Received: 22 October 2022 | Revised: 3 February 2023 | Accepted: 5 June 2023

IET Software

DOI: 10.1049/sfw2.12133

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Revisiting 'revisiting supervised methods for effort-aware cross-project defect prediction'

Fuyang Li[1] | Peixin Yang[1,2] | Jacky Wai Keung[3] | Wenhua Hu[1] | Haoyu Luo[4] | Xiao Yu[1,2,5]

[1]School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China

[2]Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China

[3]Department of Computer Science, City University of Hong Kong, Hong Kong, China

[4]College of Mathematics and Informatics, South China Agricultural University, Guangzhou, China

[5]Wuhan University of Technology Chongqing Research Institute, Chongqing, China

Correspondence

Xiao Yu, Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China.
Email: xiaoyu@whut.edu.cn

Funding information

Project of Sanya Yazhou Bay Science and Technology City, Grant/Award Number: SCKJ-JYRC-2022-17; Natural Science Foundation of China, Grant/Award Number: 62272356; Youth Fund Project of Hainan Natural Science Foundation, Grant/Award Number: 622QN344; Natural Science Foundation of Chongqing, Grant/Award Number: cstc2021jcyj-msxmX1115; Start-up Grant from Wuhan University of Technology, Grant/Award Number: 104-40120693

Abstract

Effort-aware cross-project defect prediction (EACPDP), which uses cross-project software modules to build a model to rank within-project software modules based on the defect density, has been suggested to allocate limited testing resource efficiently. Recently, Ni et al. proposed an EACPDP method called EASC, which used all cross-project modules to train a model without considering the data distribution difference between cross-project and within-project data. In addition, Ni et al. employed the different defect density calculation strategies when comparing EASC and baseline methods. To explore the effective defect density calculation strategies and methods on EACPDP, the authors compare four data filtering methods and five transfer learning methods with EASC using four commonly used defect density calculation strategies. The authors use three classification evaluation metrics and seven effort-aware metrics to assess the performance of methods on 11 PROMISE datasets comprehensively. The results show that (1) The classification before sorting (CBS+) defect density calculation strategy achieves the best overall performance. (2) Using balanced distribution adaption (BDA) and joint distribution adaptation (JDA) with the K-nearest neighbour classifier to build the EACPDP model can find 15% and 14.3% more defective modules and 11.6% and 8.9% more defects while achieving the acceptable initial false alarms (IFA). (3) Better comprehensive classification performance of the methods can bring better EACPDP performance to some extent. (4) A flexible adjustment of the defect threshold λ of the CBS+ strategy contribute to different goals. In summary, the authors recommend researchers and practitioners use to BDA and JDA with the CBS+ strategy to build the EACPDP model.

KEYWORDS

data mining, quality assurance, software engineering, software maintenance, software metrics, software quality

1 | INTRODUCTION

Computer software is widely used in various industries in society today, which may fail with quality problems. As software plays an increasingly important role in various fields, ensuring software reliability is one of the issues people are more concerned about [1–3]. Software defects are a potential sources of errors, failures, and crashes of associated systems. However, the increase in the scale of the software makes defect inspection and fixing more time-consuming. Worse yet, software

testing teams usually have limited testing resources and cannot inspect all software modules in a short period of time [4–6]. Therefore, software defect prediction (SDP) techniques have received more attention for fully using limited resources. Software testing teams build the SDP model based on the historical software data to predict the defective-proneness of the software module to be inspected. They can devote more testing resources to or first inspect those software modules that are predicted to be defective, which is very beneficial to allocating software testing resources efficiently [7, 8] and help defect localisation [9–12].

Most previous SDP models are based on binary classification algorithms, and such models predict software modules into two categories, that is, defective and clean. When software testing resources are inadequate to inspect all the predicted defective modules, the prediction results of classification-based SDP models cannot guide software testing teams regarding which predicted defective modules to inspect first. Therefore, for the first time, Mende et al. [13] proposed effort-aware defect prediction (EADP) to sort software modules based on the defect density and inspect the modules with higher defect densities first. Software testing teams can find more defects when checking a certain number of lines of code (LOC). For example, Fenton [14] and Andersson [15] et al. pointed out that approximately 20% of LOC account for 80% or more of the defects.

1.1 | Motivations

Most previous EADP studies [13, 16–21] usually built predictive models on historical labelled software modules and then predicted the defect-proneness of unlabelled modules in the same project, referred to as within-project EADP. However, in actual software development scenarios, it is hard to obtain a large amount of historical data from the same project, especially for newly developed software [22–27]. Therefore, Ni et al. [28] proposed an effort-aware cross-project defect prediction (EACPDP) method called EASC in their IEEE transactions on software engineering (TSE) paper titled 'Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction'.

EASC first trains a naive Bayes classifier using cross-project software modules and then divides all within-project software modules into two lists (i.e. defective and clean), which contain predicted defective and clean software modules, separately. Finally, EASC appends clean modules to defective ones after ranking the two lists separately by defect density (i.e. the ratio between the predicted defective probability and LOC), since EASC considers that high defective-prone software modules with less LOC should be checked, preferentially. The results of experiments on four public datasets (AEEEM [29], NASA [30, 31], PROMISE [32], and RELINK [33]) demonstrate that the EASC method significantly outperforms some unsupervised methods when considering effort-aware performance measures.

Although EASC achieves encouraging performance, Ni et al.'s study [28] still has two issues that can be further improved, which are:

(1) **EASC trains the naive Bayes classifier using all cross-project data without considering the data distribution difference between within-project and cross-project modules.** Zimmermann et al.'s study [34] shows that classification-based Cross-Project Defect Prediction (CPDP) models do not work well when all cross-project data is used for training. Since EASC utilises the built naive Bayes model to calculate the defect density, the distribution difference also degrades the performance of EASC.
(2) The experimental results showed that EASC performed better than two data filtering methods (i.e. BF filter [35] and Menzies11-RF [36]), one data transformation method (i.e. CamargoCruz09-DT [37]) and one transfer learning method (i.e. Watanabe08-DT [38]). However, the four methods used the predicted defective probability as defect density to rank software modules. **The inconsistency in defect density calculation strategies between the four compared methods and EASC leads to unfair comparisons.** Even though the four compared methods account for the data distribution difference, it is unclear whether data filtering and transfer learning methods can enhance the performance of EACPDP models, since defect densities are calculated differently using EASC.

1.2 | Our works and contributions

In light of the two issues and the high impact of the work published by Ni et al. [28] in the TSE journal, we first do a literature review to identify 32 primary EADP articles published until January 2023 and find the four commonly used defect density strategies in the studies, including Label/LOC, Prob/LOC, classification before sort (CBS+) and Prob. Then, we conduct a comprehensive empirical study on 11 software defect datasets from the PROMISE corpus by posing the following four research questions (RQs). In this section, certain notations of the methods are utilised directly, which will be thoroughly explained in Section 3.

RQ1: What is the best defect density calculation strategy for EACPDP?

We investigate the impact of the four common defect density calculation strategies for EACPDP. Since the main purpose of EACPDP is to find more defects and defective modules and to obtain a more accurate global ranking of software modules, we mainly use PofB@20% (Proportion of Bugs found when the top 20% LOC are inspected), Recall@20% and *Popt* to measure the effectiveness of the calculation strategies. We also use Precision@20% and initial false alarms (IFA) to evaluate the false positive rate, and PMI@20% (Proportion of Modules Inspected when the top 20% LOC are inspected) to measure how many software modules need to be checked. Then, we use the Scott-Knott Effect Size Difference (ESD) [39] test to group these EACPDP methods into different rankings. The results show that the CBS+ defect density calculation strategy achieves better overall performance. The Label/LOC and Prob/LOC calculation strategies perform poorly in terms of PMI@20% and IFA, while the Prob calculation strategy has a

very poor performance in terms of Recall@20% and PofB@20%. Based on these results, we recommend applying the CBS+ defect density strategy to EACPDP, which first divides the modules into predicted defective group and clean group based on the defined defect threshold λ, then sorts the modules in the two groups separately by Prob/LOC and finally ranks the defective group ahead of the clean group.

RQ2: Can data filtering and transfer learning approaches improve the EACPDP performance?

To alleviate the data distribution difference between cross-project and within-project modules, we apply the four data filtering methods (BF, PF, KF and DFAC), and the five transfer learning methods (TCA, balanced distribution adaption [BDA], joint distribution adaptation [JDA], JPDA and TNB) to cross-project data, and use K-Nearest Neighbour (KNN), Logistic Regression (LR), and Random Forest (RF) to build the EACPDP models with the CBS+ strategy. We employ the Wilcoxon signed-rank test [40] to validate the performance improvement between the four data filtering methods and five transfer learning methods with None (without any data filtering and transfer learning methods). Almost all data filtering methods cannot enhance the EACPDP performance. BDA and JDA with the KNN classifier to build EACPDP models significantly improve the Recall@20% value by 15% and 14.3% and the PofB@20% value by 11.6% and 8.9% while achieving the acceptable IFA.

RQ3: What is the relationship among the performance measures?

We first use BDA and JDA with three classifiers to build the classification model and then rank the software modules by the CBS+ strategy. In order to investigate whether better classification performance can contribute to building better EACPDP models, we employ the Kendell correlation coefficient to evaluate the relationship between the three classification metrics (i.e. Precision, Recall and F1) and the seven effort-aware metrics. The results show that Precision has a very high correlation with Precision@20% and F1@20%, and F1 has a moderate or low correlation with F1@20% on BDA and JDA with the three classifiers. This implies that better comprehensive classification performance of the methods can bring better EACPDP performance to some extent.

RQ4: How does the defect threshold λ of the CBS+ strategy affect EACPDP performance?

One key of the CBS+ strategy is to divide all modules into predicted defective and clean groups based on the defect threshold λ. To explore the effect of λ, we analyse the performance of BDA and JDA with three classifiers when we change the defect threshold λ from 0.1 to 0.9. The results show that the Recall@20%, F1@20%, PofB@20%, PMI@20% and IFA values decrease, and the Precision@20% value increases conversely when the threshold value is increased generally. When λ is set to 0.5, the EACPDP models can achieve the best Recall@20%, F1@20% and PofB@20% values with the acceptable IFA.

Our contributions can be summarised as follows.[1]

- We perform a comprehensive empirical study to explore the practical benefits of data filtering and transfer learning methods with four commonly used defect density calculation strategies for the first time.
- We use three classification evaluation metrics and seven effort-aware evaluation metrics on 11 datasets from different projects in the PROMISE corpus to comprehensively evaluate these methods and provide some implications to researchers and practitioners.
- We do a comprehensive literature review of 32 EADP studies to identify four widely used defect density calculation strategies. Researchers can utilise the set as a starting point to conduct subsequent EADP studies.
- We make the source code and dataset of our empirical study publicly available to facilitate the replication of our work and conduct future research.

1.3 | Organisation

The remainder of the paper is organised as follows: Section 2 introduces the related work on effort-aware defect prediction and cross-project defect prediction. Section 3 gives a brief description of the data filtering and transfer learning methods. Section 4 and Section 5 present the details of our experiment setup as well as the experimental results. Section 6 discusses the potential threats to validity. Section 7 gives some insights for future research from the experimental results. Section 8 draws the conclusion.

2 | RELATED WORK

2.1 | Effort-aware defect prediction

We review the relevant articles published between 2010 and 2022to explore the development of EADP. To the best of our knowledge, the first two EADP articles with the titles containing 'effort-aware' and 'defect prediction/bug prediction' were presented by Mende et al. [13] at the 14th European Conference on Software Maintenance and Reengineering (CSMR 2010), and Kamei et al. [41] at the 26th IEEE International Conference on Software Maintenance (ICSM 2010). In order to refine the search process throughout the EADP study, we set the search start time to 2010. The search criteria for this study includes accessible English articles and only the journal version if the article contains both conference and journal versions. Moreover, we only select articles where the SDP scenario focuses on ranking software modules by defect density. The reasons are as follows[2]:

(1) Some researchers (e.g. Xia et al. [19], Yang et al. [42] and Wang et al. [43]) have evaluated their proposed SDP

[1] https://github.com/AIForSys/EACPDP

[2] Cost-effectiveness is also called Recall@20%, which represents the percentage of the actual defective

approach using not only many classification-based evaluation metrics (e.g. Precision, Recall, and F1), but also a few effort-aware evaluation metrics (e.g. cost-effectiveness modules found in the top 20% LOC.). However, the main goal of the proposed method is to correctly classify the defect-proneness of software modules rather than rank modules.

(2) It is difficult and impractical to find all SDP articles that use cost-effectiveness as an evaluation metric. Therefore, we only select articles where the SDP scenario focuses on ranking software modules.

We search for related articles using Google Scholar, IEEExplore, and ACM Digital Library, among others. First, inspired by Zhou et al.'s [44] method, we perform a forward snowball search by recursively examining citations from Mende et al.'s CSMR paper [13] and Kamei et al.'s ICSM paper [41]. Specifically, we first search and find all articles citing these two papers using Google Scholar, IEEExplore, and ACM Digital Library, among others. Then, we screen out irrelevant studies and repeat this operation. Eventually, we identify 32 papers with a significant impact in the field of EADP.

Mende et al. [13] applied 'effort-aware' to defect prediction for the first time. Kamei et al. [41] further studied the performance of the classification algorithms for EADP, including linear models, regression trees and random forests. Kamei et al. [17] utilised linear regression to build the EADP model to help developers review defects more efficiently with a fixed inspection budget. Yang et al. [45] and Ma et al. [46] argued that slice-based cohesion metrics and network measures were of practical value in the context of EADP, respectively. Subsequently, Yang et al. [20] proposed an unsupervised model called ManualUp for Just-In-Time (JIT) EADP. The results showed that ManualUp was more effective than some supervised methods in Recall@20%. Panichella et al. [47] and Yang et al. [48] proposed using genetic algorithms to train EADP models and validated their effectiveness. Yang et al. [49] studied the relationship between functional-level dependency clusters and software quality for EADP. Muthukumaran et al. [50] took the joint distribution of metrics and transformed them into aggregated scores that can be expressed as probabilities to build an aggregated EADP model. Bennin et al. [51] and Yu et al. [52] investigated the best EADP algorithms, and then Bennin et al. [53] evaluated the impact of data re-sampling methods on EADP. Yan et al. [54] compared the effectiveness of ManualUp and supervised prediction models on file-level EADP. Fu et al. [55] proposed a supervised effort-aware JIT defect prediction model named OneWay, which was based on the ManualUp method proposed by Yang et al. [49]. OneWay first evaluates the unsupervised models from ManualUp on labelled training data and then selects the one with the best cost-effectiveness for prioritising the changes in testing data. The experiments conducted by Fu et al. [55] showed that OneWay outperforms a majority of unsupervised models in effort-aware metrics. Liu et al. [56] used code churn to build an unsupervised JIT EADP model called CCUM. Subsequently, Miletic et al. [57] explored the applicability of cross-release code churn for identifying critical design changes and predicting defects in software version iterations. Chen et al. [58] used a multi-objective optimisation algorithm to construct the JIT EADP model. Huang et al. [16, 59] reviewed the works of Kamei et al. [17] and Yang et al. [20] and pointed out that the value of Precision@20% could be low and the Proportion of Module Inspected (PMI@20%) and IFA values could be very high according to the recommended ranking of ManualUp when inspecting the top 20% LOC. Therefore, they proposed the two supervised models (i.e. CBS and CBS+) and verified the effectiveness of these two models on effort-aware metrics. Experiments show that both methods outperform the unsupervised method ManualUp in terms of Precision@20%, PMI@20% and IFA. Guo et al. [60] bridged the gap between effort-aware performance and high classification ability by revisiting the JIT EADP models. Qiao et al. [61] proposed a deep learning-based JIT EADP method, which utilised neural networks to select useful features for defect prediction. Qu et al. [62] and Du et al. [63] analysed software defect distributions using k-core decomposition on class-dependent networks and proposed a top-core equation to improve the EADP model. In order to overcome the lack of training data, Zhang et al. [64] proposed a semi-supervised model based on sample selection for JIT EADP. Fan et al. [65] investigated the effects of mislabelling variation of different SZZ variants on the performance and interpretation of JIT EADP models. Ulan et al. [66] proposed an unsupervised EADP method based on weighted metric aggregation. Carka et al. [67] proposed to evaluate the EADP performance using normalised PofB, which ranked software modules based on predicted defect densities. Zhao et al. [21], Xu et al. [68] and Cheng et al. [22] proposed three JIT EADP methods for Android applications. Ni et al. [28] proposed a supervised EACPDP method called EASC. The difference with CBS+ [16] is that EASC used naive Bayes as the underlying classifier. However, EASC does not take into account differences in data distribution between within-project and cross-project data and builds models using all cross-project data, which can potentially impact the performance. In addition, the inconsistency of the strategies of defect density between EASC and four compared methods may result in unfair comparisons. These retrieved papers mainly used the four defect density strategies of defect density: Label/LOC, Prob/LOC, CBS+, and Prob. The detailed introductions of the four strategies are presented in Section 3.1. Table 1 provides an overview of the EADP studies.

