


## ORIGINAL RESEARCH

# The impact of feature selection techniques on effort-aware defect prediction: An empirical study

Fuyang Li<sup>1</sup> | Wanpeng Lu<sup>1,2</sup> | Jacky Wai Keung<sup>3</sup> | Xiao Yu<sup>1,4,5</sup>  | Lina Gong<sup>6</sup> | Juan Li<sup>7</sup>

<sup>1</sup>School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China

<sup>2</sup>School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China

<sup>3</sup>Department of Computer Science, City University of Hong Kong, Hong Kong, China

<sup>4</sup>Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China

<sup>5</sup>Wuhan University of Technology Chongqing Research Institute, Chongqing, China

<sup>6</sup>School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

<sup>7</sup>School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan, China

## Correspondence

Xiao Yu, Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China.

Email: [xiaoyu@whut.edu.cn](mailto:xiaoyu@whut.edu.cn)

Lina Gong, School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.

Email: [gonglina@nuaa.edu.cn](mailto:gonglina@nuaa.edu.cn)

## Funding information

Project of Sanya Yazhou Bay Science and Technology City, Grant/Award Number: SCKJ-JYRC-2022-17; NSFC, Grant/Award Numbers: 62101393, 62102292, 62202223; Youth Fund Project of Hainan Natural Science Foundation, Grant/Award Number: 622QN344; Sanya Science and Education Innovation Park of Wuhan University of Technology, Grant/Award Number: 2021KF0031; Fundamental Research Funds for the Central Universities, Grant/Award Numbers: WUT223110002, WUT223110001; Natural Science Foundation of Chongqing, Grant/Award Numbers: cstc2021jcyj-msxmX1115, cstc2021jcyj-msxmX1148; Natural Science Foundation of Jiangsu Province, Grant/Award Number: BK20220881; Open Project of Wuhan University of Technology Chongqing Research Institute, Grant/Award Number: ZL2021-6

## Abstract

Effort-Aware Defect Prediction (EADP) methods sort software modules based on the defect density and guide the testing team to inspect the modules with high defect density first. Previous studies indicated that some feature selection methods could improve the performance of Classification-Based Defect Prediction (CBDP) models, and the Correlation-based feature subset selection method with the Best First strategy (CorBF) performed the best. However, the practical benefits of feature selection methods on EADP performance are still unknown, and blindly employing the best-performing CorBF method in CBDP to pre-process the defect datasets may not improve the performance of EADP models but possibly result in performance degradation. To assess the impact of the feature selection techniques on EADP, a total of 24 feature selection methods with 10 classifiers embedded in a state-of-the-art EADP model (CBS+) on the 41 PROMISE defect datasets were examined. We employ six evaluation metrics to assess the performance of EADP models comprehensively. The results show that (1) The impact of the feature selection methods varies in classifiers and datasets. (2) The four wrapper-based feature subset selection methods with forwards search, that is, AdaBoost with Forwards Search, Deep Forest with Forwards Search, Random Forest with Forwards Search, and XGBoost with Forwards Search (XGBF) are better than other methods across the studied classifiers and the used datasets. And XGBF with XGBoost as the embedded classifier in CBS+ performs the best on the datasets. (3) The best-performing CorBF method in CBDP does not perform well on the EADP task. (4) The selected features vary with different feature selection methods and different datasets, and the features *noc* (number of children), *ic* (inheritance coupling), *cho* (coupling between object classes), and *cbm* (coupling between methods) are frequently selected by the four wrapper-based feature subset selection methods with forwards search. (5) Using AdaBoost, deep forest, random forest, and XGBoost as the base classifiers embedded in CBS+ can achieve the best performance. In summary, we recommend the software testing team should employ XGBF with XGBoost as the embedded classifier in CBS+ to enhance the EADP performance.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2023 The Authors. *IET Software* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

## KEYWORDS

feature extraction, software engineering, software quality, software reliability

## 1 | INTRODUCTION

Recently, the scale of software systems has become increasingly complex and huge, which will lead to software systems being more prone to defects, errors, and even crashes [1–5]. Therefore, finding and fixing software bugs as early as possible is particularly significant [6]. However, there are often limited software testing resources in real life, and the software testing team cannot test every software module within a limited time [7–11]. Therefore, researchers propose Software Defect Prediction (SDP) to assist the software testing team in prioritising limited testing resources by inspecting the most likely defective modules. SDP technology first builds the prediction model by utilising datasets from historical software repositories. Then, it deploys the constructed prediction model to calculate the defect-proneness of software modules. Accurate predictions can guide the software testing team to pay attention to those predicted defective software modules and inspect them first, which helps allocate limited testing resources more optimally [10, 12–14].

