Hindawi
Scientific Programming
Volume 2019, Article ID 2384706, 13 pages
https://doi.org/10.1155/2019/2384706

Research Article

Local versus Global Models for Just-In-Time Software Defect Prediction

Xingguang Yang,[1,2] **Huiqun Yu**,[1,3] **Guisheng Fan**,[1] **Kai Shi**,[1] **and Liqiong Chen**[4]

[1]*Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China*
[2]*Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai 201112, China*
[3]*Shanghai Engineering Research Center of Smart Energy, Shanghai, China*
[4]*Department of Computer Science and Information Engineering, Shanghai Institute of Technology, Shanghai 201418, China*

Correspondence should be addressed to Huiqun Yu; yhq@ecust.edu.cn and Guisheng Fan; gsfan@ecust.edu.cn

Received 16 January 2019; Revised 14 April 2019; Accepted 23 April 2019; Published 12 June 2019

Academic Editor: Emiliano Tramontana

Just-in-time software defect prediction (JIT-SDP) is an active topic in software defect prediction, which aims to identify defect-inducing changes. Recently, some studies have found that the variability of defect data sets can affect the performance of defect predictors. By using local models, it can help improve the performance of prediction models. However, previous studies have focused on module-level defect prediction. Whether local models are still valid in the context of JIT-SDP is an important issue. To this end, we compare the performance of local and global models through a large-scale empirical study based on six open-source projects with 227417 changes. The experiment considers three evaluation scenarios of cross-validation, cross-project-validation, and timewise-cross-validation. To build local models, the experiment uses the k-medoids to divide the training set into several homogeneous regions. In addition, logistic regression and effort-aware linear regression (EALR) are used to build classification models and effort-aware prediction models, respectively. The empirical results show that local models perform worse than global models in the classification performance. However, local models have significantly better effort-aware prediction performance than global models in the cross-validation and cross-project-validation scenarios. Particularly, when the number of clusters k is set to 2, local models can obtain optimal effort-aware prediction performance. Therefore, local models are promising for effort-aware JIT-SDP.

1. Introduction

Today, software plays an important role in people's daily life. However, the defective software can bring great economic losses to users and enterprises. According to the estimates of the National Institute of Standards and Technology, the economic losses caused by software defects to the United States are as high as $60 billion per year [1]. Therefore, it is necessary to detect and repair defects in the software through a large number of software quality assurance activities.

Software defect prediction technology plays an important role in software quality assurance, and it is also an active research topic in the field of software engineering data mining [2, 3]. In the actual software development, test resources are always limited, so it is necessary to prioritize the allocation of limited test resources to modules that are more likely to be defective [4]. Software defect prediction technology aims to predict the defect proneness, defect number, or defect density of a program module by using defect prediction models based on statistical or machine learning methods [3]. The results of defect prediction models help us to assign limited test resources more reasonably and improve software quality.

Traditional defect predictions are performed at a coarse-grained level (such as package or file level). However, when large files are predicted as defect prone by predictors, it can be time-consuming to perform code checks on them [5]. In addition, since large files have been modified by multiple developers, it is not easy to find the right developer to check

the files [6]. To this end, just-in-time software defect prediction (JIT-SDP) technology is proposed [7–10]. JIT-SDP is made at change level, which has a finer granularity than module level. Once developers submit a change for the program, defect predictors can identify whether the change is defect-inducing immediately. Therefore, JIT-SDP has the advantages of fine granularity, instantaneity, and traceability compared with traditional defect prediction [7].

In order to improve the performance of software defect prediction, researchers have invested a lot of effort. The main research content includes the use of various machine learning methods, feature selection, processing class imbalance, processing noise, etc. [2]. Recently, researchers propose a novel idea called *local models* that may help improve the performance of defect prediction [11–14]. The local models divide all available training data into several regions with similar metrics and train a prediction model for each region. However, the research objects in previous are file-level or class-level defect data sets. Whether local models are still valid in the context of JIT-SDP remains for further study.

To this end, we conduct experiment on the change-level defect data sets from six open-source projects including 227417 changes. We empirically compare the classification performance and effort-aware prediction performance of local and global models through a large-scale empirical study. In order to build local models, the experiment uses the k-medoids clustering algorithm to divide the training samples into homogeneous regions. For each region, the experiment builds classification models and effort-aware prediction models based on logistic regression and effort-aware linear regression (EALR), respectively, which are always used for baseline models in prior studies [7–9].

The main contributions of this paper are as follows:

(i) We compare the classification performance and effort-aware prediction performance of local and global models in three evaluation scenarios, namely, cross-validation scenario, cross-project-validation scenario, and timewise-cross-validation scenario, for JIT-SDP.

(ii) Considering the influence of the number of clusters k on the performance of local models, we evaluate the classification performance and effort-aware prediction performance of local models with different values of k by adjusting k from 2 to 10.

(iii) The empirical results show that local models perform worse in classification performance than global models in three evaluation scenarios. In addition, local models have worse effort-aware prediction performance in the timewise-cross-validation scenario. However, local models have better effort-aware prediction performance than global models in the cross-validation scenario and cross-project-validation scenario. Particularly, when the parameter k is set to 2, local models can obtain optimal performance in the ACC and P_{opt} indicators.

The results of the experiment provide valuable knowledge about the advantages and limitations of local models in the JIT-SDP problem. In order to ensure the repeatability of the experiment and promote future research, we share the code and data of the experiment.

This paper is organized as follows: Section 2 introduces the background and related work of software defect prediction. Section 3 introduces the details of local models. The experimental setup is presented in Section 4. Section 5 demonstrates the experimental results and analysis. Section 6 introduces the threats to validity. Section 7 summarizes the paper and describes future work.

2. Background and Related Work

2.1. Background of Software Defect Prediction. Software defect prediction technology originated in the 1970s [15]. At that time, the software systems were less complex, and researchers found that the number of software defects in a program system was related to the number of lines of code. In 1971, Akiyama et al. [16] proposed a formula for the relationship between the number of software defects (D) and lines of code (LOC) in a project: $D = 4.86 + 0.018 \times \text{LOC}$. However, as software scale and complexity continue to increase, the formula is too simple to predict the relationship between D and LOC. Therefore, based on empirical knowledge, researchers designed a series of metrics associated with software defects. McCabe et al. [17] believed that code complexity is better correlated with software defects than the number of code lines and proposed a *cyclomatic complexity* metric to measure code complexity. In 1994, Chidamber and Kemerer [18] proposed *CK* metrics for the object-oriented program code. In addition to code-based metrics, researchers found that metrics based on software development processes are also associated with software defects. Nagappan and Ball [19] proposed *relative code churn* metrics to predict defect density at file level by measuring code churn during software development.

The purpose of defect prediction technology is to predict the defect proneness, number of defects, or defect density of a program module. Taking the prediction of defect proneness as an example, the general process of defect prediction technology is described in Figure 1.

