# 基于多目标优化的工作量感知即时软件缺陷预测特征构建方法

赵晨阳, 刘磊, 江贺

引用本文

赵晨阳, 刘磊, 江贺. 基于多目标优化的工作量感知即时软件缺陷预测特征构建方法[J]. 计算机科学, 2025, 52(1): 232-241.

ZHAO Chenyang, LIU Lei, JIANG He. Feature Construction for Effort-aware Just-In-Time Software Defect Prediction Based on Multi-objective Optimization [J]. Computer Science, 2025, 52(1): 232-241.

---

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240100198

# 基于多目标优化的工作量感知即时软件缺陷预测特征构建方法

赵晨阳　刘　磊　江　贺
大连理工大学软件学院　辽宁 大连 116600
(zcy19998006@mail.dlut.edu.cn)

**摘　要**　即时软件缺陷预测(JIT-SDP)是一种针对代码变更的软件缺陷预测技术,具有细粒度、即时性和可追溯性的优点。工作量感知JIT-SDP进一步考虑代码检查工作量,旨在以有限的工作量识别更多的缺陷变更。尽管目前已有不少工作量感知JIT-SDP,但这些方法大多只针对分类模型算法进行优化。为提升工作量感知JIT-SDP的性能表现与泛用性,首次从特征工程方面入手,提出了一种工作量感知场景下的进化特征构建方法EEF。首先,EEF方法通过遗传编程树来表示特征,从分类性能与工作量感知性能两个角度出发,通过基于多目标优化的进化特征构建方法来获取新的特征转换方法;之后,通过得到的特征转换方法来构建新的特征集,并基于新的特征集训练与测试分类模型。为了验证EEF方法的有效性,在6个开源项目上,通过3个不同评估方案进行了实验研究,结果证明EEF方法可以提升分类模型在工作量感知场景下的性能,并优于其他特征工程方法,而且在保证特征选取多样性的前提下,基于单一模型的EEF方法同样可以提升其他模型的性能。

**关键词**:即时缺陷预测;工作量感知;进化特征构建;多目标优化;特征工程
**中图分类号**　TP311

# Feature Construction for Effort-aware Just-In-Time Software Defect Prediction Based on Multi-objective Optimization

ZHAO Chenyang, LIU Lei and JIANG He
School of Software, Dalian University of Technology, Dalian, Liaoning 116600, China

**Abstract**　Just-in-time software defect prediction (JIT-SDP) is a software defect prediction technology for code changes, which has the advantages of fine granularity, instantaneity, and traceability. Effort-aware JIT-SDP further considers the cost of code inspection and aims to detect more defective code changes with limited testing resources. Although many effort-aware JIT-SDPs have been proposed, most of them only optimize model algorithms. In order to improve the performance and generalizability of effort-aware JIT-SDP, an effort-aware evolutionary feature construction method EEF is proposed for the first time from the aspect of feature engineering. Firstly, EEF represents features through genetic programming trees. From the two aspects of classification performance and effort-aware performance, a new feature transformation is obtained through an evolutionary feature construction method based on multi-objective optimization. After that, a new feature set is constructed through the obtained feature transformation, and the classification model is trained and tested on the new feature set. In order to verify the effectiveness of EEF, experiments are conducted in three different evaluation schemes on six open source datasets. The results prove that EEF can improve the performance of the classification model in effort-aware scenarios and performs better than other feature engineering methods. Moreover, under the premise of ensuring the diversity of feature selection, EEF based on a single model can also improve the performance of other models.

**Keywords**　Just-in-time defect prediction, Effort-aware, Evolutionary feature construction, Multi-objective optimization, Feature engineering

## 1　引言

软件缺陷往往随着系统规模、功能和复杂性的增加而增多,这同时也增加了软件维护与验证工作的成本。软件缺陷预测可以通过推断可能包含缺陷的代码段,帮助开发人员有效节省测试时间,降低软件开发成本[1]。近年来,代码变更级别的缺陷预测引起了人们极大的兴趣,这种软件质量保证方法被称为即时软件缺陷预测(JIT-SDP)。JIT-SDP在代码变更级别进行缺陷预测,具有细粒度、即时性和可追溯性三大优点[2-3]。

在SDP领域,开发人员审查缺陷所需的工作(SQA工作)也是一个重要的考虑因素。基于工作量感知(Effort-Aware)软件缺陷预测考虑了使用预测模型的成本效益[4-5]。最近的研究表明,JIT-SDP中预测模型的分类性能和工作量

到稿日期:2024-01-29　返修日期:2024-06-22
通信作者:江贺(jianghe@dlut.edu.cn)

感知预测性能之间存在明显差距[6],这是因为两个目标之间存在明显的冲突。如果我们希望模型能够识别数据集中更多的有缺陷的变更,则需要更多的 SQA 工作;如果我们希望模型减少 SQA 工作,将错过更多有缺陷的变更。实现更有效、更具有泛用性的工作量感知场景下的 JIT-SDP 方法,主要面临以下几种挑战。

挑战 1 如何从模型算法之外的角度提升即时缺陷预测模型在工作量感知场景下的性能?为适应工作量感知场景,现有的 JIT-SDP 大多针对预测模型或算法本身进行改进与优化[6-9]。设计此类算法的难度往往较大,因此这些方法具有一定的局限性。能否从模型之外的角度入手,使得缺陷预测领域中成熟的分类模型可以更加方便地应用于工作量感知场景,是当前面临的挑战之一。

挑战 2 如何从特征工程角度对工作量感知场景下的即时缺陷预测模型进行优化?现有的针对 JIT-SDP 的特征工程方法一般是从预测角度进行优化,如 JIT-SDP 研究中常用的特征归一化处理、共线性处理等,包括已有的基于深度学习的特征工程,这些方法一般是针对分类模型的分类预测性能进行优化。能否设计一种专门针对工作量感知指标的特征工程方法,也是需要应对的挑战之一。

为了应对上述挑战,本文提出了 EEF(Effort-aware Evolutionary Feature Construction)——一种工作量感知场景下的进化特征构建方法。将多目标优化算法(MOA)应用于特征构建,设置两个优化目标,第一个目标是从模型分类性能的角度设计,第二个目标是从工作量感知角度设计。基于上述多目标优化算法,我们首次从特征工程的角度出发来提升 JIT-SDP 在工作量感知场景下的性能。以往的特征工程往往只能针对预测模型的分类精度进行性能优化,而进化特征构建可以实现工作量感知场景下的性能提升。将进化算法应用于特征构造上,通过遗传编程来构造特征,将原始特征空间变换到新的特征空间,以此来应对挑战 1 与挑战 2。

与基于深度学习的特征工程方法相比,基于进化算法的特征工程有以下优势:

1)现有的基于深度学习的特征工程方法一般是针对预测模型分类性能方面的优化,构建出来的特征并不一定能够适应工作量感知场景。而针对这种特定类型指标的优化,进化算法会是一种更好的选择。

2)基于深度学习的特征工程构建出来的特征往往可解释性较差,而基于进化算法构建的特征具有良好的可解释性,也更易于直接将构建好的特征表示应用在其他模型上。

在 6 个大型数据集上,将 EEF 方法在同项目交叉验证场景、跨项目验证场景和时间感知交叉验证场景下进行了性能评估验证。结果证明,EEF 方法不仅可以显著提升工作量感知 JIT-SDP 模型的性能,而且优于其他特征工程方法。

本文的主要贡献如下:

1)提出了一种称为 EEF 的特征构建方法,从工作量感知角度入手,提高了 JIT-SDP 在工作量感知场景下的性能。

2)首次从特征工程的角度对 JIT-SDP 的工作量感知预测性能进行优化,构造新的特征集来提升工作量感知场景下的模型性能。

3)对 6 个开源项目进行了大规模实验研究,调查模型在 3 种不同模型评估场景中的性能表现,结果证明了 EEF 方法的优越性。

## 2 相关工作

### 2.1 即时缺陷预测

Audris 等[10]最早进行变更级缺陷预测的研究,他们使用一些代码变更指标来预测引起缺陷的变更风险,研究结果表明 JIT-SDP 在工业上具有应用前景。Sunghun 等[11]使用从各种来源提取的大量特征来构建预测模型,进一步拓展了特征集。Yasutaka 等[12]基于 6 个开源项目和 5 个商业项目进行了大规模的实证研究,他们的方法可以获得 68% 的准确率和 64% 的召回率;此外,他们还探讨了 JIT-SDP 背景下跨项目缺陷预测的可行性[13]。