The existing SDP methods are mainly categorised into Classification-Based Defect Prediction (CBDP) and Effort-Aware Defect Prediction (EADP) [15]. CBDP employs the binary classification algorithm to train the model and predicts whether a new software module is faulty. When too many software modules are predicted to be faulty, and the software testing team cannot inspect all the predicted ones due to the deadline, they have no idea to inspect the predicted defective modules first. Therefore, EADP techniques were proposed to guide software testers to inspect the software modules with high defect density first [15, 16]. The primary objective of EADP is finding more software bugs and defective modules when inspecting a certain amount of Lines Of Code (LOC) and obtaining a more accurate ranking of modules.

Huang et al. [17] proposed an EADP model called CBS+ (Classify Before Sorting). CBS+ first employs the binary classification algorithm (i.e., Logistic Regression (LR)) to calculate the probability of new modules being defective. Then, CBS+ suggests that the software testing team inspects the predicted defective modules with a high defect density (i.e., the ratio between the defect probability and LOC) first. If there are remaining limited testing resources after inspecting all predicted defective ones, the non-faulty modules will continue to be checked. The experimental results show that CBS+ outperforms Effort-Aware Linear Regression (EALR) [18] and ManualUp [19]. Subsequently, Ni et al. [20] investigated CBS+ for cross-project EADP and showed its superiority, and Ni et al. [21] have indicated CBS+ still outperforms the baselines on 20 JavaScript projects. Recently, Yan et al. [22] pointed out that Alibaba's Development Efficiency department took considerable interest in effort-aware bug identification and found that CBS+ performed better than EALR, ManualUp, and OneWay on Alibaba real-world software projects. In addition, they produced a tool based on CBS+, but the tool help software testers pay attention to a small number of the warned code changes (i.e.,

the performance of CBS+ is not very promising on Alibaba projects.). In another SDP study, Wan et al. [23] explored the practical value of SDP and pointed out that over 90% of practitioners would like to adopt SDP techniques. The main reason for unwillingness is that some practitioners had disbelief in the performance of SDP methods.

Similar to the CBDP task, the performance of the EADP model (e.g., CBS+) is also dependent on the quality of the software features collected from modules. The previous studies [24–29] show that the CBDP models often exhibit low classification performance due to the feature irrelevance or redundancy, and some feature selection methods enhance the CBDP performance by filtering out the useless features, while some methods may degrade the performance. In the two highly-cited and most influential articles, Xu et al. [28] and Ghotra et al. [26] studied more than 30 feature selection methods for CBDP and observed that the Correlation-based feature subset selection method with the Best First strategy (CorBF) usually achieved the best performance. However, the previous feature selection studies all focus on the CBDP task, and whether the feature selection methods can enhance the EADP performance is still unknown. Blindly employing the best-performing CorBF method in CBDP to pre-process the defect datasets may not enhance the performance of EADP models but possibly result in performance degradation.

Considering this issue, we perform a comprehensive study to examine the practical benefits of 24 feature selection methods on the performance of CBS+ with 10 classifiers. The 24 methods are categorised into four families, that is, (1) filter-based ranking, (2) filter-based subset, (3) wrapper-based, and (4) None (using all original features). The 10 classifiers embedded in CBS+ fall into the six groups, that is, statistical, decision tree-based, nearest-neighbour-based, neural network-based, support vector machine-based and ensemble-based. The 41 datasets from the PROMISE corpus are employed to conduct our experimental studies. Since the primary objective of EADP is to find more bugs and defective modules and obtain a more accurate global ranking of software modules, we mainly employ Recall@20%, PofB@20% (Proportion of the found Bugs when inspecting the top 20%LOC), and Norm(*Popt*) to measure the effects of the above-mentioned feature selection methods and classifiers. In addition, we also use Precision@20% and IFA (Initial False Alarms) to evaluate the false positive rate, and PMI@20% (Proportion of Modules Inspected when inspecting the top 20% LOC) to measure how many software modules are required to inspect. Finally, we apply the Scott-Knott Effect Size Difference (Scott-Knott ESD) test [30] to divide the feature selection methods into different rankings.

The results are as follows:

- (1) The impact of the feature selection methods varies in classifiers and testing datasets. Compared with None, most of the studied methods achieve similar or even better performance on the classifier and testing dataset levels.