(1) Program modules are extracted from the software history repository, and the granularity of the modules can be set to *package*, *file*, *change*, or *function* according to actual requirement.

(2) Based on the software code and software development process, metrics related to software defects are designed and used to measure program modules. Defect data sets are built by mining defect logs in the version control system and defect reports in the defect tracking system.

(3) Defect data sets are performed necessary preprocessing (excluding outliers, data normalization, etc.). Defect prediction models are built by using

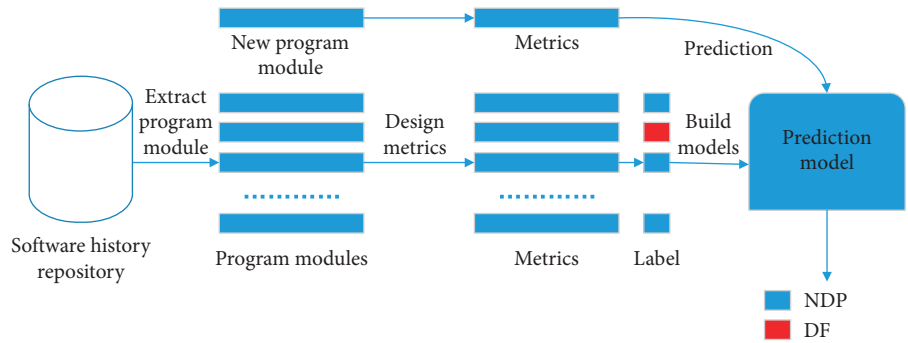

Figure 1: The process of software defect prediction.

statistical or machine learning algorithms such as Naive Bayes, random forest, and support vector machines. Prediction models are evaluated based on performance indicators such as accuracy, recall, and AUC.

(4) When a new program module appears, the same metrics are used to measure the module, and the prediction models are used to predict whether the module is defect prone (DP) or nondefect prone (NDP).

2.2. Related Work

2.2.1. Just-In-Time Software Defect Prediction. Traditional defect prediction techniques are performed at coarse granularity (package, file, or function). Although these techniques are effective in some cases, they have the following disadvantages [7]: first, the granularity of predictions is coarse, so developers need to check a lot of code to locate the defective code; second, program modules are often modified by multiple developers, so it is difficult to find a suitable expert to perform code checking on the defect-prone modules; third, since the defect prediction is performed in the late stages of the software development cycle, developers may have forgotten the details of the development, which reduces the efficiency of defects detection and repair.

In order to overcome the shortcomings of traditional defect prediction, a new technology called just-in-time software defect prediction (JIT-SDP) is proposed [7, 20]. JIT-SDP is a fine-grained prediction method whose prediction objects are code changes instead of modules, which has the following merits: first, since the prediction is performed at a fine granularity, the scope of the code inspection can be reduced to save the effort of code inspection; second, JIT-SDP is performed at the change level, and once a developer submits the modified code, the defect prediction can be executed. Therefore, defect predictions are performed more timely, and it is easy to find the right developer for buggy changes.

In recent years, research related to JIT-SDP has developed rapidly. Mockus and Weiss [20] designed the change metrics to evaluate the probability of defect-inducing changes for initial modification requests (IMR) in a 5ESS network switch project. Yang et al. [21] first applied deep learning technology to JIT-SDP and proposed a method called *Deeper*. The empirical results show that the Deeper method can detect more buggy changes and have better F1 indicator than the EALR method. Kamei et al. [10] conducted experiment on 11 open-source projects for JIT-SDP using cross-project models. They found that the cross-project models can perform well when carefully selecting training data. Yang et al. [22] proposed a two-layer ensemble learning method (TLEL), which leverages decision tree and ensemble learning method. Experimental results show that TLEL can significantly improve the PofB20 and F1 indicators compared to the three baselines. Chen et al. [9] proposed a novel supervised learning method MULTI, which applied multiobjective optimization algorithm to software defect prediction. Experimental results show that MULTI is superior to 43 state-of-the-art prediction models. Considering a commit in the software development may be partially defective, which may involve defective files and nondefective files. To this end, Pascarella et al. [23] proposed a fine-grained JIT-SDP model to predict buggy files in a commit based on ten open-source projects. Experimental results show that 43% defective commits are mixed by buggy and clean resources, and their method can obtain 82% AUC-ROC to detect defective files in a commit.

2.2.2. Effort-Aware Software Defect Prediction. In recent years, researchers pointed out that when using the defect prediction model, the cost-effectiveness of the model should be considered [24]. Due to the high cost of checking for potentially defective program modules, modules with high defect densities should be prioritized when testing resources are limited. Effort-aware defect prediction aims at finding more buggy changes in limited testing resources, which has been extensively studied in module-level (package, file, or function) defect prediction [25, 26]. Kamei et al. [7] applied effort-aware defect prediction to JIT-SDP. In their study, they used code churn (total number of modified lines of code) as a measure of the effort and proposed the EALR model based on linear regression. Their study showed that 35% defect-inducing changes can be detected by using only 20% effort.

Recently, many supervised and unsupervised learning methods are investigated in the context of effort-aware JIT-

SDP. Yang et al. [8] compared simple unsupervised models with supervised models for effort-aware JIT-SDP and found that many unsupervised models outperform the state-of-the-art supervised models. Inspired by the study of Yang et al. [8], Liu et al. [27] proposed a novel unsupervised learning method *CCUM* which is based on code churn. Their study showed that CCUM performs better than the state-of-the-art prediction model at that time. As the results Yang et al. [8] reported are startling, Fu et al. [28] repeated their experiment. Their study indicates that supervised learning methods are better than unsupervised methods when their analysis is conducted on a project-by-project basis. However, their study supports the general goal of Yang et al. [8]. Huang et al. [29] also performed a replication study based on the study of Yang et al. [8], and extended their study in the literature [30]. They found three weaknesses of the LT model proposed by Yang et al. [8]: more context switching, more false alarms, and worse F1 than supervised methods. Therefore, they proposed a novel supervised method CBS+, which combines the advantages of LT and EALR. Empirical results demonstrate that CBS + can detect more buggy changes than EALR. More importantly, CBS + can significantly reduce context switching compared to LT.

3. Software Defect Prediction Based on Local Models

Menzies et al. [11] first applied local models to defect prediction and effort estimation in 2011 and extended their study work in 2013 [12]. The basic idea of local models is simple: first, cluster all training data into homogeneous regions; then train a prediction model for each region. A test instance will be predicted by one of the predictors [31].