深度学习在即时软件缺陷预测中也有广泛的应用。Yang 等提出的 Deeper 模型[14],由深度信念网络和逻辑回归分类器组成。Hoang 等提出了 DeepJIT[15]——一种端到端的深度学习框架,其能自动从提交消息和代码变更中提取特征,并使用它们来识别缺陷;此外,他们考虑软件变更差异的层次结构,使用具有层次注意力层的卷积网络设计特征表示学习框架(CC2Vec)[16]。Jiri 等认为,除类别不平衡之外,还应该注意其他类型的数据不平衡;并结合 Siamese 网络提出了一种名为 SifterJIT 的深度神经网络模型,该模型的表现优于 CC2Vec[17]。

### 2.2 基于工作量感知的缺陷预测

Erik 等[18]考虑了使用缺陷预测模型时的成本效益。之后,Yasutaka 等[12]首次根据代码改动(代码变更修改的代码行数)评估了更改代码检查的工作量,他们提出的工作量感知线性回归(EALR)模型,使用 code churn(变更添加和删除的代码行数)来度量工作量。

Yang 等基于简单的启发式方法,提出了 LT,AGE,NUC 和 Entropy 等无监督模型[19]。这些模型会根据这些指标对软件变更进行排名,在有些情况下,其性能优于 Yasutaka 等的监督模型 EALR。此外,CBS 首先将变更分类为缺陷诱发和干净变更,然后根据启发式搜索被分类为缺陷诱发的变更[20],而 OneWay 模型会首先根据标记的变更集数据选择最佳启发式模型[21]。

Liu 等基于 code churn 提出了一种称为 CCUM 的预测模型[7]。该模型的排序性能虽然优于大部分的监督模型,但其整体的工作量消耗非常巨大。Li 等提出了一种名为 EATT 的半监督的即时软件缺陷预测模型[22],这是一种使用贪婪策略对变更进行排序的工作量感知模型。

### 2.3 进化特征构建

已有大量工作使用进化特征构建方法来构建特征集[23-25]。Michael 等[26]提出了一种使用单树表示的基于进化编程的特征构建方法,其目的是改进之前提出的进化算法,将每个原始特征转化为新特征。之前的大部分研究都是基于单树遗传编程(Genetic Programming,GP)构造特征,这种方法在高维数据集中会出现性能大幅下降的情况,而且训练往往需要大量的时间成本。Krzysztof[27]使用多树 GP 来构造

特征，结果表明，构造的特征提高了决策树模型在大多数数据集上的分类性能。之后，Zhang 等[28]将进化特征构建成功应用于更加复杂先进的机器学习模型，并取得了优秀的性能。

## 3 EEF 特征构建方法

### 3.1 基本流程

EEF 通过进化特征构建对 JIT 数据集的特征进行重新构造，在优化分类器性能的同时对工作量感知度量进行优化。在分类模型训练阶段，通过基于多目标优化方法的进化特征构建方法来构建特征集，为特征集选择合适的特征表示方式。

### 3.2 进化特征构建

EEF 算法的总体流程如图 1 所示。在训练集上，基于多目标优化的进化特征构造方法来构建特征集，进化出一个包含多个遗传编程(GP)树的种群，即种群中的每个个体都是一组 GP 树，在每个 GP 个体中，使用多棵 GP 树来表示多个特征。为了评估 GP 个体，基于该 GP 个体和训练数据构建机器学习模型，以模型性能优劣来评估 GP 个体表现，用表现好的特征个体更新种群，对种群再进行进一步的进化。反复迭代这一过程，直到达到终止条件为止。具体过程如下：

1)初始化：随机初始化种群。使用 GP 树来表示特征，由多棵 GP 树构成的特征集组成进化算法基本单位的个体，种群存储进化算法过程中表现良好的个体集合。

2)评估：根据训练数据的交叉验证来评估每个新个体的适应度值。适应度评估函数由两个目标函数组成：分类准确率评估函数与工作量感知评估函数。

3)进化：通过对父个体应用交叉和变异算子来生成新的候选个体。

4)选择：将父子两代进行合并，并基于两个目标函数，对合并后的种群进行非支配排序与拥挤度计算来进行选择。最终，使用精英策略选择高质量的个体进入下一代。

5)生成最终特征：达到进化算法的最终条件后，停止进化，得到最终种群，并从中得到最优个体，即最终特征集。利用该特征集来训练机器学习模型，从而得到最终模型。

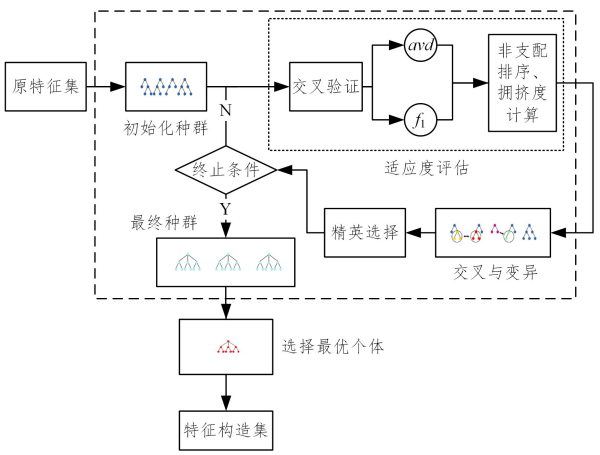



图 1 EEF 算法流程示意图

Fig. 1 Schematic diagram of EEF algorithm flow

#### 3.2.1 个体表示

本文使用 GP 树来构造特征，树的叶子节点叫做终止集，非叶子节点叫做符号集。终止集被分成了两类——常数和参数。常数在整个进化过程中保持不变；而参数是由程序输入的，在本方法中，参数就是原始特征集。

每个个体 $\varphi$ 由多棵 GP 树构成，可以表示为 $\varphi=\{\varphi_1,\varphi_1,\cdots,\varphi_m\}$，代表一组可用于构造的特征集。这些 GP 树通过对半方法随机初始化构建，即初始化的树中一半是满的，而另一半没有。每个种群又包含多个个体，可以表示为 $P=\{\phi_1,\phi_2,\cdots,\phi_s\}$。

我们选择多棵小型的 GP 树而非一棵大型的 GP 树来表示一个个体的方案，因为在之前的多个研究中已经证明[29-30]，这样的特征构建方案可以更好地提升模型性能，并且在时间性能表现上也更好。

#### 3.2.2 适应度评估

在训练数据上对分类模型进行交叉验证，用预测结果来评估每个新 GP 个体的适应度值。适应度评估函数由两个目标函数组成：分类准确率度量函数 $f1$ 与工作量度量函数 $avd$。

目标函数的具体计算过程如下所示：

$$Y_{\text{pred}}=\begin{cases}1, & y_{\text{pred}}>0.5\\ 0, & y_{\text{pred}}\leqslant 0.5\end{cases} \tag{1}$$

将 JIT-SDP 视为二元分类问题。分类模型预测的输出值 $y_{\text{pred}}$ 的范围是[0,1]。如果 $y_{\text{pred}}$ 的值大于 0.5，将代码变更分类为有错误，否则将代码变更分类为干净。

