计算机科学
COMPUTER SCIENCE

基于日志信息的不可重复构建原因分类

马钊，刘东，任志磊，江贺

引用本文

马钊，刘东，任志磊，江贺.基于日志信息的不可重复构建原因分类[J]．计算机科学, 2022, 49(12): 109-117.

MA Zhao, LIU Dong, REN Zhi-lei, JIANG He. Classification of Unreproducible Build Causes Based on Log Information [J]. Computer Science, 2022, 49(12): 109-117.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220300227

基于日志信息的不可重复构建原因分类

马 钊 刘 东 任志磊 江 贺
大连理工大学软件学院 辽宁 大连 116620
(zhao_ma@qq.com)

摘 要 可重复构建指在预定义的构建环境下重新创建二进制工件的能力。由于可重复构建具有保证软件构建环境安全和提高软件构建和分发效率的作用,许多开源软件存储库(如 Debian)开展了软件可重复构建实践。然而,由于缺乏足够的判断信息和源文件的复杂多样,确定软件不可重复构建的原因仍是一项费时费力的工作。为此,研究了基于机器学习的软件不可重复构建原因的分类检测。研究了 4 种典型的不可重复构建原因,即时间戳、文件顺序、随机性和语言环境。利用 word2vec 产生的词向量对文本日志进行表示,然后配合 logistic 回归模型,对差异日志和构建日志合并的文本语料进行学习和训练,从而实现对不可重复构建原因的自动分类。对算法进行了实现,并在 671 个不可重复构建的 Debian 软件包上进行实验,实验结果表明,该方法达到了 80.75%的宏平均精度和 86.07%的宏平均召回率,优于其他常用的机器学习算法。此外,还分析了差异日志和构建日志的相关性和重要性,实验结果表明两者对不可重复构建原因的分类都非常重要,缺一不可。该方法为不可重复构建原因自动分类提供了可靠的研究依据。

关键词:可重复构建;原因分类;差异日志;构建日志;机器学习
中图法分类号 TP311

Classification of Unreproducible Build Causes Based on Log Information

MA Zhao,LIU Dong,REN Zhi-lei and JIANG He
School of Software Engineering,Dalian University of Technology,Dalian,Liaoning 116620,China

Abstract Reproducible build is the ability to recreate binary artifacts in a predefined build environment. Due to the role of reproducible build in ensuring the security of software construction environment and improving the efficiency of software construction and distribution,many open source software repositories(such as Debian) have carried out software reproducible build practice. However,due to the lack of sufficient judgment information and the complexity and diversity of source files,it is still a time-consuming and laborious challenge to determine why software can not be built reproducibly. In order to overcome this challenge,this paper studies the classification and detection of software unreproducible build causes based on machine learning. This paper studies four typical reasons for unreproducible build,namely timestamp,fileordering,randomness and locale. This method uses the word vector generated by word2vec to represent the text log,and then cooperates with the logistic regression model to learn and train the text corpus combined with the difference log and the build log,so as to realize the automatic classification of the causes of unreproducible build. In this paper,the algorithm is implemented and tested on 671 unreproducible build Debian software packages. Experimental results show that our method achieves a macro average precision of 80.75% and a macro average recall of 86.07%,which are better than other commonly used machine learning algorithms. In addition,we also analyze the relevance and importance of difference log and build log. Result indicates that both of them are significant for the classification of unreproducible build causes. This method provides a reliable research basis for automatic classification of unreproducible build causes.

Keywords Reproducible build,Cause classification,Difference log,Build log,Machine learning

1 引言

软件构建指软件从源代码编译生成对应二进制工件(Artifact)的过程。可重复构建指任何一方在相同的构建环境下,使用相同的构建指令,对相同的源代码进行构建,都可以生成特定工件逐位相同的副本[1]。相同的源代码通常指经过软件项目托管平台审查的特定签出版本。构建环境通常包括构建工具及其依赖项、构建配置和构建系统所使用的环境变量。构建生成的工件包含可执行文件、派生的软件包和相应的文件系统镜像,但通常不包括构建日志和类似的辅助输出。实现软件的可重复构建作为近年来出现的一种重要的软件开发实践,其目的是在源代码和构建的工件间建立可信的验证路径。

在软件安全愈发重要的当下,可重复性对于构建环境的

到稿日期:2022-03-23 返修日期:2022-05-31
通信作者:江贺(jianghe@dlut.edu.cn)

安全非常重要。对于软件生态系统而言，恶意软件会攻击软件开发和构建过程，将自身的副本伴随更新分发到成千上万的机器，会造成灾难性后果。例如，在著名的 XcodeGhost 事件[2]中，超过 4 000 个 iOS 应用程序被假冒的开发环境 XcodeGhost 感染。ShadowHammer[3]以攻击华硕预装的实时更新应用程序为初始感染源，向用户设备注入后门程序，导致全球 50 万用户的网络安全受到威胁。类似的攻击事件近年来发生了 174 起[4]以上。对于这些攻击，可重复构建提供了一种可行的解决方案，如果从给定的源出发总是能获得确定的结果，那么多个第三方就能对当前的软件是否受到攻击达成共识。但不幸的是，由于现实构建环境的复杂性和不确定性，即使多次重建应用程序也很难获得两个相同的二进制文件。

在软件集成和持续集成[5]的背景下，可重复构建有助于提高构建和更新效率。如果从源文件构建生成的目标工件始终相同，就能避免在源文件未更改的情况下重新构建该目标工件，从而加快构建速度，并降低软件更新带来的带宽需求。由于其益处显著，许多开源软件项目已经启动了可重复构建验证过程。这些软件项目包括 Debian[6] 和 Guix[7]，以及 FreeBSD[8]等软件系统。特别是，为了验证软件包在不同构建环境中是否可重复构建，可重复构建验证工具链故意引入指定构建环境扰动。例如，disorderfs 能够将不确定性引入文件系统元数据[9]，用于验证文件排序问题是否会影响构建的可重复性。faketime 能够扰动时间，验证构建过程是否受构建时间的影响。自 2014 年以来，Debian 可重复软件包的数量一直在稳步增长，截至 2021 年 11 月，超过 83.4% 的 Debian 软件包可以重复构建[10]。

尽管开源软件项目为可重复构建做了很多努力，但仍有大量的软件包不可重复构建。以 Debian 的 AMD64 架构的不稳定版本为例，截至 2021 年 11 月，仍有超过 3 900 个软件包无法重复构建[10]。检测不可重复构建的原因是进行软件修复的前提条件，但当前，开发人员对不可重复构建原因的检测主要依赖于过往经验。由于一个软件包的源文件可能成百上千，这个过程费时费力。大量软件包的不可重复构建意味着对不可重复构建的原因检测效率提出了挑战。