To the best of our knowledge, there are only five studies of local models in the filed of defect prediction. Menzies et al. [11, 12] first proposed the idea of local models. Their studies attempted to explore the best way to learn lessons for defect prediction and effort estimation. They adopted *WHERE* algorithm to cluster the data into the regions with similar properties. Experimental results showed that the lessons learned from clusters have better prediction performance by comparing the lessons generated from all the data, local projects, and each cluster. Scanniello et al. [13] proposed a novel defect prediction method based on software clustering for class-level defect prediction. Their research demonstrated that for a class to be predicted in a project system, the models built from the classes related to it outperformed the classes from the entire system. Bettenburg et al. [14] compared local and global models based on statistical learning technology for software defect prediction. Experimental results demonstrated that local models had better fit and prediction performance. Herbold et al. [31] introduced local models into the context of cross-project defect prediction. They compared local models with global models and a transfer learning technique. Their findings showed that local models just made a minor difference compared with global models and the transfer learning model. Mezouar et al. [32] compared the performance of local and global models in the context of effort-aware defect prediction. Their study is based on 15 module-level defect data sets and uses *k*-medoids to build local models. Experimental results showed that local models perform worse than global models.

The process and difference between local models and global models are described in Figure 2. For global models, all training data are used to build prediction models by using statistical or machine learning methods, and the models can be used to predict any test instance. In comparison, local models first build a cluster model and divide the training data into several clusters based on the cluster model. In the prediction phase, each test instance is assigned a cluster number based on the cluster models. And each test instance is predicted by the model trained by the corresponding cluster instead of all the training data.

In our experiments, we apply local models to the JIT-SDP problem. In previous studies, various clustering methods have been used in local models, such as WHERE [11, 12], MCLUST [14], *k*-means [14], hierarchical clustering [14], BorderFlow [13], EM clustering [31], and *k*-mediods [32]. According to the suggestion of Herbold et al. [31], one can use any of these clustering methods. Our experiment adopts *k*-medoids as the clustering method in local models. *K*-medoids is a widely used clustering method. Different from *k*-means used in the previous study [14], the cluster center of *k*-medoids is an object, which has low sensitivity to outliers. Moreover, there are mature Python libraries supporting the implementation of *k*-medoids. In the stage of model building, we use logistic regression and EALR to build the classification model and the effort-aware prediction model in local models, which are used as baseline models in previous JIT-SDP studies [7–9]. The detailed process of local models for JIT-SDP is described in Algorithm 1.

4. Experimental Setup

This article answers the following three questions through experiment:

(i) RQ1: how is the classification performance and effort-aware prediction performance of local models compared to those of global models in the cross-validation scenario?

(ii) RQ2: how is the classification performance and effort-aware prediction performance of local models compared to those of global models in the cross-project-validation scenario?

(iii) RQ3: how is the classification performance and effort-aware prediction performance of local models compared to those of global models in the timewise-cross-validation scenario?

This section introduces the experimental setup from five aspects: data sets, modeling methods, performance evaluation indicators, data analysis method, and data preprocessing. Our experiment is performed on the hardware with *Intel (R) Core (TM) i7-7700 CPU 3.60 GHZ*. The operation system is *Windows 10*. The programming environment for scripts is *Python 3.6*.

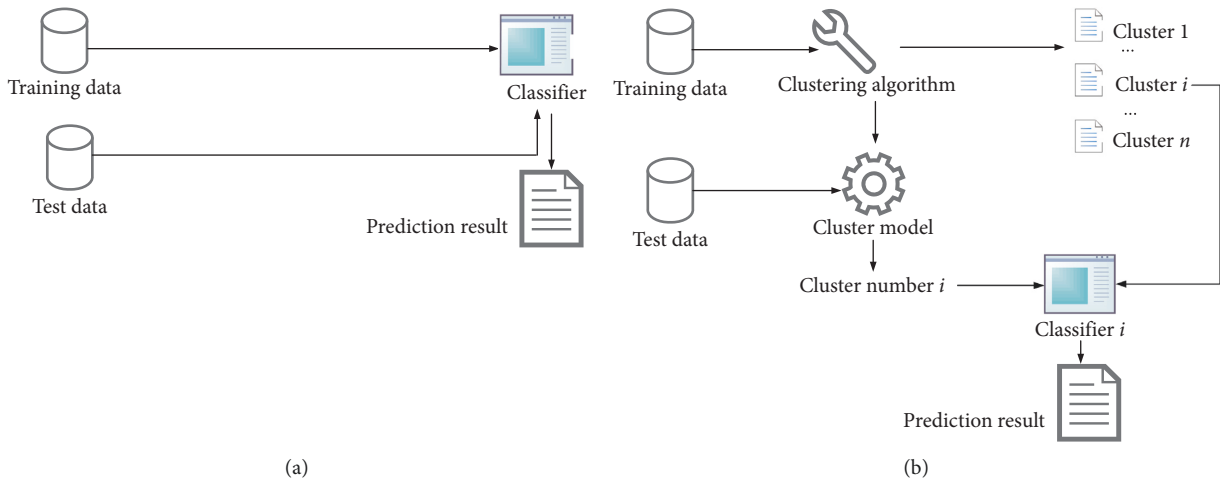

Figure 2: The process of (a) global and (b) local models.

Input: training set: $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$; test set: $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$; the number of clusters: k
Output: prediction results: R

```
(1) begin
(2)   R ⟵ ∅
(3)   //Divide the training set into k clusters
(4)   clusters, cluster_centers = k-medoids (D, k)
(5)   //Train a classifier for each cluster
(6)   classifiers = modeling_algorithm (clusters)
(7)   for Instance t in T do
(8)     //Assign a cluster number i for each instance t
(9)     i = calculate_min_distance (t, cluster_centers)
(10)    r = classifiers [i].predict (t) //perform prediction
(11)    R.append (r)
(12)  end
(13) end
```

Algorithm 1: The application process of local models for JIT-SDP.

4.1. Data Sets. The data sets used in experiment are widely used for JIT-SDP, which are shared by Kamei et al. [7]. The data sets contain six open-source projects from different application domains. For example, Bugzilla (BUG) is an open-source defect tracking system, Eclipse JDT (JDT) is a full-featured Java integrated development environment plugin for the Eclipse platform, Mozilla (MOZ) is a web browser, and PostgreSQL (POS) is a database system [9]. These data sets are extracted from CVS repositories of projects and corresponding bug reports [7]. The basic information of the data sets is shown in Table 1.

To predict whether a change is defective or not, Kamei et al. [7] design 14 metrics associated with software defects, which can be grouped into five groups (i.e., diffusion, size, purpose, history, and experience). The specific description of these metrics is shown in Table 2. A more detailed introduction to the metrics can be found in the literature [7].

4.2. Modeling Methods. There are three modeling methods that are used in our experiment. First, in the process of using local models, the training set is divided into multiple clusters. Therefore, this process is completed by the k-medoids model. Second, the classification performance and effort-aware prediction performance of local models and global models are investigated in our experiment. Therefore, we use logistic regression and EALR to build classification models and effort-aware prediction models, respectively, which are used as baseline models in previous JIT-SDP studies [7–9]. Details of the three models are shown below.