2.2 | Cross-project defect prediction

In practice, it is hard to build reliable SDP models for a new project due to the lack of historical data. Earlier, researchers suggested using cross-project modules to train SDP models and predicting defects in within-project modules and called this approach CPDP. Briand et al. [69] first proposed to use historical software projects of other systems to build SDP models and explored the applicability of CPDP in object-oriented software projects. Another CPDP study on 622 cross-project pairs by Zimmermann et al. [34] used a logistic

TABLE 1 The overview of the EADP studies with the four defect density calculation strategies.

Defect density strategies	Studies
Label/LOC	Mende et al. 2010 [13]
	Kamei et al. 2010 [41]
	Kamei et al. 2012 [17]
	Panichella et al. 2016 [47]
	Yang et al. 2016 [49]
	Bennin et al. 2016 [51]
	Bennin et al. 2016 [53]
	Muthukumaran et al. 2016 [50]
	Yang et al. 2016 [20]
	Yan et al. 2017 [54]
	Liu et al. 2017 [56]
	Fu et al. 2017 [55]
	Miletic et al. 2018 [57]
	Chen et al. 2018 [58]
	Guo et al. 2018 [60]
	Yang et al. 2021 [48]
Prob/LOC	Yang et al. 2015 [45]
	Ma et al. 2016 [46]
	Guo et al. 2018 [60]
	Qu et al. 2019 [62]
	Qiao et al. 2019 [61]
	Fan et al. 2019 [65]
	Zhang et al. 2019 [64]
	Yu et al. 2019 [52]
	Du et al. 2022 [63]
	Carka et al. 2022 [67]
CBS+	Huang et al. 2019 [16]
	Ni et al. 2020 [28]
	Zhao et al. 2020 [21]
	Xu et al. 2021 [68]
	Zhao et al. 2022 [21]
	Cheng et al. 2022 [22]
Prob	Guo et al. 2018 [60]
	Ni et al. 2020 [28]
	Ulan et al. 2021 [66]

Abbreviations: CBS+, classification before sorting, EADP, effort-aware defect prediction; LOC, lines of code.

regression classifier to build CPDP models. The experimental results of the two studies showed the poor performance of CPDP models, and CPDP is still a severe challenge. The main reason is that the cross-project and within-project modules have different data distributions. Therefore, subsequent CPDP studies [70, 71] aim to alleviate the problem in the two mainstream ways, that is, data filtering and transfer learning.

2.2.1 | Cross-project defect prediction based on data filtering

The data filtering methods aim to find similar modules based on a defined similarity index from cross-projects as the training data to build a better predictor of the within-project. According to Turhan et al.'s [35] study, the CPDP performance will be low if all available cross-projects are used to build the model. Therefore, they propose the Burak filter to select some valuable modules from cross-projects that are similar to within-project modules. Subsequently, Peters et al. [72] proposed the Peter filter. It first labels each module in cross-projects with the closest within-project module, and then each within-project module selects its closest cross-project module with the label to build the filtered cross-project data. Kawata et al. [73] proposed a density-based spatial clustering guided data filter, and Yu et al. [74] proposed an agglomerative clustering guided data filter. The two methods assume that the cross-project modules that are in the same clusters as within-project modules are the most valuable training data. Hosseini et al. [75] further embedded a genetic algorithm in the Burak filter to generate an evolving training dataset to improve CPDP. Bin et al. [76] explored the impact of nine data filtering methods for CPDP and concluded that using all cross-project modules as the training data could achieve better performance. Therefore, the subsequent CPDP studies of data filtering are limited. However, little is known about whether data filtering methods can enhance the performance of EACPDP models, which drives us to investigate the problem in the study.

2.2.2 | Cross-project defect prediction based on transfer learning

The transfer learning methods mainly borrow data or knowledge from cross-projects to facilitate the CPDP model building at the within-project. Ma et al. [77] used the gravitational gravity to assign higher weights for more relevant cross-project modules and then built a weighted naive Bayes model. Nam et al. [78] proposed the TCA + method by adding decision rules to select an appropriate normalisation method for preprocessing in Transfer Component Analysis (TCA), and verified the effectiveness of TCA and TCA + on CPDP. Xia et al. [19] proposed using genetic algorithms and ensemble learning to build a compositional model for CPDP. Jing et al. [79], Limsettho et al. [80], Wu et al. [81], Gong et al. [82], Xu et al. [83] and Li et al. [84] considered both the data distribution differences and the class imbalance problem of cross-project data, and applied some transfer learning (e.g. Semi-Supervised Transfer Component Analysis (SSTCA), Joint Distribution Adaptation (JDA), TCA and Joint Probabilistic Domain Adaptation (JPDA)) and imbalanced learning (e.g. Improved Subclass Discriminant

Analysis (ISDA), Synthetic Minority Oversampling Technique (SMOTE) and STratification embedded in Nearest Neighbour (STr-NN)) methods to solve the two problems. Xu et al. [68] applied the Balanced Distribution Adaption (BDA) method to CPDP, which not only considered the marginal and conditional distributions, but also assigned different weights to them, adaptively. Subsequently, Omondiagerbe et al. [85] proposed the Weighted-BDA + method to improve CPDP performance, which incorporated the ratio of within-project and cross-project modules into the BDA model learning process and adjusted the weights of the marginal and conditional distributions. Wu et al. [81] proposed a cost-sensitive kernelised semi-supervised dictionary learning method for CPDP, which can use limited labelled defective modules and lots of unlabelled modules in the kernel space and consider the misclassification cost during dictionary learning. Zou et al. [26] proposed a CPDP method called CFIW-TNB, which took into consideration instance-transfer learning and feature-transfer learning. Li et al. [24] proposed a selection-based kernelised discriminant subspace alignment method for CPDP, which took into account data distribution differences, non-linearity, and discriminant problems at the same time. Jin et al. [86] employed the kernel twin support vector machines based on domain adaptation to reduce the distribution difference and used a particle swarm optimisation algorithm to obtain the Support Vector Machine (SVM) parameters. Huang et al. [1] proposed a CPDP method based on multi-adaptation and nuclear norms to alleviate the data distribution differences between cross-project and within-project modules.

3 | PRELIMINARIES

3.1 | Defect density calculation methods

(1) The first defect density strategy is the ratio between *Label*(m) and *LOC*(m):

$$Density\_1(m) = \frac{Label(m)}{LOC(m)}, \tag{1}$$

where m refers to the module called m, *Label*(m) represents the predicted class label of the module m (1 or 0, 1 means defective, and 0 means clean), and *LOC*(m) is the LOC of the module m.

(2) The second defect density strategy is the ratio between Prob(m) and *LOC*(m):

$$Density\_2(m) = \frac{\text{Prob}(m)}{LOC(m)}, \tag{2}$$

where Prob(m) represents the predicted defective probability of the module m.

(3) The third defect density strategy is the CBS+ (Classification Before Sort) method proposed by Huang et al. [16]. CBS+ separates the modules into two groups, that is, defective group (the predicted defective probabilities of the modules in the group is greater than or equal to 0.5) and clean group (the predicted defective probabilities of the modules in the group is less than 0.5). Then, software testers first inspect the predicted defective modules in the defective group according to the defect density calculated in Formula (3). Then, the predicted clean modules in the clean group are inspected according to the defect density calculated in Formula (3), if there is still testing resource left.

Therefore, in Formula (3) we subtract one from the defective probability of predicted clean software modules to ensure that the predicted defective modules are ranked before the predicted clean ones and can be detected preferentially.

$$Density\_3(m) = \begin{cases} \dfrac{\text{Prob}(m)}{LOC(m)}, & \text{Prob}(m) \geq 0.5 \\ \dfrac{\text{Prob}(m) - 1}{LOC(m)}, & \text{Prob}(m) < 0.5. \end{cases} \tag{3}$$

(4) The fourth defect density strategy directly uses the predicted defective probability of the software module m as the defect density:

$$Density\_4(m) = \text{Prob}(m). \tag{4}$$

For example, suppose there are five software modules (m_1, m_2, m_3, m_4 and m_5) with 50, 100, 150, 50 and 150 LOC, respectively. A classifier predicts the defective probabilities of these five modules as 0.4, 0.6, 0.8, 0.3 and 0.9, and the class labels as 0, 1, 1, 0 and 1. The ranking of the five software modules according to the four strategies of defect density is shown in Figure 1.

3.2 | Data filtering methods

(1) **Burak Filter** (BF) [35] assumes that the cross-project modules closest to the within-project modules are the most useful modules. The specific process of BF is as follows: (a) For each within-project module, BF chooses its k nearest neighbours among all cross-project modules based on the distance measurement in the Euclidean space. (b) BF merges these neighbours into a duplicate-free training dataset.

(2) **Peters Filter** (PF) [72] also assumes that the cross-project modules closest to the within-project modules are the most valuable modules. However, PF and BF are different in finding cross-project modules: (a) PF combines within-project modules and cross-project modules to get a combined set. (b) It uses the K-Means clustering algorithm to cluster the combined dataset to obtain k distinct clusters. (c) The clusters containing at least one within-project module are preserved and others are abandoned. (d) For each within-project module, PF finds the closest module

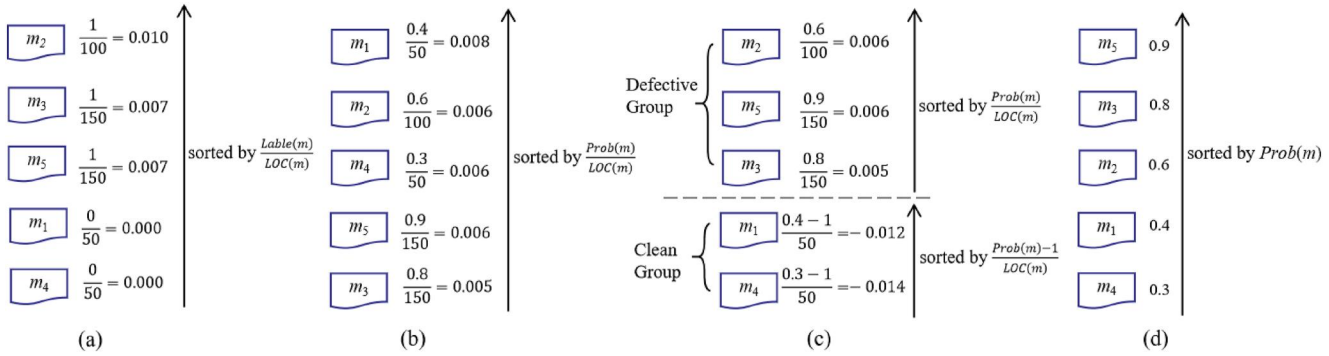


FIGURE 1 The ranking of the five software modules according to the four defect density calculation strategies. (a) Sorted by Density_1. (b) Sorted by Density_2. (c) Sorted by Density_3. (d) Sorted by Density_4.

from the cross-project data in the same cluster. (e) PF combines these modules without duplicates as the training data.

(3) **Kawata Filter** (KF) [73] assumes that the cross-project modules in the same clusters as the within-project modules are the most useful. The specific process of KF is as follows: (a) KF combines all cross-project modules and all within-project modules and employs the DBSCAN clustering algorithm to group these modules. (b) It retains the clusters containing at least one within-project module and selects all cross-project modules in the clusters as the training data.

(4) **Data Filtering based on Agglomerative Clustering** (DFAC) [74] also assumes that the cross-project modules in the same clusters as the within-project modules are the most valuable. The difference with KF is that DFAC employs the agglomerative clustering algorithm to solve the randomness problem caused by setting a large number of parameters of the DBSCAN algorithm.

3.3 | Transfer learning methods

(1) **Transfer Naive Bayes** (TNB) [77] efficiently assigns weights to the cross-project modules so that the cross-project module distribution is close to the distribution of within-project modules. Specifically, (a) TNB fetches the scope of features in each dimension of the within-project. (b) The features of each cross-project module are compared with the within-project feature scope to calculate the number of similar attributes. (c) TNB analogy to the gravitational formula, using the number of similar attributes to calculate each cross-project module weight. (d) The weighted naive Bayes model is build based on the weighted cross-project modules.

(2) **Transfer Component Analysis** (TCA) [87] aims to find a latent feature space so that the marginal distributions between different projects after transferring are similar. Therefore, TCA maps the within-project and cross-project modules together into a high-dimensional Regenerative Kernel Hilbert Space (RKHS), where the distribution distance between the within-project and cross-project modules is minimised and their respective internal properties are preserved. In addition, TCA uses Maximum Mean Discrepancy (MMD) to learn the transfer components across projects in the RKHS space.

(3) **Joint Distribution Adaptation** (JDA) [88] additionally adds the conditional distribution between different projects to optimisation objective, compared with TCA only considering the marginal distribution. In order to calculate the conditional distribution, JDA first trains a classifier to predict and optimise pseudo labels of within-project modules. Then, JDA iteratively refines their pseudo labels until they are stabilised and finally constructs the transferred feature representation of the cross-project and within-project modules.

