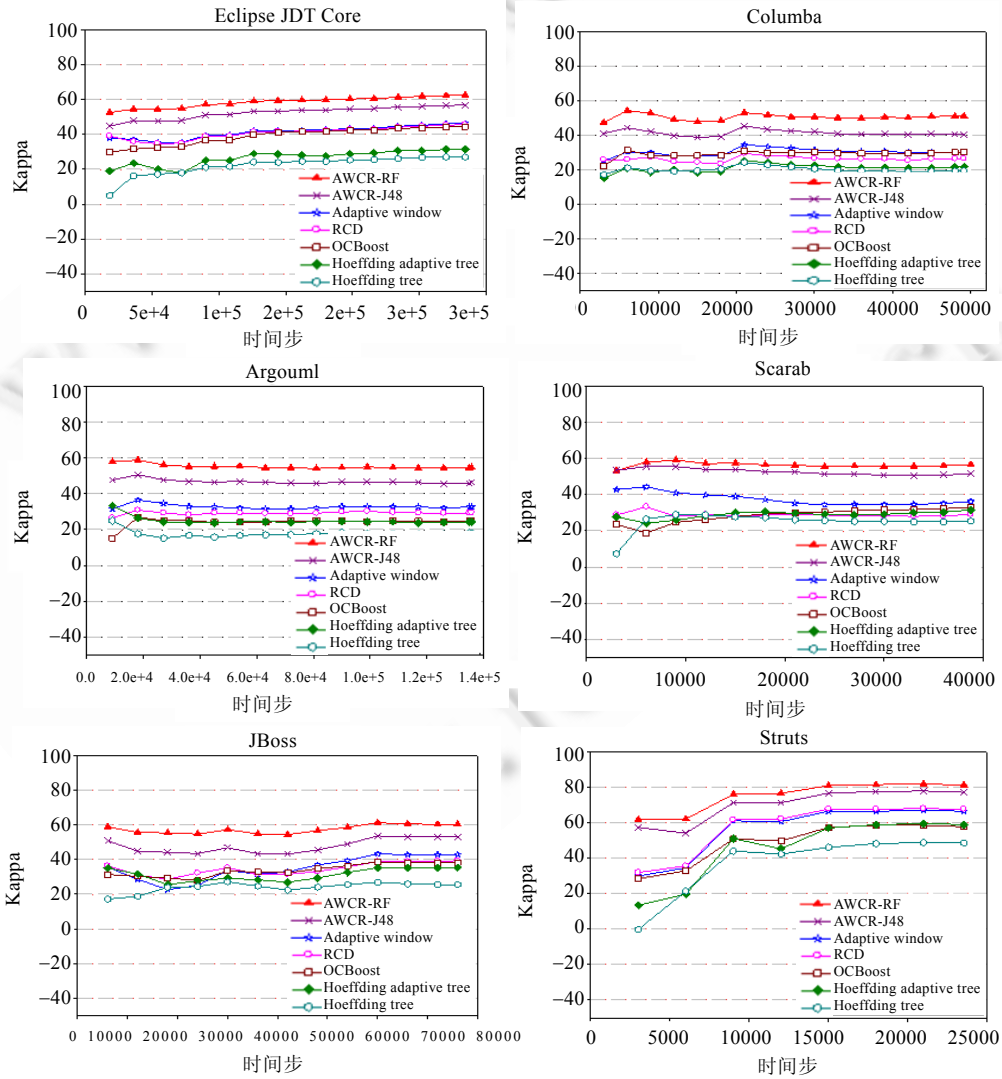


Fig.5 Performance of all classification methods

图 5 各种学习机制性能比较

可以看出,本文提出的采用随机森林作为基本分类器的 AWCR 学习机制(AWCR-RF)分类效果最优,Kappa 统计值平均为 0.61.而采用决策树作为基本分类器的 AWCR 学习机制(AWCR-J48)次之,Kappa 统计值平均为 0.54,比 AWCR-RF 低 7%.这说明数据噪声对分类效果的影响是不可忽略的,在学习过程中选择具有较强抗噪能力的基本分类器能够显著提高分类性能.排在第三、四位的是动态窗口算法和概念回顾算法,它们的 Kappa 统计值平均分别为 0.43 和 0.40.窗口算法适用于时间邻近实例较优的情况,但是它在探测到概念漂移之后会丢弃大量旧概念实例,使得窗口过小而不足以建立出可靠的基本分类器.概念回顾算法是为应对概念重现的情况提出的,但是它忽略了时间邻近的新概念实例在分类中的关键作用,当探测到概念漂移之后,通过相似度检索历史概念库中的相似概念,将其对应的分类器作为当前分类器,并丢弃当前窗口中累积的新概念实例.而本文提出的算法既保留了新概念实例,又将历史相似概念作为补充实例,弥补了动态窗口和概念回顾算法在解决细粒度源代码变更实例概念漂移问题中的不足.OCBoost 算法的分类效果优于霍夫丁动态树算法,它们的 Kappa 统计值平均分别为 0.38 和 0.34.由于 OCBoost 算法根据每一步的分类错误情况来调整训练实例和单分类器的权重,将对概念漂移的应对策略融入到每个时间步的权重调整上,从而在一定程度上减轻了概念漂移应对机制的时间延迟.此外它通过组装器做出分类决策,因此具有更好的抵抗噪声的能力.而霍夫丁动态树算法在探测到概念漂移之后需要先构建新概念子树,子树的建立会延迟应对概念漂移的时间.并且,该算法仅使用 1 棵决策树给出分类决策,其抵抗噪声的能力较弱.因此霍夫丁动态树算法分类的效果并不理想.这两种算法虽然具有较为流行的概念漂移应对机制,但是由于它们并不是专门针对时间邻近和概念重现这两种概念漂移情况而构建的,因此分类效果不如前面 4 种算法.由于霍夫丁树仅通过不断加入新实例更新决策树,并不对实例进行裁剪和加权,因此其分类效果最差.

