Software defect prediction based on nested-stacking and heterogeneous feature selection

Li-qiong Chen¹ · Can Wang¹ · Shi-long Song¹

Received: 25 May 2021 / Accepted: 22 January 2022 / Published online: 20 February 2022
© The Author(s) 2022

Abstract

Software testing guarantees the delivery of high-quality software products, and software defect prediction (SDP) has become an important part of software testing. Software defect prediction is divided into traditional software defect prediction and just-in-time software defect prediction (JIT-SDP). However, most of the existing software defect prediction frameworks are relatively simplified, which makes it extremely difficult to provide developers with more detailed reference information. To improve the effectiveness of software defect prediction and realize effective software testing resource allocation, this paper proposes a software defect prediction framework based on Nested-Stacking and heterogeneous feature selection. The framework includes three stages: data set preprocessing and feature selection, Nested-Stacking classifier, and model classification performance evaluation. The novel heterogeneous feature selection and nested custom classifiers in the framework can effectively improve the accuracy of software defect prediction. This paper conducts experiments on two software defect data sets (Kamei, PROMISE), and demonstrates the classification performance of the model through two comprehensive evaluation indicators, AUC, and F1-score. The experiment carried out large-scale within-project defect prediction (WPDP) and cross-project defect prediction (CPDP). The results show that the framework proposed in this paper has an excellent classification performance on the two types of software defect data sets, and has been greatly improved compared with the baseline models.

Keywords Software defect prediction · Ensemble learning · Stacking · Feature select · MLP

Introduction

Nowadays, we are living in the “Internet +” era, and software applications are needed for daily life or business activities. People are becoming more and more dependent on software systems in their daily lives, and their requirements for software quality are getting higher and higher. With the continuous increase of software scale, the software structure is becoming more and more complex, and the existence of software defects has become an inevitable fact [1]. Therefore, an effective software defect prediction model is urgently needed. Traditional software defect prediction is based on the analysis of files, software packages, or codebases, but in reality, software developers may cause defects every time they submit code. On the other hand, because a large amount

of code may only bring a very small number of defects, this causes code reviewers spend a lot of time. Therefore, the researchers proposed an instant defect prediction technology.

However, whether it is traditional software defect prediction or instant software defect prediction, the accuracy of model prediction is affected by the imbalance of data set categories. 20% of the defects in the software engineering field may exist in 80% of the modules [2], as shown in Fig. 1. In most cases, code changes that do not cause defects to account for a larger proportion, so there is an imbalance rate in the data set, that is, the imbalance between the minority and majority categories, which will affect the classification prediction effect of the model. Most types, that is, code changes that will not produce defects will make the model have an artificially high prediction accuracy, and it is difficult to obtain the expected results in practical applications. In addition, the data set features contain many irrelevant or redundant features, which will also affect the accuracy of the model prediction.

To reflect on the above research, a software defect prediction framework based on Nested-Stacking is proposed.

✉ Li-qiong Chen
lqchen@sit.edu.cn

Can Wang
452748706@qq.com

¹ Shanghai Institute of Technology, Shanghai, China

Fig. 1 Code repository

First, Nested-Stacking uses techniques such as heterogeneous feature selection and normalization to improve data characteristics during the data preprocessing stage, so that the model can achieve better results from the classification process. In the classification stage, Nested-Stacking technology is used to achieve various good and different ensemble learning stacking [3], and the second layer is nested with custom MLP [4] such excellent traditional neural network classifiers, allowing the model to learn from machine learning; different advantages from depth studies. In addition, two comprehensive indicators, AUC and F1-score, are used to evaluate the effect of model classification.

Finally, this paper conducts large-scale experiments on two defect data sets with different granularity levels and compares the classification performance of Nested-Stacking with other models. We get from the experimental data that the Nested-Stacking proposed in this paper is better than other models. To ensure the repeatability of the experiment and promote model change, this article also discloses the code and data of the experiment. The main contributions can be summarized as follows:

- (1) As far as we know, we are the first to propose software defect prediction based on Nested-Stacking. Nested-Stacking can learn software defect data characteristics more deeply using traditional machine learning and deep learning. Nested-Stacking first selects some of the features most suitable for the baseline model through heterogeneous feature engineering and then normalizes all the feature values. The processed data are input into the Nested-Stacking classifier for the first stage of prediction. Finally, the meta-classifier LogisticRegression is used to predict defects and output evaluation indicators.
- (2) In a large-scale empirical study, we compared the classification performance of Nested-Stacking and other models in two evaluation scenarios (WPDP/CPDP) between two data sets (Kamei/PROMISE).
- (3) The empirical results show that in the two evaluation scenarios, the classification of Nested-Stacking is better than

other models. Especially in WPDP-Kamei, the classification performance of Nested-Stacking is very competitive.

The organizational structure of this paper is as follows: The second section introduces the background and related work of software defect prediction. The third section introduces Nested-Stacking and heterogeneous feature selection in detail. In the fourth section, the experimental setup is introduced. Section 5 presents the experimental results and analysis. Section 6 introduces threats to effectiveness. Finally, the seventh section summarizes the work of this paper and looks forward to future work.

Related work

This section introduces the related work of real-time software defect prediction, as well as the related work of feature selection and integration models in the field of traditional software defects.

Just-in-time software defect prediction

In recent years, JIT-SDP has become a research hotspot in the field of defect prediction because of its fine-grained and instant traceability. In the software defect prediction problem, Khuat et al. [5] empirically evaluated the importance of sampling various classifier sets of imbalanced data by combining sampling technology and ensemble learning model and predicted positive effects for data with category imbalance problem. Zhu et al. [6] proposed a just-in-time defect prediction model DAECNN-JDP based on a denoising autoencoder and convolutional neural network. DAECNN-JDP sets different weights for position vectors of each dimension feature and trains them automatically by adaptive trainable vectors. By training the denoising autoencoder, the input features free from noise pollution can be obtained, learn more robust feature representation. Pascarella et al. [7] proposed a novel fine-grained model to predict the defective documents contained in the submission, and reduce the workload required to judge defects according to the classification performance and the degree to which the model. Yan et al. [8] proposed a two-stage framework, namely defect identification and defect location. Given a new change, first, identify whether it is a buggy change or a clean change. If a new change is identified as defective, the JIT defect location will sort the source code lines introduced by the new change according to its suspicious degree score. In the positioning stage, they use the software natural positioning method of the N-gram model. Bejjanki et al. proposed class imbalance reduction (CIR) [9], which is an algorithm that establishes symmetry between defective records and non-defective records in an imbalanced data set by considering the distribution characteristics of the

data set. Yang et al. proposed a supervision method DEJIT [10] based on Differential Evolution (DE) to build a JIT-SDP model. Specifically, they first proposed a metric called Density Percent Average (DPA), which was used as the optimization target for the training set. Then, logistic regression (LR) is used to build a predictive model. To make LR get the maximum DPA on the training set, they use the DE algorithm to determine the coefficient of LR.