(4) **Balanced Distribution Adaption** (BDA) [89] takes into account the marginal and conditional distributions of the different projects and automatically assigns weights to both distributions to obtain the best transfer components. The JDA and BDA methods suggest optimising the marginal and conditional distributions, but the BDA method notices the difference in importance between the two distinct distributions. Therefore, BDA tries to find the balance factor, which can adaptively adjust the importance of those two distributions. Specifically, BDA continuously corrects the balance factor through iterations. Then, BDA constructs the transferred feature representation of the cross-project and within-project modules after finding the optimal balance factor. In particular, when the balance factor is around 0.5, BDA considers the two distributions equally important, and BDA is similar to the JDA method.

(5) **Joint Probability Domain Adaptation** (JPDA) [90] considers the differences in joint probability distributions between different projects and the distinguishability between different classes. From a Bayesian theorem perspective, JPDA suggests computing the discrepancy of joint probability distribution between cross-project and within-project modules rather than the sum of the marginal and conditional distribution. Specifically, JPDA improves the transferability between different projects by minimising the MMD of the same class and the distinguishability between different classes by maximising the MMD of different classes.

4 | EXPERIMENTAL SETUP

4.1 | Datasets

To meet the requirements of the EADP task for the number of defects, we select 11 software project datasets from the PROMISE data repository [91] in this paper. The selection of our experimental dataset differs from that of Ni et al.'s [28] as they did not consider the information on the number of defects and chose the datasets without the information about the number of defects, such as AEEEM [29], NASA [30, 31] and RELINK [33]. However, the goal of EADP is to predict defect density (i.e. the ratio between the number of defects and LOC) and rank software modules based on the density. Therefore, we only utilise the PROMISE dataset which comprises information on the number of defects as our experimental dataset. Each module in the dataset contains 20 mutually exclusive code features and one numeric attribute that indicates the number of defects in the module. The experimental projects come from different application domains and have different sizes (i.e. containing 205 to 965 modules) and different percentages of defective modules (i.e. varying from 2.2% to 98.8%), which is conducive to improving the generalisation of our experiments. Moreover, these datasets are publicly available, so the results of this paper can be replicated to a large extent by other researchers. The description of these datasets is provided in Table 2, including the number of modules (#Module), the percentage of defective modules (%Defective), the average number of defects (AvgDefects), and the average LOC (AvgLOC).

4.2 | Evaluation metrics

4.2.1 | Effort-aware evaluation metrics

We restrict the limited effort to 20% of the total LOC of the defect dataset, similar to the previous EADP studies [16, 21, 28, 68]. Assume that there are N software modules in a defect dataset, which contain P defective modules and Q defects. When checking the top 20% LOC according to the predicted result of the EADP model, the software testing team inspects n software modules and finds p actual defective modules with q defects. In our experiments, we utilise several evaluation measures that are commonly adopted in both the software engineering [92–94] and machine learning [95–100].

Precision@20% is the ratio between the number of actual defective modules and the number of predicted defective modules in the top 20% LOC. A higher Precision@20% is significant to adopting the EADP model in practice. If the software testing team finds that many predicted defective modules do not contain defects, they may ignore all prediction results and lead to the waste of testing resources.

$$Precision@20\% = \frac{p}{n} \tag{5}$$

Recall@20% is the ratio between the number of actual defective modules found in the top 20% LOC and the number of defective modules in the dataset. A higher Recall@20% means that the software testing team can capture more defective modules.

$$Recall@20\% = \frac{p}{P} \tag{6}$$

In general, the improvement in Recall@20% comes at the expense of Precision@20%. So we further use F1@20% to comprehensively measure the EADP performance when the top 20% LOC are inspected. F1@20% takes into account both the Precision@20% and Recall@20% simultaneously, which is a harmonic average of the two.

$$F1@20\% = \frac{2 \times Precision@20\% \times Recall@20\%}{Precision@20\% + Recall@20\%} \tag{7}$$

PofB@20% is the Proportion of the found Bugs when the top 20% LOC are inspected. The higher the PofB@20%, the more defects can be detected.

$$PofB@20\% = \frac{q}{Q} \tag{8}$$

PMI@20% is the Proportion of Module Inspected when the top 20% LOC are inspected. A high PMI@20% indicates that the software testing team needs to check for more modules given the same number of LOC. Switching between different modules frequently also increases the actual effort and time cost.

$$PMI@20\% = \frac{n}{N} \tag{9}$$

PofB@20% and PMI@20% are often correlated, and when an EADP method obtains a high PofB@20%, it will also achieve a high PMI@20%.

TABLE 2 The details of the experimental datasets.

Datasets	#Module	%Defective	AvgDefects	AvgLOC
Ant 1.7	745	22.3%	2.04	280.1
Camel 1.6	965	19.5%	2.66	117.2
Ivy 2.0	352	11.4%	1.4	249.3
Jedit 4.3	492	2.2%	1.09	411.3
Log4j 1.2	205	92.2%	2.63	186.3
Lucene 2.4	340	59.7%	3.11	302.5
Poi 3.0	442	63.6%	1.78	292.6
Synapse 1.2	256	33.6%	1.69	209.0
Velocity 1.6	229	34.1%	2.44	249.0
Xalan 2.7	909	98.8%	1.35	471.5
Xerces 1.4	588	74.3%	3.65	240.1

Popt is used to evaluate how close the built prediction model is to the optimal model. The x-axis in the Alberg plot shown in Figure 2 represents the cumulative percentage of LOC, and the y-axis represents the cumulative percentage of the found defects. The prediction model sorts the software modules in descending order according to the predicted defect density; the optimal model sorts the software modules in descending order according to the actual defect density; the worst model sorts the software modules in ascending order according to the actual defect density. The *Popt* is calculated as follows:

$$Popt = 1 - \frac{Area(optimal) - Area(prediction)}{Area(optimal) - Area(worst)}, \quad (10)$$

where $Area(optimal)$ is the area under the optimal model curve, $Area(prediction)$ is the area under the prediction model curve, and $Area(worst)$ is the area under the worst model curve. The higher *Popt* value indicates that the prediction model is closer to the optimal model and has a better global ranking performance.

IFA is the number of IFA encountered before the software testing team finds the first actual defective module. Kochhar et al. [101] pointed out that the software testing team will get frustrated and stop checking other predictive defective modules if the top-k predicted defective modules by the EADP model are all false positives. In addition, their study shows that almost all respondents (close to 98%) agree that examining more than 10 actual clean modules was beyond their acceptable level.

4.2.2 | Classification evaluation metrics

In Section 5.2, we conduct a correlation analysis to explore the relationship between the effort-aware performance of the EACPDP model and the classification performance of the embedding classification model. Therefore, we employ Precision, Recall and F1 to evaluate the classification performance. The confusion matrix shown in Table 3 lists all four possible prediction results, where true positive (TP) is the number of defective modules correctly predicted as defective, false negative (FN) is the number of defective modules wrongly predicted as clean, false positive (FP) is the number of clean modules wrongly predicted as defective and true negative (TN) is the number of clean modules correctly predicted as clean.

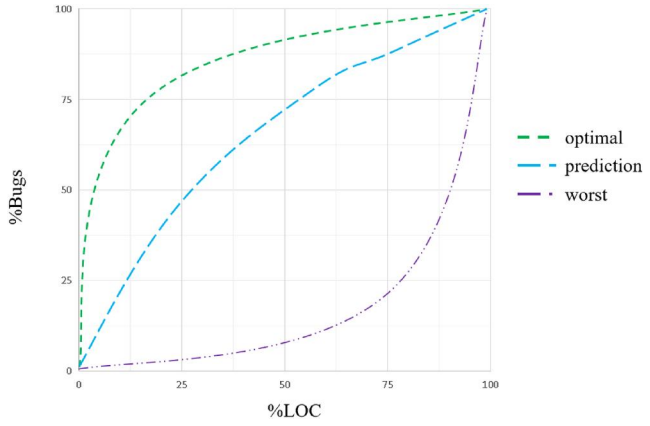


FIGURE 2 The Alberg plot.

Precision is the percentage of the modules that actually contain defects to all the predicted defective modules.

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

Recall is the percentage of the correctly predicted defective modules to all the actual defective modules in the dataset.

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

F1 is the harmonic mean of Precision and Recall, and we use the F1 to measure the model's classification performance comprehensively.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (13)$$

4.3 | Experimental process

Cross-project validation: We employ 11 software projects as the experimental datasets. It is difficult to determine which project to use as a training set to build a better model, so we choose one dataset as a within-project for testing and the remaining ten cross-projects for training at each time. The cross-project experimental setting is the same as that of Huang et al.'s [16] study. Figure 3 shows the specific process of our experiments. For example, we use Ant 1.7 as a within-project and the remaining 10 projects (i.e. Camel 1.6, Ivy 2.0, …, Xerces 1.4) as cross-projects.

(1) We use the four data filtering methods (i.e. BF, PF, KF and DFAC) to remove all irrelevant modules in the cross-projects and obtain the 10 filtered cross-projects (i.e. Camel 1.6′, Ivy 2.0′, …, Xerces 1.4′). Then, we use KNN, LR, and RF classifiers to build predictors based on the 10 filtered cross-projects. Finally, we employ the predictor to predict the class label and defective probability of the modules in Ant 1.7 and sort them according to the four calculation methods of defect density mentioned in Section 3.1.

TABLE 3 The confusion matrix.

	Predicted defective	Predicted clean
Actual defective	TP	FN
Actual clean	FP	TN

Abbreviations: FN, false negative; FP, false positive; TN, true negative; TP, true positive.

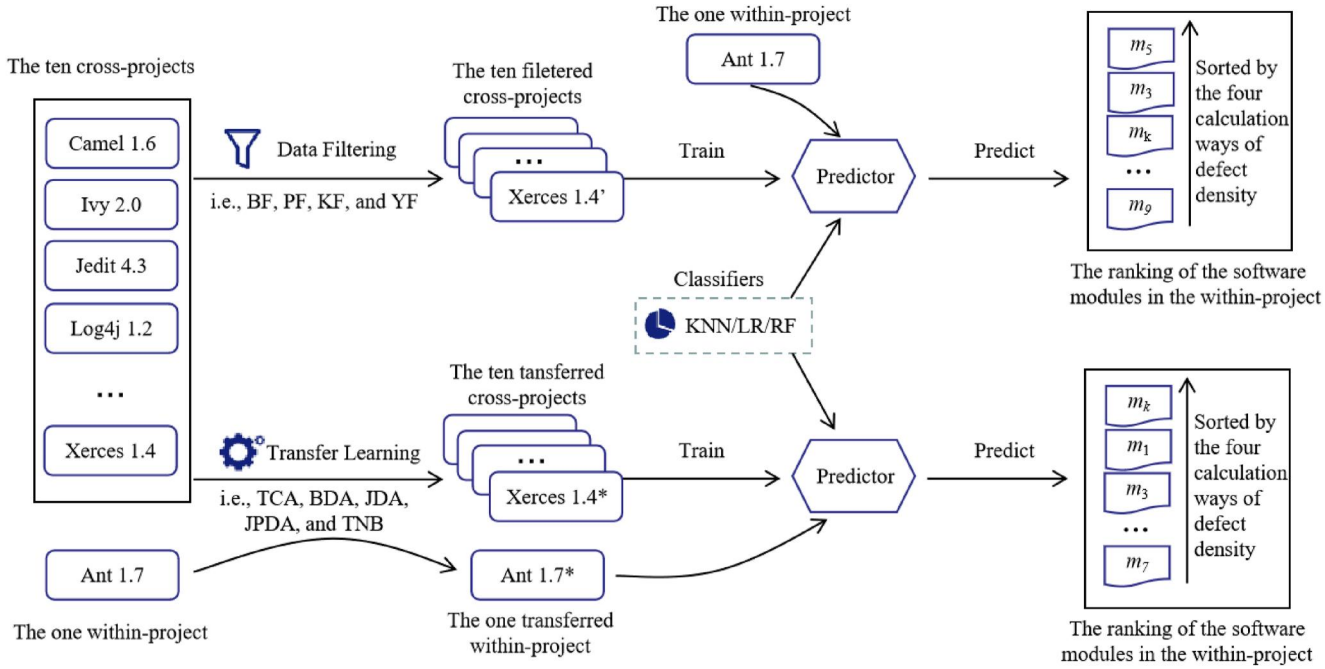


FIGURE 3 The specific process of our experiments.

(2) We apply the five transfer learning methods (i.e. TCA, BDA, JDA, JPDA and TNB) to transfer these modules' features in all the projects and obtain the 10 transferred cross-projects (i.e. Camel 1.6*, Ivy 2.0*, …, Xerces 1.4*) and the one transferred within-project (i.e. Ant 1.7*). Next, we still use the three classifiers mentioned above to build the predictor. At last, we use the predictor to sort the modules in Ant 1.7* according to the four calculation methods of defect density.

4.4 | Classifiers

The study by Ghotra et al. [102] pointed out that the choice of different binary classification algorithms has a obvious influence on the performance of SDP models. Therefore, if we build EACPDP models based on different classification techniques, we may get different results. Finally, we select the three classifiers (i.e., Logistic Regression (LR), K-Nearest Neighbour (KNN), and Random Forest (RF)), since our preliminary experiment results show that the three classifiers achieve the better performance than the widely used classifiers (including naive Bayes, support vector machine, decision tree, and multi-layer perceptron). These three classifiers were widely used in previous SDP studies [16, 28, 68, 72, 76, 83], and they belong to the three different categories, that is, regression function, instance-based algorithm, and ensemble learning.

(1) **Logistic Regression (LR)** [103] adds a non-linear mapping (Sigmoid function) to classify software modules into discrete outcomes. LR uses maximum likelihood estimation to train weights for each feature of the module, and uses the sigmoid function to map a discrete value between 0 and 1, which is used as the defective probability to classify the module.

(2) **K-Nearest Neighbour (KNN)** [104] predict the within-project module's label by the K nearest cross-project modules. KNN find the K cross-project modules that are closest to the within-project module, and use the class labels with the most occurrences as the label of the within-project module.

(3) **Random Forest (RF)** [105] generates a random forest of multiple decision trees from the cross-project by bootstrap resampling technique, and finally the prediction class with the highest votes is used as the predicted label of the within-project module.

4.5 | Statistic test

Scott-Knott ESD [39] test is a multiple comparison method that utilises hierarchical clustering to divide EACPDP methods into different groups with significant differences at the significance level of 0.05 ($\alpha = 0.05$). It ranks the EACPCP methods while ensuring the magnitude of differences is negligible for methods within the same group, and the difference is not negligible for methods between groups. In other words, there is no significant difference among the EACPDP methods in the same group, while the EACPDP methods in different groups have significant difference. For example, the Scott-Knott ESD test divides five methods into the two groups, that is, Group 1 (including methods A, D and E) and Group 2 (including methods B and C). There is no significant difference between the methods within either Group 1 or Group 2. However, the methods A, D and E in Group 1 significantly outperform the methods B and C in Group 2 according to the Scott-Knott ESD test.

Wilcoxon signed-rank [40] test is a non-parametric sample test, which is based on the ranking of the methods rather than the mean. The basic principle of hypothesis testing is the small probability principle, which states that a small probability event cannot actually occur in a single test. However, when multiple hypothesis tests are performed under the same research problem, it no longer fits the single test as stated by the small probability principle. So we also use the Benjamini–Hochberg [106] (BH) correction to control the false discovery rate. The null hypothesis is that there is no difference between two EACPDP methods. If the corrected p-value by the BH procedure is less than 0.05, we reject the null hypothesis and consider a significant difference between the EACPDP methods; otherwise, we accept the original hypothesis.