3.4 成本有效性比较实验

为了验证从语句变更层级上预测缺陷的有效性,本实验在第 3.1 节列出的开源项目上重现了 Kim 等人[2](文件级变更)和 Aversano 等人[4](事务级变更)的方法,并采用成本有效性(cost-effectiveness)评估策略对这 3 种方法进行评估.为了保证公平性,本实验选择的修正版本区间和 Kim 等人[2]在实验中选择的区间保持一致,即第 500~1 000 个修正版本(Argouml 选择第 2 500~3 000 个).由于区间较短,他们选择了适用于静态数据的学习机制和十折交叉验证方法.为了保证一致性,本实验环节不考虑概念漂移问题,选用与他们的实验相同的学习机制与验证方法.

在软件工程领域比较不同粒度层级的预测模型优劣时,越来越多的研究者开始考虑使用质量保证成本来评估预测方法优劣[52].基于成本的评估方法最大的优势在于实用性.Arisholm 等人[53]指出,针对某一模块的质量保证成本,与该模块的规模成正比.在使用成本有效性评估方法来评估预测模型时,研究者们往往使用成本有效性曲线.通常横轴代表需要测试和审查的代码行数占系统总代码行数的百分比,纵轴代表发现的缺陷数相对于系统总缺陷数的百分比.如果一种预测方法能够使得审查和测试很小百分比的代码行就可以发现系统中大部分缺陷,则认为该预测方法很有效.本文方法的预测对象是源代码变更.因此在实验中,曲线的横轴代表需要测试和审查的细粒度源代码变更数占系统细粒度源代码变更总数的百分比,纵轴代表发现的引入缺陷细粒度源代码变更数占系统中引入缺陷的细粒度变更总数的百分比.图 6 展示了开源项目在 3 种方法中的成本有效性曲线.其中,事务级、文件级变更以及本文的细粒度变更预测方法分别用 Trans-BP,File-BP 以及 SCC-BP 来表示.可以看出,在投入成本较小时, SCC-BP 方法所对应的成本有效性曲线增长最快,而 Trans-BP 方法增长最慢.当投入成本到达 20%时,SCC-BP 方法能够发现的引入缺陷细粒度源代码变更百分比平均为 79%.通过成本有效性分析可以看出,在本文的所有开源项目中,细粒度变更缺陷预测方法都显著优于事务级和文件级变更缺陷预测方法.