为了实现高效识别不可重复构建的原因，我们以构建工件差异日志为语料库，建立分类模型。具体地，不可重复构建原因分类问题可以建模为一个文本分类问题，一种可能的做法是使用差异日志和源文件作为语料库，但是它具有两个显而易见的缺点：第一，源文件数目庞大且文件类型复杂，不仅包含源代码还存在其他类型的文件，如脚本、Makefiles、配置文件等，从中抽取特征非常困难；第二，并不是所有的源文件都会参与到软件包的编译过程中，如注释、被忽略的源文件等，这可能会对最终的分类结果产生误导。另一种做法是使用构建日志和差异日志作为语料库，相比静态的源文件，构建日志类型单纯且能反映出确切参与软件包编译的文件和编译过程中的动态信息，对差异日志提供的信息是很好的补充，很好地避免了以源文件为语料库带来的缺陷。

本文提出了一个软件包不可重复构建原因分类框架。给定一个不可重复构建的软件包，以其中包含两个不同的构建工件作为输入，该框架会对软件包的不可重复构建原因进行预测。该框架分为预处理、构建特征向量和模型训练 3 个部分。预处理部分以软件包的差异日志和构建日志作为输入，基于两者中相似的语句，获得文本语料库。构建特征向量部分以文本语料作为输入，利用 word2vec[11] 获得词向量并最终获得每个软件包的特征向量。模型训练部分以用于训练的特征向量集和参数作为逻辑回归算法输入，经过参数优化后获得性能最佳的分类器，然后基于测试特征向量集输出混淆矩阵。

为了评估所提框架，我们采用 671 个不可重复构建的软件包作为真实的数据集。由于这些软件包已经被修复，因此我们确切地知道每个软件包过去影响不可重复构建的原因。将本文提出的方法与支持向量机、朴素贝叶斯、决策树和随机森林等现有分类器进行比较。实验结果表明，所提算法在检测软件包不可重复构建原因方面达到了 80.75% 的精度和 86.07% 的召回率，优于比较的其他算法。此外，我们分别对仅基于静态差异日志或动态构建日志的语料库建立算法模型，并与本文的算法模型进行比较，结果表明，差异日志和构建日志所提供的特征对软件包不可重复构建原因的分类均具有重要作用，两者相互补充。为了进一步评估所提方法的有效性，还在 Guix 软件存储库中检验了本文提出的框架。

本文的主要贡献如下：

(1)基于静态差异日志和动态构建日志信息提出了一种能有效对不可重复构建原因进行分类的方法，这是第一个研究不可重复构建原因自动分类的工作。

(2)基于该方法实现了一个原型，并对 Debian 存储库中的 671 个软件包进行了不可重复构建原因评估，实验结果表明，所提方法是有效的。

(3)此外，还对差异日志和构建日志的重要性进行了度量，以研究对不可重复构建的原因分类的影响。为了验证所提方法的有效性，我们进行了跨项目实验，实验结果表明，所提方法可以获得良好的性能。

本文第 2 节介绍了这项工作的背景和动机；第 3 节介绍了所提方法的框架；第 4 节对实验装置进行了描述；第 5 节给出了实验结果；第 6 和第 7 节描述了对研究有效性的威胁和相关工作；最后总结全文。

2 背景

本节描述可重复构建基本工作流、4 种不可重复构建原因的定义以及不可重复构建原因分类动机。

2.1 可重复构建基本工作流

从 Debian 的可重复构建实践入手，对可重复构建框架进行回顾。图 1 给出了验证软件包可重复构建的基本工作流程[12]。在开始可重复构建验证流程之前，首先需要确保构建所需的系统体系结构、构建中所依赖的软件包(包含其递归依赖项)以及构建路径，使得构建过程可以顺利进行。其次，源文件是在两个预定义的构建环境下进行编译的，构建环境通过设置环境变量和软件配置来实现，在 Debian 中是通过 reprotest 工具来实现的。表 1 列出了 Debian 可重复构建实践中常见的环境变量和软件配置片段[13]。在可行的构建环境中分别引入环境变量，然后进行构建，保存构建生成的二进制文件(步骤 1 和 2)。利用比对工具 diffoscope[14]，diffoscope

是由可重复构建团队开发的一种用于将各种二进制格式转换为更易于阅读的形式以进行比较的工具，对两轮构建所得的二进制文件进行比对。如果散列值相同，则说明该软件包可重复构建；如果散列值不同，则输出差异文件（步骤 3）。开发人员基于源文件和差异文件对可能的原因进行分析检测，然后定位到具体的问题文件或命令（步骤 4）。开发人员对可能的问题文件命令进行修复，试图消除影响可重复性构建的因素（步骤 5），然后开始新一轮可重复构建验证流程，直至该软件包构建可重复。

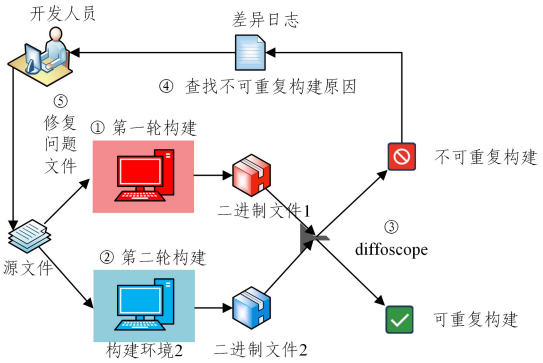

图 1　软件包可重复构建的基本工作流

Fig. 1　Basic workflow of reproducible build software package

表 1　环境变量片段

Table 1　Environment variable fragment

Variation	First build	Second build
env TZ	“GMT+12”	“GMT-14”
env LANG	“C. UTF-8”	“et_EE. UTF-8”“nl_BE. UTF-8”
domain	debian. net	i-capture-the-domainname
umask	0022	0002
filesystem	filesystem	disorderfs
—	—	—

2.2　定义

时间戳：时间戳是导致软件包不可重复构建的最大原因。记录时间戳的软件包构建结果往往是不一致的，因为软件包的多轮构建并不可能时间同步。大多数构建工具（如 Cmake）、文件系统（如 ext2）、归档命令（如 tar）都倾向于记录当前日期和时间，这会导致软件产品的构建不可重复。对于时间戳所导致的不可重复构建，目前有 3 种解决方案，即不使用时间戳、设置标准环境变量 SOURCE_DATE_EPOCH[15] 和对输出结果进行后处理[16]。