5 | EXPERIMENTAL RESULTS

5.1 | RQ1: what is the best strategy of defect density for EACPDP?

Motivation: The previous EADP studies mainly employed the four strategies of defect density, that is, Label/LOC, Prob/LOC, CBS+ and Prob. To verify which strategy performs the best, we compare the performance of the 10 methods when applying the four strategies to sort software modules.

Methods: Figures 4–10 show the performance distribution of different methods using the four strategies across all datasets in terms of Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, IFA and *Popt*. We first apply the 10 methods to the cross-project datasets and then use KNN to build the EACPDP models. The None method only uses the KNN classifier to build the model based on all cross-project data. We denote the methods as X1, X2, X3 and X4, where 'X' represents the learning method, and 1, 2, 3 and 4 represent the first, second, third and fourth strategies of defect density. The colour of each box represents the Scott-Knott ESD test ranking. From top down, the order is red, green, blue, yellow, purple, orange, pink, and grey. More specifically, the methods represented by red significantly outperforms the methods represented by other colours according to the Scott-Knott ESD test. Tables 4–10 list the medium performance values of the methods across all datasets. Due to the space limit, we only present the results with the KNN classifier. The main reasons are that KNN achieves better performance than LR and RF and the results of RQ1 with LR and RF are similar to that of KNN.

Results: The CBS+ strategy can achieve a better overall performance. The Label/LOC and Prob/LOC strategies perform poorly in terms of PMI@20% or IFA, while the Prob strategy has a poor performance in terms of Recall@20% and PofB@20%. Detailed analysis of these results are as follows:

(1) None of the methods using the **Label/LOC** strategy perform the best on all the metrics. BDA and JDA achieve the best Recall@20%, F1@20%, PofB@20%, and *Popt*. However, their medium IFA values are both 11. The previous studies [16, 101] concluded that if the first 10 software modules are false alarms, the test team will not continue to check them. Therefore, the performance of BDA and JDA on other metrics has no practical significance.
(2) All methods using the **Prob/LOC** strategy have high performance values except for Precision@20%. However, high PMI@20% and IFA values represent worse EACPDP performance. This means that the strategy has a very high overhead in frequent module switches and lots of false alarms. In particular, the medium PMI@20% value of the

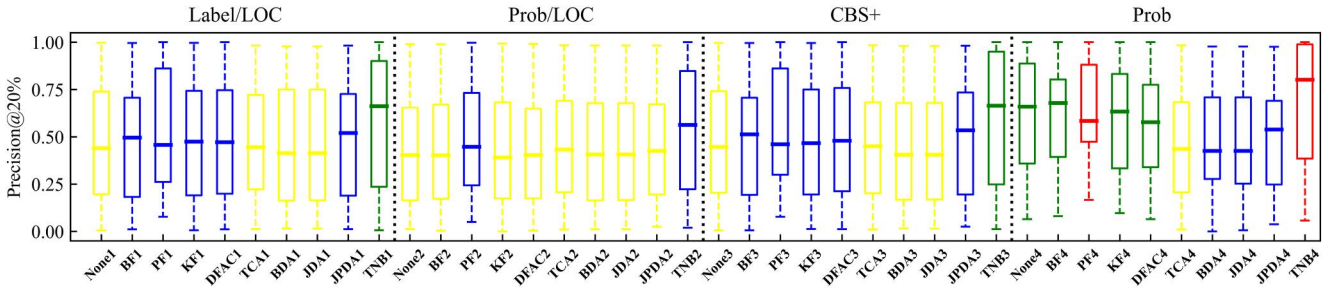


FIGURE 4 The Precision@20% of each method using the four defect density calculation strategies.

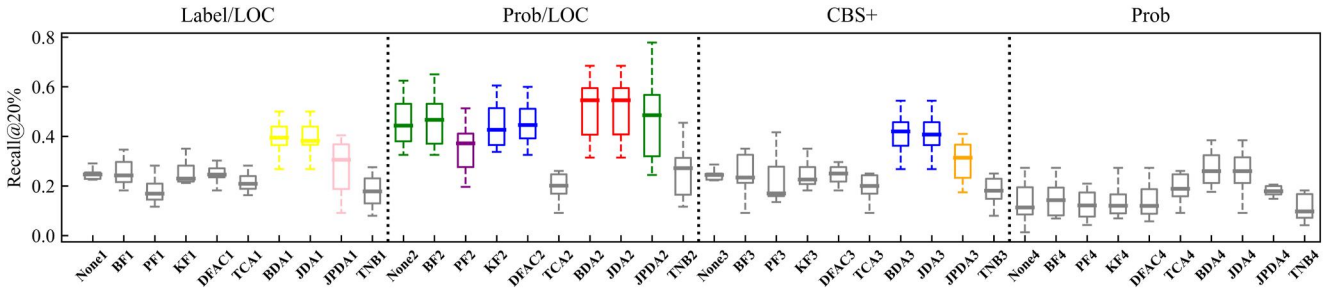


FIGURE 5 The Recall@20% of each method using the four defect density calculation strategies.

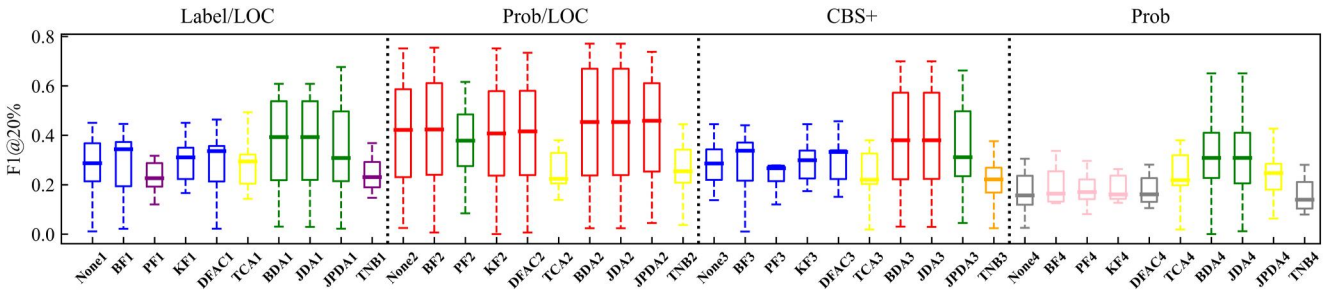


FIGURE 6 The F1@20% of each method using the four defect density calculation strategies.

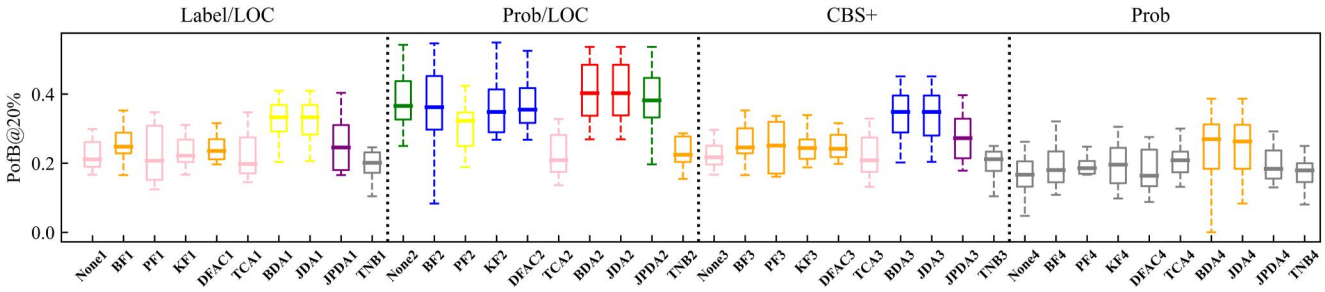


FIGURE 7 The PofB@20% of each method using the four defect density calculation strategies.

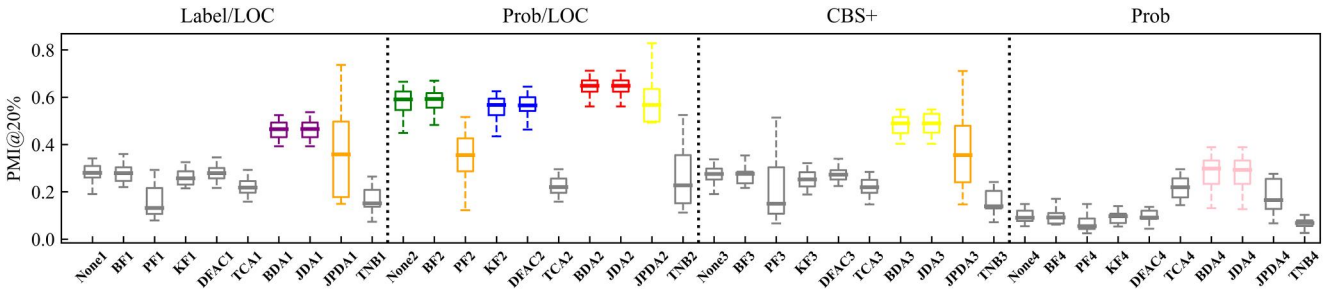


FIGURE 8 The PMI@20% of each method using the four defect density calculation strategies.

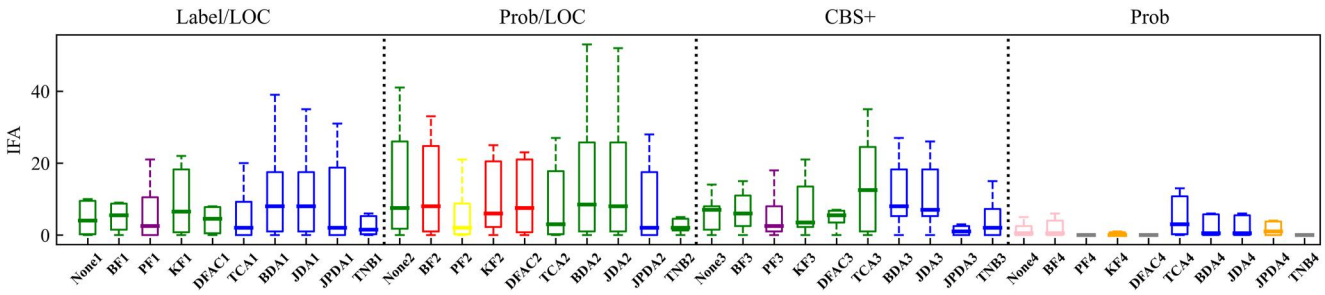


FIGURE 9 The initial false alarms (IFA) of each method using the four defect density calculation strategies.

strategy is significantly higher than that of the other ways by 8.6%–29.6%. This strategy tends to rank modules with fewer LOC first, since the modules are likely to have higher defect density. When inspecting the same amount of LOC (i.e. top 20%LOC), software testers require to check more modules (high PMI@20% value). Therefore, they have more chances to find defective modules and defects (high Recall@20% and PofB@20% values).

(3) All methods using the **Prob** strategy have low performance values except for Precision@20%. Although the PMI@20% and IFA values meet the EACPDP requirements, the strategy performs significantly worse than other ways on other important metrics (i.e. Recall@20%, F1@20% and PofB@20%). The Prob strategy ranks modules based on the predicted defect probability, and the modules with many LOC tend to be ranked first in the

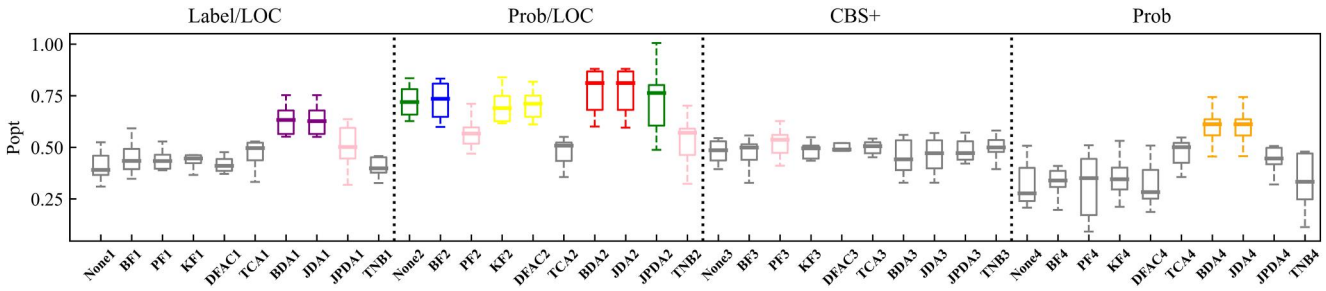


FIGURE 10 The *Popt* of each method using the four defect density calculation strategies.

TABLE 4 The median Precision@20% of each method using the four defect density calculation strategies.

	Label/LOC	Prob/LOC	CBS+	Prob
None	0.322	0.303	0.333	0.611
BF	0.409	0.303	0.422	0.618
PF	0.328	0.381	0.406	0.524
KF	0.385	0.297	0.380	0.591
DFAC	0.377	0.288	0.390	0.522
TCA	0.328	0.351	0.361	0.364
BDA	0.325	0.282	0.305	0.351
JDA	0.325	0.282	0.305	0.351
JPDA	0.476	0.333	0.500	0.471
TNB	0.600	0.511	0.606	0.714

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

TABLE 5 The median Recall@20% of each method using the four defect density calculation strategies.

	Label/LOC	Prob/LOC	CBS+	Prob
None	0.244	0.436	0.244	0.117
BF	0.242	0.447	0.229	0.122
PF	0.175	0.356	0.175	0.141
KF	0.227	0.413	0.218	0.122
DFAC	0.242	0.441	0.245	0.128
TCA	0.217	0.205	0.203	0.203
BDA	0.389	0.520	0.400	0.241
JDA	0.375	0.520	0.375	0.241
JPDA	0.256	0.473	0.273	0.176
TNB	0.179	0.267	0.182	0.104

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

TABLE 6 The median F1@20% of each method using the four defect density calculation strategies.

	Label/LOC	Prob/LOC	CBS+	Prob
None	0.284	0.381	0.281	0.158
BF	0.342	0.381	0.333	0.161
PF	0.242	0.319	0.264	0.174
KF	0.308	0.369	0.297	0.163
DFAC	0.331	0.355	0.332	0.163
TCA	0.285	0.222	0.212	0.216
BDA	0.391	0.382	0.367	0.296
JDA	0.391	0.382	0.367	0.296
JPDA	0.282	0.423	0.283	0.239
TNB	0.206	0.254	0.207	0.145

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

TABLE 7 The median PofB@20% of each method using the four defect density calculation strategies.

	Label/LOC	Prob/LOC	CBS+	Prob
None	0.200	0.340	0.205	0.167
BF	0.246	0.324	0.238	0.182
PF	0.228	0.312	0.234	0.188
KF	0.214	0.324	0.237	0.210
DFAC	0.221	0.330	0.233	0.180
TCA	0.193	0.203	0.202	0.202
BDA	0.332	0.396	0.333	0.258
JDA	0.332	0.396	0.333	0.258
JPDA	0.254	0.376	0.244	0.188
TNB	0.187	0.217	0.209	0.180

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

TABLE 8 The median PMI@20% of each method using the four defect density calculation strategies.