对于工作量感知的 JIT-SDP，构建模型时主要考虑优化目标。第一个目标是从分类模型角度考虑。对于一组数据，分类准确率度量函数 $f1$ 通过式(2)计算：

$$f1=2*\frac{precision*recall}{precision+recall} \tag{2}$$

其中，$precision$ 和 $recall$ 分别代表分类预测结果的准确率和召回率指标。

第二个目标是从工作量感知模型角度考虑。对于一组数据，根据数据集修改的代码行度量元，通过计算检测该代码变更所需的工作量，也就是 $codechurn$，来计算缺陷密度，以便进行之后的工作量计算。与 Yasutaka 等[12]的工作类似，我们使用代码改动(即添加和删除的总行数)来衡量工作量，即 $codechurn$ 的大小。对于变更 $c$，具体计算如下：

$$codechurn=(la(c)+ld(c)) \tag{3}$$

其中，$la(c)$和 $ld(c)$分别代表代码变更 $c$ 增加与删除的代码行数。

$$denisty(c)=\frac{y_{\text{pred}}}{codechurn} \tag{4}$$

基于 Elaine 等[31]的研究，对于一个代码变更 $c$，$denisty(c)$ 代表该变更的缺陷密度，那么工作量指标密度平均比例($avd$)的计算如下：

$$D=\sum_{1}^{M}denisty(c_i) \tag{5}$$

$$pd=\frac{1}{D}\sum_{m=M-i+1}^{M}denisty(c_i) \tag{6}$$

$$avd=\frac{1}{M}\sum_{i=1}^{M}pd \tag{7}$$

其中，$c_1,c_2,\cdots,c_M$ 为 $M$ 个代码变更按照预测缺陷密度升序排列；$denisty(c_i)$ 代表代码变更 $c_i$ 的实际缺陷密度，缺陷密度

之和定义为 $D$。检查每个代码变更所需的缺陷密度占总缺陷密度的比例用 $pd$ 来表示,则 $M$ 个代码变更 $pd$ 的平均值用 $avd$ 可以计算得出。工作量感知模型性能越好,则缺陷密度大的代码变的更排序会越靠前,即 $avd$ 越大;如果 $avd$ 越小,则表明模型工作量感知性能越差。

如果只考虑最大化 $avd$,则模型会将更多代码变更的标签标记为有缺陷,导致模型预测性能大幅下降,在实际工作场景中,反而会增加代码检查人员的工作量。因此,需要设置另一个目标函数,即 $f1$ 值,来保证模型在工作量感知场景下的准确度。将两个指标结合起来,可以保证模型在工作量感知场景下的性能表现更好。此外,可以通过设置两个指标的系数来适应不同的 JIT-SDP 方案。

为了获取每个个体的适应度函数,采用五折交叉验证方案来估计泛化评估函数。

3.2.3 选择、交叉与变异

每个个体都包含一组 GP 树。在后代生成中,通过锦标赛选择法选取个体,每次随机选择两个个体,每次优先选择 Pareto 排序等级高的个体,当排序等级一样时,则优选选择拥挤度大的个体。Pareto 排序等级与拥挤度具体在下一节里介绍。选择出个体后,再从每个个体中随机挑选 GP 树来执行变异和交叉操作。

1)交叉算子:用来自另一个父代的子树随机替换来自一个父代的子树。

2)变异算子:随机选择一棵子树并用随机生成的树替换它。

为了保持种群多样性,本研究中强制变异算子生成新的个体。

3.2.4 多目标优化

基于之前的分析,将 JIT-SDP 形式化为多目标优化(MOA)问题,两个适应度函数分别为 $f1$ 和 $avd$,代表预测性能与工作量感知两个目标。为了便于后续描述,首先给出一些关于 MOA 的定义。

**定义 1**(Pareto 支配) 假设 $\phi_i$ 和 $\phi_j$ 是种群中的两个个体,当且仅当 $f1(\phi_i)>f1(\phi_j)$ 且 $avd(\phi_i)\geqslant avd(\phi_j)$ 或 $f1(\phi_i)\geqslant f1(\phi_j)$ 且 $avd(\phi_i)>avd(\phi_j)$ 时,$\phi_i$ 对 $\phi_j$ 是 Pareto 支配。

**定义 2**(Pareto 最优解) 对于一个个体 $\phi$,当且仅当不存在其他个体对 $\phi$ 是 Pareto 支配时,$\phi$ 是 Pareto 最优解。

**定义 3**(Pareto 最优解集) 由所有 Pareto 最优解组成的集合称为 Pareto 最优解集。

**定义 4**(Pareto 前沿) 由所有 Pareto 最优解对应的向量组成的曲面称为 Pareto 前沿。

我们用一个实例来进一步解释这些定义。如图 2 所示,假设给定数据集,进化方法生成 7 个构造特征集。在该图中,$x$ 轴表示分类模型在构造特征集上的预测结果的 $avd$ 值,$y$ 轴表示分类模型在构造特征集上的预测结果的 $f1$ 值。构造特征集个体 $A,B,C,D$ 是其他个体的 Pareto 支配,因为它们的 $avd$ 值和 $f1$ 值均大于其他个体。而个体 $A,B,C,D$ 组成了 Pareto 最优解集,因为没有其他个体对它们具有 Pareto 支配。因此,曲面 $A$-$B$-$C$-$D$ 构成了这些个体中的 Pareto 前沿。

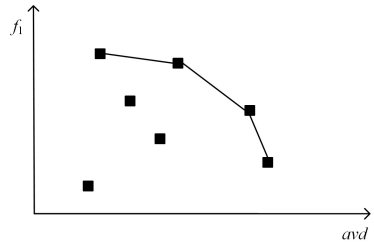



图 2 多目标优化示例

Fig. 2 Instance of MOA

将父子两代进行合并,基于 NSGA-II[32] 算法对合并后的种群进行多目标优化选择,具体步骤如下。

首先,通过快速非支配排序算法对个体之间的 Pareto 支配与非支配关系进行排序分层,将其分为不同的 Pareto 前沿。然后,为保证群体的多样性,需要在每一层级 Pareto 前沿上计算每个个体的拥挤度,从而在每个层级上比较个体的优异程度。种群中的每个个体都设定一个拥挤度参数,针对每个目标函数,找出与该解函数值相邻的两个解,并计算这两个解之间的函数差值。当前解的拥挤距离就是所有目标函数获得的相邻解函数差值之和,拥挤距离越小就代表该个体周围越拥挤。

最后,采用精英策略来防止优秀个体的流失。首先将父代种群和子代种群合并,根据以下偏序规则从合并后的种群中生成新的父代种群。

如果两个个体具有不同的 Pareto 前沿层级,则偏向于低 Pareto 前沿层级的个体;如果两个个体具有相同的 Pareto 前沿层级,则偏向于拥挤距离较大的个体。

3.2.5 构造特征集生成

重复上述进化算法过程,直至满足进化算法的最终条件后,停止进化并得到最终种群,也就是最优解集。根据 Xing 等[33] 的研究,选择最优解集中分类准确度度量即 $f1$ 值最大的解作为最优解,即最终特征表示集,可以获得性能更好的特征集。将该特征集用于后续机器学习模型训练,以得到最终模型。

## 4 实验设置

### 4.1 研究问题

为了研究 EEF 方法的有效性,设计了以下 3 个问题进行后续的实验研究。

1)RQ1:EEF 方法对工作量感知场景下的 JIT-SDP 模型性能提升效果如何?

2)RQ2:与其他的特征工程方法相比,EEF 方法对工作量感知场景下的 JIT-SDP 模型性能提升效果如何?

3)RQ3:基于单一模型的 EEF 方法构建的特征有效性如何?

