doi:10.11920/xnmdzk.2025.01.013

基于时序特征的即时缺陷预测研究

黄　磊[1],李英玲[1,2],辛　罡[1,2]
(1. 西南民族大学计算机与人工智能学院,四川 成都 610041;2. 西南民族大学计算机系统国家民委重点实验室,四川 成都 610041)

摘　要:即时缺陷预测是持续集成代码提交时进行代码缺陷预测,从而节省测试代码资源消耗以及减少测试时间,提升集成构建速度.然而现有的即时缺陷预测方法,只采用传统人工特征、代码结构和语义信息,忽略了在集成历史中产生的时序信息.受到时序模型相关研究启发,可从集成历史版本信息提取时序特征,用于缺陷预测.因此,提出基于时序特征的即时缺陷预测模型(TSF-JIT),首先从项目数据的集成版本历史中,提取历史轨迹信息并分段处理;接着,基于分段轨迹使用双向门控循环单元以及多头自注意力机制提取序列中的时序特征;最后,将时序特征卷积降维并构建缺陷预测模型,预测提交代码是否包含缺陷.与当前具有代表性的 2 个基线方法在 6 个项目数据进行比较,在精确率、召回率、F1 值和 AUC 结果平均值上,模型比基线方法高出 4.74%~7.21%、8.05%~8.74%、5.56%~8.31%、2.92%~4.79%.实验结果说明,提出的方法能够有效提高缺陷预测的精确率,从而降低开发中测试资源和时间消耗.

关键词:即时缺陷预测;时序特征;双向门控循环单元;自注意力机制
中图分类号:TP311.5　　**文献标志码**:A　　**文章编号**:2095-4271(2025)01-0092-08

Research on Just-In-Time defect prediction with temporal features

HUANG Lei [1], LI Yingling[1,2], XIN Gang[1,2]
(1. School of Computer and Artificial Intelligence, Southwest Minzu University, Chengdu 610041, China;
2. The Key Laboratory for Computer Systems of State Ethnic Affairs Commission, Southwest Minzu University, Chengdu 610041, China)

Abstract: Just-In-Time (JIT) defect prediction aims to identify code defects in continuous integration commits to optimize testing resource usage and reduce testing times, thereby accelerating integration build processes. However, the temporal information inherent in the integration history is overlooked by existing JIT defect prediction methods, which only utilize traditional manual features, code structure and semantic information. Inspired by research on temporal models, temporal features can be extracted from the integration history information for defect prediction. Therefore, this paper proposed a Just-In-Time defect prediction model with temporal features (TSF-JIT). It began by extracting historical trajectory information from the project dataset's integration version history and processing it in segments. Then, it used a bidirectional GRU and multi-head self-attention mechanism to extract temporal features from the segmented trajectories. Finally, the model applied convolution to reduce the dimensionality of temporal features before constructing the defect prediction model to predict whether the committed code contained defects. The proposed method was compared with two current representative baseline methods across six project datasets, showing improvements of 4.74%~7.21%, 8.05%~8.74%, 5.56%~8.31% and 2.92%~4.79% over those of the baselines in precision, recall, F1 score and AUC, respectively. The experimental results demonstrated that the proposed TSF-JIT method could effectively improve the precision of defect prediction, thereby reducing the consumption of testing resources and time in project development.

Keywords: Just-In-Time defect prediction; temporal feature; bidirectional GRU neural network; self-attention mechanism

收稿日期:2024-03-14
通信作者:辛罡(1983-),男,实验师,博士,研究方向:可学习模型的智能优化算法、边缘计算. E-mail:xin_gang@swun.edu.cn
基金项目:国家自然科学基金项目(62302408);中央高校基本科研业务费专项资金优秀学生培养工程项目(2023NYXXS035)

当前大型项目开发,系统以极其频繁的速度进行代码的集成构建,而极其频繁的集成构建需要消耗大量的测试资源和时间. 为了解决这一问题,即时(Just-In-Time,JIT)缺陷预测方法被提出[1]. 其原理是从项目中的大量集成历史数据进行特征提取,构建即时缺陷预测模型. 通过预测结果判断是否调用测试资源,从而减少不必要的测试资源使用,减少开发成本.