	Label/LOC	Prob/LOC	CBS+	Prob
None	0.288	0.576	0.284	0.093
BF	0.285	0.576	0.279	0.100
PF	0.139	0.359	0.166	0.060
KF	0.270	0.566	0.267	0.098
DFAC	0.291	0.555	0.283	0.093
TCA	0.222	0.234	0.234	0.234
BDA	0.468	0.637	0.494	0.278
JDA	0.470	0.637	0.494	0.278
JPDA	0.351	0.544	0.351	0.151
TNB	0.163	0.249	0.140	0.068

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

TABLE 9 The median IFA of each method using the four defect density calculation strategies.

	Label/LOC	Prob/LOC	CBS+	Prob
None	5	10	8	1
BF	7	8	7	0
PF	3	3	1	0
KF	9	7	4	0
DFAC	6	8	6	0
TCA	3	5	16	3
BDA	11	8	10	1
JDA	11	7	10	1
JPDA	2	3	1	1
TNB	2	2	3	0

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

testing sequence. When inspecting the same amount of LOC (i.e. top 20%LOC), software testers require to check fewer modules (low PMI@20% value). Therefore, the Recall@20% and PofB@20% values are also low accordingly, since fewer inspected modules bring fewer opportunities to find defective modules and defects.

(4) The **CBS+** calculation strategy loses to the Prob strategy but significantly outperforms Prob/LOC for a large portion of the methods in terms of Precision@20%. Except for the TCA, BDA, and JDA methods, the CBS+ strategy performs similarly to Label/LOC and significantly outperforms the Prob strategy on Recall@20% and F1@20%. Compared with Label/LOC and Prob, CBS+ has a significant improvement on PofB@20%. In terms of PMI@20%, CBS+ strategy and Label/LOC are not significantly different, but they are significantly better than Prob/LOC strategy on the methods except BDA and JDA. The medium IFA values of all the methods except TCA using the CBS+ strategy are controlled within 10. In terms of *Popt*, the CBS + strategy outperforms the Label/LOC and Prob strategies in all methods except BDA, JDA and JPDA. The CBS+ strategy first groups modules according to their defect probability, then ranks the defective group in front of the clean group. The modules ranked at the top are more likely to be defective and have relatively more LOC. Therefore, the CBS+ strategy has better PMI@20% or IFA performance than Label/LOC and Prob/LOC.

(5) It is worth noting that BDA and JDA always achieve the best performance among the methods in terms of Recall@20%, F1@20%, PofB@20% and *Popt* under all four defect density strategies.

TABLE 10 The median *Popt* of each method using the four defect density calculation strategies.

	Label/LOC	Prob/LOC	CBS+	Prob
None	0.399	0.707	0.482	0.288
BF	0.433	0.735	0.499	0.355
PF	0.439	0.575	0.534	0.378
KF	0.451	0.685	0.494	0.357
DFAC	0.416	0.711	0.484	0.292
TCA	0.502	0.508	0.505	0.505
BDA	0.624	0.791	0.448	0.611
JDA	0.612	0.791	0.464	0.611
JPDA	0.488	0.762	0.462	0.433
TNB	0.400	0.572	0.495	0.346

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

However, the implementation of Label/LOC and Prob/LOC strategies produces unsatisfactory PMI@20% or IFA results for these two methods. Therefore, the CBS+ strategy is the most suitable for EACPDP under overall consideration.

Answer to RQ1

The CBS+ strategy can achieve the relatively high Recall@20%, F1@20%, PofB@20%, and *Popt* values with the acceptable IFA.

5.2 | RQ2: can data filtering and transfer learning approaches improve the EACPDP performance?

Motivation: Ni et al.'s study [28] used naive Bayes to model EASC without taking into account the data distribution difference. Therefore, we compare four data filtering methods and five transfer learning methods with the EASC method in order to verify whether these methods can alleviate the data difference distribution problem to improve the performance of EACPDP further.

Methods: We apply the four data filtering methods and the five transfer learning methods to the cross-project datasets, and then use KNN, LR and RF classifiers to build the EACPDP models, respectively. Tables 11–13 and Figures 11–13 show the performance differences between the data filtering and transfer learning methods and None with these three classifiers using the CBS+ strategy. We employ the Wilcoxon signed-rank test and draw boxplots to examine the significant differences, where red boxes indicate that the corresponding method is significantly different from None and black boxes indicate no significant difference between the corresponding method and None. Similarly, in Tables 11–13 we bold the results of the Wilcoxon signed-rank test to indicate that significant difference between the corresponding method and None.

Results: All data filtering methods have weak performance improvement or even degradation in classification and effort-aware scenarios, and the transfer learning methods (i.e. BDA and JDA) can significantly improve the performance of EACPDP in terms of Recall, Recall@20% and PofB@20%. We analyse the data filtering and transfer learning methods in detail, and the results are shown below.

(1) Almost all data filtering methods do not differ significantly from None on the evaluation metrics, and some methods have degraded the performance. The PF with the KNN classifier has a significant decrease of 20.7% on Recall. The KF with the LR classifier has a significant decrease of 4.1% on Precision and 3.5% on PMI@20%. The PF of the RF classifier decreases significantly by 22.9% on Recall. However, there are also some data filtering methods that slightly improve the performance. For example, DFAC with the KNN classifier improve the Precision value by 0.9%, but the improvement is not statistically significant.

TABLE 11 The relative improvement of each method with the KNN classifier using the CBS+ strategy.

	Precision	Recall	F1	Precision@20%	Recall@20%	F1@20%	PofB@20%	PMI@20%	Popt	IFA
BF	**0.003**	0.032	0.019	0.007	0.004	0.007	0.008	−0.002	0.017	0
PF	0.003	**−0.207**	−0.233	0.055	−0.011	0.003	0.047	−0.037	0.004	0
KF	0.012	0.005	0.008	0.02	0	0	0.014	−0.019	0.004	−1
DFAC	**0.009**	0.02	0.008	0.009	0.01	0.011	0.008	−0.005	0.009	−1
TCA	−0.019	0.064	0.018	0.004	−0.005	−0.007	0.003	−0.034	−0.003	8
BDA	−0.004	0	0.002	−0.016	**0.15**	0.061	**0.116**	**0.193**	−0.003	5
JDA	**0.03**	0.221	0.138	−0.016	**0.143**	0.061	**0.089**	**0.195**	−0.003	4
JPDA	0.006	0.084	0.034	−0.002	0.069	0.034	0.022	0.076	0.011	−2
TNB	**0.081**	−0.103	−0.002	**0.133**	−0.032	−0.007	−0.002	**−0.117**	0.002	−1

Note: Bold values indicate the results of the Wilcoxon signed-rank test to indicate that significant difference between the corresponding method and None.

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

TABLE 12 The relative improvement of each method with the LR classifier using the CBS+ strategy.

	Precision	Recall	F1	Precision@20%	Recall@20%	F1@20%	PofB@20%	PMI@20%	Popt	IFA
BF	0.001	0	0.01	0	0.009	0.014	0.012	0.008	−0.005	0
PF	0.021	−0.125	−0.139	0.024	−0.032	−0.045	0	−0.079	0.005	0
KF	**−0.041**	0.077	0.02	−0.045	0.025	0.01	0	**0.035**	−0.004	1
DFAC	0	0.008	0.01	−0.001	0.006	0.001	0	0.009	−0.01	0
TCA	**−0.104**	−0.053	**−0.084**	**−0.116**	**0.276**	0.031	**0.161**	**0.371**	0	1
BDA	**−0.122**	**0.537**	0.024	−0.027	0.098	0.04	0.06	**0.16**	−0.005	2
JDA	**−0.131**	**0.537**	0.024	−0.027	0.098	0.04	0.06	**0.16**	−0.005	2
JPDA	**−0.585**	−0.259	**−0.395**	**−0.113**	**0.182**	0.015	**0.166**	**0.403**	0.063	9
TNB	0.012	−0.103	−0.112	0.014	−0.048	−0.06	0	−0.035	−0.004	−1

Note: Bold values indicate the results of the Wilcoxon signed-rank test to indicate that significant difference between the corresponding method and None.

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

TABLE 13 The relative improvement of each method with the RF classifier using the CBS+ strategy.

	Precision	Recall	F1	Precision@20%	Recall@20%	F1@20%	PofB@20%	PMI@20%	Popt	IFA
BF	−0.001	0	0	−0.005	0	−0.001	−0.006	0.007	−0.004	0
PF	0.081	**−0.229**	−0.082	0.079	−0.075	−0.026	0.007	−0.154	−0.008	−5
KF	0.005	−0.005	0	0.001	−0.013	−0.015	−0.022	−0.01	−0.008	0
DFAC	0.012	0	0.009	0.005	−0.016	−0.003	0	−0.008	0.002	0
TCA	−0.005	0.004	−0.001	−0.021	**0.222**	0.037	**0.126**	**0.265**	−0.015	0
BDA	−0.025	0.09	0.001	**−0.037**	**0.269**	0.023	**0.158**	**0.331**	0.039	0
JDA	−0.044	0.111	−0.006	**−0.029**	**0.207**	0.022	**0.112**	**0.29**	0.012	0
JPDA	−0.007	−0.02	−0.009	−0.036	**0.16**	0.042	**0.12**	**0.304**	0.012	4
TNB	**0.043**	−0.127	−0.146	0.039	−0.072	−0.104	−0.04	**−0.09**	−0.01	0

Note: Bold values indicate the results of the Wilcoxon signed-rank test to indicate that significant difference between the corresponding method and None.

Abbreviations: BDA, balanced distribution adaption; BF, Burak Filter; CBS+, classification before sorting; DFAC, data filtering based on agglomerative clustering; JDA, joint distribution adaptation; JPDA, joint probability domain adaptation; KF, Kawata Filter; LOC, lines of code; PF, peters filter; TCA, transfer component analysis; TNB, transfer Naive Bayes.

(2) BDA and JDA with the KNN classifier show significant improvements on **Recall@20%** and **PofB@20%**, with BDA improving 15% on Recall@20% and 11.6% on PofB@20%, and JDA improving 14.3% and 8.9%, respectively. When embedding the LR classifier, TCA, BDA, JDA and JPDA increase the Recall@20% value by 27.6%, 9.8%, 9.8%, and 18.2% and the PofB@20% value by 16.1%, 6.0%, 6.0% and 16.6%, respectively. Moreover, the improvements of TCA and JPDA are statistically significant.

When embedding the RF classifier, TCA, BDA, JDA and JPDA significantly increase the Recall@20% value by 22.2%, 26.9%, 20.7% and 16%, and the PofB@20% value by 12.6%, 15.8%, 11.2% and 12%, respectively. In addition, we find that the BDA and JDA methods achieve the best performance no matter which of the three classifiers is used.

(3) JDA and TNB significantly improve the **Precision** value by 3% and 8% when embedding the KNN classifier. But TCA, BDA, JDA and JPDA significantly decrease the Precision value by 10.4%, 12.2%, 13.1% and 58.5% when embedding the LR classifier. TNB significantly improves the Precision value by 4.3% when embedding the RF classifier. The **Recall** values of BDA and JDA are both significantly improved by 53.7% when embedding the LR classifier. TCA and JPDA embedding the LR classifier significantly decrease the **F1** value by 8.4%, and 39.5%, respectively. Other transfer learning methods have no significant difference from None in terms of the three evaluation metrics.
(4) TNB significantly improves the **Precision@20%** value by 13.3% when embedding the KNN classifier. The Precision@20% values of TCA and JPDA are significantly decreased by 11.6% and 11.3% when embedding the LR classifier. BDA and JDA significantly decrease the Precision@20% values by 3.7% and 2.9% under the RF classifier.

The **PMI@20%** values of BDA and JDA are significantly improved by 19.3% and 19.5% under the KNN classifier, by 16% and 16.0% under the LR classifier, and by 33.1% and 29% under the RF classifier, respectively. In terms of **Popt** and **IFA**, all methods do not differ significantly from None when embedding the three classifiers.

Answer to RQ2

The BDA and JDA methods with the KNN classifier to build EACPDP models can significantly improve the Recall@20% and PofB@20% values while achieving the acceptable IFA.

5.3 | RQ3: what is the relationship among the performance measures?

Motivations: Since we first employ BDA and JDA to build the classification models, and then use the CBS+ strategy to rank software modules. We wonder whether better classification performance can contribute to building better EACPDP models. In our study, we employ three classification evaluation metrics and seven effort-aware evaluation metrics. Therefore, we perform the correlation analysis to explore the relationship between the classification performance and the effort-aware performance.

Methods: We employ the Kendell correlation coefficient (i.e. r) to evaluate the correlation among the evaluation metrics and use a heat map to show the results, as shown in Figure 14. The heavier the colour, the stronger the correlation between every two evaluation metrics. According to the study presented by Hinkle et al. [107], the correlation coefficient is considered negligible ($|r| < 0.3$), low ($0.3 \leq |r| < 0.5$), moderate ($0.5 \leq |r| < 0.7$), high ($0.7 \leq |r| < 0.9$), and very high ($0.9 \leq |r| \leq 1$).

Results:

(1) Precision has a very high correlation with Precision@20% (0.93, 0.82, 0.89, 0.89, 0.82, and 0.89, respectively) and with

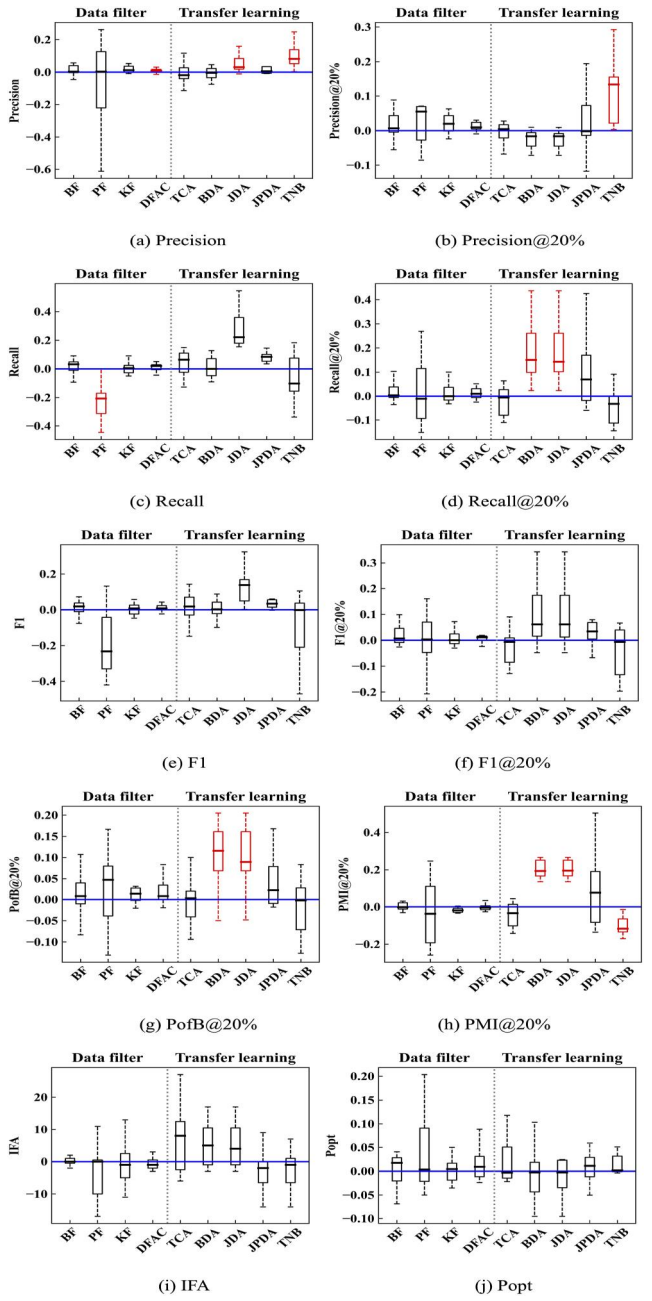


FIGURE 11 The performance difference between the data filtering and transfer learning methods and None with the K-Nearest Neighbour classifier using the CBS+ strategy. CBS+, classification before sorting; KNN, K-Nearest Neighbour.