### 4.2 实验框架

JIT-SDP 的实验流程如图 3 所示。首先,通过收集软件度量元与软件实体标签来构造即时软件缺陷预测数据集,并对数据集进行预处理。然后,根据具体方案划分训练数据集和测试数据集,通过 EEF 方法重新构建训练数据集并在新特征集上对模型进行训练,并将同样的特征构建方案应用在测试集上,用于后续的模型测试与评估。

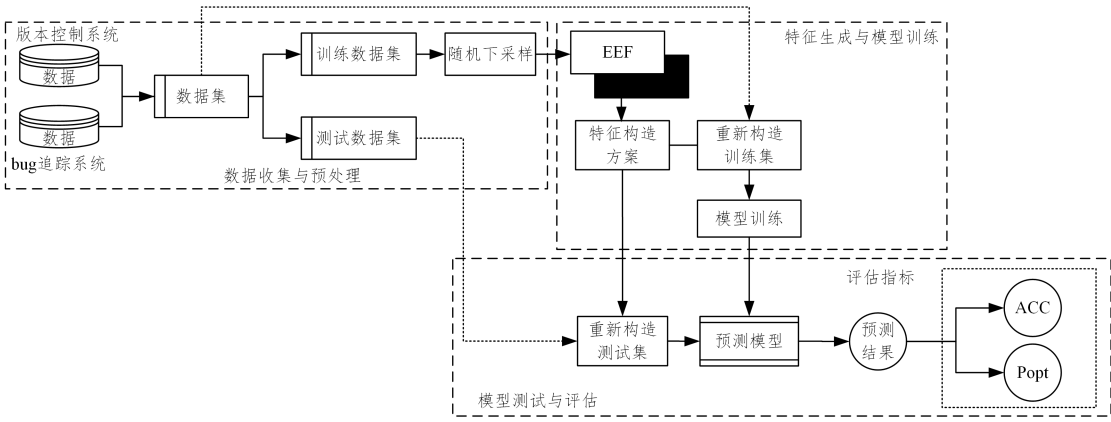



图 3 JIT-SDP 实验流程示意图

Fig. 3 Schematic diagram of JIT-SDP experiment process

### 4.3 数据预处理

如表 1 所列，从项目的源代码控制存储库数据中派生出 14 个指标，即数据集特征，这些指标分为代码分布、规模、目的、历史和经验 5 个维度。

表 1 原始特征集

Table 1 Original feature set

| 维度 | 特征 | 描述 |
|---|---|---|
| 代码分布 | NS | 代码变更修改的程序子系统数量 |
| | ND | 代码变更修改的代码目录数量 |
| | NF | 代码变更修改的文件数量 |
| | Entropy | 修改的代码在变更文件中的分布 |
| 规模 | LA | 代码变更增加的代码行数 |
| | LD | 代码变更减少的代码行数 |
| | LT | 代码变更相关的文件在变更提交之前的代码行数 |
| 目的 | FIX | 代码变更是否修复缺陷 |
| 历史 | NDEV | 代码变更相关文件修改过的开发者数量 |
| | PD | 代码变更相关文件最近变更与该变更的时间差 |
| | NUC | 代码变更相关文件修改过的次数 |
| 经验 | EXP | 开发者已提交的代码变更数量 |
| | REXP | 开发者近期提交的代码变更数目 |
| | SEXP | 开发者已提交的代码变更中影响到子系统的数目 |

此外，由于提取的数据集相对不平衡，因此我们使用随机下采样方法来解决这个问题，通过随机删除多数类的实例对训练数据进行重新采样，使得数据分布变得较为平衡。类不平衡处理只在训练集上使用，测试集等待模型训练完成后直接测试，以保证测试数据的真实性。

为了评估工作量感知场景下 JIT-SDP 模型的性能表现，我们分别采取 3 种实验评估方案。

1)同项目交叉验证场景：对于同项目交叉验证场景，采取十折交叉验证，将数据集分为 10 份，其中 9 份作为训练数据，其余部分用于测试评估。

2)跨项目验证场景：对于跨项目验证，进行跨不同项目的测试评估。基于一个项目(源项目)构建模型，并使用该模型来预测其他项目(目标项目)的代码变更。

3)时间交叉验证场景[34]：对于时间交叉验证，将同一项目内的代码变更条目按照日期进行排名，并将同一个月的代码变更分在同一组中，按照月份分为 $n$ 组。之后，首先根据第 $i$ 组和第 $i+1$ 组的数据构建模型，然后使用构建的模型来对第 $i+4$ 组和第 $i+5$ 组的数据进行预测，$n$ 组数据会生成 $n-5$ 条运行结果。

### 4.4 数据集

我们在 Yasutaka 等共享的 6 个大型开源项目进行了实验[2]。这 6 个项目都是大型长期项目，涵盖广泛的领域和规模，是即时软件缺陷预测领域最为经典且具有代表性的数据集。表 2 总结了这些项目的统计数据，数据集中的每个实例都对应于提交到版本控制系统的代码变更，所有的代码更改均从项目的 CVS 存储库中提取，并通过分析错误报告和更改日志进行标记。

表 2 数据集

Table 2 Dataset

| 项目 | 变更数目 | 缺陷数目 | 缺陷率 |
|---|---|---|---|
| BUG | 4620 | 1696 | 36.71 |
| COL | 4455 | 1361 | 30.55 |
| JDT | 35386 | 5089 | 14.38 |
| MOZ | 98275 | 5149 | 5.24 |
| PLA | 64250 | 9452 | 14.71 |
| POS | 20431 | 5119 | 25.06 |

### 4.5 评估指标

采用 ACC 和 $P_{opt}$ 来评估工作量感知场景下 JIT-SDP 的性能，这两个指标是衡量工作量感知场景下分类模型性能的常用指标[35]。

ACC 表示使用 20% 的工作量来检查排序靠前的代码变更时，所观察到的有缺陷的变更数量占所有有缺陷的变更数量的比值，即召回率。按照式(8)可以计算得到 ACC 指标。

$$ACC=N_d/N_{id} \tag{8}$$

其中，$N_d$ 表示测试集中有缺陷的提交实例的总数，$N_{id}$ 表示花费 20% 的工作量检查到的有缺陷的提交实例的总数。

$P_{opt}$ 表示当前模型与最好模型之间的差距，定义为 $1-\Delta_{opt}$，其中 $\Delta_{opt}$ 指基于工作量感知的提升图中最优模型和预测模型之间面积与最优模型和最差模型之间面积的比值，该图的 $x$ 轴表示工作量的累计百分比，$y$ 轴表示检测到的包含缺陷的更改的累计百分比。$P_{opt}$ 具体可以根据式(9)计算。

$$P_{opt}=1-\frac{Area(Optimal)-Area(Predict)}{Area(Optimal)-Area(Worst)} \tag{9}$$

如图 4 所示，$Area(optimal)$，$Area(Predict)$，$Area(worst)$ 分别表示 3 条曲线与 $x$ 轴围成的面积。

为了检查不同方法之间的预测性能是否存在显著差异，

我们采用威尔科克森符号秩检（Wilcoxon Signed-rank Test）[36]来检查差异的显著性水平，使用 p-value 来检查不同模型在 0.05 的显著性水平上是否具有统计学意义。如果统计检验显示存在显著差异，则使用 Cliff's $\delta$ 来衡量差异的大小。不同 Cliff's $\delta$ 值的含义及其相应的解释如表 3 所列。

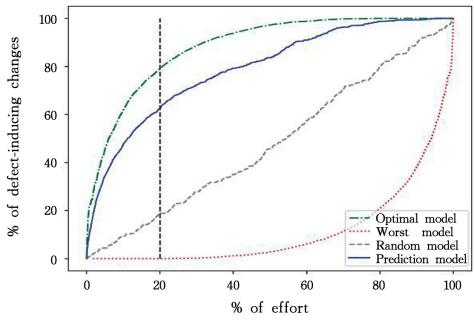



图 4 $P_{opt}$ 指标

Fig. 4 $P_{opt}$ metric

表 3 Cliff's $\delta$ 对应的差异性水平

Table 3 Cliff's $\delta$ values and their corresponding difference level

| Cliff's $\delta$ | 差异性水平 |
|---|---|
| $\|\delta\|<0.147$ | 极小 |
| $0.147\leqslant\|\delta\|<0.33$ | 小 |
| $0.33\leqslant\|\delta\|<0.474$ | 中 |
| $0.474\leqslant\|\delta\|$ | 大 |

### 4.6 参数配置

我们采用 Python3.8 来构建即时缺陷预测模型. 实验运行环境为 window10 系统。EEF 方法所涉及到的具体参数设置如表 4 所列。

表 4 参数设置

Table 4 Parameter settings

| 参数 | 值 |
|---|---|
| 种群规模 | 100 |
| 迭代进化次数 | 200 |
| 交叉与变异概率 | 0.7 与 0.1 |
| 最大树高度 | 12 |
| 符号集 | +，−，*，/，log，sqrt，square |
| 个体规模 | 10 |