文件顺序：不确定的输入和输出顺序会导致软件包的不可重复构建，如 Makefile 文件中常用的通配符函数 wildcard 输出顺序不稳定。Perl 的哈希、Python 的字典等类型数据结构在每次运行时都以不同的顺序输出 key 值，以限制算法复杂度攻击[17]。但在软件包的可重复构建验证中，这往往会导致其不可重复。解决方案为对其进行排序，显式地进行输入和输出。

随机性：记录随机数据的构建结果往往是不可重复的，在不同的场景下有不同的解决方案。如针对使用随机数据作为构建输入，则使用具有预定值的伪随机数生成器，当构建时，从预定义的文件或版本控制系统中读取该值[18]。

语言环境：Locale 是根据计算机用户所使用语言、所在国家或者地区，以及当地文化传统所定义的一个软件运行时的语言环境[18]，受语言环境影响的工具会使其成为可重复构建问题的根源。如时间格式化函数将根据当前区域设置输出，排序函数受环境变量 *LC_COLLATE* 的影响等。对于语言环境所导致的不可重复构建问题，解决方案是使用环境变量 *LC_ALL*[19]。

2.3　动机

针对时间戳、文件顺序、随机性和语言环境这 4 种类型的不可重复构建问题，开发人员在长期实践中已经积累了较多的解决方案，可以使得不可重复构建的软件包快速进行修复。但是，由于导致软件包不可重复构建的原因众多，开发人员很难凭手工劳作在成千上万的源文件中确定不可重复构建的原因。因此，检测软件包不可重复构建的原因具有现实意义。本文应用分类算法对软件包不可重复构建的原因进行分类检测。

3　方法

本文重点研究 4 种导致软件包不可重复构建的原因，结合差异日志和构建日志来构建算法模型，并通过该模型对软件包不可重复构建的原因进行分类预测。本节将从日志获取、数据预处理、特征向量的构建和基于日志信息的不可重复构建原因分类算法这 4 个方面来详细描述本文提出的分类框架。图 2 描述了基于日志信息的不可重复构建原因分类框架的基本工作流程。

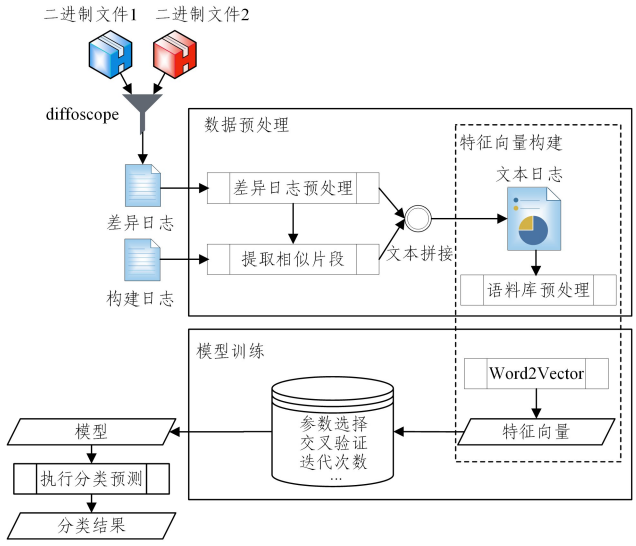

图 2　不可重复构建原因分类框架

Fig. 2　Classification framework of unreproducible build causes

3.1　日志获取

首先，在给定的构建环境中对软件包进行两次构建，获得相对应的二进制文件 1 和二进制文件 2。然后，使用 diffoscope 工具生成差异日志，图 3 给出了软件包 cmtk 的差异日志片段，差异日志详尽列举了两次构建中不一致的文件名称以及相应的递归展开。但是，单纯使用静态的差异日志来判断软件包不可重复构建的原因具有局限性，因此，研究中通过获取软件包在构建过程中的构建信息来建立构建日志。图 4 给出了软件包 cmtk 的构建日志片段。可以观察到，动态获取

的构建日志具有更多的信息并且对静态的差异日志信息进行了补充。更具体地说，首先根据 make 命令生成的“进入/离开目录”标记(见图 3 中的第 1 行和第 12 行)，将构建日志拆分为构建命令段，通过此操作，可以将在同一目录下调用的命令组合在一起，作为差异日志的语料库的补充。尽管在两个构建环境中，有两个版本的构建日志，但由于本研究只对 build 命令感兴趣，因此选择任何一个版本的构建日志都不会对结果产生影响。

```
1. /usr/lib/cmtk/bin/epiunwarp
2. readelf --wide --notes {}
3. @@ -3,8 +3,8 @@
4. Owner   Data sizeDescription
5. GNU   0x00000010NT_GNU_ABI_TAG
6. OS:Linux,ABI:2.6.32
7. Displaying notes found in:.note.gnu.build-id
8. Owner   Data sizeDescription
9. GNU   0x00000014NT_GNU_BUILD_ID
10. Build ID:77cbf03cfe35200ee4e9a555a4e74fe415fd9f46
11. Build ID:8f2d45e8d2e6006c7bf1934df4bcd6c2eda5d9c9
```

图 3 cmtk 的差异日志片段

Fig. 3 cmtk's difference log fragment

```
1. make[1]:Entering directory '/build/1st/cmtk-3.2.2/obj-x86_64-
2. […]
3. make[3]:Entering directory '/build/1st/cmtk-3.2.2/obj-x86_64-
4. Building CXX object apps/CMakeFiles/mk_adni_phantom.cxx.o
5. cd /build/1st/cmtk-3.2.2/obj-x86_64-linux-gnu/apps && /usr/bin/
6. […]
7. cd /build/1st/cmtk-3.2.2/obj-x86_64-linux-gnu/apps && /usr/bin/
8. Linking CXX executable../bin/epiunwarp
9. /usr/bin/c++ -fopenmp […] -o../bin/epiunwarp
10. make[3]:Leaving directory '/build/1st/cmtk-3.2.2/obj-x86_64-
11. […]
12. make[1]:Leaving directory '/build/1st/cmtk-3.2.2/obj-x86_64-
```

图 4 cmtk 的构建日志片段

Fig. 4 cmtk's build log fragment

3.2 数据预处理

在进入建立模型过程之前，需要对现有的获取到的差异日志和构建日志进行数据处理，以保证数据的完整性和可用性。以 cmtk 的差异日志为例，其中包含超过 4 000 行信息，但可用于判断不可重复构建的原因的信息却很少，因此将与研究不相关的部分予以清理，只保留在差异日志中所提示的不一致的行信息(如图 3 中前面有“-,+”符号的行)。然后，对于相对应的两行不一致信息进行合并，对于两行中相同的部分，只保留一次，将不同的部分依次排列，最终提取出一行有用信息。此外，实验中发现部分行信息中保留了散列值以及文件大小等信息，这些信息非常容易受到环境信息的扰动而发生变化，但对于研究软件包不可重复构建的原因而言，属于不可控的因素(只要有不可重复构建发生，这些信息就会不同且无规律可循)，因此在研究中将其剔除。