4.2.1. K-Medoids. The first step in local models is to use a clustering algorithm to divide all training data into several homogeneous regions. In this process, the experiment uses k-medoids as a clustering algorithm, which is widely used in different clustering tasks. Different from k-means, the coordinate center of each cluster of k-medoids is a representative object rather than a centroid. Hence, k-medoids is less sensitive to the outliers. The objective function of k-medoids can be defined by using Equation (1), where p is a sample in the cluster C_i and O_i is a medoid in the cluster C_i:

Table 1: The basic information of data sets.

Project	Period	Number of defective changes	Number of changes	% defect rate
BUG	1998/08/26~2006/12/16	1696	4620	36.71
COL	2002/11/25~2006/07/27	1361	4455	30.55
JDT	2001/05/02~2007/12/31	5089	35386	14.38
MOZ	2000/01/01~2006/12/17	5149	98275	5.24
PLA	2001/05/02~2007/12/31	9452	64250	14.71
POS	1996/07/09~2010/05/15	5119	20431	25.06

Table 2: The description of metrics.

Dimension	Metric	Description
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
	AGE	Average time interval between the last and the current change
	NUC	Number of unique last changes to the files
Experience	EXP	Developer experience
	REXP	Recent developer experience
	SEXP	Developer experience on a subsystem

$$E = \sum_{i=1}^{k} \sum_{p \epsilon C_i} |p - O_i|^2. \quad (1)$$

Among different k-medoids algorithms, the partitioning around medoids (PAM) algorithm is used in the experiment, which was proposed by Kaufman el al. [33] and is one of the most popularly used algorithms for k-medoids.

4.2.2. Logistic Regression. Similar to prior studies, logistic regression is used to build classification models [7]. Logistic regression is a widely used regression model that can be used for a variety of classification and regression tasks. Assuming that there are n metrics for a change c, logistic regression can be expressed by using Equation (2), where $w_0, w_1, \ldots, w_n$ denote the coefficients of logistic regression and the independent variables $m_1, \ldots, m_n$ represent the metrics of a change, which are introduced in Table 2. The output of the model $y(c)$ indicates the probability that a change is buggy:

$$y(c) = \frac{1}{1 + e^{-(w_0 + w_1 m_1 + \ldots + w_n m_n)}}. \quad (2)$$

In our experiment, logistic regression is used to solve classification problems. However, the output of the model is a continuous value with a range of 0 to 1. Therefore, for a change c, if the value of $y(c)$ is greater than 0.5, then c is classified as buggy, otherwise it is classified as clean. The process can be defined as follows:

$$Y(c) = \begin{cases} 1, & \text{if } y(c) > 0.5, \\ 0, & \text{if } y(c) \leq 0.5. \end{cases} \quad (3)$$

4.2.3. Effort-Aware Linear Regression. Arisholm et al. [24] pointed out that the use of defect prediction models should consider not only the classification performance of the model, but also the cost-effectiveness of code inspection. Afterwards, a large number of studies focus on the effort-aware defect prediction [8, 9, 11, 12]. Effort-aware linear regression (EALR) was firstly proposed by Kamei et al. [7] for solving effort-aware JIT-SDP, which is based on linear regression models. Different from the above logistic regression, the dependent variable of EALR is $Y(c)$/Effort(c), where $Y(c)$ denotes the defect proneness of a change and Effort(c) denotes the effort required to code checking for the change c. According to the suggestion of Kamei et al. [7], the effort for a change can obtained by calculating the number of modified lines of code.

4.3. Performance Indicators. There are four performance indicators used to evaluate and compare the performance of local and global models for JIT-SDP (i.e., F1, AUC, ACC, and P_{opt}). Specifically, we use F1 and AUC to evaluate the classification performance of prediction models and use ACC and P_{opt} to evaluate the effort-aware prediction performance of prediction models. The details are as follows.

The experiment first uses F1 and AUC to evaluate the classification performance of models. JIT-SDP technology aims to predict the defect proneness of a change, which is a classification task. The predicted results can be expressed as a confusion matrix shown in Table 3, including true positive (TP), false negative (FN), false positive (FP), and true negative (TN).

For classification problems, precision (P) and recall (R) are often used to evaluate the prediction performance of the model:

$$\begin{aligned} P &= \frac{TP}{TP + FP}, \\ R &= \frac{TP}{TP + FN}. \end{aligned} \tag{4}$$

However, the two indicators are usually contradictory in their results. In order to evaluate a prediction model comprehensively, we use F1 to evaluate the classification performance of the model, which is a harmonic mean of precision and recall and is shown in the following equation:

$$F1 = \frac{2 \times P \times R}{P + R}. \tag{5}$$

The second indicator is AUC. Data class imbalance problems are often encountered in software defect prediction. Our data sets are also imbalanced (the number of buggy changes is much less than clean changes). In the class-imbalanced problem, threshold-based evaluation indicators (such as accuracy, recall rate, and F1) are sensitive to the threshold [4]. Therefore, a threshold-independent evaluation indicator AUC is used to evaluate the classification performance of a model. AUC (area under curve) is the area under the ROC (receiver operating characteristic) curve. The drawing process of the ROC curve is as follows. First, sort the instances in a descending order of probability that is predicted to be positive, and then treat them as positive examples in turn. In each iteration, the true positive rate (TPR) and the false positive rate (FPR) are calculated as the ordinate and the abscissa, respectively, to obtain an ROC curve:

$$\begin{aligned} TPR &= \frac{TP}{TP + FN}, \\ FPR &= \frac{FP}{TN + FP}. \end{aligned} \tag{6}$$

The experiment uses ACC and P_{opt} to evaluate the effort-aware prediction performance of local and global models. ACC and P_{opt} are commonly used performance indicators, which have widely been used in previous research [7–9, 25].

The calculation method of ACC and P_{opt} is shown in Figure 3. In Figure 3, x-axis and y-axis represent the cumulative percentage of code churn (i.e., the number of lines of code added and deleted) and defect-inducing changes. To compute the values of ACC and P_{opt}, three curves are included in Figure 3: optimal model curve, prediction model curve, and worst model curve [8]. In the optimal model curve, the changes are sorted in a descending order based on their actual defect densities. In the worst model curve, the changes are sorted in the ascending order according to their actual defect densities. In the prediction model curve, the changes are sorted in the descending order based on their predicted defect densities. Then the area between the three curves and the x-axis, respectively, $Area(optimal)$, $Area(m)$, and $Area(worst)$ are calculated. ACC indicates the recall of defect-inducing changes when 20% of the effort is invested based on the prediction model curve. According to [7], P_{opt} can be defined by the following equation:

$$P_{opt} = 1 - \frac{area(optimal) - area(m)}{area(optimal) - optimal(worst)}. \tag{7}$$

Table 3: Confusion matrix.