近年来,相关研究人员提出了多种机器学习和深度学习方法来构建即时缺陷预测模型. 机器学习模型方面,Kamei 等人提出了基于逻辑回归的即时缺陷预测模型,该模型是由逻辑回归模型以及 14 个人工特征构建[2]. Yang 等人提出了使用深度置信网络(DBN)的即时缺陷预测模型[3]. 深度学习方面,Hoang 等人提出了使用文本卷积神经网络(TextCNN)自动从代码更改和提交日志的内容中自动提取特征[4]. 后续 Hoang 等人还提出了利用代码结构构建特征的缺陷预测模型,该模型通过学习由日志信息引起的代码更改特征[5]. 此外,学术界和工业界联合提出的基于深度学习的工作量感知即时缺陷预测、基于监督学习和无监督学习的即时缺陷预测,项目内和跨项目场景下的即时缺陷预测.

现有即时缺陷预测研究虽然取得了不错效果,但是未细粒度地考虑集成过程中产生的时序特征. 在现有研究中,采用的特征是各集成版本中的代码结构特征和代码语义特征,以及传统的人工特征,而忽略了集成历史轨迹中产生的时序信息特征.

针对即时缺陷预测中未使用时序信息特征,受到时间序列模型相关研究启发,可采用时间序列相关技术提取集成历史版本中的时序信息,通过时序信息特征构建即时缺陷预测模型. 故本研究提出基于时序特征的即时缺陷预测(TSF-JIT)方法,使用持续集成在历史中产生的时序性特征构建即时缺陷预测模型进行缺陷预测,核心采用基于自注意力机制的双向 GRU 神经网络提取集成历史版本中的时序特征. 该模型分为三个阶段:首先,基于集成历史的轨迹分段处理阶段,该阶段将集成历史版本信息提取时序信息,构建版本历史轨迹,并进行轨迹分段处理;接着,基于集成历史的时序特征提取阶段,该阶段使用双向 GRU 以及多头自注意力机制技术提取集成版本序列段中的时序特征;最后,基于集成时序特征的缺陷预测阶段,将提取的时序特征作为输入,使用分类预测函数预测该次集成版本提交是否包含缺陷.

实验结果表明,本文提出的方法预测性能,在 6 个大型开源数据集上,与当前具有代表性的 2 个方法进行比较,从精准率,召回率,F1 值和 AUC 值四个评价指标的平均值上来看,分别高出基线方法 4.74% ~ 7.21%、8.05% ~ 8.74%、5.56% ~ 8.31% 和 2.92% ~ 4.79%.

1　相关工作

1.1　即时缺陷预测

为了节省开发时间和测试资源,软件工程研究人员提出了即时缺陷预测技术,即预测开发人员提交的每一次代码更改是否存在缺陷.

即时缺陷预测技术基于实时监测和分析开发人员的代码提交,通过挖掘每次提交中的潜在缺陷信息,以及与历史数据的对比,来预测代码变更是否可能引入新的缺陷[6-7]. 该技术借助机器学习和深度学习等先进方法,从代码变更的特征、commit 提交信息、开发人员的过往行为等多个维度进行综合分析,提高了预测的准确性[8].

例如,Zeng 等人提出了 LAPredict 模型,仅根据提交中添加的代码行数来进行缺陷预测,提交中添加的代码行越多,其出现缺陷的概率就越高[9]. Wang 等人利用深度置信网络,从源代码中学习抽象语法树(AST)的语义信息[10]. Li 等人使用 CNN 提取 AST 中的语义信息,并将提取的语义特征与传统手工特征相结合,以提高预测性能[11]. 此外,Qu 等人采用了网络嵌入技术(即 node2vec)来自动学习代码提交中的结构信息[12]. Xu 等人通过提取每个组件模块中的调用图依赖关系,然后使用 UCINET 工具提取依赖关系中的结构特征[13].

不同于已有即时缺陷预测研究,本文采用了集成历史中产生的时序信息,通过时序信息构建基于时序特征的即时缺陷预测模型.

1.2　时间序列模型

时间序列模型是一种专门针对时间序列数据的预测方法,其目标是通过分析时间序列数据的趋势、季节性和周期性等特征,来预测未来数据的走向.

时间序列预测的方法有多种,包括统计方法、经

典机器学习方法和深度学习方法[14]. 统计方法包括使用移动平均、指数平滑和自回归积分移动平均(ARIMA)模型[15-16]. 一般来说,深度学习方法在识别时间序列数据中的异常方面更有效,特别是当数据是高维或非线性时[17-18]. 例如,Altan 等人利用长短期记忆网络和随机优化器构成的混合优化器来捕获风速时间序列数据集中的非线性特征[19]. Lai 等人提出了长短期时间序列网络(LSTNet),使用 CNN 来捕获短期模式,并使用 LSTM 记忆相对长期的模式[20]. Gu 等人利用循环结构并通过时间步之间的状态转化,隐式捕获时序变化的信息[21]. Zhou 等人使用自注意力机制,提取时间点之间的时间依赖性[22].