对于构建日志，根据 make 生成的“进入/离开目录”标记(见图 4 的第 1 行和第 12 行)，将构建日志拆分为构建命令片段。基于此操作，可以将在同一目录下调用的命令组合在一起，作为扩充语料库的文件。在本文中，对于构建命令片段 d 中的一个词汇 t，其 TF-IDF 值[20]是基于 $f_{t,d}\times\log((n+1)/(n_t+1)+1)$ 计算的，其中 $f_{t,d}$ 表示词汇 t 在构建命令片段 d 中出现的次数，n_t 表示词汇 t 出现的构建命令片段数，n 表示构建命令片段总数。定义 TF-IDF 后，将每个构建命令片段表示为一个向量，并计算其与每行差异日志的余弦相似值[21]。对于处理后的每行差异日志信息，利用信息检索模型[22]在构建日志片段中搜索相关文件，提取余弦相似值最高的构建日志片段，将其拼接在当前该行差异日志信息后作为最新的文本日志。

运行实例：以 cmtk 为例，将 cmtk 的差异日志中第 520 行“-./usr/lib/cmtk/bin/epiunwarp”和第 1 432 行“+./usr/lib/cmtk/bin/epiunwarp”合并成一行不一致信息“usr/lib/cmtk/bin/epiunwarp”；然后基于获得的该行不一致信息，利用信息检索模型，在构建日志片段中进行搜索，寻找到相关度最高的构建命令“/usr/bin/c++ -fopenmp […] -o../bin/epiunwarp”(见图 4 第 9 行)；最后，将该构建命令所在片段拼接在不一致信息“./usr/lib/cmtk/bin/epiunwarp”后形成新的文本日志中。

3.3 特征向量构建

在数据预处理阶段，通过对每个不可重复构建软件包的差异日志和构建日志的处理获得相应的文本日志。为了后续机器学习算法能够基于软件包文本日志来训练预测模型，需要提取出每个软件包文本日志的特征向量。为了获得特征向量，首先要对文本日志进行词向量处理。

Word2vec 是由 Google 的 Mikolov 等[11]于 2013 年提出的一种神经网络概率语言模型，用于快速地对文本进行训练并获得低维词向量表示。Word2vec 有连续词袋 CBOW 和 Skip-Gram 两种模型，都是仅包含输入层、隐藏层和输出层的轻量级神经网络。这两种模型的不同之处主要在于输入输出，连续词袋模型 CBOW 是在确定词 W_t 的上下文 W_{t-2}，W_{t-1} 和 W_{t+1}，W_{t+2} 的情况下对当前词 W_t 进行预测，而 Skip-Gram 模型是在确定词 W_t 的情况下对其上下文 W_{t-2}，W_{t-1} 和 W_{t+1}，W_{t+2} 进行预测，图 5 给出了两种模型的架构。Word2vec 采用 Hierarchical softmax 和负采样两种策略来提升训练的性能[23]。

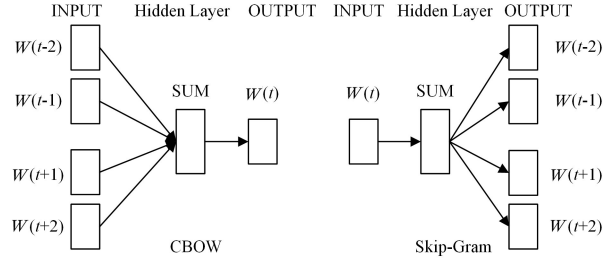

图 5 CBOW 模型和 Skip-Gram 模型

Fig. 5 CBOW and Skip-Gram models

Word2vec 作为一个 NLP 处理工具，可以将文本中的词向量化，在词与词之间定量度量它们之间的关系，挖掘词之间的联系。本文选择 Word2vec 模型训练得到文本日志的词向量，然后根据词向量来获得文本日志的特征向量。图 2 中的

特征向量构建模块展示了文本日志特征向量的构建流程。

(1)收集所有软件包的文本日志,形成语料库。

(2)对获得的语料库进行数据预处理,包括去除标点符号,文件分隔符替换成空格,停用词等一系列清洗和转换工作。最后,将处理后的文档进行保存,文档全部由词汇组成。

(3)将处理后的文本数据利用第三方库 Gensim 的模型 word2vec 进行训练,从而得到词汇的词向量表。

(4)在完成对词汇的词向量表的更新后,取每个文本日志的词向量平均值作为该文本日志的特征向量。

3.4　不可重复构建原因分类算法

3.4.1　逻辑回归算法

本文将基于差异日志的不可重复构建原因检测视为一个多分类问题,而逻辑回归作为经典的多分类机器学习算法能够适应本研究所在场景。普通的逻辑回归是一个二分类问题,结合本研究的多分类场景,我们将其推广至多分类。

假设集合$\{1,2,\cdots,K\}$是离散型随机变量 Y 的取值集合,即 Y 共有 K 类,则可以推导出多分类场景下逻辑回归模型为:

$$P(Y=j|\boldsymbol{x})=\frac{e^{w_j-x}}{1+\sum_{i=1}^{K-1}e^{w_j-x}},j=1,2,3,\cdots,K-1 \tag{1}$$

$$P(Y=K|\boldsymbol{x})=\frac{1}{1+\sum_{i=1}^{K-1}e^{w_j-x}},x\in R^{n+1},w_j\in R^{n+1} \tag{2}$$

其中,$Y=j$ 是取 $1,2,3,\cdots,K-1$ 中的一类,$Y=K$ 指第 K 类,$P(Y=K|x)$即 1 减去 Y 取其他 j 值的概率。多分类逻辑回归的 softmax 函数如下:

$$\mathrm{softmax}(\boldsymbol{x})=\frac{e^j}{\sum_{j=1}^{C}e^j} \tag{3}$$

基于差异日志不可重复构建原因分类算法的伪代码如算法 1 所示。

算法 1　不可重复构建原因分类算法

```
输入:二进制文件包 binary1,binary2,参数
输出:分类结果 result
1. /* 数据预处理模块 */
2. diff_log ← diffoscope(binary1,binary2)
3. diff_log ← process_diff_log(diff_log)
4. build_log ← process_build_log(build_log)
5. relevant_build_log ← retrieve_revelant(diff_log,build_log)
6. text_log ← splice(diff_log,relevant_build_log)
7. /* 特征向量构建模块 */
8. corpus ← preprocess(text_log)
9. word_vectors ← word2vec(corpus)
10. feature_vectors ← build_feature _vector(corpus,word vectors)
11. /* 模型训练模块 */
12. train_data,test_data ← split(feature_vectors)
13. model_parameters ← tune(train_data,maxIts,multi_class,
    solver,…)
14. train_mode ←train(Logistic,train_data,model_parameters)
15. num ← length of test_data
16. while count < num do
17.    result ← predict(train_model,test_data)
18.    count ← count+1
19. end while
20. return result
```