Actual value	Prediction result	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

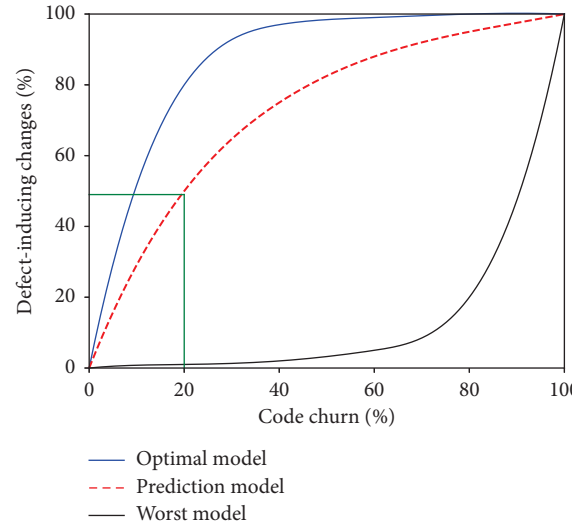

Figure 3: The performance indicators of ACC and P_{opt}.

4.4. Data Analysis Method. The experiment considers three scenarios to evaluate the performance of prediction models, which are 10 times 10-fold cross-validation, cross-project-validation, and timewise-cross-validation, respectively. The details are as follows.

4.4.1. 10 Times 10-Fold Cross-Validation. Cross-validation is performed within the same project, and the basic process of 10 times 10-fold cross-validation is as follows: first, the data sets of a project are divided into 10 parts with equal size. Subsequently, nine parts of them are used to train a prediction model and the remaining part is used as the test set. This process is repeated 10 times in order to reduce the randomness deviation. Therefore, 10 times 10-fold cross-validation can produce $10 \times 10 = 100$ results.

4.4.2. Cross-Project-Validation. Cross-project defect prediction has received widespread attention in recent years [34]. The basic idea of cross-project defect prediction is to train a defect prediction model with the defect data sets of a project to predict defects of another project [8]. Given data sets containing n projects, $n \times (n-1)$ run results can be generated for a prediction model. The data sets used in our experiment contain six open-source projects, so $6 \times (6-1) = 30$ prediction results can be generated for a prediction model.

4.4.3. Timewise-Cross-Validation. Timewise-cross-validation is also performed within the same project, but it considers the chronological order of changes in the data sets [35]. Defect data sets are collected in the chronological order during the actual software development process. Therefore, changes committed early in the data sets can be used to predict the defect proneness of changes committed late. In the timewise-cross-validation scenario, all changes in the data sets are arranged in the chronological order, and the changes in the same month are grouped in the same group. Suppose the data sets are divided into n groups. Group i and group $i+1$ are used to train a prediction model, and the group $i+4$ and group $i+5$ are used as a test set. Therefore, assuming that a project contains n months of data sets, $n-5$ run results can be generated for a prediction model.

The experiment uses the Wilcoxon signed-rank test [36] and Cliff's δ [37] to test the significance of differences in classification performance and prediction performance between local and global models. The Wilcoxon signed-rank test is a popular nonparametric statistical hypothesis test. We use p values to examine if the difference of the performance of local and global models is statistically significant at a significance level of 0.05. The experiment also uses Cliff's δ to determine the magnitude of the difference in the prediction performance between local models and global models. Usually the magnitude of the difference can be divided into four levels, i.e., trivial ($(|\delta| < 0.147)$), small ($0.147 \le |\delta| < 0.33$), moderate ($0.33 \le |\delta| < 0.474$), and large ($\ge 0.474$) [38]. In summary, when the p value is less than 0.05 and Cliff's δ is greater than or equal to 0.147, local models and global models have significant differences in prediction performance.

4.5. Data Preprocessing. According to suggestion of Kamei et al. [7], it is necessary to preprocess the data sets. It includes the following three steps:

(1) *Removing Highly Correlated Metrics.* The ND and REXP metrics are excluded because NF and ND, REXP, and EXP are highly correlated. LA and LD are normalized by dividing by LT, since LA and LD are highly correlated. LT and NUC are normalized by dividing by NF since LT and NUC are highly correlated with NF.

(2) *Logarithmic Transformation.* Since most metrics are highly skewed, each metric executes logarithmic transformation (except for fix).

(3) *Dealing with Class Imbalance.* The data sets used in the experiment have the problem of class imbalance, i.e., the number of defect-inducing changes is far less than the number of clean changes. Therefore, we perform random undersampling on the training set. By randomly deleting clean changes, the number of buggy changes remains the same as the number of clean changes. Note that the undersampling is not performed in the test set.

5. Experimental Results and Analysis

5.1. Analysis for RQ1. To investigate and compare the performance of local and global models in the 10 times 10-fold cross-validation scenario, we conduct a large-scale empirical study based on data sets from six open-source projects. The experimental results are shown in Table 4. As is shown in Table 4, the first column represents the evaluation indicators for prediction models, including AUC, F1, ACC, and P_{opt}; the second column denotes the project names of six data sets; the third column shows the performance values of the global models in the four evaluation indicators; from the fourth column to the twelfth column, the performance values of the local models at different k values are shown. Considering that the number of clusters k may affect the performance of local models, the value of parameter k is tuned from 2 to 10. Since 100 prediction results are generated in the 10 times 10-fold cross-validation scenario, the values in the table are medians of the experimental results. We do the following processing on the data in Table 4:

(i) We performed the significance test defined in Section 4.4 on the performance of the local and global models. The performance values of local models that are significantly different from the performance values of global model are bolded.

(ii) The performance values of the local models which are significantly better than those of the global models are given in italics.

(iii) The W/D/L values (i.e., the number of projects for which local models can obtain a better, equal, and worse prediction performance than global models) are calculated under different k values and different evaluation indicators.

First, we analyze and compare the classification performance of local and global models. As is shown in Table 4, firstly, all of the performance of local models with different parameters k and different projects are worse than that of global models in the AUC indicator. Secondly, in most cases, the performance of local models are worse than that of global models (except for four values in italics) in the F1 indicator. Therefore, it can be concluded that global models perform better than local models in the classification performance.

Second, we investigate and compare effort-aware prediction performance for local and global models. As can be seen from Table 4, local models are valid for effort-aware JIT-SDP. However, local models have different effects on different projects and different parameters k.

Table 4: Local vs. global models in the 10 times 10-fold cross-validation scenario.