受到时间序列模型启发,集成历史版本也可看作时间序列数据,可根据持续集成历史版本构建序列轨迹,并采用时序模型提取轨迹特征,构建缺陷预测模型,用于预测下一次集成是否包含缺陷.

本文使用双向门控循环单元(Bi-GRU)以及多头自注意力机制提取集成历史版本中的时序特征,从而构建包含时序特征的即时缺陷预测模型.

2 基于时序特征的即时缺陷预测模型

本文提出基于时序特征的即时缺陷预测模型(如图 1 所示),该模型一共分为 3 个阶段:基于集成历史的轨迹分段处理、基于集成版本的时序特征提取和基于集成时序特征的缺陷预测. 具体过程如下:1)基于集成历史的轨迹分段处理阶段,是提取集成历史版本的时序标签以构建集成历史轨迹. 对于集成历史轨迹,使用滑动窗口技术进行轨迹分段处理,获得训练和测试序列. 2)基于集成版本的时序特征提取阶段,采用双向 GRU 以及多头自注意力机制技术,以提取集成轨迹序列段中的时序特征. 3)基于集成时序特征的缺陷预测阶段,使用卷积技术对提取的时序特征降维,并使用分类函数 softmax 进行缺陷预测.

2.1 基于集成历史的轨迹分段处理

该阶段对项目数据集中的历史版本信息使用 pandas 框架进行数据预处理操作. 对于清洗好的数据,按照时间顺序进行排列. 通过对集成版本结果构建集成历史轨迹 T,提交结果为 1 说明集成版本包含缺陷,提交结果为 0 则说明集成版本不包含缺陷. 集成历史轨迹通过设定固定的步长 Step,将集成历史轨迹 T 划分为同样大小的时序片段数据,最终得到用于训练的序列 T_{train} 和测试的分段序列 T_{test}.

2.2 基于集成版本的时序特征提取

基于集成版本的时序特征提取阶段,使用双向 GRU 技术对输入的序列段提取浅层时序特征,使用多头自注意力机制以及双向 GRU 技术提取深层时序特征.

2.2.1 浅层时序特征提取

对于集成历史序列,一个时间点的集成版本如果产生缺陷,则后续的时间点很有可能同样会引入缺陷. 因为一个缺陷的引入,会影响到相关调用的代码,导致多处的代码缺陷,从而不能第一时间发现引入缺陷的代码块,进而导致后续的集成版本都包含缺陷. 从逆向时序来看,同样存在时序信息,若后续的多个版本都不包含缺陷的版本,则前面的集成版本很有可能也是不包含缺陷的.

双向 GRU 在处理序列数据时能同时考虑正向和逆向两个方向的时序信息. 它由两个独立的 GRU 组成,一个按照正序处理输入序列,另一个按照逆序处理输入序列. 最终两个方向上的隐藏状态联合起来,提供整个序列的全局信息.

故本方法采用双向 GRU 以获取双向的时序信息. 序列段 T_i 作为输入,使用双向 GRU 提取两个方向的序列信息,其前向隐藏层信息和逆向隐藏层信息分别表示为:

$$\overrightarrow{h_{forward}} = \overrightarrow{GRU}(T_i). \tag{1}$$

$$\overleftarrow{h_{backward}} = \overleftarrow{GRU}(T_i). \tag{2}$$

通过拼接两个方向的隐藏信息形成浅层时序特征向量 TSF_s,表示为:

$$TSF_s = [\overrightarrow{h_{forward}} \oplus \overleftarrow{h_{backward}}]. \tag{3}$$

2.2.2 深层时序特征提取

对于集成历史序列,在一个时间窗口的数据上,距离预测时间点越远的时间点在预测时对结果的影响会更小,而距离预测时间点越近的集成历史会对结果影响更大. 同时,从数据集缺陷率上可知,大多数的数据集上包含缺陷的版本远小于不包含缺陷的版本,故在时序预测上,有缺陷的版本对预测结果的影响会更大.

本文采用多头自注意力机制,将经过双层 GRU 输出的浅层时序特征向量 TSF_s 输入多头自注意力机

制中进行注意力权重嵌入,对影响预测结果更重要的集成版本时间点,给予较大的权重;对影响预测结果不重要的集成版本时间点,给予较小的权重. 注意力嵌入过程如下:

$$head_i = Attention(Q_i, K_i, V_i). \tag{4}$$

$$MultiHead(Q, V, K) = Concact(head_1 \cdots head_n)W. \tag{5}$$

其中 Q_i, K_i, V_i 分别代表第 i 个浅层时序特征向量 TSF_s 查询、键、值矩阵, $head$ 代表的是单头自注意力机制. $MultiHead$ 是多头自注意力机制, $Concact$ 代表将多个头的自注意力结果拼接起来, W 是可学习的权重参数.