3.5　参数优化

为了取得更好的实验效果,特征向量构建模块选取 Skip-Gram 模型,词向量的维数设置为 100 维,迭代次数设置为 10。在模型训练模块,本文采用交叉验证来对多种参数组合进行验证,找到其中最佳的参数组合。在执行交叉验证时,需要从训练数据集中划分出验证集以评估模型的性能。在模型训练时,逻辑回归需要调优的参数包括最大迭代次数 *maxIts*、正则化选择参数 *penalty*、优化算法选择参数 *solver*、分类方式选择参数 *multi_class*、类型权重参数 *class_weight* 以及样本权重参数 *sample_weight*。

4　实验

本文实验过程如下:首先,收集数据。本文共收集了 671 个不可重复构建的软件包,并利用工具链获得这些软件包的差异日志和构建日志。将数据集定义为:

$$\boldsymbol{D}=\{l_1,l_2,l_3,\cdots,l_i\},i=1,2,3,\cdots \tag{4}$$

其中,$\boldsymbol{D}$ 是我们的数据集,由一系列的日志文件组成,l_i 表示一个被标记为特定的不可重复构建原因的软件包的差异日志和构建日志。

然后,对于数据集 $\boldsymbol{D}$ 中的每个 l_i,提取其特征值表示为一个特征向量。将这些指标定义为:

$$\boldsymbol{X}=\{x_1,x_2,x_3,\cdots,x_j\},j=1,2,3,\cdots \tag{5}$$

其中,$\boldsymbol{X}$ 是该软件包文本日志的特征向量,$\boldsymbol{x}_j$ 是特征向量的分量,j 的大小由特征向量的维数确定。

接下来,构建一个用于训练的模型,使用交叉验证进行参数优化以选择最佳参数,将优化后的模型用测试数据集进行评估。

4.1　数据收集

本文参考文献[24]工作中的数据集,其中包含 671 个不可重复构建的软件包。该数据集是通对 Debian 的缺陷跟踪系统(Bug Tracking System,BTS)的挖掘构建的。Debian 是唯一提供过去版本包和可重复性相关补丁的存储库,这些补丁对于确定不可重复构建的原因至关重要。该数据集包含 4 类不可重复构建的软件包,分别是时间戳(462 个)、文件顺序(118 个)、随机性(50 个)和语言环境(41 个)。然后,应用可重复构建验证工具链(如图 1 中步骤 1、步骤 2 和步骤 3)来获得相应的差异日志和构建日志,作为本次研究的数据集。将这些数据集按 5∶2∶3 的比例划分为训练集、验证集和测试集。训练集用于模型的训练,验证集用于调整模型参数以提升性能,测试集用于对模型进行评估。需要说明的是,Debian 有 16 类不可重复构建原因[25],但是如工具链、环境、从源开始无法构建等所导致的不可重复构建问题并不在本次研究的考虑范围之内,这主要是因为这些类别所属的软件包在当前环境下无法构建成功或者数量过于稀少。

4.2　工具实现及对比算法

本文提出的算法是在 Python3.7 和 Java 17.0.1 中实现的。实验在 Windows10 Intel CI(R) i7-7700 CPU @ 3.60 GHz,32 GB 内存机器上运行,日志提取部分在内核为 4.9.0

的 GNU/Linux 上实现。本文用到了 Gensim 的 TF-IDF 和 word2vec 算法模型，并用余弦相似度 API 接口计算与差异日志相似度最高的构建日志片段，使用第三方提供的机器学习库 sklearn 来建立和评估多个机器学习模型。

将本文提出的算法与 4 种常用的机器学习算法分类器进行比较，包括支持向量机、朴素贝叶斯、决策树、随机森林。对于上面的每个分类器，使用多次交叉验证来进行参数优化以保证实验的公平性。此外，因为先前没有解决这个问题的方法，本文还考虑了两个算法各自的两种变体，分别为仅基于构建日志的不可重复构建原因分类算法和仅基于差异日志的不可重复构建分类算法，用于检查其作为特征的重要性。

4.3 评估指标

本文中使用准确率、召回率和 F_1 指标来衡量不同算法在每一个类别上的分类性能；采用宏平均(macro average)来评估算法在整个数据集上的性能，宏平均是每一个类的性能指标的算术平均值。在 n 分类问题上($n>2$)，将分类器转化为 n 个二分类问题，最终得到 n 个二分类混淆矩阵，二分类混淆矩阵如表 2 所列。

表 2 二维混淆矩阵

Table 2 Two dimensional confusion matrix

真实原因	预测结果	
	原因一(正例)	原因二(反例)
原因一(正例)	真正例 TP	假反例 FN
原因二(反例)	假正例 FP	真反例 TN

宏平均精度是先对每一个类别算出精度，再对所有的类求算术平均值，其计算式如下：

$$P_{\text{macro}}=\frac{1}{n}\sum_{i=1}^{n}P_i=\frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i+FP_i} \tag{6}$$

其中，TP_i 是第 i 个混淆矩阵中正确预测原因一的实例个数，FP_i 是第 i 个混淆矩阵中将原因二错误预测为原因一的实例个数。

同理，宏平均召回率是先对每一个类别算出召回率，再对所有的类求算术平均值，其计算式如下：

$$R_{\text{macro}}=\frac{1}{n}\sum_{i=1}^{n}R_i=\frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i+FN_i} \tag{7}$$

其中，FN_i 是第 i 个混淆矩阵中错误预测原因二的实例个数。

宏平均 F_1 指标是宏平均精度和宏平均召回率的调和平均，其计算式如下：

$$F_{1\text{-macro}}=\frac{2*P_{\text{macro}}*R_{\text{macro}}}{P_{\text{macro}}+R_{\text{macro}}} \tag{8}$$