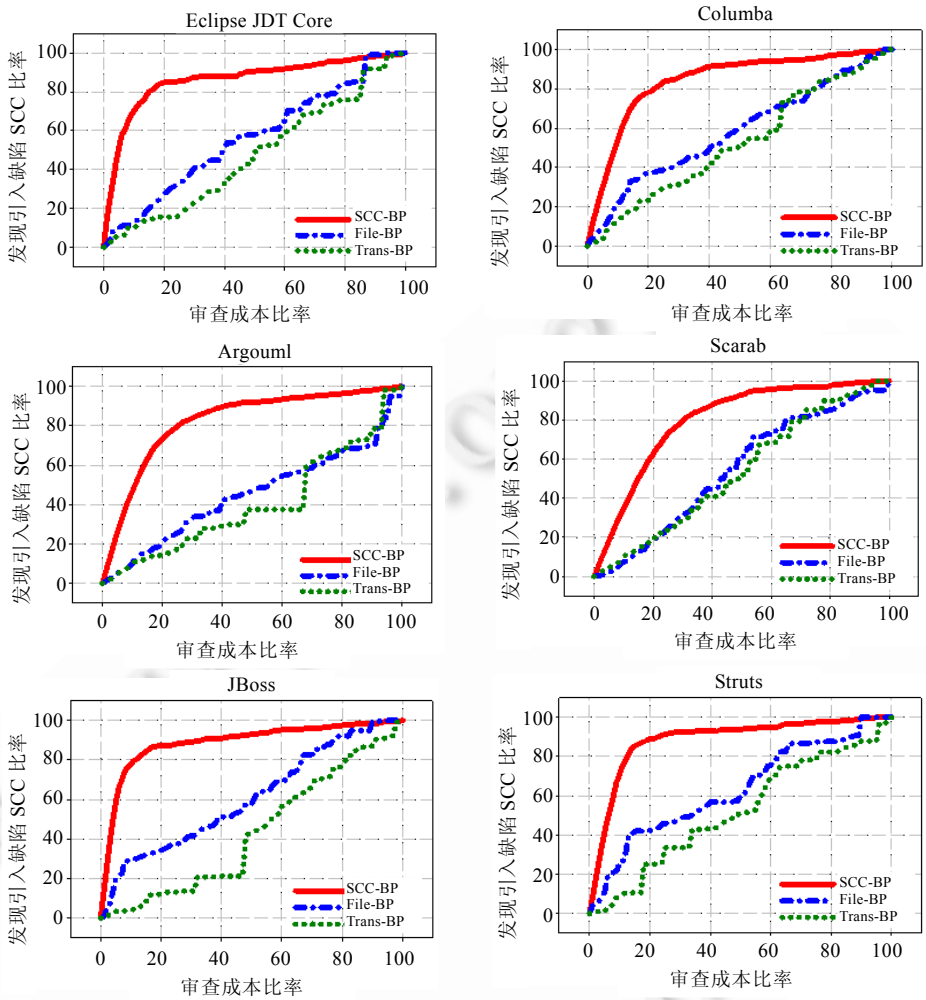


Fig.6 Cost-Effectiveness curves of the three bug prediction methods

图 6 3 种缺陷预测方法的成本有效性曲线

4 结束语

本文提出了一种面向细粒度源代码变更的缺陷预测方法.该方法将细粒度源代码变更作为学习和预测对象,从而有效降低了质量保证成本.通过深入挖掘软件库中的程序结构信息、自然语言语义信息以及历史信息,该方法从变更发生的上下文、变更内容、变更时间以及操纵变更的开发人员这 4 个方面构建特征集.针对学习过程中概念漂移问题的两大特点(时间邻近的实例较优和概念重现性),该方法通过一种伴随概念回顾的动态窗口算法来构建学习机制.实验首先分析了各个特征对缺陷预测的贡献,并发现使用软件库中的自然语言语义信息构建出的特征对缺陷预测具有重要作用.其次,采用特征熵差值计算法分析了概念漂移问题的特点,并通过和几个流行的应对概念漂移问题的学习机制进行比较,验证了本文提出的学习机制的有效性.最后,通过成本有效性验证得出:细粒度源代码变更缺陷预测方法明显优于文件级和事务级变更缺陷预测方法.

在未来的研究中,将进一步开展以下几个方面的工作:

(1) 扩展数据源.本文选用的数据集都是采用开源开发模式的 Java 项目,因此实验验证具有一定的局限性.在商业项目和其他语言实现的项目数据集上验证方法的有效性是下一步工作的重点.

(2) 完善特征集.本文方法在构建开发人员维度的特征时,仅涉及了独立经验(项目经验和文件经验),并未考虑开发人员之间的协作交流以及开发人员所在团队的结构对变更质量的影响.未来计划通过深入挖掘开发人员在版本控制系统中的合作修改信息、邮件列表中的交流信息以及缺陷报告库中的共同评论信息来构建人员网络,并探索网络结构和细粒度变更质量的关系,从而进一步完善现有特征集.