接着将包含自注意力权重的浅层时序特征向量输入到双向 GRU 中进行学习时序信息,最终得到包含深层时序特征的向量表示为 TSF_d .

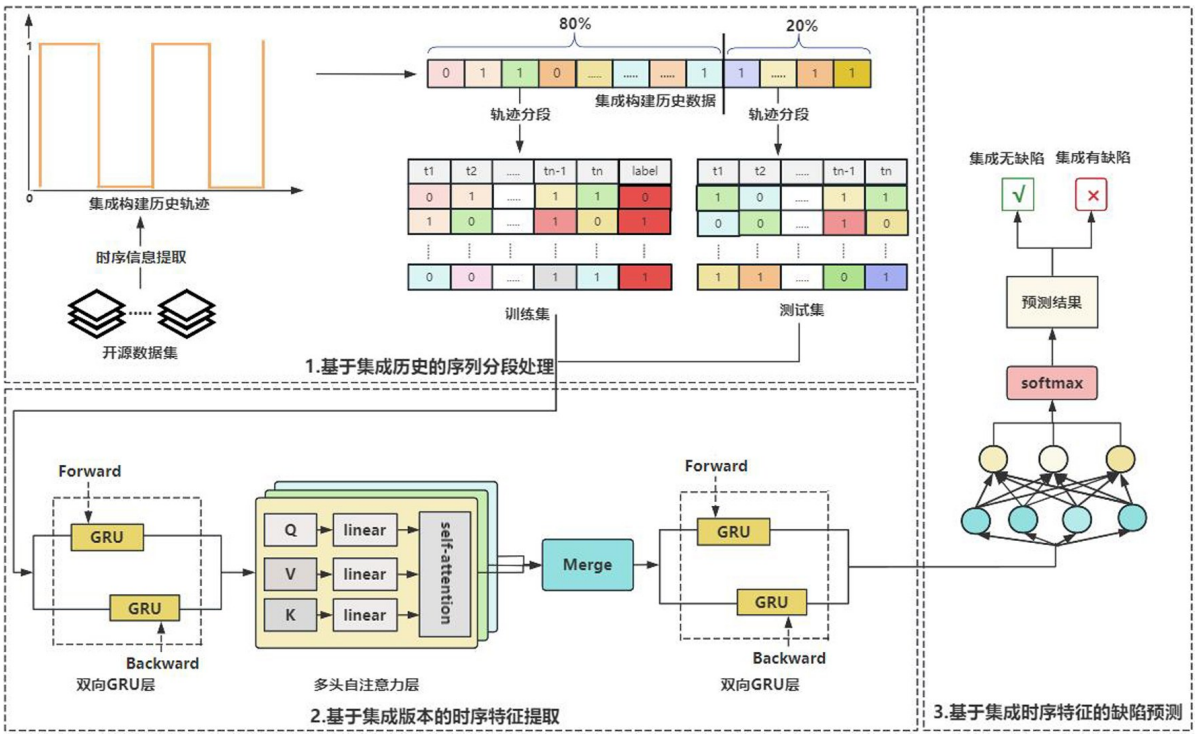

图 1　基于时序特征的即时缺陷预测方法

Fig. 1　Just-In-Time Defect Prediction Method with Temporal Features

2.3　基于集成时序特征的缺陷预测

经过基于集成版本的时序特征提取阶段的深层时序特征向量 TSF_d ,已经包含了版本时间序列中前向、逆向的时序特征,以及对于不同重要程度的版本时间点的注意力权重信息. 基于集成时序特征的缺陷预测阶段,对时间序列向量采用卷积操作,对输入的时间序列进行扫描,进行卷积降维;随后,将卷积过后的向量输入到 softmax 函数中进行预测操作,预测此次集成版本是否包含缺陷. 如图 2 所示.