In JIT-SDP-related research, many processing methods tend to change the data distribution. In real data, the distribution of training and test data cannot be confirmed, and changes in the data distribution may lead to a decrease in model accuracy. This article does not perform any up-sampling or down-sampling of the data, nor does it modify the data distribution, which ensures that the model learns the most primitive data distribution.

Software defect prediction based on ensemble model

In the field of software defect prediction, there have also been many related pieces of research on integrated models. Alsawalqah [11] proposed a hybrid classification method for software defect prediction. The main idea of this method is to develop experts and robust classification models based on similar pattern groups. Malhotra [12] conducted an empirical comparison of software defect prediction models developed using various integration methods based on boosting in three open-source JAVA projects, all of which included resampling technology. The results show that the use of resampling technology before ensemble classification can significantly improve the prediction accuracy of the model. Matloob et al. [13] proposed an integrated classification framework based on multiple feature selection algorithms and compared the results with ten machine learning classifiers. Li et al. [14] proposed a new two-stage integrated learning (TSEL) method, which includes an integrated multi-core domain adaptation (EMDA) stage and an integrated data sampling (EDS) stage. Iqbal et al. [15] presented a classification framework which uses Multi-Filter feature selection technique and Multi-Layer Perceptron (MLP) to predict defect prone software modules. Öztürk [16] studied the effect of hyperparameter optimization on the integrated learning algorithm in terms of defect prediction performance. The purpose of Kakkar et al.'s study [17] was to determine the most appropriate attribution technique for handling missing values in SDP datasets.

At present, the integration models in the field of software defects are mostly single soft integration or traditional stacking, such as bagging, voting, and stacking. From the perspective of the prediction effect, it may not be as effective as the tree algorithm, and it takes a long time. In addition, the common K-Fold segmentation data used in most recent studies have not yet considered the impact of class imbalance in

software defect data. The Nested-Stacking proposed in this paper uses StratifiedKfold and integrates the advantages of various boost algorithms to improve the accuracy of software defect prediction.

Software defect prediction based on feature selection

Ni et al. proposed a multi-objective feature selection method MOFES (Multi-Objective Feature Selection) [18], which simultaneously optimizes two objectives. An optimization goal is to minimize the number of selected features. This goal is related to the cost analysis of this problem. Another goal is to maximize the performance of the constructed model. This goal is related to the benefit analysis of the problem. Balogun A O uses 4 different classifiers to evaluate 5 software defect data sets in the NASA software defect database with 4 filter feature ranking (FFR) and 14 filter feature subset selection (FSS) methods [19]. The feature selection based on the wrapper in the traditional software defect prediction model has good prediction performance, but it has a high computational cost and lack of generalization. Therefore, Oluwagbemiga B A et al. proposed a hybrid multi-filter packaging method [20], which uses the relationship between filter–filter and filter–wrapper to give the optimal feature subset and reduce the time-consuming model. Bashir K et al proposed a new method based on Maximum Likelihood Logistic Regression (MLLR) [21]. In the data set used in the study, analysis of variance (ANOVA) F test results verifies that the proposed method is superior to all FS techniques, both in sampled and unsampled data.

The amount of software defect data is large and the feature dimension is high. A large number of redundant features will affect the model classification effect, so feature selection is necessary. At present, the methods of feature selection can be divided into three types: filtering, embedded, and encapsulated. First of all, filtering algorithms (such as Pearson correlation coefficient [22] and Spearman rank correlation coefficient [23]) select the optimal feature subset independent of the algorithm and have good flexibility. However, the accuracy of the selected features is limited. The encapsulated algorithm selects the optimal feature subset based on the performance of the classification algorithm based on multiple rounds of iterations. However, due to the large search space leading to high computational complexity, the encapsulated algorithm selects the optimal feature subset based on the performance of the classification algorithm based on multiple rounds of iteration. The huge search space leads to high computational complexity, and the method has limited versatility, that is, when the algorithm changes, a new round of complex training will start again. Software defect data exist in most of the features with small variance to determine

the label, so the above-mentioned common methods cannot better solve the data problem.

This study proposes Nested-Stacking to predict the software defect data set, and choose different features according to different baseline models at the data level. The experimental results found that nesting and stacking various boosting algorithms and MLP can also reduce the impact of class imbalance in the data set on the classification effect. The innovations of this article are as follows:

(1) Customized baseline model can be nested to adapt to various distributions of software defect data. (2) The data segmentation algorithm in Nested-Stacking uses Stratified-KFold instead of traditional KFold to adapt to the imbalanced distribution of data. (3) Each baseline model in Nested-Stacking can randomly select or specify the feature vector for training and prediction. For example, specify Random-Forest to use only the first, fourth, seventh, and eighth column features for training and prediction, or AdaBoost randomly selects 10% of the features for training and prediction.

Nested-Stacking framework

In this section, we will elaborate on the process and steps of the model framework proposed in this article, as shown in Fig. 2.

(1) Data collection and preprocessing: First, collect two open-source software defect data sets with different granularity levels. Second, different baseline models perform heterogeneous feature selection based on the calculation results of their respective feature importance, to filter out features with low importance, and to achieve the goal of reducing data dimensions, improving model accuracy, and computing speed. Finally, the data are normalized to prevent the difference in feature levels from affecting the internal processing of the model. Heterogeneous feature selection will be detailed in this section. (2) Model construction: First, determine the combination of baseline model and meta-classifier through repeated experiments. After each baseline model is stacked, the prediction result is input to the Meta Classifier, and the final classification prediction is performed. (3) Model evaluation: using the within-project Stratified K-Fold ($K = 10$) [24] cross-validation and cross-project verification experimental methods, two comprehensive evaluation indicators: Area Under Curve (AUC), and F1-score on the generalization performance of the model authenticating.