## 5 实验结果评估与分析

### 5.1 RQ1 实验与分析

RQ1：EEF 方法对工作量感知场景下的 JIT-SDP 模型性能提升效果如何？

动机：如果所提出的 EEF 方法可以提高不同评估方案下工作量感知场景中常用的不同分类器的性能，则可以证明 EEF 方法的有效性。

方法：本实验选择了以往 JIT 工作中常用的分类器，即逻辑回归模型（Logistic Regression，LR）、朴素贝叶斯模型（Naive Bayes，NB）、决策树模型（Decision Tree，DT）以及随机森林模型（Random Forest，RF），方法使用 sklearn 库中的默认参数。在 3 个评估场景下进行实验，来检验 EEF 算法的有效性。

实验结果：表 5、表 6 和表 7 分别展示了在同项目交叉验证、跨项目验证与时间交叉验证中，基于原始特征集建立的预测模型和基于 EEF 方法构造的新特征数据集建立的分类模型的性能，表中 AVE 行代表所有数据集评价指标结果的平均值。此外，跨项目验证中的项目列，指以该项目为训练集，以其他 5 个项目集为测试集所得出的评估结果的平均值。

结果分析：从表 5 可以看出，在同项目交叉验证场景中，应用 EEF 方法后，LR 模型性能提升最明显，*ACC* 和 $P_{opt}$ 分别提升 40%和 38.7%；其次是 NB 模型，分别提升 29.1%和 27.1%；RF 模型分别提升 22.6%和 21.4%；而 DT 模型性能提升效果最差，分别提升 11.6%和 21.4%。

从表 6 可以看出，在跨项目验证场景中，应用 EEF 方法后，LR 模型的性能提升最为明显，*ACC* 和 $P_{opt}$ 分别提升 69.5%和 51.9%；DT 模型性能提升最差，分别提升 29.0%和 26.6%；NB 模型和 RF 模型分别提升 35.6%和 29.8%、40.0%和 31.2%。

从表 7 可以看出，在时间感知交叉验证中，应用 EEF 方法后，NB 模型的性能提升最明显，*ACC* 和 $P_{opt}$ 分别提升 63.3%和 60.6%；LR 模型分别提升 22.4%和 17.7%；DT 模型分别提升 27.7%和 16.8%；RF 模型分别提升 28.8%和 22.4%。可以看出，因为 EEF 方法构建特征的过程依赖于模型训练，对于时间感知评估场景来说，训练数据集的规模远小于前两种场景，因此 EEF 方法的性能也受到了影响。

表 5 十折交叉验证

Table 5 10-fold cross-validation

| | 项目 | LR 模型 | | NB 模型 | | DT 模型 | | RF 模型 | |
|---|---|---|---|---|---|---|---|---|---|
| | | 原始 | EEF | 原始 | EEF | 原始 | EEF | 原始 | EEF |
| *ACC* | BUG | 0.602 | 0.731 | 0.756 | 0.798 | 0.570 | 0.612 | 0.580 | 0.692 |
| | COL | 0.520 | 0.810 | 0.473 | 0.774 | 0.534 | 0.592 | 0.567 | 0.631 |
| | JDT | 0.547 | 0.701 | 0.575 | 0.733 | 0.539 | 0.581 | 0.573 | 0.684 |
| | MOZ | 0.451 | 0.673 | 0.450 | 0.706 | 0.502 | 0.611 | 0.531 | 0.716 |
| | PLA | 0.629 | 0.838 | 0.738 | 0.814 | 0.567 | 0.618 | 0.624 | 0.739 |
| | POS | 0.482 | 0.768 | 0.516 | 0.705 | 0.531 | 0.607 | 0.556 | 0.741 |
| | AVE | 0.539 | 0.754 | 0.585 | 0.755 | 0.541 | 0.604 | 0.572 | 0.701 |
| $P_{opt}$ | BUG | 0.729 | 0.893 | 0.887 | 0.929 | 0.706 | 0.797 | 0.727 | 0.837 |
| | COL | 0.592 | 0.910 | 0.571 | 0.912 | 0.617 | 0.768 | 0.644 | 0.782 |
| | JDT | 0.637 | 0.870 | 0.695 | 0.907 | 0.639 | 0.797 | 0.667 | 0.807 |
| | MOZ | 0.593 | 0.866 | 0.633 | 0.886 | 0.651 | 0.805 | 0.685 | 0.855 |
| | PLA | 0.711 | 0.914 | 0.829 | 0.913 | 0.667 | 0.811 | 0.714 | 0.860 |
| | POS | 0.596 | 0.899 | 0.660 | 0.884 | 0.642 | 0.785 | 0.684 | 0.863 |
| | AVE | 0.643 | 0.892 | 0.713 | 0.905 | 0.654 | 0.794 | 0.687 | 0.834 |

表 6 跨项目验证

Table 6 Cross-project-validation

| | 项目 | LR 模型 | | NB 模型 | | DT 模型 | | RF 模型 | |
|---|---|---|---|---|---|---|---|---|---|
| | | 原始 | EEF | 原始 | EEF | 原始 | EEF | 原始 | EEF |
| *ACC* | BUG | 0.392 | 0.628 | 0.413 | 0.631 | 0.427 | 0.583 | 0.405 | 0.611 |
| | COL | 0.515 | 0.692 | 0.407 | 0.729 | 0.475 | 0.625 | 0.370 | 0.654 |
| | JDT | 0.585 | 0.809 | 0.561 | 0.723 | 0.514 | 0.701 | 0.548 | 0.712 |
| | MOZ | 0.249 | 0.778 | 0.638 | 0.721 | 0.485 | 0.602 | 0.443 | 0.625 |
| | PLA | 0.536 | 0.744 | 0.696 | 0.753 | 0.508 | 0.623 | 0.531 | 0.694 |
| | POS | 0.411 | 0.771 | 0.526 | 0.837 | 0.484 | 0.599 | 0.508 | 0.632 |
| | AVE | 0.448 | 0.737 | 0.540 | 0.732 | 0.482 | 0.622 | 0.468 | 0.655 |
| $P_{opt}$ | BUG | 0.519 | 0.834 | 0.568 | 0.751 | 0.624 | 0.687 | 0.610 | 0.733 |
| | COL | 0.633 | 0.855 | 0.544 | 0.884 | 0.542 | 0.765 | 0.481 | 0.764 |
| | JDT | 0.702 | 0.900 | 0.704 | 0.895 | 0.627 | 0.805 | 0.657 | 0.811 |
| | MOZ | 0.392 | 0.911 | 0.739 | 0.867 | 0.592 | 0.755 | 0.530 | 0.759 |
| | PLA | 0.651 | 0.838 | 0.827 | 0.933 | 0.610 | 0.787 | 0.637 | 0.804 |
| | POS | 0.533 | 0.878 | 0.662 | 0.921 | 0.591 | 0.741 | 0.598 | 0.740 |
| | AVE | 0.5720.505 | 0.869 | 0.674 | 0.875 | 0.598 | 0.757 | 0.586 | 0.769 |

表 7 时间感知交叉验证

Table 7 Timewise-cross-validation

| | 项目 | LR 模型 | | NB 模型 | | DT 模型 | | RF 模型 | |
|---|---|---|---|---|---|---|---|---|---|
| | | 原始 | EEF | 原始 | EEF | 原始 | EEF | 原始 | EEF |
| *ACC* | BUG | 0.475 | 0.557 | 0.491 | 0.551 | 0.507 | 0.611 | 0.476 | 0.610 |
| | COL | 0.452 | 0.542 | 0.347 | 0.604 | 0.485 | 0.572 | 0.438 | 0.574 |
| | JDT | 0.462 | 0.565 | 0.334 | 0.595 | 0.416 | 0.529 | 0.412 | 0.558 |
| | MOZ | 0.461 | 0.562 | 0.317 | 0.581 | 0.459 | 0.557 | 0.465 | 0.597 |
| | PLA | 0.504 | 0.592 | 0.301 | 0.653 | 0.471 | 0.604 | 0.491 | 0.581 |
| | POS | 0.438 | 0.593 | 0.372 | 0.544 | 0.452 | 0.612 | 0.447 | 0.596 |
| | AVE | 0.465 | 0.569 | 0.360 | 0.588 | 0.465 | 0.581 | 0.455 | 0.586 |
| $P_{opt}$ | BUG | 0.639 | 0.691 | 0.631 | 0.732 | 0.644 | 0.754 | 0.613 | 0.747 |
| | COL | 0.601 | 0.717 | 0.478 | 0.818 | 0.639 | 0.699 | 0.574 | 0.718 |
| | JDT | 0.605 | 0.727 | 0.506 | 0.805 | 0.567 | 0.687 | 0.559 | 0.694 |
| | MOZ | 0.611 | 0.721 | 0.495 | 0.778 | 0.618 | 0.701 | 0.619 | 0.738 |
| | PLA | 0.625 | 0.726 | 0.469 | 0.823 | 0.601 | 0.676 | 0.605 | 0.736 |
| | POS | 0.584 | 0.734 | 0.519 | 0.767 | 0.599 | 0.612 | 0.589 | 0.721 |
| | AVE | 0.611 | 0.719 | 0.490 | 0.7870.68 | 0.589 | 0.688 | 0.593 | 0.726 |