Indicator	Project	Global	Local								
			$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
AUC	BUG	0.730	**0.704**	**0.687**	**0.684**	**0.698**	**0.688**	**0.686**	**0.686**	**0.693**	**0.685**
	COL	0.742	**0.719**	**0.708**	**0.701**	**0.703**	**0.657**	**0.666**	**0.670**	**0.668**	**0.660**
	JDT	0.725	**0.719**	**0.677**	**0.670**	**0.675**	**0.697**	**0.671**	**0.674**	**0.671**	**0.654**
	MOZ	0.769	**0.540**	**0.741**	**0.715**	**0.703**	**0.730**	**0.730**	**0.725**	**0.731**	**0.722**
	PLA	0.742	**0.733**	**0.702**	**0.693**	**0.654**	**0.677**	**0.640**	**0.638**	**0.588**	**0.575**
	POS	0.777	**0.764**	**0.758**	**0.760**	**0.758**	**0.753**	**0.755**	**0.705**	**0.757**	**0.757**
	W/D/L	—	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6	0/0/6
F1	BUG	0.663	**0.644**	**0.628**	**0.616**	**0.603**	**0.598**	**0.608**	**0.610**	**0.601**	**0.592**
	COL	0.696	**0.672**	**0.649**	**0.669**	**0.678**	**0.634**	**0.537**	**0.596**	**0.604**	**0.564**
	JDT	0.723	**0.706**	*0.764*	*0.757*	0.734	**0.695**	**0.615**	**0.643**	0.732	0.722
	MOZ	0.825	**0.374**	*0.853*	**0.758**	**0.785**	**0.741**	**0.718**	**0.737**	**0.731**	**0.721**
	PLA	0.704	**0.678**	**0.675**	**0.527**	**0.477**	**0.557**	0.743	*0.771*	**0.627**	**0.604**
	POS	0.762	**0.699**	**0.747**	0.759	**0.746**	**0.698**	**0.697**	**0.339**	**0.699**	**0.700**
	W/D/L	—	0/0/6	2/0/4	1/1/4	0/1/5	0/0/6	0/1/5	1/0/5	0/1/5	0/1/5
ACC	BUG	0.421	*0.452*	*0.471*	0.458	0.424	0.451	0.450	*0.463*	0.446	*0.462*
	COL	0.553	**0.458**	**0.375**	**0.253**	**0.443**	**0.068**	**0.195**	**0.459**	**0.403**	**0.421**
	JDT	0.354	*0.532*	0.374	*0.431*	**0.291**	*0.481*	*0.439*	**0.314**	**0.287**	0.334
	MOZ	0.189	*0.386*	*0.328*	*0.355*	*0.364*	*0.301*	*0.270*	*0.312*	*0.312*	*0.312*
	PLA	0.368	*0.472*	*0.509*	*0.542*	*0.497*	*0.509*	*0.456*	0.442	*0.456*	*0.495*
	POS	0.336	*0.490*	*0.397*	*0.379*	*0.457*	*0.442*	*0.406*	0.341	*0.400*	*0.383*
	W/D/L	—	5/0/1	4/1/1	4/1/1	3/1/2	4/1/1	4/1/1	2/2/2	3/1/2	4/1/1
P_{opt}	BUG	0.742 055	0.747	0.744	**0.726**	0.738	0.749	0.750	0.760	0.750	0.766
	COL	0.726	**0.661**	**0.624**	**0.401**	0.731	**0.295**	**0.421**	**0.644**	**0.628**	**0.681**
	JDT	0.660	*0.779*	**0.625**	0.698	**0.568**	*0.725*	*0.708*	**0.603**	**0.599**	**0.634**
	MOZ	0.555	*0.679*	*0.626*	*0.645*	*0.659*	*0.588*	*0.589*	*0.643*	*0.634*	*0.630*
	PLA	0.664	*0.723*	*0.727*	*0.784*	*0.746*	*0.758*	0.718	0.673	0.716	*0.747*
	POS	0.673	*0.729*	0.680	**0.651**	*0.709*	*0.739*	0.683	**0.596**	0.669	**0.644**
	W/D/L	—	4/1/1	2/2/2	2/1/3	3/2/1	4/1/1	2/3/1	1/2/3	1/3/2	2/1/3

First, local models have different effects on different data sets. To describe the effectiveness of local models on different data sets, we calculate the number of times that local models perform significantly better than global models on 9 k values and two evaluation indicators (i.e., ACC and P_{opt}) for each data set. The statistical results are shown in Table 5, where the second row is the number of times that local models perform better than global models for each project. It can be seen that local models have better performance on the projects MOZ and PLA and poor performance on the projects BUG and COL. Based on the number of times that local models outperform global models in six projects, we rank the project as shown in the third row. In addition, we list the sizes of data sets and their rank as shown in the fourth row. It can be seen that the performance of local models on data sets has a strong correlation with the size of data sets. Generally, the larger the size of the data set, the more significant the performance improvement of local models.

In view of this situation, we consider that if the size of the data set is too small when using local models, since the data sets are divided into multiple clusters, the training samples of each cluster will be further reduced, which will lead to the performance degradation of models. While the size of data sets is large, the performance of the model is not easily limited by the size of the data sets so that the advantages of local models are better reflected.

In addition, the number of clusters k also has an impact on the performance of local models. As can be seen from the table, according to the W/D/L analysis, local models can obtain the best ACC indicator when k is set to 2 and obtain the best P_{opt} indicator when k is set to 2 and 6. Therefore, we recommend that in the cross-validation scenario, when using local models to solve the effort-aware JIT-SDP problem, the parameter k can be set to 2.

This section analyzes and compares the classification performance and effort-aware prediction performance of local and global models in 10 times 10-fold cross-validation scenario. The empirical results show that compared with global models, local models perform poorly in the classification performance but perform better in the effort-aware prediction performance. However, the performance of local models is affected by the parameter k and the size of the data sets. The experimental results show that local models can obtain better effort-aware prediction performance on the project with larger size of data sets and can obtain better effort-aware prediction performance when k is set to 2.

5.2. Analysis for RQ2. This section discusses the performance differences between local and global models in the cross-project-validation scenario for JIT-SDP. The results of the experiment are shown in Table 6, in which the first column indicates four evaluation indicators including AUC, F1,

Table 5: The number of wins of local models at different k values.

Project	BUG	COL	JDT	MOZ	PLA	POS
Count	4	0	7	18	14	11
Rank	5	6	4	1	2	3
Number of changes (rank)	4620 (5)	4455 (6)	35386 (3)	98275 (1)	64250 (2)	20431 (3)

ACC, and P_{opt}; the second column represents the performance values of local models; from the third column to the eleventh column, the prediction performance of local models under different parameters k is represented. In our experiment, six data sets are used in the cross-project-validation scenario. Therefore, each prediction model produces 30 prediction results. The values in Table 6 represent the median of the prediction results.

For the values in Table 6, we do the following processing:

(i) Using the significance test method, the performance values of local models that are significantly different from those of global models are bolded.

(ii) The performance values of local models that are significantly better than those of global models are given in italics.

(iii) The W/D/L column analyzes the number of times that local models perform significantly better than global models at different parameters k.

First, we compare the difference in classification performance between local models and global models. As can be seen from Table 6, in the AUC and F1 indicators, the classification performance of local models under different parameter k is worse than or equal to that of global models. Therefore, we can conclude that in the cross-project-validation scenario, local models perform worse than global models in the classification performance.