Nested-Stacking classifier

The core of Nested-Stacking Classifier is to build multiple different baseline modes and stack them to obtain better classification performance. The flowchart is shown in Fig. 3. From the perspective of the baseline model, model stacking

can be either homogeneous or heterogeneous. If the model is generated using the same induction algorithm, then the ensemble is called isomorphism; otherwise, it is heterogeneous. Both are now used for software defect prediction. In addition, from the perspective of the generation strategy of the baseline model, the integrated learning methods can be divided into two categories: the parallel method mainly represented by bagging, and the sequential method mainly represented by boosting. In addition to the bagging and boosting methods mentioned above, stacking is another very effective integrated learning method. The core idea is to reduce the generalization error by adding a variety of good and different baseline models and using a meta-fclassifier to combine the results of baseline model predictions, thereby improving the accuracy of software defect prediction. In this article, we stacked the representative algorithms of these two methods and nested a layer of basic Stacking. Figure 3 compares the similarities and differences between the integrated model framework and the Nested-Stacking Classifier framework in the previous study [25]. The heterogeneous integration of different types of basic learners can improve the generalization ability of the model; in addition, combining the prediction results with meta-learners can improve the fitting ability of the model. Nested-Stacking makes the model further improve the classification effect through nested rules.

As shown in Fig. 3, the input data of the Nested-Stacking classifier are selected and normalized by heterogeneous features. The Nested-Stacking classifier contains a total of three layers. The first layer is integrated by three boosting algorithms: LightGBM, CatBoost, and AdaBoost, and a simple stacking model including MLP and RandomForest is nested. The meta-classifier is Gradient Boosting Decision Tree. The last layer is the meta-classifier of LogisticRegression, which performs the final classification prediction. In the early stage of the experiment, to obtain a baseline model with better generalization ability and accuracy, we chose CART, SVM, Naïve Bayes, k-Nearest Neighbor, BaggingClassifier, DecisionTree, and other different types and numbers of baseline models to conduct a wide range of combined experiments. The results show that when LightGBM, CatBoost, and AdaBoost are used as baseline models, the model has the best predictive performance. Compared with other ensemble learning algorithms, Nested-Stacking uses the predicted results of the primary model as the input of the second model when stacking. Therefore, the model deviation is effectively reduced, the accuracy is improved, and the prediction result of the software defect data set is optimized.

Heterogeneous feature selection

In the Nested-Stacking framework, each baseline model can perform different feature selection, which we call heterogeneous feature selection. Using the baseline models to select

Fig. 2 Framework diagram of the model proposed in our study

the best-performing features in each model (the features are not necessarily the same), to stack their prediction results, can improve the overall classification effect of Nested-Stacking. In the heterogeneous feature selection, you can also set the baseline model to perform feature selection by percentage, thereby reducing the overall running time of the model. Next, we will introduce the algorithm flow of feature selection for part of the baseline model in the Nested-Stacking framework.

AdaBoost [26] is an embedded algorithm in the field of feature selection. It uses integrated learning weak classifiers to select the best-performing features to obtain the optimal target feature subset. Its weak classifiers are usually composed of some decision stubs, BP neural networks and SVM, and other compositions. Some code changes usually have only minor feature differences, which are usually easily ignored by soft thresholds such as SVM and BP neural networks. Therefore, to improve the model's sensitivity to these small differences, a threshold-based decision stub is used as the weak classifier of AdaBoost. Specifically, the AdaBoost

feature selection process for the software defect data set is as follows:

Input: Software defect data set

$$X = \left\{ \left(x_1^1, x_1^2, \dots, x_1^m \right), \right. \\ \left(x_2^1, x_2^2, \dots, x_2^m \right), \dots, \\ \left. \left(x_n^1, x_n^2, \dots, x_n^m \right) \right\}, \quad (1)$$

each defect data label

$$Y = \{y_1, y_2, \dots, y_n\}, \quad (2)$$

where m is the number of features and n is the number of samples.

Output: The optimal target feature subset F .

Step 1. For a software defect data feature j , train a corresponding weak classifier h_j to evaluate its importance.

Fig. 3 Nested-Stacking classifier

Step 2. Set the correspondence hypothesis between features and labels

$$h_t = \{x_i^j \rightarrow Y\}. \quad (3)$$

Step 3. The error corresponding to $D(X_i^j)$ is expressed as

$$\varepsilon_t = \sum_{i:h_t(x_i^j) \neq y_i} D_t(x_i^j). \quad (4)$$

Step 4. The features with the smallest error ε_t are sequentially deleted from the initial recognition feature set X , and added to the optimal target feature subset F at the same time: $X = X - \{f\}$, $F = F + \{f\}$.

Step 5. Use the best-performing classifier h_t error to update the weight of each weak classifier

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right). \quad (5)$$

Step 6. Update $D(X_i^j)$ to

$$D_{t+1}(x_i^j) = \frac{D_t(x_i^j)}{N_t} \times \begin{cases} e^{-\beta_t}, & \text{if } h_t(x_i^j) = y_i \\ e^{\beta_t}, & \text{otherwise.} \end{cases} \quad (6)$$

Among them, N_t is a regularization constant term, such that

$$\sum_{i=1}^m D_t(x_i^j) = 1. \quad (7)$$

Step 7. After completing the iteration, the final strong classifier is obtained

$$H(x_i^j) = \text{sign} \left(\sum_{t=1}^T \beta_t \times h_t(x_i^j) \right). \quad (8)$$

By setting the weak classifier to traverse all the features, the optimal target feature subset is formed according to the

optimal feature selected in each iteration

$$F = \left\{ \left(x_1^1, x_1^2, \dots, x_1^k \right), \right. \\ \left(x_2^1, x_2^2, \dots, x_2^k \right), \dots, \\ \left. \left(x_n^1, x_n^2, \dots, x_n^k \right) \right\}, \quad (9)$$

where k represents the number of features of the optimal target feature obtained.