- (2) On the testing dataset level, the four wrapper-based feature subset selection methods with forwards search (i.e., AdaBoost with Forwards Search (ADBF), Deep Forest with Forwards Search (DFF), Random Forest with Forwards Search (RFF), and XGBoost with Forwards Search (XGBF)) outperform other feature selection methods. All of these four methods can be in the top-3 ranking in the range of 60%–97% of the studied testing datasets in terms of PofB@20%, Recall@20%, and Norm(*Popt*) and also achieve acceptable performance in terms of Precision@20%, PMI@20%, and IFA. In addition, RFF and XGBF perform the best among these four methods.
- (3) On the classifier level, ADBF, DFF, RFF, and XGBF obtain better performance than others. All of these four methods can be in the top-2 ranking in the range of 70%–100% of the studied classifiers in terms of PofB@20%, Recall@20%, and Norm(*Popt*), and they also achieve the acceptable Precision@20%, PMI@20%, and IFA values. In addition, XGBF with XGBoost as the embedded classifier in CBS+ achieves the highest average PofB@20%, Recall@20%, and Norm(*Popt*) values on the testing datasets.
- (4) The best-performing CorBF method in CBDP does not perform well on the EADP task. Compared with the None method, CorBF can only achieve similar performance on most classifiers and testing datasets.
- (5) The selected features vary with different feature selection methods and different datasets. The features *noc*, *ic*, *cho*, and *cbm* are selected by the four wrapper-based feature subset selection methods with forwards search for the most times among the testing datasets, which indicates that these four features help construct the EADP models more effectively.
- (6) Employing AdaBoost (ADB), Deep Forest (DF), Random Forest (RF), and XGBoost (XGB) as the base classifiers embedded in CBS+ can achieve the best performance in terms of Precision@20%, Recall@20%, PofB@20%, and Norm(*Popt*).

Our contributions can be concluded as the following two points:

- We, for the first time, perform such a comprehensive empirical study to explore the practical benefits of 24 feature selection methods and 10 classifiers for EADP.
- We use six evaluation metrics on 41 datasets from the PROMISE corpus to comprehensively evaluate these methods, discuss the experimental results on both classifier and testing dataset levels, and provide some implications to researchers and practitioners.

## 2 | PRELIMINARIES

We give an overview of the studied feature selection methods. We also treat the None method that uses all original features as a feature selection method. Therefore, we totally apply 24 feature selection methods for EADP. The selection of the methods is the same as Ghotra et al.'s [26] empirical study. In addition, the 24 methods are widely used in previous SDP

studies [28, 31–33] and cover the four families, that is, 11 filter-based feature ranking methods, four filter-based feature subset selection methods, eight wrapper-based feature subset selection methods, and None.

### 2.1 | Filter-based feature ranking

The process of the methods is to first evaluate the importance value of each software feature and then rank the features according to the value. A higher value indicates that the corresponding software feature correlates more strongly with the class labels.

Statistic-based methods:

- Chi-Square (CS) measures the importance value of each feature by calculating the chi-square statistic value between the features and the class labels.
- CorRelation (CR) evaluates the importance value of each feature by calculating the Pearson correlation coefficient value between the features and the class labels.
- Clustering Variation (CV) evaluates the importance value of each feature by calculating the coefficient of variation value between the features and the class labels.

Probability-based methods:

- Probabilistic Significance (PS) is a conditional probability-based method in which each feature is assigned a significance value depending on how effectively it discerns each class label.
- Information Gain (IG) is an entropy-based method in which features are sorted and selected based on the uncertainty of the class labels. The higher IG value indicates a stronger capability of eliminating uncertainty.
- Gain Ratio (GR) is an improvement to IG concerning the preference for the features with a larger number of possible values.
- Symmetrical Uncertainty (SU) similarly alleviates IG's bias towards multi-valued features and normalises the values within the range from zero to one by calculating the SU between one set of features and another.

Instance-based methods:

- ReliefF(REF) chooses a software module at random and its nearest neighbours from the defective modules and non-defective ones. Then, the correlation value of each feature is updated by comparing this module and its nearest neighbours.
- ReliefF-Weight (RW) can be regarded as a parameter tuning of ReliefF where the nearest neighbours will be weighed by their distance to the chosen software module.

Classifier-based methods:

- One Rule based Feature selection counts all the features and the number of their occurrences in the case of each class label

and then calculates the error rate for each feature as the only rule. The feature with the minimal error rate will be selected.

- Support Vector Machine-based Feature selection (SVMF) sorts the software feature based on the square of the weight assigned by the SVM algorithm.

For the above-mentioned methods, we choose the top  $\log_2 n$  features suggested by Khoshgoftaar et al. [34], where  $n$  is the total number of software features.