总的来说，使用 EEF 的工作量感知场景下，JIT 预测模型在 3 种不同评估方案下的性能优于原始模型，证明了 EEF 方法的有效性。

### 5.2 RQ2 实验与分析

RQ2：与其他的特征工程方法相比，EEF 对工作量感知场景下的 JIT-SDP 模型性能提升效果如何？

动机：EEF 方法改变原始特征空间，重新构建新的特征集，属于一种特征工程算法。为了进一步检验 EEF 方法的有效性，需要与其他特征工程方法进行对比实验。

方法：本实验将 EEF 算法与 3 种常用的特征工程方法进行比较。这 3 种方法为：主成分分析（Principal Component Analysis，PCA）、线性判别分析（Linear Discriminant Analysis，，LDA）以及自动编码器（Autoencoders，AE），方法使用 sklearn 库中的默认参数。分类器选择之前工作量感知场景中表现良好的 NB 模型和工作量感知场景研究中广泛使用的 LR 模型[8-9]，通过在 3 个评估场景下进行对比实验来检验 EEF 算法的有效性。

实验结果：表 8 列出了不同评估场景下基于 LR 模型的 EEF 与其他特征工程方法的性能对比；表 9 列了不同评估场景下基于 NB 模型的 EEF 与其他特征工程方法的性能对比，其中 W/D/L 行展示了应用 EEF 方法后对比应用 EEF 方法前，预测模型在 6 个数据集上预测结果指标基于威尔科克森符号秩检验下表现更好/相等/更差的次数。

结果分析：从表 8 可以看出，EEF 方法在十折交叉验证与跨项目验证场景下，基于 LR 模型的性能表现均优于其他几种特征工程方法；在时间感知评估场景下，EEF 方法在 BUG 数据集上的 ACC 和 $P_{opt}$ 指标表现不如 LDA 方法，在 MOZ 数据集上的 ACC 指标表现不如 AE 方法，但在其他数据集上都优于其他方法。时间感知评估场景下 EEF 方法表现较差的原因仍如前文所说。

从表 9 可以看出，EEF 方法在十折交叉验证与跨项目验证场景下，基于 NB 模型的性能表现均基本优于其他几种特征工程方法；在时间感知评估场景下，EEF 方法在 BUG 数据集和 POS 数据集上的 ACC 指标表现不如 LDA 方法，其他数据集上的性能表现都基本优于其他特征工程方法。

总体而言，EEF 方法在不同评估场景中的大部分数据集上的表现都明显优于其他特征工程方法；在时间感知评估场景下，由于训练集样本的减少，EEF 方法的性能可能存在一定程度的下降。

表 8 基于 LR 模型的 EEF 和其他特征工程方法的性能对比

Table 8 Performance comparison of LR model based EEF and other feature engineering methods

| | 项目 | 十折交叉验证 | | | | 跨项目验证 | | | | 时间感知交叉验证 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | EEF | PCA | LDA | AE | EEF | PCA | LDA | AE | EEF | PCA | LDA | AE |
| *ACC* | BUG | 0.731 | 0.564 | 0.555 | 0.631 | 0.628 | 0.436 | 0.564 | 0.426 | 0.557 | 0.465 | 0.668 | 0.554 |
| | COL | 0.701 | 0.391 | 0.582 | 0.530 | 0.692 | 0.433 | 0.575 | 0.391 | 0.542 | 0.418 | 0.571 | 0.443 |
| | JDT | 0.810 | 0.556 | 0.608 | 0.502 | 0.809 | 0.619 | 0.559 | 0.590 | 0.565 | 0.436 | 0.550 | 0.399 |
| | MOZ | 0.673 | 0.402 | 0.503 | 0.454 | 0.778 | 0.292 | 0.586 | 0.499 | 0.562 | 0.427 | 0.476 | 0.596 |
| | PLA | 0.838 | 0.573 | 0.667 | 0.470 | 0.744 | 0.523 | 0.545 | 0.250 | 0.592 | 0.411 | 0.609 | 0.425 |
| | POS | 0.768 | 0.446 | 0.503 | 0.466 | 0.771 | 0.491 | 0.595 | 0.635 | 0.593 | 0.426 | 0.507 | 0.415 |
| | AVE | 0.754 | 0.489 | 0.570 | 0.509 | 0.737 | 0.466 | 0.570 | 0.465 | 0.569 | 0.431 | 0.564 | 0.472 |
| | W/D/L | | 6/0/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 2/3/1 | 4/1/1 |
| $P_{opt}$ | BUG | 0.893 | 0.702 | 0.692 | 0.762 | 0.834 | 0.567 | 0.663 | 0.623 | 0.691 | 0.627 | 0.773 | 0.639 |
| | COL | 0.870 | 0.554 | 0.652 | 0.500 | 0.855 | 0.552 | 0.683 | 0.531 | 0.717 | 0.582 | 0.653 | 0.594 |
| | JDT | 0.910 | 0.643 | 0.691 | 0.615 | 0.900 | 0.734 | 0.664 | 0.742 | 0.727 | 0.587 | 0.709 | 0.571 |
| | MOZ | 0.866 | 0.545 | 0.653 | 0.609 | 0.911 | 0.451 | 0.678 | 0.621 | 0.721 | 0.588 | 0.701 | 0.577 |
| | PLA | 0.914 | 0.675 | 0.748 | 0.604 | 0.838 | 0.641 | 0.652 | 0.438 | 0.726 | 0.547 | 0.724 | 0.570 |
| | POS | 0.899 | 0.559 | 0.621 | 0.586 | 0.878 | 0.538 | 0.692 | 0.745 | 0.734 | 0.571 | 0.681 | 0.571 |
| | AVE | 0.892 | 0.613 | 0.676 | 0.613 | 0.869 | 0.581 | 0.672 | 0.617 | 0.719 | 0.584 | 0.707 | 0.587 |
| | W/D/L | | 6/0/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 4/1/1 | 6/0/0 |