RandomForest [27] is a combination classifier based on the decision tree, which can be used for feature selection. RandomForest uses the Bagging method to randomly and repeatably draw samples from the original sample set for classifier training. About 37% of the sample data will not be selected. These data are called Out of Bag (OOB). When calculating the importance of a certain column of defect features, use OOB data as the base learner after the test set test training, and the test error rate is recorded as errOOB ; add noise to the important features to be calculated in the OOB sample, and calculate errOOB again; calculate all base learning. The average error of the test of the device is calculated using the average accuracy decrease rate (MDA) as an indicator for feature importance calculation

$$MDA = \frac{1}{n} \sum_{t=1}^n (\text{errOOB}_t - \text{errOOB}'_t). \quad (10)$$

In Eq. (11), n is the number of base learners; errOOB' is the error outside the bag after adding noise. The more the MDA index declines, the greater the impact of the corresponding feature on the prediction result, and the higher its importance. This feature importance calculation method is called RandomForest's out-of-bag estimation. According to this method, the importance of software defect data-related features is ranked and feature selection is performed.

Data preprocessing

In terms of data preprocessing, this paper uses StandardScaler to normalize the experimental data, so that Nested-Stacking can process and fit the data faster.

Experimental setup

This section introduces the experimental setup from five aspects: data sets, performance evaluation indicators, and data analysis method. Our experiment is performed on the hardware with Intel (R) Core (TM) i5-10600KF CPU 4.10 GHz, 16 GB of RAM, and an RTX3090GPU. The operating system is Windows 10 Pro. The programming environment for scripts is Python 3.8.

With the analysis of the experimental results, this article answers the following two questions:

- (1) In the within-project cross-validation scenario, how does the classification performance of the Nested-Stacking model compared to other models?
- (2) In the cross-project verification scenario, how does the classification performance of the Nested-Stacking model compared to other models?

Datasets

To verify the effectiveness of our proposed framework, this study uses Kamei [28] and PROMISE [29] as experimental data sets. The data set information provided by Kamei is shown in Table 1, and its features are shown in Table 2.

Table 3 shows the specific information of the PROMISE data set, including the project name, project version, number of code files, and defect rate. In addition, the 20 static metric features selected in this experiment are all extracted by Jureczko et al. [30] for object-oriented programming language design, including the depth of inheritance tree (Depth of Inheritance Tree), the number of subclasses (Number of Children), the number of lines of code (Line of Code), and related code complexity features, etc., as shown in Table 4.

Performance indicators

This section will introduce the evaluation indicators selected in this experiment, such as confusion matrix, precision, recall, F1-score, and area under the ROC curve (AUC). The software defect prediction problem solved in this paper is a two-class problem, so comprehensive indicators such as AUC and F1-score are used to evaluate the prediction model. The confusion matrix is shown in Table 5. TP (True Positive) represents the number of defects that are correctly predicted; TN (True Negative) represents the number of non-defects that are correctly predicted; FP (False Positive) represents that it is incorrectly classified as defective, but The actual number of defects; FN (False Negative) represents the number of defects that are incorrectly classified as non-defects.

Precision is called precision rate; Recall is called recall rate; F1-score is usually used as a model evaluation indicator in the field of software defect prediction, which is the harmonic average of precision and recall. Their calculation formula is as follows:

$$Precision = \frac{TP}{TP + FP} \quad (11)$$

$$Recall = \frac{TP}{TP + FN} \quad (12)$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}. \quad (13)$$

Table 1 Datasets information (Kamei)

Projects	Period	Number of defects	Number of changes	Defect rate
Bugzilla	1998.08.23~2006.12.16	1696	4620	36.71%
Columba	2002.11.25~2006.07.27	1361	4455	30.55%
Eclipse JDT	2001.05.02~2007.12.31	5089	35,386	14.38%
Mozilla	2000.01.01~2006.12.17	5149	98,275	5.24%
Eclipse Platform	2001.05.02~2007.12.31	9452	64,250	14.71%
PostgreSQL	1996.07.09~2010.05.15	5119	20431	25.06%

Table 2 Features of datasets (Kamei)

Class	Name	Definition
Diffusion	NS	Number of modified subsystems
	ND	Number of modified directories
	NF	Number of modified files
	Entropy	Distribution of modified code across each file
Size	LA	Lines of code added
	LD	Lines of code deleted
	LT	Lines of code in a file before the change
Purpose	FIX	Whether or not the change is a defect fix
History	NDEV	Number of developers that changed the files
	PD	Average time interval between the last and the current change
	NPT	Number of unique last changes to the files
Experience	EXP	Developer experience
	REXP	Recent developer experience
	SEXP	Developer experience on a subsystem

Table 3 Datasets information (PROMISE)

Projects	Version	Number of defects	Number of changes	Defect rate
Ant	1.3/1.4/1.5/1.6/1.7	350	1692	20.69%
Camel	1.0/1.2/1.4/1.6	562	2784	20.19%
Jedit	3.2/4.0/4.1/4.2/4.3	303	1749	17.32%
Log4j	1.0/1.1/1.2	260	449	57.91%
Lucene	2.0/2.2/2.4	438	782	56.01%
Poi	1.5/2.0/2.5/3.0	707	1378	51.31%
Synapse	1.0/1.1/1.2	162	635	25.51%
Velocity	1.4/1.5/1.6	367	639	57.43%
Xalan	2.4/2.5/2.6/2.7	1806	3320	54.40%
Xerces	1.2/1.3/1.4	577	1481	38.96%
Ivy	1.4/2.0	56	798	7.02%

Area Under Curve (AUC): AUC is defined as the area under the ROC curve and the coordinate axis. The corresponding points in the ROC space are calculated according to the model classification results, and the ROC curve is formed by connecting these points. The abscissa is False-Positive Rate (FPR), and the ordinate is True-Positive Rate (TPR).

Data analysis method

The experiment considered two actual business scenarios to evaluate the predictive performance of the framework proposed in this article, which is the within-project StratifiedKFold ($K = 10$) cross-validation and cross-project validation. These experimental methods are described in detail below.