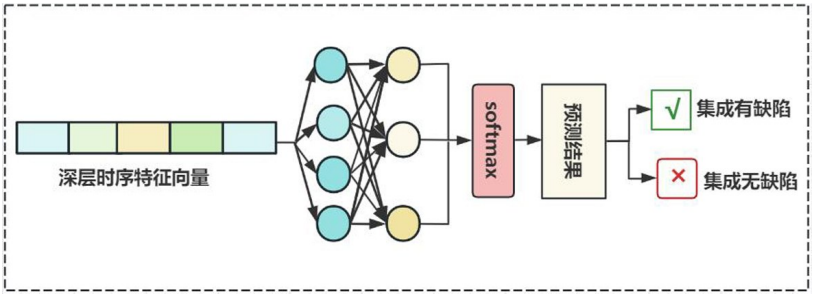

图 2　基于集成时序特征的缺陷预测

Fig. 2　Defect Prediction with CI Temporal Features

预测的过程中 softmax 函数如公式(6)所示:

$$Y = softmax(W^* H_m + b). \quad (6)$$

其中 W 代表 softmax 层的权重矩阵, b 代表偏置项. 本文采用二元交叉熵损失(Binary Cross-Entropy)构建的损失函数,采用了反向传播算法进行模型训练,学习模型的参数如公式(7)所示:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i})]. \quad (7)$$

其中,N 代表的是样本数, y_i 代表的是第 i 个样本的真实标签(0 或 1), $\hat{y_i}$ 代表模型对第 i 个样本的预测概率值. 损失函数会使得模型的预测概率接近真实标签的概率,即当真实标签为 1 时,模型预测为 1 的概率越大越好.

3 实验设置与结果分析

3.1 数据集准备和处理

本文使用的数据是基于 TravisTorrent[23] 的数据集,根据构建记录的数量从中选择 6 个项目作为本实验中的数据集. 对于获得的数据集,包含许多缺失数据行和问题数据行,本文采用 pandas 相关数据处理工具对数据进行数据清理;最后,将清洗后的数据按时间顺序排列,获得集成版本序列轨迹,并且按照特定步长进行分段操作处理为训练集和测试集. 其中清理后的数据如表 1 所示.

表 1 数据集介绍

Table 1 Dataset description

项目名称	集成次数/次	缺陷次数/次	缺陷率/%
cloudify	5 742	1 470	26
jackrabbit-oak	8 205	3 488	42
metasploit-framework	8 839	704	8
open-build-service	5 199	1 360	29
rails	19 447	6 719	35
ruby	15 388	3 422	22

由表中数据可知,大部分项目中的集成缺陷次数都远少于没有缺陷的集成次数,如 metasploit-framework 缺陷率仅有 8%,而 jackrabbit-oak 和 rails 数据相对平衡缺陷率分别为 42% 和 35%.

3.2 评价指标

本文参考了相关文献,选取了时序预测中常用的 4 个评价指标:精准率(Precision)、召回率(Recall)、F1 值和 AUC 值.

1)精确率

该评价指标是针对预测结果而言的,其含义是在被所有预测为正的样本中实际为正样本的概率. 精确率代表对正样本结果中的预测准确程度:

$$Precision = \frac{TP}{TP + FP}. \quad (8)$$

2)召回率

该评价指标,也叫查全率,是覆盖面的度量. 衡量了分类器对正例的识别能力.

$$Recall = \frac{TP}{TP + FN}. \quad (9)$$

3)F1 值

该评价指标是精确率(Precision)和召回率(Recall)的调和平均数.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}. \quad (10)$$

4)AUC 值

该评价指标是 ROC 曲线下的面积,其数值范围在 0 到 1 之间,完美分类器的 AUC 为 1,随机分类器的 AUC 约为 0.5. AUC 越接近 1,说明模型性能越好,能更好地将正样本排在负样本前面. AUC 指标的公式并不直接给出,而是通过对 ROC 曲线下的积分计算得到的,ROC 曲线是以假正例率(FPR)为横轴,真正例率(TPR)为纵轴所绘制的曲线,其中:

$$TPR = \frac{TP}{TP + FN}. \quad (11)$$

$$FPR = \frac{FP}{FP + TN}. \quad (12)$$

3.3 基线方法

本文选取了缺陷预测领域和时序预测领域两个相关方法作为本实验的基线.

1)基于遗传算法和长短期记忆模型的缺陷预测模型[24](DL-CIBuild):DL-CIBuild 是基于集成过程中产生的人工特征,使用长短期记忆模型(LSTM)提取每个版本中特征,方法中特别使用了遗传算法(GA)获取最佳的模型参数. 这种方法能够很好地提取人工特征中地每个特征之间的关系.

2)基于循环神经网络的序列预测模型(DeepAR)[25]:DeepAR 是一种基于时序信息的预测模型,常用于时序预测. 该方法使用了经典的 RNN 模型构建时间序列预测模型,可以学习时序数据中的长期依

赖关系和局部模式,是时序预测领域中一个经典的基线模型.