表 9 基于 NB 模型的 EEF 和其他特征工程方法的性能对比

Table 9 Performance comparison of LR model based EEF and other feature engineerinjg methods

| | 项目 | 十折交叉验证 | | | | 跨项目验证 | | | | 时间感知交叉验证 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | EEF | PCA | LDA | AE | EEF | PCA | LDA | AE | EEF | PCA | LDA | AE |
| *ACC* | BUG | 0.798 | 0.759 | 0.568 | 0.750 | 0.631 | 0.352 | 0.563 | 0.425 | 0.551 | 0.519 | 0.600 | 0.493 |
| | COL | 0.774 | 0.477 | 0.525 | 0.515 | 0.729 | 0.302 | 0.515 | 0.464 | 0.604 | 0.393 | 0.549 | 0.383 |
| | JDT | 0.733 | 0.729 | 0.569 | 0.574 | 0.723 | 0.685 | 0.530 | 0.491 | 0.595 | 0.369 | 0.537 | 0.341 |
| | MOZ | 0.706 | 0.596 | 0.453 | 0.581 | 0.721 | 0.455 | 0.529 | 0.331 | 0.581 | 0.313 | 0.517 | 0.318 |
| | PLA | 0.814 | 0.756 | 0.660 | 0.450 | 0.753 | 0.691 | 0.516 | 0.258 | 0.653 | 0.309 | 0.598 | 0.327 |
| | POS | 0.705 | 0.545 | 0.458 | 0.498 | 0.837 | 0.647 | 0.498 | 0.630 | 0.544 | 0.381 | 0.558 | 0.385 |
| | AVE | 0.755 | 0.644 | 0.539 | 0.561 | 0.732 | 0.522 | 0.525 | 0.433 | 0.588 | 0.381 | 0.560 | 0.375 |
| | W/D/L | | 5/1/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 4/0/2 | 6/0/0 |
| $P_{opt}$ | BUG | 0.929 | 0.888 | 0.691 | 0.882 | 0.751 | 0.555 | 0.663 | 0.556 | 0.732 | 0.667 | 0.735 | 0.659 |
| | COL | 0.912 | 0.518 | 0.601 | 0.614 | 0.884 | 0.446 | 0.619 | 0.651 | 0.818 | 0.549 | 0.671 | 0.595 |
| | JDT | 0.907 | 0.849 | 0.660 | 0.741 | 0.895 | 0.821 | 0.631 | 0.623 | 0.805 | 0.541 | 0.676 | 0.572 |
| | MOZ | 0.886 | 0.780 | 0.595 | 0.493 | 0.867 | 0.599 | 0.619 | 0.482 | 0.778 | 0.489 | 0.664 | 0.546 |
| | PLA | 0.913 | 0.859 | 0.736 | 0.539 | 0.933 | 0.814 | 0.624 | 0.419 | 0.823 | 0.484 | 0.704 | 0.544 |
| | POS | 0.884 | 0.720 | 0.559 | 0.650 | 0.921 | 0.774 | 0.590 | 0.861 | 0.767 | 0.534 | 0.647 | 0.594 |
| | AVE | 0.905 | 0.769 | 0.640 | 0.653 | 0.875 | 0.668 | 0.624 | 0.599 | 0.787 | 0.544 | 0.683 | 0.585 |
| | W/D/L | | 6/0/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 6/0/0 | 6/0/0 | | 6/0/0 | 5/1/0 | 6/0/0 |

### 5.3 RQ3 实验与分析

RQ3：基于单一模型的 EEF 方法构建的特征有效性如何？

动机：前面两个实验是针对不同的模型构建对应的特征集来检验 EEF 方法的有效性，为了进一步验证 EEF 方法能否构建出有效特征，同时为了检验 EEF 方法构建特征的泛用性，需要研究能否通过单一模型所构建的特征来提升其他模型的性能。

方法：本实验中选择 DT 模型作为构建特征集的模型，选择 LR 模型、NB 模型和 RF 模型在 6 个数据集上进行十折交叉验证。首先，基于 DT 模型，通过 EEF 方法构建特征集，并将其直接应用在其他模型上，这种方案用 EEF-DT 表示。但由于最终特征集的多样性较差，因此这种方案很有可能并不适用其他模型。

为进一步研究 EEF 方法构建特征集的可拓展性，我们提出另一种方案，对最终特征集的选取方案做出改变：不采取直接选用最终种群中分类性能表现最好的个体的方案，而是根据特征重要性对最终种群中所有个体中的所有特征进行排序，从中选出表现良好的特征，以此来保证选取特征集的多样性。我们通过分析决策树模型的不纯度下降来计算特征重要性，而 EEF 方法的其他部分不做改变，我们将这种方案用 EEF-FI 表示。同时，为了验证这种方案对原模型性能的提升程度，也将 EEF-FI 应用在 DT 模型上。

实验结果：实验结果如表 10 所列。

表 10 十折交叉验证

Table 10 10-fold cross-validation

| | 项目 | LR 模型 | | | NB 模型 | | | RF 模型 | | | DT 模型 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 原始 | EEF-DT | EEF-FI | 原始 | EEF-DT | EEF-FI | 原始 | EEF-DT | EEF-FI | 原始 | EEF-FI |
| *ACC* | BUG | 0.602 | 0.490 | 0.630 | 0.756 | 0.778 | 0.785 | 0.580 | 0.615 | 0.592 | 0.570 | 0.584 |
| | COL | 0.520 | 0.415 | 0.508 | 0.473 | 0.434 | 0.759 | 0.567 | 0.556 | 0.579 | 0.534 | 0.557 |
| | JDT | 0.547 | 0.469 | 0.526 | 0.575 | 0.648 | 0.701 | 0.573 | 0.566 | 0.591 | 0.539 | 0.566 |
| | MOZ | 0.451 | 0.431 | 0.440 | 0.450 | 0.610 | 0.653 | 0.531 | 0.546 | 0.571 | 0.502 | 0.535 |
| | PLA | 0.629 | 0.560 | 0.673 | 0.738 | 0.753 | 0.748 | 0.624 | 0.608 | 0.670 | 0.567 | 0.585 |
| | POS | 0.482 | 0.496 | 0.579 | 0.516 | 0.483 | 0.674 | 0.556 | 0.573 | 0.672 | 0.531 | 0.548 |
| | AVE | 0.539 | 0.477 | 0.559 | 0.585 | 0.618 | 0.720 | 0.572 | 0.577 | 0.613 | 0.541 | 0.563 |

（续表）

| | 项目 | LR 模型 | | | NB 模型 | | | RF 模型 | | | DT 模型 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 原始 | EEF-DT | EEF-FI | 原始 | EEF-DT | EEF-FI | 原始 | EEF-DT | EEF-FI | 原始 | EEF-FI |
| $P_{opt}$ | BUG | 0.729 | 0.660 | 0.768 | 0.887 | 0.901 | 0.910 | 0.727 | 0.731 | 0.718 | 0.706 | 0.737 |
| | COL | 0.592 | 0.493 | 0.529 | 0.571 | 0.535 | 0.872 | 0.644 | 0.665 | 0.717 | 0.617 | 0.723 |
| | JDT | 0.637 | 0.568 | 0.613 | 0.695 | 0.744 | 0.844 | 0.667 | 0.705 | 0.715 | 0.639 | 0.704 |
| | MOZ | 0.593 | 0.582 | 0.599 | 0.633 | 0.814 | 0.840 | 0.685 | 0.643 | 0.655 | 0.651 | 0.678 |
| | PLA | 0.711 | 0.676 | 0.783 | 0.829 | 0.863 | 0.867 | 0.714 | 0.710 | 0.739 | 0.667 | 0.725 |
| | POS | 0.596 | 0.610 | 0.712 | 0.660 | 0.554 | 0.821 | 0.684 | 0.698 | 0.780 | 0.642 | 0.689 |
| | AVE | 0.643 | 0.598 | 0.667 | 0.713 | 0.735 | 0.859 | 0.687 | 0.692 | 0.721 | 0.654 | 0.709 |

结果分析：从表 10 可以看出，直接将 DT 模型生成的特征集应用在其他模型上，性能提升效果并不好，甚至会导致 LR 模型性能下降，RF 模型和 NB 模型性能提升效果也不稳定；但在保证选取特征集多样性的前提下，EEF-FI 方案生成的特征可以在一定程度上提升其他分类器模型在工作量感知场景下的性能表现，虽然提升效果不如原 EEF 方法，但这种方案有更好的泛用性，也进一步验证了 EEF 方法构建的特征集的有效性。此外，我们通过 DT 模型上的结果对比，发现即使基于同一模型生成的特征，EEF-FI 方案的表现也不如原 EEF 方法，这是因为根据特征重要性评估，只考虑模型的分类性能，忽视了工作量感知性能，导致工作量感知性能提升程度不如原方法。

**结束语** 本文提出一种工作量感知场景下的进化特征构建方法 EEF，将多目标优化算法应用于特征构建，然后将构建好的特征应用在机器学习模型上，从而可以提升即时软件缺陷预测模型在工作量感知下的性能表现。

通过在 3 种评估场景下 6 个大型数据集上的实验，证明了 EEF 方法的有效性。结果证明，EEF 方法不仅可以大幅提升分类模型在工作量感知场景下的性能，而且也优于其他特征工程方法。此外，我们还发现，在保证特征选取的多样性前提下，单一模型生成特征集同样可以在一定程度上提升其他模型的性能。

尽管目前的实验结果已证明 EEF 方法可以取得不错的结果，但仍有进一步的研究空间。

1）EEF-FI 方案针对单一模型生成特征集的性能提升效果有限，在未来的工作中需要对 EEF 方法的泛用性进行进一步的研究与实验。

2）目前的研究是针对 JIT-SDP 研究中常用的数据集进行实验，需进一步在其他类型数据集中进行 EEF 方法的实验与研究，以拓展实验结论。

3）此外，在未来的工作中，EEF 方法在时间交叉感知场景下的性能也需要进一步研究优化。

## 参 考 文 献


[1] MENDE T, RAINER K. Revisiting the evaluation of defect prediction models[C]// 5th International Conference on Predictor Models in Software Engineering. 2009: 1-10.