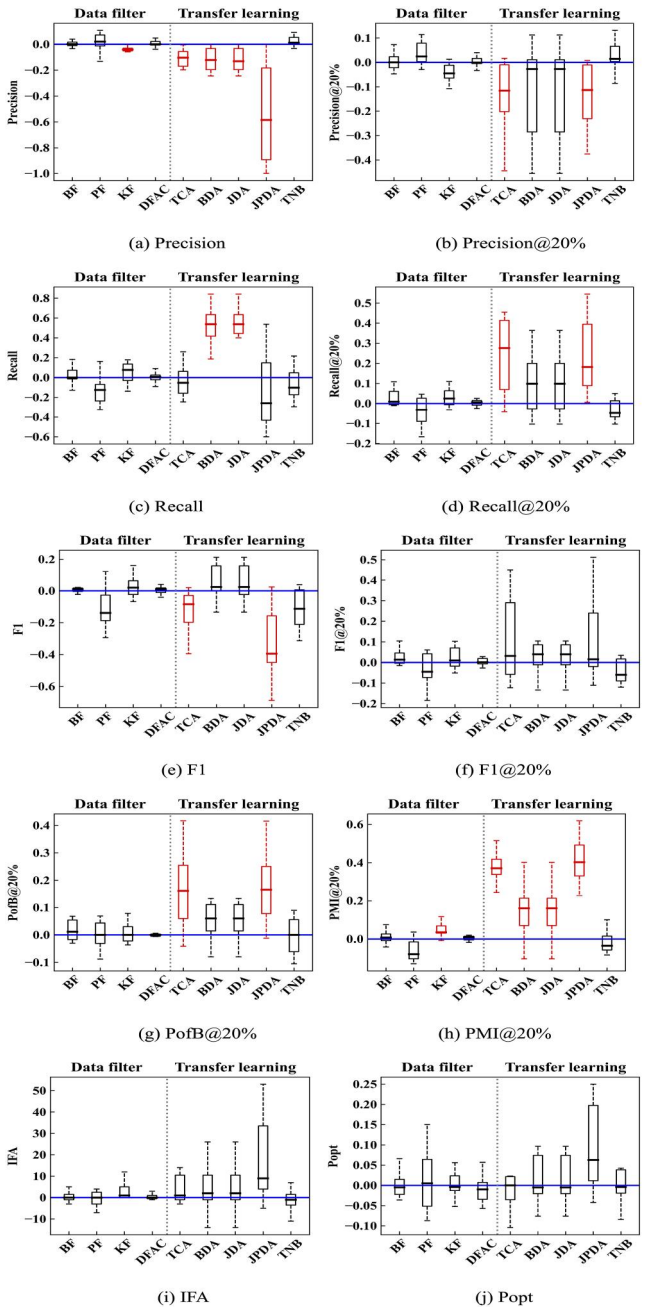


FIGURE 12 The performance difference between the data filtering and transfer learning methods and None with the LR classifier using the CBS+ strategy. CBS+, classification before sorting; LR, logistic regression.

F1@20% (0.85, 0.75, 0.85, 0.85, 0.75 and 0.85, respectively) on BDA and JDA with the three classifiers. However, the correlations between Recall and Recall@20% are almost not obvious (−0.24, −0.018, −0.13, −0.6, −0.15 and −0.35, respectively.) F1 has a moderate or low correlations with F1@20% (0.67, 0.42, 0.45, 0.31, 0.42 and 0.38, respectively). It indicates the better comprehensive classification performance of the methods can contribute to the superiority of EACPDP models to some extent.

(2) Recall@20% has a high or moderate correlation with PofB@20% (0.71, 0.82, 0.6, 0.67, 0.82, 0.82 and 0.82, respectively), since the more defective modules captured when checking the top 20% LOC, the more defects can be found.

(3) PMI@20% always has a relatively obvious correlation with PofB@20% (0.27, 0.53, 0.2, 0.27, 0.53 and 0.49, respectively) and Recall@20% (0.42, 0.71, 0.6, 0.38, 0.71 and 0.67, respectively), because more inspected modules increases the chances to find more defective modules and defects. Precision@20% has a moderate correlation with IFA (−0.58, −0.38,-0.58, −0.58, −0.38 and −0.55, respectively), since the more false alarms in the top 20% LOC, the more likely the IFA value is high.

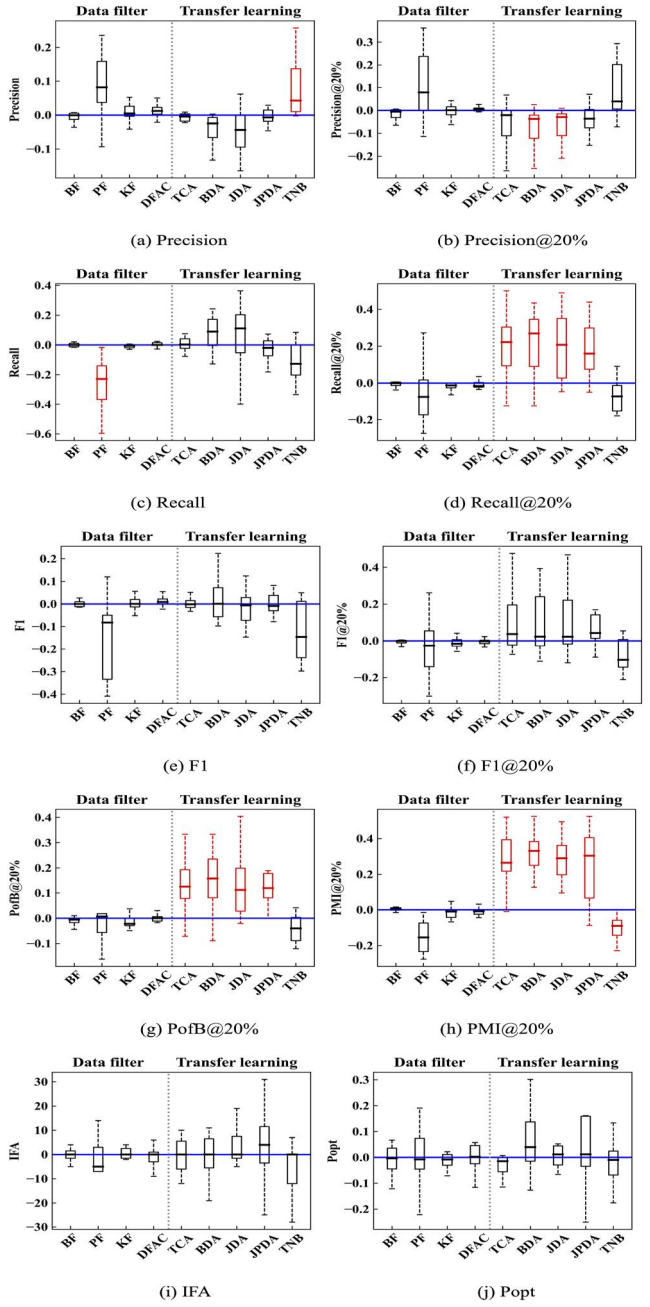


FIGURE 13 The performance difference between the data filtering and transfer learning methods and None with the RF classifier using the CBS+ strategy. CBS+, classification before sorting; RF, random forest.

Answer to RQ3

The better comprehensive classification performance of the methods can bring better EACPDP performance to some extent.

5.4 | RQ4: how does the defect threshold λ affect EACPDP performance?

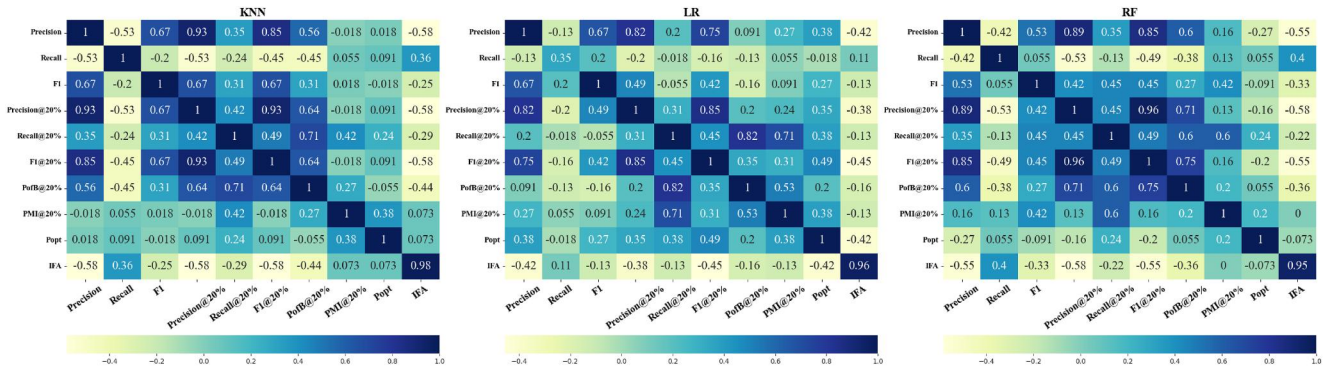
Motivation: One of the keys of the CBS+ strategy is to divide all modules into defective and clean groups. The defective group contains modules predicted to be defective, while the clean group contains modules predicted to be clean. The contents of these two groups are directly decided by the threshold λ, which is utilised to distinguish whether a new module is defective. In other words, if the predicted probability of a new module is larger than λ, the module is regarded to be defective. Moreover, we sort the defective group before the clean one, so the threshold λ also directly determines the detection order of modules. In the case of limited testing resources, only the first few modules in the clean group will be checked or even not checked. Therefore, to investigate whether changing the defect threshold λ can further improve the EACPDP performance, we perform detailed analysis of BDA and JDA using the CBS+ strategy with different λ values.

Methods: Figure 15 presents the Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, IFA and *Popt* values of BDA and JDA using the CBS+ strategy, when the threshold λ is changed from 0.1 to 0.9 with an interval of 0.1. We do not set λ to 0 or 1, because all modules will be predicted as defective or clean.

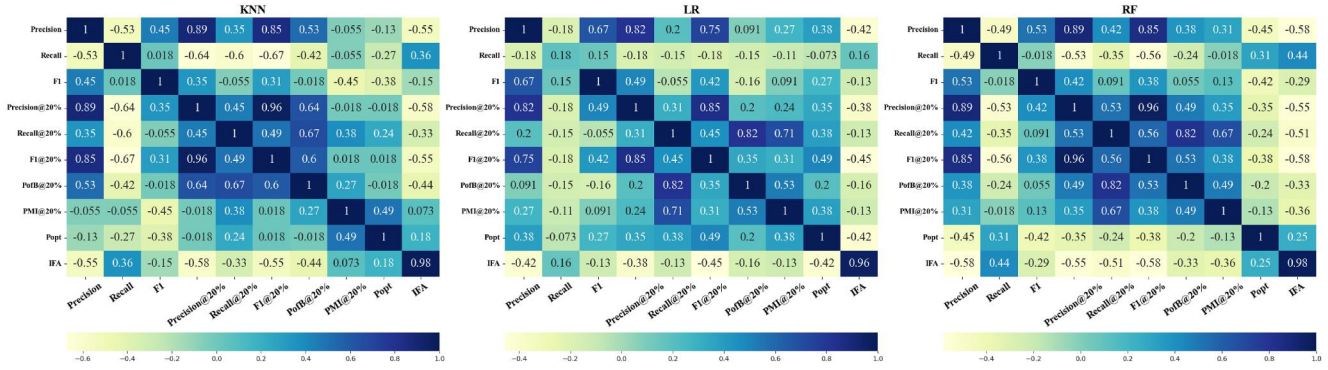
Results: Generally, when the threshold λ increases, the Recall@20%, F1@20%, PofB@20%, PMI@20% and IFA values all decrease, and the Precision@20% value increases conversely. Furthermore, the *Popt* value decreases before the threshold λ is set to 0.6 and increases after. One exception is that when the threshold λ is set to 0.9, the Precision@20% value decreases, while all other metric values increase dramatically.

(1) When we set λ to 0.1, most modules will be predicted to be defective and put into the defective group. Since the modules with fewer LOC have higher defect density, software testers would first check many modules with fewer LOC. When inspecting the same amount of LOC (i.e. top 20% LOC), software testers require to check more modules (i.e. higher PMI@20% value). As a result, the Recall@20% and PofB@20% values increase accordingly, since more inspected modules bring more opportunities to find defective modules and defects. Since the modules with fewer LOC are less likely to be defective, the IFA value is large and the Precision@20% value is low.

(2) When we increase the λ value from 0.1 to 0.8, more modules with more LOC would be put into the defective group, since the modules with more LOC are more likely to be defective. When inspecting the same amount of LOC (i.e. top 20% LOC), software testers require to check fewer modules (i.e. lower PMI@20% value). As a result, the numbers of found defective modules (i.e. Recall@20%) and defects (i.e. PofB@20%) and the IFA value decrease accordingly. When the threshold λ is set to 0.4, all metric values except Precision@20% are higher than those with $\lambda = 0.5$, which means that we can sacrifice a small part of

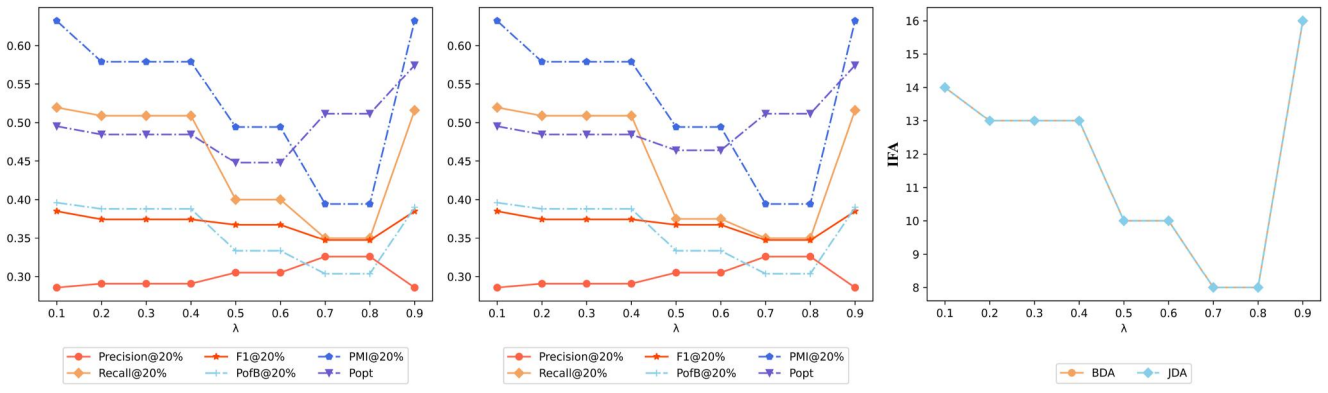


(a) BDA



(b) JDA

FIGURE 14 The correlation among all performance measures on BDA and JDA with the three classifiers.