3.4　实验结果及分析

本章节首先将本文提出的方法从精确率、召回率、F1 值和 AUC 这 4 个方面,与基线方法在 6 个项目数据集上进行对比试验,验证模型效果,然后是消融实验,证明本方法提取的时序特征的有效性.

3.4.1　对比实验

表 2 展示了本文提出的方法 TSF-JIT 和基线方法在不同项目上的性能. 总体上来看(在 6 个项目数据集平均值),本文提出的方法 TSF-JIT 在所有评价指标上均具有明显优势. 对比基线方法 DL-CIBuild 精确率、召回率、F1 值和 AUC 值分别高出 7.21%、8.74%、8.31%和 4.79%;比基线方法 DeepAR,精确率、召回率、F1 值和 AUC 值分别高出 4.74%、8.05%、5.56%和 2.92%.

从各项目上看,本文提出的方法 TSF-JIT 在 6 个项目数据集上均高于基线方法. 在 cloudify 项目数据集上,TSF-JIT 在四个评价指标得分最高,准确率、F1 值和 AUC 值,取得了最佳效果,分别为 89.24%,84.14%和 91.30%;在 jackrabbit-oak 项目数据集上,TSF-JIT 召回率达到 84.95%,取得最高.

值得注意的是,在 metasploit-framework 项目数据集上,TSF-JIT 与基线方法在精确率、召回率和 F1 值上表现与其他项目数据集有较大差距,主要原因是 metasploit-framework 项目数据集过于不平衡,包含缺陷的集成版本过少(仅占整体的 8%),导致模型对缺陷特征的识别度不够高,故在预测结果时误差较大. 但 TSF-JIT 方法比基线方法 DL-CIBuild 在精确率有 15.26%的提升,在召回率有 6.15%的提升,在 F1 值上有 14.44%的提升,在 AUC 值上有 3.97%的提升;比基线方法 DeepAR 在精确率有 24.86%的提升,在召回率有 15%的提升,在 F1 值上有 23.98%的提升,在 AUC 值上有 5.34%的提升.

总的来说,TSF-JIT 对比基线方法 DL-CIBuild 和 DeepAR,在所有评价指标上都优于基线,因为 TSF-JIT 方法中双向 GRU 能够提取前向和后向的时序信息,而基线方法 DL-CIBuild 只能提取集成版本中单个时间点的特征间的关系,DeepAR 只能提取单一方向的时序信息. 因此,本文提出的方法在提取特征时更具优势,模型效果更佳.

表 2　TSF-JIT 与基线方法结果统计

Table 2　Comparison of TSF-JIT and baseline performance

项目数据集	模型	准确率	召回率	F1 值	AUC 值
cloudify	TSF-JIT	**0.892 4**	**0.796 0**	**0.841 4**	**0.913 0**
	DL-CIBuild	0.890 4	0.780 0	0.827 2	0.900 3
	DeepAR	0.874 5	0.768 0	0.817 8	0.897 8
jackrabbit-oak	TSF-JIT	**0.725 7**	**0.849 5**	**0.782 8**	**0.716 3**
	DL-CIBuild	0.718 1	0.794 5	0.754 4	0.658 0
	DeepAR	0.703 0	0.726 6	0.719 7	0.678 0
metasploit-framework	TSF-JIT	**0.483 3**	**0.191 7**	**0.274 5**	**0.756 5**
	DL-CIBuild	0.419 4	0.180 6	0.252 4	0.727 7
	DeepAR	0.387 1	0.166 7	0.233 0	0.718 2
open-build-service	TSF-JIT	**0.681 2**	**0.515 8**	**0.587 1**	**0.795 0**
	DL-CIBuild	0.575 0	0.484 2	0.525 7	0.764 6
	DeepAR	0.676 9	0.463 2	0.539 8	0.781 3
rails	TSF-JIT	**0.741 8**	**0.545 9**	**0.628 9**	**0.763 2**
	DL-CIBuild	0.663 7	0.478 3	0.556 0	0.676 1
	DeepAR	0.710 6	0.523 0	0.602 5	0.734 9
ruby	TSF-JIT	**0.765 1**	**0.542 9**	**0.635 1**	**0.842 4**
	DL-CIBuild	0.734 4	0.447 6	0.556 2	0.841 0
	DeepAR	0.743 4	0.538 1	0.624 3	0.840 3
平均值	TSF-JIT	**0.714 9**	**0.573 6**	**0.626 7**	**0.797 7**
	DL-CIBuild	0.666 8	0.527 5	0.578 6	0.761 3
	DeepAR	0.682 6	0.530 9	0.593 7	0.775 1

此外,对于极度不平衡数据集 metasploit-framework, TSF-JIT 相比基线方法提升效果显著,造成这种结果的原因是本方法中采用了多头自注意力机制对时序特征进行注意力嵌入.对于影响结果的集成版本节点给予更大注意力权重,使得模型更加关心缺陷样本数据.