(3) 应用于实际项目的开发过程.本文已采用多个实际开源项目验证了本文方法的有效性,下一步计划将该方法应用于实际的开发过程.主要包括两步:首先,将本文方法集成至实际项目的版本控制系统,在软件人员每次完成变更即将提交时,给出精确至语句级别的风险变更列表,从而帮助软件人员及时有效发现并定位缺陷;其次,通过问卷调查等质性方法收集软件人员的反馈信息,从而进一步验证方法的实用性.

References:

[1] Song QB, Shepperd M, Cartwright M, Mair C. Software defect association mining and defect correction effort prediction. IEEE Trans. on Software Engineering, 2006,32(2):69−82. [doi: 10.1109/TSE.2006.19]

[2] Kim S, Whitehead EJ, Zhang Y. Classifying software changes: Clean or buggy? IEEE Trans. on Software Engineering, 2008,34(2): 181−196. [doi: 10.1109/TSE.2007.70773]

[3] Zimmermann T, Nagappan N, Zeller A. Predicting bugs from history. Software Evolution, 2008,4(1):69−88. [doi: 10.1007/978-3-540-76440-3]

[4] Aversano L, Cerulo L, Grosso CD. Learning from bug-introducing changes to prevent fault prone code. In: Proc. of the Int'l Workshop on Principles of Software Evolution. New York: ACM, 2007. 19−26. [doi: 10.1145/1294948.1294954]

[5] Qin LN. Research on static prediction of software defects [MS. Thesis]. Wuhan: University of Central China Normal, 2011 (in Chinese with English abstract).

[6] Zimmermann T, Premraj R, Zeller A. Predicting defects for Eclipse. In: Proc. of the Predictor Models in Software Engineering. Washington: IEEE Computer Society, 2007. 9−16. [doi: 10.1109/PROMISE.2007.10]

[7] Zhang Y, Yuan ZH, Jiang HY. An early defects prediction approach for object oriented software. Computer Technology and Development, 2010,20(8):37−44 (in Chinese with English abstract).

[8] Zimmermann T, Nagappan N. Predicting subsystem failures using dependency graph complexities. In: Proc. of the 18th IEEE Int'l Symp. on Software Reliability. Washington: IEEE Computer Society, 2007. 227−236. [doi: 10.1109/ISSRE.2007.19]

[9] Abdulsalam H. Streaming random forests [Ph.D. Thesis]. Kingston: Queen's University, 2008.

[10] Sliwerski J, Zimmermann T, Zeller A. When do changes induce fixes? ACM Sigsoft Software Engineering Notes, 2005,30(4):1−5. [doi: 10.1145/1082983.1083147]

[11] Eyolfson J, Tan L, Lam P. Do time of day and developer experience affect commit bugginess? In: Proc. of the 8th IEEE Working Conf. on Mining Software Repositories. New York: ACM, 2011. 153−162. [doi: 10.1145/1985441.1985464]

[12] Rahman F, Devanbu PT. Ownership, experience and defects: A fine-grained study of authorship. In: Proc. of the 33rd Int'l Conf. on Software Engineering. New York: ACM, 2011. 491−500. [doi: 10.1145/1985793.1985860]

[13] Abreu R, Premraj R. How developer communication frequency relates to bug introducing changes. In: Proc. of the Joint Int'l Workshop on Principles of Software Evolution. New York: ACM, 2009. 153−158. [doi: 10.1145/1595808.1595835]

[14] Tsymbal A. The problem of concept drift: Definitions and related work. Technical Report, TCD-CS-2004-15, Dublin: Trinity College at Dublin, 2004.

[15] Lane T, Brodley CE. Temporal sequence learning and data reduction for anomaly detection. ACM Trans. on Information and System Security, 1999,2(3):295−331. [doi: 10.1145/322510.322526]

[16] Giacomini R, Rossi B. Detecting and predicting forecast breakdowns. The Review of Economic Studies, 2009,76(2):669−705. [doi: 10.1111/j.1467-937X.2009.00545.x]

[17] Delany S, Cunningham P, Tsymbal A. A comparison of ensemble and case-base maintenance techniques for handling concept drift in spam filtering. In: Proc. of the 19th Int'l Conf. on Artificial Intelligence. Florida AI Research Society, 2006. 340−345.