Second, we compare the effort-aware prediction performance of local and global models. It can be seen from Table 6 that the performance of local models is better than or equal to that of global models in the ACC and P_{opt} indicators when the parameter k is from 2 to 10. Therefore, local models are valid for effort-aware JIT-SDP in the cross-validation scenario. In particular, when the number k of clusters is set to 2, local models can obtain better ACC values and P_{opt} values than global models; when k is set to 5, local models can obtain better ACC values. Therefore, it is appropriate to set k to 2, which can find 49.3% defect-inducing changes when using 20% effort and increase the ACC indicator by 57.0% and increase the P_{opt} indicator by 16.4%.

In the cross-project-validation scenario, local models perform worse in the classification performance but perform better in the effort-aware prediction performance than global models. However, the setting of k in local models has an impact on the effort-aware prediction performance for local models. The empirical results show that when k is set to 2, local models can obtain the optimal effort-aware prediction performance.

5.3. Analysis for RQ3. This section compares the prediction performance of local and global models in the timewise-cross-validation scenario. Since in the timewise-cross-validation scenario only two months of data in each data sets are used to train prediction models, the size of the training sample in each cluster is small. Through experiment, we find that if the parameter k of local models is set too large, the number of training samples in each cluster may be too small or even equal to 0. Therefore, the experiment sets the parameter k of local models to 2. The experimental results are shown in Table 7. Suppose a project contains n months of data, and prediction models can produce $n-5$ prediction results in the timewise-cross-validation scenario. Therefore, the third and fourth columns in Table 7 represent the median and standard deviation of prediction results for global and local models. Based on the significance test method, the performance values of local models with significant differences from those of global models are bold. The row W/D/L summarizes the number of projects for which local models can obtain a better, equal, and worse performance than global models in each indicator.

First, we compare the classification performance of local and global models. It can be seen from Table 7 that local models perform worse on the AUC and F1 indicators than global models on the 6 data sets. Therefore, it can be concluded that local models perform poorly in classification performance compared to global models in the timewise-cross-validation scenario.

Second, we compare the effort-aware prediction performance of local and global models. It can be seen from Table 7 that in the projects BUG and COL, local models have worse ACC and P_{opt} indicators than global models, while in the other four projects (JDT, MOZ, PLA, and POS), local models and global models have no significant difference in the ACC and P_{opt} indicators. Therefore, local models perform worse in effort-aware prediction performance than global models in the timewise-cross-validation scenario. This conclusion is inconsistent with the conclusions in Sections 5.1 and 5.2. For this problem, we consider that the main reason lies in the size of the training sample. In the timewise-cross-validation scenario, only two months of data in the data sets are used to train prediction models (there are only 452 changes per month on average). When local models are used, since the training set is divided into several clusters, the number of training samples in each cluster is further decreased, which results in a decrease in the effort-aware prediction performance of local models.

In this section, we compare the classification performance and effort-aware prediction performance of local and global models in the timewise-cross-validation scenario. Empirical results show that local models perform worse than global models in the classification performance and effort-aware prediction performance. Therefore, we recommend

Table 6: Local vs. global models in the cross-project-validation scenario.

Indicator	Global	Local									W/D/L
		$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	
AUC	0.706	**0.683**	**0.681**	**0.674**	**0.673**	**0.673**	**0.668**	**0.659**	**0.680**	**0.674**	0/0/9
F1	0.678	**0.629**	0.703	0.696	0.677	**0.613**	**0.620**	**0.554**	**0.629**	0.637	0/4/5
ACC	0.314	*0.493*	0.396	0.348	*0.408*	0.393	0.326	0.34	0.342	0.345	2/7/0
P_{opt}	0.656	*0.764*	0.691	0.672	0.719	0.687	0.619	0.632	0.667	0.624	1/8/0

Table 7: Local vs. global models in the timewise-cross-validation scenario.

Indicators	Project	Global	Local ($k=2$)
AUC	BUG	0.672 ± 0.125	$\mathbf{0.545 \pm 0.124}$
	COL	0.717 ± 0.075	$\mathbf{0.636 \pm 0.135}$
	JDT	0.708 ± 0.043	$\mathbf{0.645 \pm 0.073}$
	MOZ	0.749 ± 0.034	$\mathbf{0.648 \pm 0.128}$
	PLA	0.709 ± 0.054	$\mathbf{0.642 \pm 0.076}$
	POS	0.743 ± 0.072	$\mathbf{0.702 \pm 0.095}$
	W/D/L	—	0/0/6
F1	BUG	0.587 ± 0.108	$\mathbf{0.515 \pm 0.128}$
	COL	0.651 ± 0.066	$\mathbf{0.599 \pm 0.125}$
	JDT	0.722 ± 0.048	$\mathbf{0.623 \pm 0.161}$
	MOZ	0.804 ± 0.050	$\mathbf{0.696 \pm 0.190}$
	PLA	0.702 ± 0.053	$\mathbf{0.618 \pm 0.162}$
	POS	0.714 ± 0.065	$\mathbf{0.657 \pm 0.128}$
	W/D/L	—	0/0/6
ACC	BUG	0.357 ± 0.212	$\mathbf{0.242 \pm 0.210}$
	COL	0.426 ± 0.168	$\mathbf{0.321 \pm 0.186}$
	JDT	0.356 ± 0.148	0.329 ± 0.189
	MOZ	0.218 ± 0.132	0.242 ± 0.125
	PLA	0.358 ± 0.181	0.371 ± 0.196
	POS	0.345 ± 0.159	0.306 ± 0.205
	W/D/L	—	0/4/2
P_{opt}	BUG	0.622 ± 0.194	$\mathbf{0.458 \pm 0.189}$
	COL	0.669 ± 0.150	$\mathbf{0.553 \pm 0.204}$
	JDT	0.654 ± 0.101	0.624 ± 0.142
	MOZ	0.537 ± 0.110	0.537 ± 0.122
	PLA	0.631 ± 0.115	0.645 ± 0.151
	POS	0.615 ± 0.131	0.583 ± 0.167
	W/D/L	—	0/4/2

using global models to solve the JIT-SDP problem in the timewise-cross-validation scenario.

6. Threats to Validity

6.1. External Validity. Although our experiment uses public data sets that have extensively been used in previous studies, we cannot yet guarantee that the discovery of the experiment can be applied to all other change-level defect data sets. Therefore, the defect data sets of more projects should be mined in order to verify the generalization of the experimental results.

6.2. Construct Validity. We use F1, AUC, ACC, and P_{opt} to evaluate the prediction performance of local and global models. Because defect data sets are often imbalanced, AUC is more appropriate as a threshold-independent evaluation indicator to evaluate the classification performance of models [4]. In addition, we use F1, a threshold-based evaluation indicator, as a supplement to AUC. Besides, similar to prior study [7, 9], the experiment uses ACC and P_{opt} to evaluate the effort-aware prediction performance of models. However, we cannot guarantee that the experimental results are valid in all other evaluation indicators such as precision, recall, F_β, and PofB20.