(a) BDA (b) JDA (c) IFA

FIGURE 15 The performance impact of changing the threshold λ on BDA and JDA methods.

Precision@20%, PMI@20% and IFA in exchange for better performance on all other metrics.

(3) When the threshold λ is set to 0.9, very few modules are predicted to be defective for inspection, and there is a large budget to inspect modules predicted to be clean. By performing these additional checks, software testers can find more defective modules and defects, which also means that they would sacrifice the PMI@20% and Precision@20% values. In addition, the IFA value exceeds other cases and is unacceptable (greater than 10). When we set λ to 0.9, only very few modules are predicted to be defective, and these modules may not have defects. Therefore, software testers

have to check the modules in the clean group when examining modules in the top 20% LOC to find the first defective module. However, the top-ranked modules in the clean group contain fewer LOC, which are more likely to clean. Therefore, the IFA value increases dramatically.

Answer to RQ4

In general, the Recall@20%, F1@20%, PofB@20%, PMI@20% and IFA values decrease, and the Precision@20% value increases conversely, when we increase the threshold λ.

6 | THREATS TO VALIDITY[3,4,5,6,7]

(1) Our findings are based on 11 PROMISE datasets collected by Jureczko et al. [32] using the BugInfo tool, which have been used and validated by numerous SDP studies [28, 46, 54, 58]. Our experimental datasets belong to different application fields and have different numbers of modules (from 205 to 965) with different levels of defective proportions (from 2.2% to 98.8%). However, many other software projects in other fields with other characteristics or programming languages are not used in our work. In addition, all software projects used in our work were developed by the open-source community, and it is not clear whether our conclusions can apply to commercial projects. In the future work, we will further reduce the threat by analysing more modules from other defect datasets.

(2) We employ the four data filtering methods and five transfer learning methods in our empirical study, since they are the most classical methods and are also widely investigated in previous SDP studies. In addition, we acknowledge the existence of some other data filtering and transfer learning methods. The adoption of the unused methods in our work is left for future work.

(3) We employ not only widely used Precision, Recall and F1 metrics in the classification scenario [59, 108–110], but also Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, IFA and *Popt* in the effort-aware scenario [16, 28]. Since the EACPDP model is designed to find more defects and defective modules and to obtain an accurate global ranking based on the predicted defect density, we used PofB@20%, Recall@20%, and *Popt*. We use Precision@20%, because Precision@20% and Recall@20% are usually paired. F1@20% balances the tradeoff between Precision@20% and Recall@20%, so we use F1@20% to correct the bias that may result from using Precision@20% and Recall@20%. We use PMI@20% because checking too many modules introduces additional effort costs. In addition, we use IFA because previous studies [16, 101] have concluded that if the IFA value is too large, it will significantly reduce software testers' confidence. In addition, we use the non-parametric statistical Wilcoxon signed-rank test and Scott-Knott ESD test to compare the performance of the different data filtering and transfer learning methods to ensure that the differences are statistically significant.

(4) In order to alleviate the technical defects in our experiments as much as possible, we implement the classifiers based on the SKlearn library. The transfer learning methods are based on their own third-party libraries (i.e. TCA, BDA, JDA and JPDA). We carefully implement the code for other methods by strictly following the original papers' descriptions.

(5) Our study directly employs the default hyper-parameter setting for all data filtering and transfer learning methods and the three classifiers. Although Tantithamthavorn et al.'s study [111] have shown that optimising the hyper-parameters of classifiers would lead to a significant improvement for SDP, we leave the hyper-parameter tuning as a future work for several reasons: (a) There are several effort-aware evaluation metrics in our work, and it is difficult to choose the optimisation target for tuning the methods, because it is impossible to decide which performance measure is the most important. Therefore, the hyper-parameter tuning of the methods is regarded as a multi-objective optimisation problem, which is much more complex than the optimisation problem considering only a single evaluation metric (e.g. the AUC metric in Tantithamthavorn et al.'s study [111]). (b) The grid search method requires traversing all possible combinations of parameters, which is very time-consuming in the face of large datasets and multiple parameters of some transfer learning methods. The time-consuming tuning process makes the methods less practical. (c) In our study, adjusting the defect threshold λ can achieve the trade-off among Pecision@20%, Recall@20%, PofB@20% and PMI@20%.

7 | IMPLICATIONS

We outline some implications that researchers and practitioners can derive from our experimental results.

(1) **Researchers and practitioners should use the CBS+ strategy to rank software modules.** Ni et al.'s EACPDP study [28] has used different defect density calculation strategies when comparing different methods. Section 5.1 compares four frequently used defect density strategies in previous EADP studies. The results show that the CBS+ defect density calculation strategy outperforms Label/LOC and Prob/LOC in terms of Precision@20%, PMI@20% and IFA, and is better than Prob/LOC in other metrics (i.e. Recall@20%, F1@20%, and PofB@20%). Therefore, under overall consideration, we recommend using CBS+

[3] https://github.com/scikit-learn
[4] https://github.com/jindongwang/transferlearning/tree/master/code/traditional/TCA
[5] https://github.com/jindongwang/transferlearning/tree/master/code/traditional/BDA
[6] https://github.com/jindongwang/transferlearning/tree/master/code/traditional/JDA
[7] https://github.com/chamwen/JPDA

defect density strategy to find more defective modules and defects (higher Recall@20% and PofB@20% values), while ensuring that the IFA value is within the acceptable range.

(2) **Future EACPDP studies should consider exploring whether more advanced transfer learning methods could further enhance the performance.** We explore the performance of data filtering and transfer learning methods for EACPDP, and the results in Section 5.2 show that the transfer learning methods (i.e. BDA and JDA) have the best performance among the investigated methods. It indicates that considering both marginal and conditional distributions for feature dimensionality reduction can effectively improve the EACPDP performance. We also find that the classification performance has a direct impact on the EACPDP performance. If the classification model has better classification capability, then the EACPDP model built based on this classification model also has better EADP performance. Therefore, we believe that future EACPDP research can further consider more advanced transfer learning methods to improve the performance of EASC models.

(3) **Future EACPDP studies can consider flexible adjustment of the defect threshold λ to achieve different goals.** The optimal EACPDP model should obtain high Recall@20%, PofB@20%, Precision@20%, and *Popt* values, while keeping the PMI@20% and IFA values as low as possible. However, software testers can change the threshold λ to balance the trade-off among the metrics. If they are not sensitive to switching modules frequently and false alarms, it is worthwhile to set a low threshold to sacrifice PMI@20% and IFA in exchange for higher Recall@20% and PofB@20% values. For example, setting λ to 0.4 is very beneficial to improve Recall@20% and PofB@20% values, although it slightly increases the PMI@20% and IFA values. If only a few software testers inspect the top 20% LOC and the negative impacts of IFA on their confidence in the EACPDP method are seriously considered, we should set a very high threshold.

8 | CONCLUSION

This paper explores the effects of different defect density calculation strategies, data filtering methods, and transfer learning methods for EACPDP on 11 datasets from the PROMISE corpus. We use Precision, Recall, F1, Precision@20%, Recall@20%, F1@20%, PofB@20%, PMI@20%, IFA and *Popt* to evaluate the performance and apply the Scott-Knott ESD test and Wilcoxon signed-rank test to analyse experimental results. We observe that the CBS+ defect density strategy has the best overall performance, and BDA and JDA perform the best among all data filtering and transfer learning methods. By embedding three classifiers (i.e. KNN, LR and RF), we find that the classification ability directly affects the EACPDP performance to some extent. Therefore, we suggest that researchers use the CBS+ defect density strategy and better transfer learning methods (e.g. BDA and JDA) to enhance EACPDP performance further. Finally, we observe that the defect threshold λ of the CBS+ strategy has a significant impact on the performance, and a flexible adjustment can contribute to different EACPDP goals.

AUTHOR CONTRIBUTIONS

Fuyang Li: Investigation, Formal analysis, Methodology, Writing – original draft. **Peixin Yang**: Data curation, Software, Writing – original draft. **Jacky Wai Keung**: Project administration, Methodology, Validation. **Wenhua Hu**: Data curation, Visualisation, Writing – review & editing. **Haoyu Luo**: Investigation, Software, Visualisation. **Xiao Yu**: Formal analysis, Methodology, Supervision, Writing – review & editing.

ACKNOWLEDGEMENTS

This work was in part supported by the Project of Sanya Yazhou Bay Science and Technology City (SCKJ-JYRC-2022-17), the Natural Science Foundation of China (62272356), the Youth Fund Project of Hainan Natural Science Foundation (622QN344), the Natural Science Foundation of Chongqing (cstc2021jcyj-msxmX1115), and the Start–up Grant from Wuhan University of Technology (104–40120693).

CONFLICT OF INTEREST STATEMENT

We declare that we have no conflict of interest.

DATA AVAILABILITY STATEMENT

Data openly available in a public repository that does not issue DOIs.

ORCID

Xiao Yu https://orcid.org/0000-0002-4473-3068

REFERENCES

1. Huang, Q., et al.: A cross-project defect prediction method based on multi-adaptation and nuclear norm. IET Softw. 16(2), 200–213 (2022). https://doi.org/10.1049/sfw2.12053
2. Manchala, P., Bisi, M.: Diversity based imbalance learning approach for software fault prediction using machine learning models. Appl. Soft Comput. 124, 109069 (2022). https://doi.org/10.1016/j.asoc.2022.109069
3. Stradowski, S., Madeyski, L.: Machine learning in software defect prediction: a business-driven systematic mapping study. Inf. Software Technol. 155, 107128 (2022). https://doi.org/10.1016/j.infsof.2022.107128
4. Feng, S., et al.: Investigation on the stability of smote-based oversampling techniques in software defect prediction. Inf. Software Technol. 139, 106662 (2021). https://doi.org/10.1016/j.infsof.2021.106662
5. Yu, X., et al.: Predicting the precise number of software defects: are we there yet? Inf. Software Technol. 146, 106847 (2022). https://doi.org/10.1016/j.infsof.2022.106847
6. Zhou, C., et al.: Software defect prediction with semantic and structural information of codes based on graph neural networks. Inf. Software Technol. 152, 107057 (2022). https://doi.org/10.1016/j.infsof.2022.107057
7. Bennin, K.E., et al.: An empirical study on the effectiveness of data resampling approaches for cross-project software defect prediction. IET Softw. 16(2), 185–199 (2022). https://doi.org/10.1049/sfw2.12052
8. Kabir, M.A., et al.: Inter-release defect prediction with feature selection using temporal chunk-based learning: an empirical study. Appl. Soft Comput. 113, 107870 (2021). https://doi.org/10.1016/j.asoc.2021.107870

9. Xie, H., et al.: A universal data augmentation approach for fault localization. In: Proceedings of the 44th International Conference on Software Engineering, pp. 48–60 (2022)
10. Yang, D., et al.: Seeing the whole elephant: Systematically understanding and uncovering evaluation biases in automated program repair. ACM Trans. Software Eng. Methodol. 32(3), 1–37 (2022). https://doi.org/10.1145/3561382
11. Yu, X., et al.: The bayesian network based program dependence graph and its application to fault localization. J. Syst. Software 134, 44–53 (2017). https://doi.org/10.1016/j.jss.2017.08.025
12. Zhang, Z., et al.: Influential global and local contexts guided trace representation for fault localization. ACM Trans. Software Eng. Methodol. 32(3), 1–27 (2022). https://doi.org/10.1145/3576043
13. Mende, T., Koschke, R.: Effort-aware defect prediction models. In: 2010 14th European Conference on Software Maintenance and Reengineering, pp. 107–116. IEEE (2010)
14. Fenton, N.E., Ohlsson, N.: Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Software Eng. 26(8), 797–814 (2000). https://doi.org/10.1109/32.879815
15. Andersson, C., Runeson, P.: A replicated quantitative analysis of fault distributions in complex software systems. IEEE Trans. Software Eng. 33(5), 273–286 (2007). https://doi.org/10.1109/tse.2007.1005
16. Huang, Q., Xia, X., Lo, D.: Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. Empir. Software Eng. 24(5), 2823–2862 (2019). https://doi.org/10.1007/s10664-018-9661-2
17. Kamei, Y., et al.: A large-scale empirical study of just-in-time quality assurance. IEEE Trans. Software Eng. 39(6), 757–773 (2012). https://doi.org/10.1109/tse.2012.70
18. Rao, J., et al.: Learning to rank software modules for effort-aware defect prediction. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion, pp. 372–380. IEEE (2021)
19. Xia, X., et al.: Hydra: Massively compositional model for cross-project defect prediction. IEEE Trans. Software Eng. 42(10), 977–998 (2016). https://doi.org/10.1109/tse.2016.2543218
20. Yang, Y., et al.: Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 157–168 (2016)
21. Zhao, K., et al.: A compositional model for effort-aware just-in-time defect prediction on android apps. IET Softw. 16(3), 259–278 (2022). https://doi.org/10.1049/sfw2.12040
22. Cheng, T., et al.: Effort-aware cross-project just-in-time defect prediction framework for mobile apps. Front. Comput. Sci. 16(6), 1–15 (2022). https://doi.org/10.1007/s11704-021-1013-5
23. Khatri, Y., Singh, S.K.: Towards building a pragmatic cross-project defect prediction model combining non-effort based and effort based performance measures for a balanced evaluation. Inf. Software Technol. 150, 106980 (2022). https://doi.org/10.1016/j.infsof.2022.106980
24. Li, Z., et al.: Cross-project defect prediction via landmark selection-based kernelized discriminant subspace alignment. IEEE Trans. Reliab. 70(3), 1–18 (2021). https://doi.org/10.1109/tr.2021.3074660
25. Sun, Z., et al.: Cfps: Collaborative filtering based source projects selection for cross-project defect prediction. Appl. Soft Comput. 99, 106940 (2021). https://doi.org/10.1016/j.asoc.2020.106940
26. Zou, Q., et al.: Correlation feature and instance weights transfer learning for cross project software defect prediction. IET Softw. 15(1), 55–74 (2021). https://doi.org/10.1049/sfw2.12012
27. Zou, Q., et al.: Joint feature representation learning and progressive distribution matching for cross-project defect prediction. Inf. Software Technol. 137, 106588 (2021). https://doi.org/10.1016/j.infsof.2021.106588
28. Ni, C., et al.: Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. IEEE Trans. Software Eng. 48(3), 786–802 (2022). https://doi.org/10.1109/tse.2020.3001739
29. D'Ambros, M., Lanza, M., Robbes, R.: An extensive comparison of bug prediction approaches. In: 2010 7th IEEE Working Conference on Mining Software Repositories, pp. 31–41. IEEE (2010)
30. Gray, D., et al.: The misuse of the nasa metrics data program data sets for automated software defect prediction. In: 15th Annual Conference on Evaluation & Assessment in Software Engineering, pp. 96–103. IET (2011)
31. Shepperd, M., et al.: Data quality: some comments on the nasa software defect datasets. IEEE Trans. Software Eng. 39(9), 1208–1215 (2013). https://doi.org/10.1109/tse.2013.11
32. Jureczko, M., Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, pp. 1–10 (2010)
33. Wu, R., et al.: Relink: recovering links between bugs and changes. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 15–25 (2011)
34. Zimmermann, T., et al.: Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 91–100 (2009)
35. Turhan, B., et al.: On the relative value of cross-company and within-company data for defect prediction. Empir. Software Eng. 14(5), 540–578 (2009). https://doi.org/10.1007/s10664-008-9103-7
36. Menzies, T., et al.: Local vs. global models for effort estimation and defect prediction. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 343–351. IEEE (2011)
37. Cruz, A.E.C., Ochimizu, K.: Towards logistic regression models for predicting fault-prone code across software projects. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 460–463. IEEE (2009)
38. Watanabe, S., Kaiya, H., Kaijiri, K.: Adapting a fault prediction model to allow inter languagereuse. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, pp. 19–24 (2008)
39. Tantithamthavorn, C., et al.: An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Software Eng. 43, 1–18 (2016). https://doi.org/10.1109/tse.2016.2584050
40. Benjamini, Y., Yekutieli, D.: The control of the false discovery rate in multiple testing under dependency. Ann. Stat. 29(4), 1165–1188 (2001). https://doi.org/10.1214/aos/1013699998
41. Kamei, Y., et al.: Revisiting common bug prediction findings using effort-aware models. In: 2010 IEEE International Conference on Software Maintenance, pp. 1–10. IEEE (2010)
42. Yang, X., et al.: Tlel: a two-layer ensemble learning approach for just-in-time defect prediction. Inf. Software Technol. 87, 206–220 (2017). https://doi.org/10.1016/j.infsof.2017.03.007
43. Wang, H., Zhuang, W., Zhang, X.: Software defect prediction based on gated hierarchical lstms. IEEE Trans. Reliab. 70(2), 711–727 (2021). https://doi.org/10.1109/tr.2020.3047396
44. Zhou, Y., et al.: How far we have progressed in the journey? An examination of cross-project defect prediction. ACM Trans. Software Eng. Methodol. 27, 1–51 (2018). https://doi.org/10.1145/3183339
45. Yang, Y., et al.: Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. Software Engineering IEEE Transactions on 41(4), 331–357 (2015). https://doi.org/10.1109/tse.2014.2370048
46. Ma, W., et al.: Empirical analysis of network measures for effort-aware fault-proneness prediction. Inf. Software Technol. 69, 50–70 (2016). https://doi.org/10.1016/j.infsof.2015.09.001
47. Panichella, A., et al.: A search-based training algorithm for cost-aware defect prediction. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp. 1077–1084 (2016)
48. Yang, X., et al.: Dejit: a differential evolution algorithm for effort-aware just-in-time software defect prediction. Int. J. Software Eng. Knowl.