3.4.2 消融实验

为了验证 TSF-JIT 模型设计的有效性,本章节进行了消融实验.去掉多头自注意力机制的 TSF-JIT 模型(命名为 TSF-JIT-A),在相同的参数定义和数据集上与 TSF-JIT 进行对比,结果如表 3 所示.

总体上来看,TSF-JIT 模型相比去掉多头自注意力机制的 TSF-JIT-A 在四个评价指标上,在 6 个项目数据集平均值结果上,精确率上高出 3.88%,在召回率上高出 4.85%,在 F1 值上高出 4.14% 和在 AUC 值上高出 2.26%.

特别地,在 metasploit-framework 数据集上,TSF-JIT 相比 TSF-JIT-A 在四个评价指标精确率、召回率、F1 值和 AUC 值上分别高出 11.82%、19.40%、17.21% 和 4.34%.

总的来说,TSF-JIT 模型使用多头自注意力技术能够明显提升模型整体效果.同时,在极度不平衡的项目数据集上,TSF-JIT 相比 TSF-JIT-A 提升效果最为显著,说明多头自注意力机制更能提取出不平衡数据集中少数类样本的时序信息,从而提升预测结果精确率.

表 3 消融实验结果

Table 3 Performance of ablation experiment

项目数据集	模型	准确率	召回率	F1 值	AUC 值
cloudify	TSF-JIT	**0.892 4**	**0.796 0**	**0.841 4**	**0.913 0**
	TSF-JIT-A	0.883 5	0.728 0	0.798 2	0.875 0
jackrabbit-oak	TSF-JIT	**0.725 7**	**0.849 5**	**0.782 8**	**0.716 3**
	TSF-JIT-A	0.705 2	0.843 4	0.780 3	0.701 5
metasploit-framework	TSF-JIT	**0.483 3**	**0.191 7**	**0.274 5**	**0.756 5**
	TSF-JIT-A	0.432 2	0.160 6	0.234 2	0.725 0
open-build-service	TSF-JIT	**0.681 2**	**0.515 8**	**0.587 1**	**0.795 0**
	TSF-JIT-A	0.636 4	0.485 1	0.550 6	0.793 3
rails	TSF-JIT	**0.741 8**	**0.545 9**	**0.628 9**	**0.763 2**
	TSF-JIT-A	0.734 1	0.545 9	0.626 2	0.759 0
ruby	TSF-JIT	**0.765 1**	**0.542 9**	**0.635 1**	**0.842 4**
	TSF-JIT-A	0.737 8	0.521 8	0.611 3	0.827 0
平均值	TSF-JIT	**0.714 9**	**0.573 6**	**0.626 7**	**0.797 7**
	TSF-JIT-A	0.688 2	0.547 5	0.600 1	0.780 1

4 总结

持续集成中,开发人员极其频繁地系统构建和测试集成代码需要大量的测试资源.受到时序领域相关研究启发,本文提出基于时序特征的即时缺陷预测方法,通过从集成历史版本中提出序列轨迹学习时序特征,构建即时缺陷预测模型.本文提出的方法,在 6 个项目数据集上的平均值来看,比最优的基线方法精确率高出 4.74%,召回率高出 8.05%, F1 值高出 5.56%,AUC 值高出 2.92%.本方法旨在提高软件开发过程中缺陷检测的准确性,减少项目中测试资源浪费.

虽然本文提出的方法取得了较好的实验结果,在极度不平衡数据上,整体效果显著优于基线方法,然而整体效果还是不尽人意.所以,未来在处理不平衡数据集时需要进行特殊的不平衡处理.另外,本方法中仅使用了时序特征作为模型训练特征,后续工作中会将现有研究中的人工特征、代码结构和代码语义特征加入模型中,检验增加这些特征是否能进一步提升模型性能.

参考文献

[1]葛建,虞慧群,范贵生,等.面向智能计算框架的即时缺陷预测[J].软件学报,2023,34(9):3966-3980.

[2]KAMEI Y,SHIHAB E,ADAMS B,et al. A large-scale empirical study of just-in-time quality assurance[J]. IEEE Transactions on Software Engineering,2013,39(6):757-773.