6.3. Internal Validity. Internal validity involves errors in the experimental code. In order to reduce this risk, we carefully examined the code of the experiment and referred to the code published in previous studies [7, 8, 29] so as to improve the reliability of the experiment code.

7. Conclusions and Future Work

In this article, we firstly apply local models to JIT-SDP. Our study aims to compare the prediction performance of local and global models under cross-validation, cross-project-validation, and timewise-cross-validation for JIT-SDP. In order to compare the classification performance and effort-aware prediction performance of local and global models, we first build local models based on the k-medoids method. Afterwards, logistic regression and EALR are used to build classification models and effort-aware prediction models for local and global models. The empirical results show that local models perform worse in the classification performance than global models in three evaluation scenarios. Moreover, local models have worse effort-aware prediction performance in the timewise-cross-validation scenario. However, local models are significantly better than global models in the effort-aware prediction performance in the cross-validation-scenario and timewise-cross-validation scenario. Particularly, the optimal effort-aware prediction performance can be obtained when the parameter k of local models is set to 2. Therefore, local models are still promising in the context of effort-aware JIT-SDP.

In the future, first, we plan to consider more commercial projects to further verify the validity of the experimental conclusions. Second, logistic regression and EALR are used in local and global models. Considering that the choice of modeling methods will affect the prediction performance of local and global models, we hope to add more baselines to local and global models to verify the generalization of experimental conclusions.

Data Availability

The experiment uses public data sets shared by Kamei et al. [7], and they have already published the download address of

the data sets in their paper. In addition, in order to verify the repeatability of our experiment and promote future research, all the experimental code in this article can be downloaded at https://github.com/yangxingguang/LocalJIT.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the NSF of China under Grant nos. 61772200 and 61702334, Shanghai Pujiang Talent Program under Grant no. 17PJ1401900, Shanghai Municipal Natural Science Foundation under Grant nos. 17ZR1406900 and 17ZR1429700, Educational Research Fund of ECUST under Grant no. ZH1726108, and Collaborative Innovation Foundation of Shanghai Institute of Technology under Grant no. XTCX2016-20.

References

[1] M. Newman, *Software Errors Cost US Economy $59.5 Billion Annually, NIST Assesses Technical Needs of Industry to Improve Software-Testing*, 2002, http://www.abeacha.com/NIST_press_release_bugs_cost.htm.

[2] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[3] X. Chen, Q. Gu, W. Liu, S. Liu, and C. Ni, "Survey of static software defect prediction," *Ruan Jian Xue Bao/Journal of Software*, vol. 27, no. 1, 2016, in Chinese.

[4] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 2017.

[5] A. G. Koru, D. Dongsong Zhang, K. El Emam, and H. Hongfang Liu, "An investigation into the functional form of the size-defect relationship for software modules," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 293–304, 2009.

[6] S. Kim, E. J. Whitehead, and Y. Zhang, "Classifying software changes: clean or buggy?," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, 2008.

[7] Y. Kamei, E. Shihab, B. Adams et al., "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 2013.

[8] Y. Yang, Y. Zhou, J. Liu et al., "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 157–168, Hong Kong, China, November 2016.

[9] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: multi-objective effort-aware just-in-time software defect prediction," *Information and Software Technology*, vol. 93, pp. 1–13, 2018.

[10] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2072–2106, 2016.

[11] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. R. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 343–351, Lawrence, KS, USA, November 2011.

[12] T. Menzies, A. Butcher, D. Cok et al., "Local versus global lessons for defect prediction and effort estimation," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 822–834, 2013.

[13] G. Scanniello, C. Gravino, A. Marcus, and T. Menzies, "Class level fault prediction using software clustering," in *Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 640–645, Silicon Valley, CA, USA, November 2013.

[14] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Towards improving statistical modeling of software engineering data: think locally, act globally!," *Empirical Software Engineering*, vol. 20, no. 2, pp. 294–335, 2015.

[15] Q. Wang, S. Wu, and M. Li, "Software defect prediction," *Journal of Software*, vol. 19, no. 7, pp. 1565–1580, 2008, in Chinese.

[16] F. Akiyama, "An example of software system debugging," *Information Processing*, vol. 71, pp. 353–359, 1971.

[17] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[18] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[19] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pp. 284–292, Saint Louis, MO, USA, May 2005.

[20] A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 169–180, 2000.

[21] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 17–26, Vancouver, BC, Canada, August 2015.

[22] X. Yang, D. Lo, X. Xia, and J. Sun, "TLEL: a two-layer ensemble learning approach for just-in-time defect prediction," *Information and Software Technology*, vol. 87, pp. 206–220, 2017.

[23] L. Pascarella, F. Palomba, and A. Bacchelli, "Fine-grained just-in-time defect prediction," *Journal of Systems and Software*, vol. 150, pp. 22–36, 2019.

[24] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[25] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM)*, pp. 1–10, Shanghai, China, September 2010.

[26] Y. Zhou, B. Xu, H. Leung, and L. Chen, "An in-depth study of the potentially confounding effect of class size in fault prediction," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 1, pp. 1–51, 2014.

[27] J. Liu, Y. Zhou, Y. Yang, H. Lu, and B. Xu, "Code churn: a neglected metric in effort-aware just-in-time defect prediction," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 11–19, Toronto, ON, Canada, November 2017.

[28] W. Fu and T. Menzies, "Revisiting unsupervised learning for defect prediction," in *Proceedings of the 2017 11th Joint*

Meeting on Foundations of Software Engineering (ESEC/FSE), pp. 72–83, Paderborn, Germany, September 2017.

[29] Q. Huang, X. Xia, and D. Lo, "Supervised vs. unsupervised models: a holistic look at effort-aware just-in-time defect prediction," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 159–170, Shanghai, China, September 2017.

[30] Q. Huang, X. Xia, and D. Lo, "Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction," *Empirical Software Engineering*, pp. 1–40, 2018.

[31] S. Herbold, A. Trautsch, and J. Grabowski, "Global vs. local models for cross-project defect prediction," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1866–1902, 2017.

[32] M. E. Mezouar, F. Zhang, and Y. Zou, "Local versus global models for effort-aware defect prediction," in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering (CASCON)*, pp. 178–187, Toronto, ON, Canada, October 2016.

[33] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, Hoboken, NJ, USA, 1990.

[34] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 111–147, 2019.

[35] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," *Mining Software Repositories*, vol. 30, no. 4, pp. 1–5, 2005.

[36] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[37] N. Cliff, *Ordinal Methods for Behavioral Data Analysis*, Psychology Press, London, UK, 2014.

[38] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: should we really be using *t*-test and cohensd for evaluating group differences on the nsse and other surveys," in *Annual Meeting of the Florida Association of Institutional Research*, pp. 1–33, 2006.