Table 4 Features of datasets (PROMISE)

Name	Definition
wmc	No. of methods in a class
dit	Depth of inheritance levels from top
noc	No. of immediate subclasses of the class
cbo	Coupling between the objects of different classes
rfe	No. of methods executed when an object receives a message
lcom	No. of methods in a class that are not related through the sharing of some of the class fields
lcom3	No. of methods and attributes, and access to those methods on another class
npm	No. of public methods in a class
loc	Lines of code
dam	Ratio of the no. of private and protected attributes to the total number of attributes
moa	Extent of the part-whole relationship, realized by using attributes
mfa	No. of methods inherited by a class per number of methods accessible by its methods
cam	Relatedness among methods of a class based upon the parameter list of the methods
ic	No. of parent classes to which a given class is coupled
cbm	No. of new and redefined methods to which all the inherited methods are coupled
amc	Average method size for each class
ca	No. of other classes which depend on the class
ce	No. of other classes on which the class is depended
max_cc	Max value for Cyclomatic Complexity metric
avg_cc	Average value for Cyclomatic Complexity metric

Within-project stratified K-Fold ($K = 10$) cross-validation

First, you need to randomly shuffle the data set, and then divide the data set into ten parts, that is, after tenfold, each onefold is used as the test set, and the other tenfold are used as the training set. The prediction model is trained and verified to calculate the performance indicators. A total of 10 runs. In the second experiment, the performance index obtained each time was averaged as the evaluation result of tenfold cross-validation. The usage of StratifiedKFold is similar to KFold, but because of its stratified sampling principle, it ensures that the proportion of samples in each category in the training set and test set is the same as the original data set.

Cross-project validation

In recent years, cross-project defect prediction has received widespread attention. The basic idea of cross-project defect prediction is to train a defect prediction model with the defect data set of one project, and then use it to predict the defects of another project.

Statistical test

The above experiments all use the Wilcoxon signed-rank test [31] and Cliff's δ test [32] to show the significance of the performance difference between Nested-Stacking and other

Table 5 Confusion matrix

	Actual defect	Actual non-defect
Predict defects	TP	FP
Predict non-defects	FN	TN

Table 6 Cliff's δ values and their effective levels

Cliff's δ	Effective levels
$ \delta < 0.147$	Negligible
$0.147 \leq \delta < 0.33$	Small
$0.33 \leq \delta < 0.474$	Medium
$0.474 \leq \delta $	Large

models. Wilcoxon signed-rank test is a commonly used non-parametric statistical hypothesis test. This article uses the ρ -value to test whether the model's performance difference is statistically significant at the 0.05 significance level. The experiment also uses Cliff's δ to determine the magnitude of the difference in prediction performance between models. Usually, the size of the difference can be divided into four levels, as shown in Table 6. In summary, when the ρ -value is less than 0.05 and Cliff's δ is greater than or equal to 0.147, there is a significant difference in the prediction performance between the models.

Experimental results and analysis

This section will use the with-in project StratifiedKFold ($K = 10$) cross-validation and cross-project validation on two data sets to compare the classification performance differences between Nested-Stacking and other models.

Analysis of the RQ1

The experiment compares Nested-Stacking with other models and uses the with-in project StratifiedKFold ($K = 10$) cross-validation to classify and predict the two types of open-source project data sets. Since each model produces ten prediction results in the with-in project StratifiedKFold ($K = 10$) cross-validation scenario, the AUC and F1-score in the table are the averages of the experimental results. The better AUC and F1-score will be marked in bold.

WPDP-Kamei

In the experiment of Kamei data set, this study follows LocalJIT [33] and Kamei's experimental methods and performance indicators (AUC, F1-score), and compares them with their experimental results. The experimental results are shown in Fig. 4 and Table 7.

In Table 7, rows 2–7, respectively, give the AUC results of Nested-Stacking and the other three models on 6 data sets. Line 8 is the average AUC predicted value of each model. The 9th line (W/D/L) shows the number of wins for each prediction model on the 6 data sets. The highest AUC value predicted by the other three models is Stacking-0.8198, and the lowest is LocalJIT-0.704. The Nested-Stacking average AUC prediction value is 0.8322, which is an average increase of 8.25% compared to the other three models. It is worth noting that the AUC prediction result of Nested-Stacking in the Mozilla data set is 0.8517, which is the highest AUC index obtained by predicting a single item in all models.

The experiments in this section analyze and compare the classification performance of Nested-Stacking and other JIT-SDP models. The experimental results based on the within-project StratifiedKFold ($K = 10$) cross-validation show that, compared with other JIT-SDP models, Nested-Stacking has an average increase of 8.25% in AUC and an average increase of 22.08% in F1-score, which proves the rationality of Nested-Stacking as the core classifier in this paper. It can be concluded from Fig. 4 that the classification performance of Nested-Stacking on JIT-SDP is better than other JIT-SDP models.

WPDP-PROMISE

In the WPDP experiment of the PROMISE data set, this study follows the experimental method and performance indica-

tor (F1-score) of ImprovedCNN [34] and COSTE [35]. The experimental results are shown in Fig. 5 and Table 8.

In Table 8, the first column is the PROMISE data set, and the other columns are the prediction model and its prediction results. Among them, the second column is the Nested-Stacking proposed in this article; the third column is the Stacking model with the nested part removed, and only three boosting algorithms are retained; the fourth column is the improved CNN model predicted by PROMISE; the last column is an excellent class imbalanced data processing model-COSTE.

The 12th line is the average predicted value of AUC for each model, and the 13th line (W/D/L) shows the number of wins for each model on PROMISE. The prediction results of Nested-Stacking in ant, camel, jedit, synapse, and xalan are better than the other three prediction models; traditional Stacking has better prediction results in log4j, poi, and xerces; ImprovedCNN has the best prediction results in lucene and velocity. Although the average effect of COSTE is relatively good, it does not have a very eye-catching performance in a certain data set. In addition to Nested-Stacking, the highest average F1-score predicted value is traditional Stacking-0.7912, and the lowest is ImprovedCNN-0.6003. The average F1-score of Nested-Stacking is 0.7988, which is an average increase of 14.83% compared to other models.

It can be seen from Table 8 and Fig. 5 that this part of the experiment compares the classification performance of Nested-Stacking and other prediction models on PROMISE. The experimental results based on the within-project StratifiedKFold ($K = 10$) cross-validation show that the F1-score of Nested-Stacking has increased by 14.83% on average, which proves the rationality of Nested-Stacking as a traditional software defect prediction model.