宏平均 F_1 值从整体上考虑这两个指标，防止两个指标差距过大，用于在整体上评估算法的效果。

4.4 研究问题

在本研究中，我们通过调查以下问题，对所提方法进行系统分析。

研究问题 1：各种机器学习分类算法的执行效率如何？

研究问题 2：各种机器学习分类算法在宏平均精度、宏平均召回率、宏平均 F_1 值上表现如何？

研究问题 3：从构建日志和差异日志中提取的特征对分类性能的影响如何？

研究问题 4：本实验训练的模型用于其他项目的不可重复构建原因检测时效果如何？

5 实验结果

本节分析和讨论了实验结果，并对 4 个研究问题进行了回答。

5.1 研究问题 1：计算时间的比较

在研究问题 1 中，我们讨论了不可重复构建原因分类算法的计算时间。表 3 列出了本文提出的算法同其他 4 种算法在模型训练时间和测试时间上的差异，需要说明的是，表中的每个值都是实验运行 10 次后的平均值。

表 3 逻辑回归和其他算法计算时间上的比较

Table 3 Comparison of calculation time between logistic regression and other algorithms

(单位：s)

Algorithm	Training time	Testing time
支持向量机	0.86	0.51
朴素贝叶斯	1.25	0.51
决策树	0.89	0.26
随机森林	0.84	0.28
逻辑回归	0.85	0.24

从表 3 中可以看出，基于逻辑回归算法模型的训练时间为 0.85 s，测试时间为 0.24 s。相比训练时间用时最少的随机森林，基于逻辑回归算法模型的训练时间仅比其多 0.01 s，而测试时间 z 在当前几种机器学习算法中用时最少。除朴素贝叶斯算法外，基于逻辑回归算法模型的训练时间和测试时间少于其他 3 种算法。

研究问题 1 的答案：总体上，在对软件包不可重复构建的原因进行分类时，逻辑回归算法的效率略微占优。

5.2 研究问题 2：算法效果的比较

研究问题 2 评估了不同机器学习算法在软件包不可重复构建原因分类方面的性能表现。表 4 列出了逻辑回归算法和其他 4 种机器学习算法的宏平均精度、宏平均召回率和宏平均 F_1 值。实验表明，所提模型精度在 80.75%左右，召回率在 86.07%左右，F_1 值在 83.33%左右。需要说明的是，实验结果为 10 次运行结果的平均值，以最大程度地降低机器学习算法带来的不确定性。与其他 4 种机器学习算法相比，逻辑回归在对 4 种软件包不可重复构建原因分类法方面表现更好。为了得出一种算法是否优于另一种算法的可靠结论，我们结合表 4 进行了统计测试。本文进行了 T 检验[26]，并使用一个零假设——比较的算法结果之间不存在差异，以验证 4 种算法与逻辑回归在宏平均精度、宏平均召回率、宏平均 F_1 值上的差异。我们认可 95%置信水平，即 P 值小于 0.05，则表示两个样本之间存在显著差异，T 检验的结果如表 5 所列。

表 4 逻辑回归和其他算法性能上的比较

Table 4 Performance comparison between logistic regression and other algorithms

(单位：%)

Algorithm	P_{macro}	R_{macro}	$F_{1\text{-macro}}$
支持向量机	59.89	75.65	66.85
朴素贝叶斯	72.18	83.24	77.32
决策树	65.90	81.65	73.08
随机森林	73.52	84.55	78.65
逻辑回归	80.75	86.07	83.33

表 5　4 种机器学习算法 T 检验的 P 值

Table 5　P values of T-test of four machine learning algorithms

Algorithm	支持向量机	朴素贝叶斯	决策树	随机森林
P	0.0171	0.0365	0.0413	0.0567

在统计检验中,支持向量机、朴素贝叶斯、决策树和逻辑回归算法之间的差异显著。而随机森林算法在宏平均召回率和宏平均 F_1 值方面都与逻辑回归算法非常接近,因此它与逻辑回归之间没有显著差异。在软件包不可重复构建的原因分类上,随机森林和逻辑回归获得了几乎相同的效果,而支持向量机、朴素贝叶斯、决策树的宏平均 F_1 值偏低。

基于以上的分析可以看出,逻辑回归在多数情况下表现良好。针对其相对于其他机器学习算法良好的性能,我们分析有以下几个原因。第一,逻辑回归更适合当前分类自变量;第二,支持向量机在面对当前任务时,很难找到一个合适的核函数;第三,朴素贝叶斯假设属性之间相互独立,这在现实中往往是不成立的,在属性个数相对较多或者属性之间相关性较大时,分类效果不好;最后,与决策树针对数据局部结构不同,逻辑回归对数据整体结构的分析把握较好。在当前框架中,基于软件包不可重复构建原因分类这个特定领域,逻辑回归可以取得较好性能。

研究问题 2 的答案:逻辑回归在软件包不可重复构建原因分类上的宏平均精度、宏平均召回率和宏平均 F_1 值方面优于支持向量机、决策树、随机森林和朴素贝叶斯算法。

5.3　研究问题 3:特征重要性评估

在研究问题 3 中,本文通过检查随机森林和逻辑回归这两种算法的各自变体来衡量基于差异日志和构建日志的特征重要性。表 6 列出了两种算法各自的变体和两种算法的实验结果。RF(DL)表示基于差异日志特征建立的随机森林算法,RF(BL)表示基于构建日志特征建立的随机森林算法,LR(DL)表示基于差异日志特征建立的逻辑回归算法,LR(BL)表示基于构建日志建立的逻辑回归算法。表的组织如下:第一列为两种算法及其两种变体,其余部分给出每个变体的性能评估指标。

表 6　算法及其变体之间的比较

Table 6　Comparison between algorithm and its variants

(单位:%)

Algorithm	P_{macro}	R_{macro}	$F_{1\text{-macro}}$
RF(DL)	63.63	75.92	69.23
RF(BL)	69.32	81.41	74.88
随机森林	73.52	84.55	78.65
LR(DL)	65.83	78.25	71.50
LR(BL)	69.83	82.62	75.69
逻辑回归	80.75	86.07	83.33

从实验结果来看,随机森林及其两种变体 RF(DL),RF(BL)和逻辑回归及其两种变体 LR(DL),LR(BL)展现出了相同的变化趋势,即仅基于差异日志特征建立的不可重复构建分类算法 RF(DL),LR(DL)性能不如仅基于构建日志特征建立的不可重复构建分类算法 RF(BL),LR(BL),仅基于构建日志特征建立的不可重复构建分类算法 RF(BL),LR(BL)不如依赖两种日志建立的不可重复构建原因分类算法。仅基于差异日志或构建日志构建的分类算法,其宏平均精度始终低于 70%,宏平均 F_1 值始终低于 80%,单一数据源提出的特征建立的算法模型 RF(DL),RF(BL),LR(DL),LR(BL)在对软件包不可重复构建原因分类方面的性能劣于多种数据来源提出特征建立的算法模型(随机森林、逻辑回归)。