[18] Billsus D, Pazzani M. A hybrid user model for news story classification. In: Proc. of the 7th Int'l Conf. on User Modeling. New York: Springer-Verlag, 1999. 99−108. [doi: 10.1007/978-3-7091-2490-1_10]

[19] Koren Y. Collaborative filtering with temporal dynamics. In: Proc. of the 15th Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM, 2009. 447−456. [doi: 10.1145/1557019.1557072]

[20] Lazarescu M, Venkatesh S. Using selective memory to track concept effectively. In: Proc. of the Int'l Conf. on Intelligent Systems and Control. ACTA, 2003. 14−20.

[21] Koychev I. Gradual forgetting for adaptation to concept drift. In: Proc. of the ECAI 2000 Workshop Current Issues in Spatio-Temporal Reasoning. 2000. 101−106.

[22] Pelossof R, Jones M, Vovsha I, Rudin C. Online coordinate boosting. In: Proc. of the 12th Int'l Conf. on Computer Vision Workshops. New York: IEEE, 2009. 1354−1361. [doi: 10.1109/ICCVW.2009.5457454]

[23] Ekanayake J, Tappolet J, Gall H, Bernstein A. Tracking concept drift of software projects using defect prediction quality. In: Proc. of the 6th IEEE Working Conf. on Mining Software Repositories. Washington: IEEE Computer Society, 2009. 51−60. [doi: 10.1109/MSR.2009.5069480]

[24] Chrupala G. Learning from evolving data streams: Online triage of bug reports. In: Proc. of the 13th Conf. of the European Chapter of the Association for Computational Linguistics. Stroudsburg: Association for Computational Linguistics, 2012. 613−622. [doi: 10.1023/A:1018046501280]

[25] Fluri B, Würsch M, Pinzger M, Gall H. Change distilling: Tree differencing for fine-grained source code change extraction. IEEE Trans. on Software Engineering, 2007,33(11):725−743. [doi: 10.1109/TSE.2007.70731]

[26] Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A. Defect prediction from static code features: Current results, limitations, new approaches. Automated Software Engineering, 2010,17(4):375−407. [doi: 10.1007/s10515-010-0069-5]

[27] McCabe TJ. A complexity measure. IEEE Trans. on Software Engineering, 1976,4:308−320. [doi: 10.1109/TSE.1976.233837]

[28] Halstead MH. Elements of Software Science. Amsterdam: Elsevier North-Holland Press, 1977.

[29] Lukins SK, Kraft NA, Etzkorn LH. Source code retrieval for bug localization using Latent Dirichlet allocation. In: Proc. of the 15th Working Conf. on Reverse Engineering. Washington: IEEE Computer Society, 2008. 155−164. [doi: 10.1109/WCRE.2008.33]

[30] Maletic JI, Marcus A. Supporting program comprehension using semantic and structural information. In: Proc. of the 23rd Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2001. 103−112. [doi: 10.1109/ICSE.2001.919085]

[31] Blei DM, Ng AY, Jordan MI. Latent Dirichlet allocation. Journal of Machine Learning Research, 2003,3:993−1022.

[32] Thomas SW, Adams B, Hassan AE, Blostein D. Modeling the evolution of topics in source code histories. In: Proc. of the 8th Working Conf. on Mining Software Repositories. New York: ACM, 2011. 173−182. [doi: 10.1145/1985441.1985467]

[33] Canini KR, Shi L, Griffiths TL. Online inference of topics with Latent Dirichlet allocation. In: Proc. of the Int'l Conf. on Artificial Intelligence and Statistics. 2009. 65−72.

[34] Yan R, Huang C, Tang J, Zhang Y, Li X. To better stand on the shoulder of giants. In: Proc. of the 12th ACM/IEEE-CS Joint Conf. on Digital Libraries. New York: ACM, 2012. 51−60. [doi: 10.1145/2232817.2232831]

[35] Graves TL, Karr AF, Marron JS, Siy HP. Predicting fault incidence using software change history. IEEE Trans. on Software Engineering, 2000,26(7):653−661. [doi: 10.1109/32.859533]

[36] Pan K, Kim S, Whitehead EJ. Toward an understanding of bug fix patterns. Empirical Software Engineering, 2009,14(3):286−315. [doi: 10.1007/s10664-008-9077-5]