Based on the above two WPDP experiments, it can be concluded that Nested-Stacking is superior to other prediction models in classification performance on software defect data sets of different granularities.

Analysis of the RQ2

This section discusses the classification performance difference between Nested-Stacking and other models in cross-project validation scenarios. The AUC or F1-score in the graph and table is the average of the experimental results.

CPDP-Kamei

In this CPDP experiment, we use the six data sets provided by Kamei, so each software defect prediction model produces 30 prediction results, and the results are averaged. The experimental results are shown in Fig. 6.

The AUC prediction comparison of CPDP-Kamei is shown in Fig. 6. The average of the 30 prediction results

Table 7 Within-project StratifiedKFold ($K = 10$) cross-validation (Kamei)

Indicators	Projects	Nested-stacking	Stacking	LocalJIT	Kamei
AUC	Bugzilla	0.8291	0.8275	0.7046	0.7464
	Columba	0.8283	0.8053	0.7197	0.7578
	JDT	0.8144	0.8018	0.7191	0.7456
	Mozilla	0.8517	0.8482	0.7412	0.7721
	Platform	0.8276	0.8258	0.7337	0.7589
	Postgres	0.8421	0.8099	0.7644	0.7946
	AVG	0.8322	0.8198	0.7305	0.7626
	W/D/L	6/0/0	0/0/6	0/0/6	0/0/6
F1-score	Bugzilla	0.7481	0.7554	0.6445	0.6031
	Columba	0.7754	0.7690	0.6725	0.5869
	JDT	0.8268	0.8245	0.7643	0.3747
	Mozilla	0.9304	0.9344	0.8531	0.2289
	Platform	0.8410	0.8372	0.7717	0.3856
	Postgres	0.7986	0.7976	0.7599	0.5613
	AVG	0.8200	0.8197	0.7443	0.4568
	W/D/L	4/0/2	2/0/4	0/0/6	0/0/6

Fig. 4 WPDP-Kamei **Table 8** Within-project StratifiedKFold ($K = 10$) cross-validation (PROMISE)

Projects	Nested-Stacking	Stacking	ImprovedCNN	COSTE
Ant	0.7890	0.7662	0.4200	0.7160
Camel	0.8072	0.7943	0.5175	0.6500
jedit	0.8393	0.8362	0.4320	0.7603
Log4j	0.8245	0.8308	0.7133	0.8063
Lucene	0.6314	0.6111	0.7133	0.7068
Poi	0.8038	0.8068	0.6000	0.6450
Synapse	0.8121	0.7987	0.5500	0.7487
Velocity	0.8116	0.7989	0.8367	0.6987
Xalan	0.8414	0.8316	0.6775	0.7870
Xerces	0.8274	0.8376	0.5400	0.7291
AVG	0.7988	0.7912	0.6003	0.7248
W/D/L	6/0/5	3/0/8	2/0/9	0/0/11

Fig. 5 WPDP-PROMISE

Fig. 6 Cross-project validation (Kamei)

produced by Nested-Stacking is 0.6713, and the average AUC of the remaining three models is 0.6648, 0.6065, and 0.6958, respectively. In comparison, Nested-Stacking is 0.9% higher than traditional Stacking's cross-project verification; compared with LocalJIT, it has a larger increase of 10.6%; however, compared with the originator of JIT-SDP, Kamei, it is reduced by 3.6%. It may be due to overfitting of Nested-Stacking. To sum up, compared with other models, Nested-Stacking has an average increase of 2.4% in cross-project verification scenarios, which proves that the model has a certain degree of advancement.

The comparison of CPDP-Kamei's F1-score prediction results is shown in Fig. 6. It can be seen from the figure that the average of the 30 prediction results produced by Nested-Stacking is 0.7073, and the average AUC of the remaining three models are 0.6940, 0.5758, and 0.3814, respectively. In comparison, Nested-Stacking in the cross-project validation

scenario is 1.33% higher than the F1-score predicted value of traditional Stacking; compared with LocalJIT, it has a larger improvement of 22.84%; compared with the Kamei, it is increased by 85.45%, and has a very amazing lifting effect. To sum up, compared with other models, Nested-Stacking has an average increase of 28.51% in cross-project validation scenarios, which proves that the model proposed in this paper still has high practical value in cross-project validation scenarios.

CPDP-PROMISE

In the CPDP experiment of the PROMISE data set, this study follows the experimental methods and performance indicator (F1-score) of DBN [36], TCA+ [37], and DTL-DP [38], and compares with their experimental results. The results are shown in Fig. 7 and Table 9.

In the CPDP experiment in this section, Nested-Stacking is compared with the other three models. Under the guidance of previous research, we conducted 22 experiments. In each experiment, we randomly select two project versions from two different projects, one as the training set and the other as the test set. Nested-Stacking's average F1-score prediction value is 0.6712, and DBN, TCA+, and DTL-DP reach 0.5681, 0.4799, and 0.6179, respectively. Compared with the other three models, Nested-Stacking has increased 18.15%, 39.65%, and 8.46%, respectively. It can be seen from Fig. 7 and Table 9 that the model proposed in this paper works well in most experiments, followed by DTL-DP. The four models in the experiment won 12 times, 2 times, 0 times, and 8 times, respectively.

Combining the CPDP experiment of the two data sets, it can be concluded that in the CPDP scenarios of the two different granularity data sets, the classification performance of Nested-Stacking is improved compared with the other models. Therefore, the model proposed in this article will show its superiority in actual test scenarios.

Discussion

From the experimental part of this study, we demonstrated the superiority of Nested-Stacking using two large data sets. But why does Nested-Stacking perform well? We will scientifically discuss the main reasons why the proposed framework is superior to other baseline models in three directions.