以逻辑回归算法为例,相比仅基于差异日志的逻辑回归算法 LR(DL),其在软件包不可重复构建原因分类上的宏平均精度提高了 14.92%,宏平均召回率提高了 7.82%,宏平均 F_1 值提高 11.83%。相比仅基于构建日志建立的逻辑回归算法 LR(BL),本文算法在软件包不可重复构建原因分类上的宏平均精度提高了 10.92%,宏平均召回率提高了 3.45%,宏平均 F_1 值提高了 7.64%。基于表 6 中的数据计算不同日志特征构建的相应算法之间的 T 检验 P 值,LR(DL)的 P 值为 0.0152,LR(BL)的 P 值为 0.0385。从统计学角度看,基于差异日志和构建日志建立的逻辑回归分类算法与仅基于其中一种日志构建的逻辑回归算法存在显著差异。根据以上分析,可以确定,作为基于两种日志特征建立的分类算法,本文方法显著优于仅依赖于单一日志特征的算法模型。

研究问题 3 的答案:单一数据源所提出的特征对不可重复构建原因的分类效果差于多种数据来源所提出的特征。通过两种日志特征建立的不可重复构建原因分类模型优于单一日志特征建立的分类模型。

5.4　研究问题 4:跨项目检测

在研究问题 1—研究问题 3 中,我们评估了本文算法的性能,采用了来自 BTS 已经修复的软件包。为了评估本文算法在其他项目上的效果,我们进行了跨项目实验。在跨项目实验中,采用来自 GUIX 存储库的 24 个已经修复的软件包,其中时间戳、随机性、文件顺序、语言环境各 6 个。选择 GUIX 存储库的原因是:1)GUIX 致力于可重复构建实践[27];2)GUIX 包管理器提供的本地验证包可重复功能有助于实验设计。此外,我们仍然采用最佳参数配置和性能评价指标。

实验结果如表 7 所列。从表 7 中我们可以看到,相比对 Debian 的不可重复构建软件包原因进行分类,对 GUIX 存储库中的不可重复构建软件包原因分类的性能有所下降,但 P 值大于 0.05,说明差异并不显著。总体而言,基于日志信息的不可重复构建原因检测方,在跨项目实验的合理范围内的效果有所降低,但检测结果变化不大,这表明所提方法具有良好的性能。

表 7　跨项目实验的结果

Table 7　Results of cross project experiment

(单位:%)

Project	P_{marco}	R_{marco}	$F_{1\text{-marco}}$	P value
GUIX	71.35	83.77	77.06	0.0501

研究问题 4 的答案:在跨项目实验的合理范围内,不可重复构建原因的分类结果有所降低。

6　对有效性的威胁

首先,基础数据集来自 Debian,这是通过对其缺陷追踪系统的挖掘建立的。此外,我们还通过可重复软件包的验证过程收集差异日志和构建日志。因此,一个风险是所提方法

可能会过于依赖 Debian 而不利于推广到其他软件存储库中。如问题研究 4 中所示，我们成功地将软件包不可重复构建原因分类算法应用到 Guix 的不可重复构建软件包中。因此，本文提出的方法可以推广到其他开源项目中。在今后的工作中，我们将考虑在更多的开源项目上来验证所提方法。

其次，本研究没有考虑工具链、环境等外部因素所导致的不可重复构建软件包问题。以工具链为例，软件包的不可重复构建问题是其依赖包引入的而非当前包自身的问题。此外，对工具链的修复有助于更多软件包的可重复构建。例如，当 Gcc 接受可重复构建相关补丁时，200 多个依赖于 Gcc 的软件包可以做到重复构建[28]。在未来的工作中，我们将探索工具链、环境等外部因素所导致的不可重复构建问题。

7 相关工作

2018 年，Ren 等[24]提出 RepLoc 框架，用于对源代码中导致不可重复构建的问题文件进行定位。RepLoc 将基于启发式规则的过滤组件同基于构建日志的查询增强组件相结合，给出一个按可能性排序的问题文件列表。2019 年，Ren 等进一步提出了 REPTRACE 框架[29]，该框架使用系统调用追踪工具来获取不同构建环境中构建时的系统调用信息，通过分析这些信息来建立构建过程中的依赖关系图，进而使用遍历关系图来确定不可重复构建的根源。

Wheeler[30]描述了一种名为“多样化双重编译”的实用的技术。该技术通过使用不同的编译器编译源文件两次，并验证编译后的二进制文件，可以检测并防止某些类型的恶意攻击。根据 Debian 的文档，这项工作在一定程度上推动了可复制构建实践。

Navarro 等[31]在用户空间中实现了一种名为 DetTrace 的 Linux 可重现抽象容器。DetTrace 将一个轻量级容器与系统调用拦截结合起来，使得容器中的程序与容器外的程序和文件隔离开，为最初不可重复构建的软件带来可重现性，却无须任何硬件、操作系统或应用程序的更改。

Ohm 等[32]观察到，在软件版本的良性更新间，新引入的可观察对象数量较少。但当软件变成恶意软件时，生成的可观察异常数量会增加。基于这些观察，Ohm 等提出用于动态分析软件及其第三方依赖的框架 Buildwatch，以帮助开发者评估依赖项更新是否引入恶意程序。

Xiong 等[33]专门为 Java 系统的构建可重复性提出了一种系统性的方法。该方法由 3 部分组成：一个统一的构建过程、一个在构建过程中动态控制非确定性的工具、一个通过后处理构建工件来消除不可重复部分的工具。该方法突破了过去 Java 项目构建可重复依赖于特定构建工具的桎梏。

He 等[34]提出一种有助于生成可重复工件的自动化工具 ConstBin。它在构建过程对不可重建的命令进行捕捉，并根据可扩展的规则集自动将它们替换为对应的的修正命令，通过这种方式来完成对不可重复构建软件包的自动修复工作。

结束语 本文基于差异日志和构建日志提取特征，使用逻辑回归算法对 671 个软件包的不可重复构建原因进行分类检测。在前两个实验中，我们比较了所提算法模型和其他机器学习算法的计算时间和性能，实验结果表明，所提方法在计算时间、精度、召回率以及 F_1 值上均优于其他算法。在第三个实验中，我们将本文提出的算法同它的两种变体进行比较，实验结果表明基于两种日志提取的特征更有利于训练出性能更好的模型。最后，在开源项目 Guix 上进行了跨项目的实验评估，实验结果证实了所提方法的普适性。在未来的工作中，我们将在更多的不可重复构建原因类别和开源项目上进行扩展。

参 考 文 献

[1] LAMB C, ZACCHIROLI S. Reproducible Builds: Increasing the Integrity of Software Supply Chains[J]. IEEE Software, 2021, 39(2): 62-70.