[2] YUNHUA Z, KOSTADIN D, HUI C. A Systematic Survey of Just-in-Time Software Defect Prediction[J]. ACM Comput. Surv., 2023, 55(10): 1-35.

[3] YASUTAKA K, EMAD S, BRAM A, et al. A large - scale empirical study of just-in-time quality assurance[J]. IEEE Transactions on Software Engineering, 2013, 39(6): 757-733.

[4] YANG X G, YU H Q, FAN G S. An Empirical Study on Progressive Sampling for Just-in-Time Software Defect Prediction [C]// Proceedings of the 7th International Workshop on Quantitative Approaches to Software Quality. 2019: 12-18.

[5] MENDE T, RAINER K. Effort-aware defect prediction models [C]// 14th European Conference on Software Maintenance and Reengineering. 2010: 107-116.

[6] GUO Y C, MARTIN S, LI N. Bridging effort-aware prediction and strong classification: a just-in-time software defect prediction study[C]// IEEE/ACM 40th International Conference on Software Engineering: Companion. 2018: 325-326.

[7] LIU J P, ZHOU Y M, YANG Y B, et al. Code churn: A neglected metric in effort-aware just-in-time defect prediction[C]// ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2017: 11-19.

[8] CHEN X, ZHAO Y Q, WANG Q P, et al. MULTI: Multi-objective effort-aware just-in-time software defect prediction[J]. Information and Software Technology, 2018, 93: 1-13.

[9] YANG X G, YU H Q FAN G S. A differential evolution-based approach for effort-aware just-in-time software defect prediction [C]// Proceedings of the 1st ACM SIGSOFT International Workshop on Representation Learning for Software Engineering and Program Languages. 2020: 13-16.

[10] AUDRIS M, DAVID W. Predicting risk of software changes[J]. Bell Labs Tech, 2000, 5(2): 169-180.

[11] SUNGHUN K, JAMES W, YI Z. Classifying software changes: clean or buggy[J]. IEEE Trans. Softw. Eng., 2008, 34(2): 181-196.

[12] YASUTAKA K, EMAD S, BRAM A, et al. A large-scale empirical study of just-in-time quality assurance[J]. IEEE Transactions on Software Engineering, 2013, 39(6): 757-733.

[13] KAMEI Y, FUKUSHIMA T, MCINTOSH S, et al. Studying just-in-time defect prediction using cross-project models[J]. Empir. Softw. Eng., 2016, 21(5): 2072-2106.

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

[15] THONG H, HOA K, YASUTAKA K, et al. DeepJIT: An end-to-end deep learning framework for just-in-time defect prediction [C]// IEEE/ACM 16th International Conference on Mining Software Repositories(MSR'19). IEEE, 2019: 34-45.

[16] THONG H, HONG J, DAVID L, et al. CC2Vec: Distributed representations of code changes[C]// ACM/IEEE 42nd International Conference on Software Engineering. 2020: 518-529.

[17] JIRI G, JIAWEI L, IFTEKHAR A. An Empirical Examination

of the Impact of Bias on Just-in-time Defect Prediction[C]// ACM/IEEE International Symposium on Empirical Software Engineering and Measurement(ESEM). 2021:1-12.

[18] ERIK A,LIONEL B,EIVIND J. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models[J]. Journal of Systems and Software,2010,83(1):2-17.

[19] YANG Y,ZHOU Y,LIU J P,et al. Effort-aware just-in-time defect prediction:simple unsupervised models could be better than supervised models[C]// 24th ACM SIGSOFTInternational Symposium on Foundations of Software Engineering. 2016:157-168.

[20] CHAO N,XIN X,DAVID L. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction[J]. Empirical Software Engineering,2019,24(5):2823-2862.

[21] WEI F,TIM M. Revisiting unsupervised learning for defect prediction[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017:72-83.

[22] LI W W,ZHANG W Z,JIA X Y. Effort-Aware Semi-Supervised Just-in-Time Defect Prediction[J]. Information and Software Technology,2020,126:106351-106364.

[23] BI Y,XUE B,ZHANG M J. Genetic programming with image-related operators and a flexible program structure for feature learning in image classification[J]. IEEE Trans. Evol. Comput.,2021,25(1):87-101.

[24] BALIGH A,QI C,BING X. Multi-tree genetic programming for feature construction-based domain adaptation in symbolic regression with incomplete data[C]// Proceedings of the 2020 Genetic and Evolutionary Computation Conference(GECCO'20). Association for Computing Machinery. 2020:913-921.

[25] LENSEN A,XUE B,ZHANG M J. Genetic programming for evolving similarity functions for clustering:Representations and analysis[J]. Evol. Comput.,2020,28(4):531-561.

[26] MICHAEL L R,WILLIAM F P,ERIK D G,et al. Genetic programming for improved data mining:application to the biochemistryof protein interactions[C]// Proceedings of the 1st Annual Conference on Genetic Programming. 996:375-380.

[27] KRZYSZTOF K. Genetic programming-based construction of features for machine learning and knowledge discovery tasks [J]. Genet. Program. Evol. Mach.,2002,3:329-343.

[28] ZHANG H Z,ZHOU A M,ZHANG H. An Evolutionary Forest for Regression[J]. IEEE Transactions on Evolutionary Computation,2022,26(4):735-749.

[29] BINH T,BING X,MENG J Z. Genetic programming for multiple feature construction on high-dimensional classification[J]. Pattern Recognit,2019,93:404-417.

[30] WILLIAM L,JASON H. Learning feature spaces for regression with genetic programming[J]. Genet. Program. Evol. Mach.,2020,21:433-467.

[31] ELAINE J,THOMAS J,ROBERT M. Comparing the effectiveness of several modeling methods for fault prediction[J]. Empirical Software Engineering,2010,15(3):277-295.

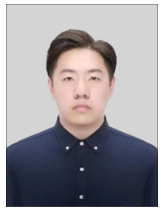
[32] KALYAN D,AMRIT P,SAMEER A,et al. A fast and elitist multiobjective genetic algorithm:NSGA-II[J]. IEEE Trans. Evol. Comput.,2002,6(2):182-197.

[33] XING G Y,HUI Q Y,GUI S F. An empirical study on optimal solutions selection strategies for effort-aware just-in-time software defect prediction[C]// Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering. 2019:319-324.

[34] JACEK S,THOMAS Z,ANDREAS Z. When do changes induce fixes[C]// Proceedings of the International Workshop on Mining Software Repositories. 2005:1-5.

[35] YI B Y,YU M Z,JIN P L,et al. Effort-aware just-in-time defect prediction:simple unsupervised models could be better than supervised models[C]// Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016:157-168.

[36] ROMANO J,KROMREY J,CORAGGIO J,et al. Exploring methods for evaluating group differences on the NSSE and other surveys:Are the t-test and cohens d indices the most appropriate choices[C]// Annual Meeting of the Southern Association for Institutional Research. 2006:1-51.



**ZHAO Chenyang**, born in 1999, postgraduate. His main research interests include just-in-time software testing and so on.

**JIANG He**, born in 1980, professor, Ph.D supervisor, is a member of CCF (No. 08846D). His main research interests include system software and intelligent software engineering.

(责任编辑:柯颖)