First, in JIT-SDP-related studies, many processing methods tend to change the data distribution. The distribution of training and test data in real data cannot be confirmed, and the change of data distribution may lead to a decrease in model accuracy. Nested-Stacking does not perform any up-sampling or down-sampling of the original data, nor does it

Fig. 7 CPDP-PROMISE

Table 9 Cross-project validation (PROMISE)

Source	Target	Nested-Stacking	DBN	TCA+	DTL-DP
Ant-1.6	Camel-1.4	0.7410	0.3162	0.2922	0.3954
Jedit-4.1		0.7345	0.6934	0.3304	0.4072
Camel-1.4	Ant-1.6	0.7550	0.9796	0.6169	0.5913
Poi-3.0		0.6944	0.4788	0.5988	0.6931
Camel-1.4	Jedit-4.1	0.7660	0.6151	0.5377	0.5313
Log4j-1.1		0.8205	0.5033	0.4192	0.6394
Jedit-4.1	Log4j1.1	0.8440	0.6455	0.5744	0.7838
Lucene-2.2		0.6239	0.6187	0.5713	0.7949
Lucene-2.2	Xalan2.5	0.5529	0.5537	0.5305	0.6894
Xerces-1.3		0.5243	0.5727	0.5811	0.6861
Xalan-2.5	Lucene-2.2	0.5256	0.5943	0.5616	0.7832
Log4j-1.1		0.5506	0.6921	0.5245	0.7694
Xalan-2.5	Xerces-1.3	0.6645	0.3864	0.3941	0.4005
Ivy-2.0		0.8565	0.4262	0.3988	0.4207
Xerces-1.3	Ivy-2.0	0.8910	0.4536	0.4099	0.4729
Synapse-1.2		0.6960	0.8247	0.3837	0.4946
Ivy-1.4	Synapse-1.1	0.7297	0.4899	0.3483	0.5457
Poi-2.5		0.4685	0.4254	0.3762	0.5975
Ivy-2.0	Synapse-1.2	0.6914	0.4333	0.5706	0.6204
Poi-3.0		0.6352	0.5141	0.5425	0.6236
Synapse-1.2	Poi-3.0	0.4548	0.6615	0.6514	0.8272
Ant-1.6		0.5249	0.6196	0.3431	0.8270
AVG		0.6702	0.5681	0.4799	0.6179
W/D/L		12/0/10	2/0/20	0/0/22	8/0/14

modify the data distribution, ensuring that the software defect prediction model learns the original data distribution.

Second, most recent studies use ordinary KFold data partitioning and have not considered the effect of class imbalance in software defect data. In this paper, we propose that StratifiedKFold, a data partitioning method, and different boosting algorithms and MLP can significantly reduce the effect of class imbalance on the classification effect.

Third, traditional feature selection methods can be divided into filtering, embedded, and encapsulated types. However, software defect data exist in most feature determination data with small variance. Therefore, the above commonly used methods cannot solve the feature engineering problems better. In Nested-Stacking, different baseline models select heterogeneous features based on their respective feature importance calculation results, to filter out low-importance features, so as reduce data dimension, and improve model accuracy and computing speed.

Finally, integration models in software defect prediction models are mostly single soft integrations or traditional stacks, such as bagging, voting, and stacking strategies. In terms of the prediction effect, it may not be as good as the single tree model and take longer. In the early stage, to obtain a classifier with better generalization ability and accuracy, we

selected baseline models of different types and quantities to conduct extensive combined experiments. Nested-stacking classifier contains three layers, of which the first layer is integrated with LightGBM, CatBoost, and AdaBoost algorithms, and a simple stacking model containing MLP and Random-Forest is nested. The meta-classifier is Gradient Boosting Decision Tree. The end meta-classifier performs the final classification prediction for the LogisticRegression. In contrast to other ensemble learning algorithms, Nested-Stacking takes the predicted results of the first-level model as input to the second-level model when stacking. Therefore, the model deviation is effectively reduced and the accuracy is improved, and the prediction result of the software defect data set is optimized.

Threats to validity

This section analyzes the effectiveness of the proposed software defect prediction model from two main perspectives: external effectiveness and internal effectiveness.

The external validity reflects whether the conclusions drawn from the experimental research are universal: this paper uses two granular software defect data sets (Kamei,

PROMISE), both of which can obtain all static metrics of the project program modules from the open-source data sets, these representative data sets can ensure the correctness of the research conclusions. Another aspect of external validity is the experimental part. Although we strictly follow the process described in the comparison model, they may not be fully restored to their original details. Furthermore, methods based on traditional machine learning or deep learning are random, which also makes the experimental results different from those provided in the original literature.

Internal validity affects the accuracy of experimental results: the code written in this article is mainly based on the Scikit-learn machine learning package of Python to ensure the accuracy of model construction. From the perspective of model accuracy and stability, AUC and F1-score are used to ensure the reliability of the metric. These two comprehensive evaluation indicators have been used in much previous literature to evaluate defect prediction models.

Conclusion validity refers to the rationality of the relationship between treatment and outcome. The experiment was repeated ten times to minimize random deviation. In addition, at the significance level of p -value=0.05, two non-parametric hypothesis testing methods, Wilcoxon signed-rank test, and the Cliff's δ test were used to compare the significant difference and effect size. Further research on other statistical testing methods and significance levels will be explored to analyze our Nested-Stacking model.

Conclusions and future work

The Nested-Stacking framework proposed in this paper is based on the superposition criterion, combined with various Boosting algorithms and custom deep learning algorithms, and maximizes the scientific nature of feature selection. The Nested-Stacking framework conducts a large number of experiments on feature engineering and baseline model selection. The experiments have passed the within-project StratifiedKFold ($K = 10$) cross-validation and cross-project validation, showing the effectiveness of the model on software defect data at two levels of granularity. This proves that Nested-Stacking can provide reliable decision support for software testing. Furthermore, the framework adopts a heterogeneous integration method, and basic learners and meta-learners are algorithms in various fields. Therefore, this method can improve the fitting ability and generalization ability of the model, and is better than the current popular software defect prediction method.

However, the disadvantage of Nested-Stacking is that the optimal combination of the baseline model is derived from complex experiments, and the process is not very efficient. In the future, at the model combination level, our goal is to build a more intelligent and automated prediction system, in

which all model combinations are traversed until the optimal combination is found and its parameters are optimized. Finally, the whole Nested-Stacking process becomes more efficient and intelligent.

Open access

To verify the reproducibility of the experimental data in this article and to promote model changes, all the data and codes in this study are available at <https://github.com/WangHuoShanPY>.