[37] Chen T, Thomas SW, Nagappan M, Hassan AE. Explaining software defects using topic models. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories. 2012. 189−198. [doi: 10.1109/MSR.2012.6224280]

[38] Hindle A, Godfrey MW, Holt RC. Mining recurrent activities: Fourier analysis of change events. In: Proc. of the 31st Int'l Conf. on Software Engineering. New York: IEEE, 2009. 295−298. [doi: 10.1109/ICSE-COMPANION.2009.5071005]

[39] Widmer G, Kubat M. Learning in the presence of concept drift and hidden contexts. Machine Learning, 1996,23(1):69−101. [doi: 10.1023/A:1018046501280]

[40] Gama J, Medas P, Castillo G, Rodrigues P. Learning with drift detection. In: Proc. of the SBIA Brazilian Symp. on Artificial Intelligence. Berlin: Springer-Verlag, 2004. 286−295. [doi: 10.1007/978-3-540-28645-5_29]

[41] Schilling MF. Multivariate two-sample tests based on nearest neighbors. Journal of the American Statistical Association, 1986,81(395):799−806. [doi: 10.1080/01621459.1986.10478337]

[42] Hassan AE, Holt RC. The top ten list: Dynamic fault prediction. In: Proc. of the 21st IEEE Int'l Conf. on Software Maintenance. Washington: IEEE Computer Society, 2005. 263−272. [doi: 10.1109/ICSM.2005.91]

[43] Yang Y, Pedersen JO. A comparative study on feature selection in text categorization. In: Proc. of the 14th Int'l Conf. on Machine Learning. Morgan Kaufmann Publishers, 1997. 412−420.

[44] Widmer G, Kubat M. Effective learning in dynamic environments by explicit context tracking. In: Proc. of the 6th European Conf. on Machine Learning. London: Springer-Verlag, 1993. 69−101. [doi: 10.1007/3-540-56602-3_139]

[45] Albert B, Ricard G. Adaptive parameter-free learning from evolving data streams. In: Proc. of the 8th Int'l Symp. on Intelligent Data Analysis. Berlin: Springer-Verlag, 2009. 249−260. [doi: 10.1007/978-3-642-03915-7_22]

[46] Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In: Proc. of the 2001 Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM, 2001. 97−106. [doi: 10.1145/502512.502529]

[47] Melville P, Shah N, Mihalkova L, Mooney RJ. Experiments on ensembles with missing and noisy data. In: Proc. of the Workshop on Multiple Classifier Systems. 2004. 293−302.

[48] Breiman L. Random forests. Machine Learning, 2001,45(1):5−32. [doi: 10.1023/A:1010933404324]

[49] Bifet A, Holmes G, Kirkby R, Pfahringer B. Moa: Massive online analysis. The Journal of Machine Learning Research, 2010,99: 1601−1604.

[50] Cohen J. A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 1960,20(3):37−46. [doi: 10.1177/001316446002000104]

[51] Wu TT. Research on classifier performance evaluation [MS. Thesis]. Beijing: Beijing Jiaotong University, 2010 (in Chinese with English abstract).

[52] Hata H, Mizuno O, Kikuno T. Bug prediction based on fine-grained module histories. In: Proc. of the 34th Int'l Conf. on Software Engineering. New York: IEEE, 2012. 200−210. [doi: 10.1109/ICSE.2012.6227193]

[53] Arisholm E, Briand LC, Johannessen EB. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software, 2010,83(1):2−17. [doi: 10.1016/j.jss.2009.06.055]

附中文参考文献:

[5] 秦丽娜.软件缺陷静态预测研究[硕士学位论文].武汉:华中师范大学,2011.

[7] 张垚,袁志海,江海燕.一种面向对象软件缺陷的早期预测方法.计算机技术与发展,2010,20(8):37−44.

[51] 武婷婷.分类器性能评价研究[硕士学位论文].北京:北京交通大学,2010.

原子(1983－),女,河南开封人,博士生,主要研究领域为软件工程,软件库挖掘,软件演化,缺陷预测.
E-mail: yuanzi@sei.buaa.edu.cn

刘超(1958－),男,教授,博士生导师,CCF高级会员,主要研究领域为软件工程,软件测试,软件演化,数据挖掘.
E-mail: liuchao@buaa.edu.cn

于莉莉(1978－),女,博士,主要研究领域为软件工程,回归测试用例选择,扎根理论.
E-mail: xueer-123@263.net