[3]YANG X L,LO D,XIA X,et al. Deep learning for just-in-time defect prediction[C]//2015 IEEE International Conference on Software Quality,Reliability and Security. Vancouver,BC,Canada:IEEE,2015:17-26.

[4]HOANG T,KHANH DAM H,KAMEI Y,et al. DeepJIT:An end-to-end deep learning framework for just-in-time defect prediction[C]//2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). Montreal,QC,Canada:IEEE,2019:34-45.

[5]HOANG T,KANG H J,LO D,et al. CC2Vec:Distributed representations of code changes[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Seoul South Korea:ACM,2020:518-529.

[6]PHILLIPS P C B,JIN S N. Business cycles,trend elimination,and the HP filter[J]. International Economic Review,2021,62(2):469-520.

[7]LI D,CHEN D C,JIN B H,et al. MAD-GAN:Multivariate anomaly detection for time series data with generative adversarial networks[M]// Artificial Neural Networks and Machine Learning - ICANN 2019:Text and Time Series. Cham:Springer International Publishing,2019:703-716.

[8]刘旭同,郭肇强,刘释然,等. 软件缺陷预测模型间的比较实验:问题、进展与挑战[J]. 软件学报,2023,34(2):582-624.

[9]ZENG Z R,ZHANG Y Q,ZHANG H T,et al. Deep just-in-time defect prediction:how far are we? [C]. Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York,NY,USA:ACM,2021:427-438.

[10]WANG S,LIU T Y,NAM J,et al. Deep semantic feature learning for software defect prediction[J]. IEEE Transactions on Software Engineering,2020,46(12):1267-1293.

[11]LI J,HE P J,ZHU J M,et al. Software defect prediction via convolutional neural network[C]//2017 IEEE International Conference on Software Quality,Reliability and Security (QRS). Prague,Czech Republic:IEEE,2017:318-328.

[12]QU Y,LIU T,CHI J L,et al. node2defect:Using network embedding to improve software defect prediction[C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Montpellier,France:ACM,2018:844-849.

[13]XU Z,LI S,XU J,et al. LDFR:Learning deep feature representation for software defect prediction[J]. Journal of Systems and Software,2019,158:110402.

[14]蔡亮,范元瑞,鄢萌,等. 即时软件缺陷预测研究进展[J]. 软件学报,2019,30(5):1288-1307.

[15]张献,贲可荣,曾杰. 基于代码自然性的切片粒度缺陷预测方法[J]. 软件学报,2021,32(7):2219-2241.

[16]ZHANG T,YU Y,MAO X J,et al. FENSE:A feature-based ensemble modeling approach to cross-project just-in-time defect prediction[J]. Empirical Software Engineering,2022,27(7):162.

[17]WANG S,LIU T Y,TAN L,et al. Automatically learning semantic features for defect prediction[C]//Proceedings of the 38th International Conference on Software Engineering. Austin,Texas,USA:ACM,2016:297-308.

[18]LIANG H L,YU Y,JIANG L,et al. Seml:A semantic LSTM model for software defect prediction[J]. IEEE Access,2019,7:83812-83824.

[19]ALTAN A,KARASU S,ZIO E. A new hybrid model for wind speed forecasting combining long short-term memory neural network,decomposition methods and grey wolf optimizer[J]. Applied Soft Computing,2021,100:106996.

[20]LAI G K,CHANG W C,YANG Y M,et al. Modeling long- and short-term temporal patterns with deep neural networks[C]//The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. Ann Arbor,MI,USA:ACM,2018:95-104.

[21]GU A,GOEL K,RÉ C. Efficiently modeling long sequences with structured state spaces[J]. arXiv preprint arXiv:2021,2111.00396.

[22]ZHOU H,ZHANG S,PENG J,et al. Beyond efficient transformer for long sequence time-series forecasting [J]. arXiv preprint arXiv,2021,17325:11106-11115.

[23]BELLER M,GOUSIOS G,ZAIDMAN A. TravisTorrent:Synthesizing travis CI and GitHub for full-stack research on continuous integration [C]//2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). Buenos Aires,Argentina:IEEE,2017:447-450.

[24]SAIDANI I,OUNI A,MKAOUER M W. Improving the prediction of continuous integration build failures using deep learning[J]. Automated Software Engineering,2022,29(1):21.

[25]SALINAS D,FLUNKERT V,GASTHAUS J,et al. DeepAR:Probabilistic forecasting with autoregressive recurrent networks[J]. International Journal of Forecasting,2020,36(3):1181-1191.

(责任编辑:张阳,殷锋,付强,和力新,肖丽;英文编辑:周序林,郑玉才)