Eng. 31(03), 289–310 (2021). https://doi.org/10.1142/s0218194021500108
49. Yang, Y., et al.: An empirical study on dependence clusters for effort-aware fault-proneness prediction. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 296–307. IEEE (2016)
50. Muthukumaran, K., et al.: Testing and code review based effort-aware bug prediction model. In: Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 17–30. Springer (2016)
51. Bennin, K.E., et al.: Empirical evaluation of cross-release effort-aware defect prediction models. In: 2016 IEEE International Conference on Software Quality, Reliability and Security, pp. 214–221. IEEE (2016)
52. Yu, X., et al.: An empirical study of learning to rank techniques for effort-aware defect prediction. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering, pp. 298–309. IEEE (2019)
53. Bennin, K.E., et al.: Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models. In: 2016 IEEE 40th Annual Computer Software and Applications Conference, pp. 154–163. IEEE (2016)
54. Yan, M., et al.: File-level defect prediction: unsupervised vs. supervised models. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 344–353. IEEE (2017)
55. Fu, W., Menzies, T.: Revisiting unsupervised learning for defect prediction. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 72–83 (2017)
56. Liu, J., et al.: Code churn: a neglected metric in effort-aware just-in-time defect prediction. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 11–19. IEEE (2017)
57. Miletić, M., et al.: Cross-release code churn impact on effort-aware software defect prediction. In: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 1460–1466. IEEE (2018)
58. Chen, X., et al.: Multi: multi-objective effort-aware just-in-time software defect prediction. Inf. Software Technol. 93, 1–13 (2018). https://doi.org/10.1016/j.infsof.2017.08.004
59. Huang, Q., et al.: Supervised vs unsupervised models: a holistic look at effort–aware just–in–time defect prediction. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 159–170. IEEE (2017)
60. Guo, Y., Shepperd, M., Li, N.: Bridging effort-aware prediction and strong classification: a just-in-time software defect prediction study. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, pp. 325–326 (2018)
61. Qiao, L., Wang, Y.: Effort-aware and just-in-time defect prediction with neural network. PLoS One 14(2), e0211359 (2019). https://doi.org/10.1371/journal.pone.0211359
62. Qu, Y., et al.: Using k-core decomposition on class dependency networks to improve bug prediction model's practical performance. IEEE Trans. Software Eng. 47(2), 348–366 (2019). https://doi.org/10.1109/tse.2019.2892959
63. Du, X., et al.: Corebug: improving effort-aware bug prediction in software systems using generalized k-core decomposition in class dependency networks. Axioms 11(5), 205 (2022). https://doi.org/10.3390/axioms11050205
64. Zhang, W., Li, W., Jia, X.: Effort-aware tri-training for semi-supervised just-in-time defect prediction. In: Pacific-asia Conference on Knowledge Discovery and Data Mining, pp. 293–304. Springer (2019)
65. Fan, Y., et al.: The impact of changes mislabeled by SZZ on just-in-time defect prediction. IEEE Trans. Software Eng., 1–26 (2019)
66. Ulan, M., et al.: Weighted software metrics aggregation and its application to defect prediction. Empir. Software Eng. 26(5), 1–34 (2021). https://doi.org/10.1007/s10664-021-09984-2
67. Çarka, J., Esposito, M., Falessi, D.: On effort-aware metrics for defect prediction. Empir. Software Eng. 27(6), 1–38 (2022). https://doi.org/10.1007/s10664-022-10186-7
68. Xu, Z., et al.: Effort-aware just-in-time bug prediction for mobile apps via cross-triplet deep feature embedding. IEEE Trans. Reliab. 71(1), 204–220 (2021). https://doi.org/10.1109/tr.2021.3066170
69. Briand, L.C., Melo, W.L., Wust, J.: Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Trans. Software Eng. 28(7), 706–720 (2002). https://doi.org/10.1109/tse.2002.1019484
70. Bai, J., Jia, J., Capretz, L.F.: A three-stage transfer learning framework for multi-source cross-project software defect prediction. Inf. Software Technol. 150, 106985 (2022). https://doi.org/10.1016/j.infsof.2022.106985
71. Chen, H., et al.: Aligned metric representation based balanced multiset ensemble learning for heterogeneous defect prediction. Inf. Software Technol. 147, 106892 (2022). https://doi.org/10.1016/j.infsof.2022.106892
72. Peters, F., Menzies, T., Marcus, A.: Better cross company defect prediction. In: 2013 10th Working Conference on Mining Software Repositories, pp. 409–418. IEEE (2013)
73. Kawata, K., Amasaki, S., Yokogawa, T.: Improving relevancy filter methods for cross-project defect prediction. In: 2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence, pp. 2–7. IEEE (2015)
74. Yu, X., et al.: Improving cross-company defect prediction with data filtering. Int. J. Software Eng. Knowl. Eng. 27(09n10), 1427–1438 (2017). https://doi.org/10.1142/s0218194017400046
75. Hosseini, S., Turhan, B., Mäntylä, M.: A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. Inf. Software Technol. 95, 296–312 (2018). https://doi.org/10.1016/j.infsof.2017.06.004
76. Bin, Y., et al.: Training data selection for cross-project defection prediction: which approach is better? In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 354–363. IEEE (2017)
77. Ma, Y., et al.: Transfer learning for cross-company software defect prediction. Inf. Software Technol. 54(3), 248–256 (2012). https://doi.org/10.1016/j.infsof.2011.09.007
78. Nam, J., Pan, S.J., Kim, S.: Transfer defect learning. In: 2013 35th International Conference on Software Engineering, pp. 382–391. IEEE (2013)
79. Jing, X.Y., et al.: An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans. Software Eng. 43(4), 321–339 (2016). https://doi.org/10.1109/tse.2016.2597849
80. Limsettho, N., et al.: Cross project defect prediction using class distribution estimation and oversampling. Inf. Software Technol. 100, 87–102 (2018). https://doi.org/10.1016/j.infsof.2018.04.001
81. Wu, F., et al.: Cross-project and within-project semisupervised software defect prediction: a unified approach. IEEE Trans. Reliab. 67(2), 581–597 (2018). https://doi.org/10.1109/tr.2018.2804922
82. Gong, L., et al.: A novel class-imbalance learning approach for both within-project and cross-project defect prediction. IEEE Trans. Reliab. 69(1), 40–54 (2019). https://doi.org/10.1109/tr.2019.2895462
83. Xu, B., et al.: Cross-project aging-related bug prediction based on joint distribution adaptation and improved subclass discriminant analysis. In: 2020 IEEE 31st International Symposium on Software Reliability Engineering, pp. 325–334. IEEE (2020)
84. Li, D., et al.: A cross-project aging-related bug prediction approach based on joint probability domain adaptation and k-means smote. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion, pp. 350–358. IEEE (2021)
85. Omondiagbe, O.P., Licorish, S.A., MacDonell, S.G.: Improving transfer learning for cross project defect prediction. TechRxiv

preprint techrxiv 19517029 (2022). https://doi.org/10.36227/techrxiv.19517029
86. Jin, C.: Cross-project software defect prediction based on domain adaptation learning and optimization. Expert Syst. Appl. 171, 114637 (2021). https://doi.org/10.1016/j.eswa.2021.114637
87. Pan, S.J., et al.: Domain adaptation via transfer component analysis. IEEE Trans. Neural Network. 22(2), 199–210 (2010). https://doi.org/10.1109/tnn.2010.2091281
88. Long, M., et al.: Transfer feature learning with joint distribution adaptation. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2200–2207 (2013)
89. Wang, J., et al.: Balanced distribution adaptation for transfer learning. In: 2017 IEEE International Conference on Data Mining, pp. 1129–1134. IEEE (2017)
90. Zhang, W., Wu, D.: Discriminative joint probability maximum mean discrepancy (djp-mmd) for domain adaptation. In: 2020 International Joint Conference on Neural Networks, pp. 1–8. IEEE (2020)
91. Boetticher, G.: The promise repository of empirical software engineering data. http://promisedata.org/repository (2007)
92. Chen, Y., et al.: Improving ponzi scheme contract detection using multi-channel textcnn and transformer. Sensors 21(19), 6417 (2021). https://doi.org/10.3390/s21196417
93. Ma, X., et al.: Casms: combining clustering with attention semantic model for identifying security bug reports. Inf. Software Technol. 147, 106906 (2022). https://doi.org/10.1016/j.infsof.2022.106906
94. Zhen, Y., et al.: On the significance of category prediction for code-comment synchronization. ACM Trans. Software Eng. Methodol. (2022). https://doi.org/10.1145/3534117
95. Chen, Y., Lu, X., Li, X.: Supervised deep hashing with a joint deep network. Pattern Recogn. 105, 107368 (2020). https://doi.org/10.1016/j.patcog.2020.107368
96. Chen, Y., Lu, X., Wang, S.: Deep cross-modal image–voice retrieval in remote sensing. IEEE Trans. Geosci. Rem. Sens. 58(10), 7049–7061 (2020). https://doi.org/10.1109/tgrs.2020.2979273
97. Chen, Y., et al.: Deep quadruple-based hashing for remote sensing image-sound retrieval. IEEE Trans. Geosci. Rem. Sens. 60, 1–14 (2022). https://doi.org/10.1109/tgrs.2022.3155283
98. He, C., Wu, J., Zhang, Q.: Characterizing research leadership on geographically weighted collaboration network. Scientometrics 126(5), 4005–4037 (2021). https://doi.org/10.1007/s11192-021-03943-w
99. He, C., Wu, J., Zhang, Q.: Proximity-aware research leadership recommendation in research collaboration via deep neural networks. J. Assoc. Inf. Sci. Technol. 73(1), 70–89 (2022). https://doi.org/10.1002/asi.24546
100. Yang, Z., et al.: Acomnn: attention enhanced compound neural network for financial time-series forecasting with cross-regional features. Appl. Soft Comput. 111, 107649 (2021). https://doi.org/10.1016/j.asoc.2021.107649
101. Kochhar, P.S., et al.: Practitioners' expectations on automated fault localization. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, pp. 165–176 (2016)
102. Ghotra, B., McIntosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 789–800. IEEE (2015)
103. LaValley, M.P.: Logistic regression. Circulation 117(18), 2395–2399 (2008). https://doi.org/10.1161/circulationaha.106.682658
104. Hautamaki, V., Karkkainen, I., Franti, P.: Outlier detection using k-nearest neighbour graph. In: Proceedings of the 17th International Conference on Pattern Recognition, 2004, pp. 430–433. ICPR 2004, IEEE (2004)
105. Liaw, A., Wiener, M.: Classification and regression by randomforest. R. News 2, 18–22 (2002)
106. Ferreira, J.A., Zwinderman, A.H.: On the benjamini–hochberg method. Ann. Stat. 34(4), 1827–1849 (2006). https://doi.org/10.1214/009053606000000425
107. Hinkle, D.E., Wiersma, W., Jurs, S.G.: Applied statistics for the behavioral sciences. Houghton Mifflin College Division. vol. 663. https://books.google.com.tw/books?id=7tntAAAAMAAJ (2003)
108. He, Z., et al.: An investigation on the feasibility of cross-project defect prediction. Autom. Software Eng. 19(2), 167–199 (2012). https://doi.org/10.1007/s10515-011-0090-3
109. Stuckman, J., Walden, J., Scandariato, R.: The effect of dimensionality reduction on software vulnerability prediction models. IEEE Trans. Reliab. 66, 1–21 (2017). https://doi.org/10.1109/tr.2016.2630503
110. Xin, X., et al.: Hydra: Massively compositional model for cross-project defect prediction. IEEE Trans. Software Eng. 42(10), 977–998 (2016). https://doi.org/10.1109/tse.2016.2543218
111. Tantithamthavorn, C., et al.: Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th International Conference on Software Engineering, pp. 321–332 (2016)

How to cite this article: Li, F., et al.: Revisiting 'revisiting supervised methods for effort-aware cross-project defect prediction'. IET Soft. 17(4), 472–495 (2023). https://doi.org/10.1049/sfw2.12133