[2] GUI X, LIU J, CHI M, et al. Analysis of malware application based on massive network traffic[J]. China Communications, 2016, 13(8): 209-221.

[3] The editorial department. Journal International chapter of the inventory of network security events in 2019 [J]. Confidential Science and Technology, 2019(12): 2.

[4] OHM M, PLATE H, SYKOSCH A, et al. Backstabber's knife collection: A review of open source software supply chain attacks[C]// International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Cham: Springer, 2020: 23-43.

[5] DI RUSCIO D, PELLICCIONE P. Simulating upgrades of complex systems: The case of Free and Open Source Software[J]. Information and Software Technology, 2014, 56(4): 438-462.

[6] MACKINNON J G. The Linux operating system: Debian GNU/Linux[J]. Journal of Applied Econometrics, 1999, 14(4): 443-452.

[7] COURTÈS L, WURMUS R. Reproducible and user-controlled software environments in HPC with Guix[C]// European Conference on Parallel Processing. Cham: Springer, 2015: 579-591.

[8] MASTE E. Reproducible Builds in FreeBSD[EB/OL]. https://people. freebsd. org/~ emaste/2017-03-12-AsiaBSDCon-Reproducible-Builds-FreeBSD. pdf.

[9] VANGOOR B K R, AGARWAL P, MATHEW M, et al. Performance and Resource Utilization of FUSE User-Space File Systems[J]. ACM Transactions on Storage, 2019, 15(2): 1-49.

[10] HOLGER L, ADRIAN B, ALEXANDER C, et al. Overview of reproducible builds for packages in unstable for amd64[EB/OL]. (2014-10-01) [2021-11-25]. https://tests. reproducible-builds. org/debian/unstable/index_suite_amd64_stats. html.

[11] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space[J]. arXiv: 1301. 3781, 2013.

[12] HOLGER L, ADRIAN B, ALEXANDER C, et al. Reproducible Builds Experimental Toolchain[EB/OL]. https://wiki. debian. org/ReproducibleBuilds/ExperimentalToolchain.

[13] HOLGER L, ADRIAN B, ALEXANDER C, et al. Variations introduced when testing Debian packages[EB/OL]. (2014-10-01) [2021-11-25]. https://tests. reproduciblebuilds. org/debian/index_variations. html.

[14] GLUKHOVA M. Tools for ensuring reproducible builds for

open-source software[J]. Lappeenranta University of Technology,2017,3(1):11-12.
[15] CHRIS L,HOLGER L,MATTIA R,et al. SOURCE DATE EPOCH specification[EB/OL]. (2017-11-27)[2021-11-28]. https://reproduciblebuilds.org/specs/source-date-epoch/.
[16] HOLGER L,ADRIAN B,ALEXANDER C,et al. Strip Nondeterminism:a Perl library for stripping non-deterministic information[EB/OL]. (2014-10-01) [2021-11-28]. https://packages.debian.org/sid/strip-nondeterminism.
[17] BAR-YOSEF N,WOOL A. Remote algorithmic complexity attacks against randomized hash tables[C]// International Conference on E-Business and Telecommunications. Berlin:Springer,2007:162-174.
[18] HOLGER L,ADRIAN B,ALEXANDER C,et al. Achieve deterministic builds:Randomness[EB/OL]. (2017-05-18)[2021-11-25]. https://reproducible-builds.org/docs/randomness/.
[19] HOLGER L,ADRIAN B,ALEXANDER C,et al. Achieve deterministic builds:Locales [EB/OL]. (2017-06-24)[2021-11-25]. https://reproducible-builds.org/docs/locales/.
[20] HAVRLANT L,KREINOVICH V. A simple probabilistic explanation of term frequency-inverse document frequency(tf-idf) heuristic(and variations motivated by this explanation)[J]. International Journal of General Systems,2017,46(1):27-36.
[21] ZHENG L,IDRISSI K,GARCIA C,et al. Logistic similarity metric learning for face verification[C]// 2015 IEEE International Conference on Acoustics,Speech and Signal Processing (ICASSP). IEEE,2015:1951-1955.
[22] MANNING C,RAGHAVAN P,SCHÜTZE H. Introduction to information retrieval[J]. Natural Language Engineering,2010,16(1):100-103.
[23] JOHNS B T,MEWHORT D J K,JONES M N. The role of negative information in distributional semantic learning[J]. Cognitive Science,2019,43(5):e12730.
[24] REN Z,JIANG H,XUAN J,et al. Automated localization for unreproducible builds[C]// Proceedings of the 40th International Conference on Software Engineering. 2018:71-81.
[25] HOLGER L,ADRIAN B,ALEXANDER C,et al. Reproducible builds bugs filed [EB/OL]. (2014-10-01) [2021-11-25]. https://tests.reproducible.builds.org/debian/index_bugs.html.
[26] ZHI D X,XU X J,ZHANG H B. Comparison of Z-test and t-test [J]. Statistics and decision making,2014(20):4.
[27] COURTÈS L,WURMUS R. Reproducible and user-controlled software environments in HPC with Guix[C]// European Conference on Parallel Processing. Cham:Springer,2015:579-591.
[28] HOLGER L,ADRIAN B,ALEXANDER C,et al. Reproducible builds:week 54 in Stretch cycle [EB/OL]. (2016-5-10) [2021-11-25]. https://reproducible-builds.org/blog/posts/54/.
[29] REN Z,LIU C,XIAO X,et al. Root cause localization for unreproducible builds via causality analysis over system call tracing [C]// 2019 34th IEEE/ACM International Conference on Automated Software Engineering(ASE). IEEE,2019:527-538.
[30] WHEELER D A. Countering trusting trust through diverse double-compiling[C]// 21st Annual Computer Security Applications Conference(ACSAC'05). IEEE,2005:13-48.
[31] NAVARRO LEIJA O S,SHIPTOSKI K,SCOTT R G,et al. Reproducible containers[C]// Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020:167-182.
[32] OHM M,SYKOSCH A,MEIER M. Towards detection of software supply chain attacks by forensic artifacts[C]// Proceedings of the 15th International Conference on Availability,Reliability and Security. 2020:1-6.
[33] XIONG J,SHI Y,CHEN B,et al. Towards Build Verifiability for Java-based Systems[J]. arXiv:2202.05906,2022.
[34] HE H,CAO J,DU L,et al. ConstBin:A Tool for Automatic Fixing of Unreproducible Builds[C]// 2020 IEEE International Symposium on Software Reliability Engineering Workshops(ISSREW). IEEE,2020:97-102.

MA Zhao, born in 1996, postgraduate. His main research interests include reproducible build and so on.

JIANG He, born in 1980, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include system software and software engineering.

(责任编辑:何杨)