Acknowledgements This work is partially supported by Development Fund for young and middle-aged scientific and technological talents of Shanghai Institute of Technology under Grant No. ZQ2021-19.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Bowes D, Hall T, Petrić J (2018) Software defect prediction: do different classifiers find the same defects? *Softw Qual J* 26(2):525–552
2. Bashir K, Li T, Yohannese CW et al (2020) SMOTEFris-INFFC: handling the challenge of borderline and noisy examples in imbalanced learning for software defect prediction. *J Intell Fuzzy Syst* 38(1):917–933
3. Goyal S (2020) Heterogeneous Stacked Ensemble Classifier for Software Defect Prediction. 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC). IEEE, 6–8 November 2020, pp 126–130
4. Massaoudi M, Refaat SS, Chihi I et al (2021) A novel stacked generalization ensemble-based hybrid LGBM-XGB-MLP model for Short-Term Load Forecasting. *Energy* 214:118874
5. Khuat T T, Le M H (2020) Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems. *SN Computer Science* 1(2):1–16
6. Zhu K, Zhang N, Ying S et al (2020) Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network. *IET Softw* 14(3):185–195
7. Pascarella L, Palomba F, Bacchelli A (2019) Fine-grained just-in-time defect prediction. *J Syst Softw* 150:22–36
8. Yan M, Xia X, Fan Y et al (2020) Just-in-time defect identification and localization: a two-phase framework. *IEEE Trans Softw Eng*
9. Bejjanki KK, Gyani J, Gugulothu N (2020) Class imbalance reduction (CIR): a novel approach to software defect prediction in the presence of class imbalance. *Symmetry* 2(3):407

10. Yang X, Yu H, Fan G et al (2021) DEJIT: a differential evolution algorithm for effort-aware just-in-time software defect prediction. *Int J Softw Eng Knowl Eng* 31(03):289–310
11. Alsawalqah H, Hijazi N, Eshtay M et al (2020) Software defect prediction using heterogeneous ensemble classification based on segmented patterns. *Appl Sci* 10(5):1745
12. Malhotra R, Jain J (2020) Handling imbalanced data using ensemble learning in software defect prediction. In: 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, 29–31 January 2020, pp 300–304
13. Matloob F, Aftab S, Iqbal A (2019) A framework for software defect prediction using feature selection and ensemble learning techniques. *Int J Modern Educ Comput Sci* 11(12)
14. Li Z, Jing X Y, Zhu X, et al (2019) Heterogeneous defect prediction with two-stage ensemble learning. *Autom Softw Eng* 26(3):599–651
15. Iqbal A, Aftab S (2020) A classification framework for software defect prediction using multi-filter feature selection technique and MLP. *Int J Modern Educ Comput Sci* 12(1)
16. Maruf OM (2019) The impact of parameter optimization of ensemble learning on defect prediction. *Comput Sci J Moldova* 79(1):85–128
17. Kakkar M, Jain S, Bansal A, et al (2021) Combining data preprocessing methods with imputation techniques for software defect prediction. *Research Anthology on Recent Trends, Tools, and Implications of Computer Programming*. IGI Global, pp 1792–1811
18. Ni C, Chen X, Wu F et al (2019) An empirical study on pareto based multi-objective feature selection for software defect prediction. *J Syst Softw* 152:215–238
19. Balogun AO, Basri S, Abdulkadir SJ et al (2019) Performance analysis of feature selection methods in software defect prediction: a search method approach. *Appl Sci* 9(13):2764
20. Oluwagbemiga BA, Shuib B, Abdulkadir SJ, et al (2019) A hybrid multi-filter wrapper feature selection method for software defect predictors. *Int J Supply Chain Manag* 8(2):916–922
21. Bashir K, Li T, Yahaya M (2020) A novel feature selection method based on maximum likelihood logistic regression for imbalanced learning in software defect prediction. *Int Arab J Inf Technol* 17(5):721–730
22. Liu Y, Mu Y, Chen K, et al (2020) Daily activity feature selection in smart homes based on pearson correlation coefficient. *Neural Process Lett* pp 1–17
23. Cavallo B (2020) Functional relations and Spearman correlation between consistency indices. *J Oper Res Soc* 71(2):301–311
24. Novaes MT, de Carvalho OLF, Ferreira PHG, et al (2021) Prediction of secondary testosterone deficiency using machine learning: a comparative analysis of ensemble and base classifiers, probability calibration, and sampling strategies in a slightly imbalanced dataset. *Inf Med Unlock* 23:100538
25. Saifan AA, Abu-wardih L (2020) Software defect prediction based on feature subset selection and ensemble classification. *ECTI Trans Comput Inf Technol (ECTI-CIT)* 14(2):213–228
26. Wu Y, Ke Y, Chen Z et al (2020) Application of alternating decision tree with AdaBoost and bagging ensembles for landslide susceptibility mapping. *Catena* 187:104396
27. Li X K, Chen W, Zhang Q et al (2020) Building auto-encoder intrusion detection system based on random forest feature selection. *Comput Secur* 95:101851
28. Kamei Y, Shihab E, Adams B et al (2012) A large-scale empirical study of just-in-time quality assurance. *IEEE Trans Softw Eng* 39(6):757–773
29. Sohan M F, Kabir M A, Rahman M et al (2020) Prevalence of machine learning techniques in software defect prediction. In: International Conference on Cyber Security and Computer Science, Springer, Cham, 15–16 February 2020, pp 257–269
30. Jureczko M, Madeyski L (2010) Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th international conference on predictive models in software engineering September 2010, pp 1–10
31. Fay M P, Proschan MA (2010) Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Stat Surv* 4:1
32. Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J R Stat Soc: Ser B (Methodol)* 57(1):289–300
33. Yang X, Yu H, Fan G et al (2019) Local versus global models for just-in-time software defect prediction. *Sci Program*, pp 1–13
34. Pan C, Lu M, Xu B, et al (2019) An improved cnn model for within-project software defect prediction. *Appl Sci* 9(10):2138
35. Feng S, Keung J, Yu X, et al (2019) COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. *Inf Softw Technol* 129:106432
36. Wang S, Liu T, Nam J et al (2018) Deep semantic feature learning for software defect prediction. *IEEE Trans Softw Eng* 46(12):1267–1293
37. Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: 2013 35th international conference on software engineering (ICSE). IEEE, 18–26 May 2013, pp 382–391
38. Chen J, Hu K, Yu Y et al (2020) Software visualization and deep transfer learning for effective software defect prediction. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering, June 2020, pp 578–589

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.