## 2.2 | Filter-based feature subset selection

The idea of the methods is to select a subset of features from all original features instead of evaluating the importance value of each software feature individually.

- Correlation-based feature subset selection (Cor) employs the heuristics method to evaluate and rank feature subsets rather than an individual feature and chooses the better subset where features are strongly correlated with the class label but not correlated with each other.
- Consistency-based feature subset selection (Con) selects the smallest feature subset whose consistency is equal to the consistency of all original features.

We employ two search strategies to generate feature subsets using the aforementioned consistency-based and correlation-based methods.

- Best First (BF) implements the greedy hill-climbing method with backtracking to generate feature subsets. It can search forward from an empty feature set, search backward from the complete feature set, or search bidirectionally from an intermediate point.
- Greedy Stepwise (GS) greedily searches the feature subset space forward or backward without backtracking till adding or removing a feature leads to performance degradation.

Therefore, we totally obtain four (2 filter-based feature subset selection methods  $\times$  2 search strategies) feature selection methods, that is, Cor with Best First, Cor with Greedy Stepwise, Con with Best First, and Con with Greedy Stepwise. We abbreviate them to CorBF, CorGS, ConBF, and ConGS, respectively.

## 2.3 | Wrapper-based feature subset selection

The methods employ pre-determined classifiers and performance measures to search for a best-performing feature subset. In this study, we use the four classifiers, that is, ADB, XGB, DF, and RF as the base classification model because our preliminary experimental results show that the four classifiers perform the best when using all original software features. We employ PofB@20% as the performance measure because the primary objective of EADP is to discover more software bugs by

inspecting a certain amount of LOC. In addition, we use two search strategies in the process of wrapper-based feature subset selection, which are starting forwards search from the empty feature set and starting backwards search from the full feature set. Therefore, we totally obtain the eight (4 classifiers  $\times$  2 search strategies) feature selection methods, that is, ADB with Forwards Search, ADB with Backwards Search, XGB with Forwards Search, XGB with Backwards Search, DF with Forwards Search, DF with Backwards Search, RF with Forwards Search, and RF with Backwards Search. We abbreviate them to ADBF, ADBB, XGBF, XGBB, DFF, DFB, RFF, and RFB, respectively.

## 3 | EXPERIMENTAL SETUP

### 3.1 | Datasets

Many public software defect datasets only have the information of the class label (i.e., defective or not). Since EADP models aim to find more bugs, in this experiment, we select the PROMISE defect datasets [35] that have information on bug numbers. In addition, we conduct the cross-version validation, so we only select the projects that contain three or more versions. Table 1 shows the detailed information of the 41 experimental datasets, where #Module is the number of modules, %Defects is the proportion of the defective ones, AvgDefects is the average number of defects, and AvgLOC is the average number of LOC. There are 20 software features in the datasets, as shown in Table 2.

### 3.2 | Evaluation metric

Due to the limitation of the testing resources in actual defect inspection, it is vital to utilise limited resources to find more software defects. Therefore, it is necessary to take the effort into consideration for defect prediction. In this work, we deploy six different effort-aware evaluation metrics to measure the prediction results of EADP models, some of which are also widely used in the machine learning field [3, 36–43]. Similar to the previous EADP studies, we restrict the limited effort to 20% of the total LOC of one dataset in our work. There are  $M$  software modules in a defect dataset, and it contains  $P$  defective modules and  $Q$  bugs. When inspecting the top 20% LOC based on the prediction results of an EADP model, the software testing team inspects  $m$  software modules and finds  $p$  actual defective modules and  $q$  bugs. Based on this, the six effort-aware evaluation metrics are calculated as follows:

**PofB@20%** indicates the proportion of the number of inspected bugs to the total number of defects. The higher PofB@20% value means more defects can be found.

$$PofB@20\% = \frac{q}{Q} \quad (1)$$

**Recall@20%** indicates the proportion of the number of inspected actual defective software modules to the total



**TABLE 1** The details of the experimental datasets

Datasets	#Module	%Defects	AvgDefects	AvgLOC
Ant-1.3	125	16%	1.65	301.6
Ant-1.4	178	22.5%	1.18	304.5
Ant-1.5	293	10.9%	1.09	297.1
Ant-1.6	351	26.2%	2.00	322.6
Ant-1.7	745	22.3%	2.04	280.1
Camel-1.0	339	3.8%	1.08	99.5
Camel-1.2	608	35.5%	2.42	109.0
Camel-1.4	872	16.6%	2.31	112.5
Camel-1.6	965	19.5%	2.66	117.2
Ivy-1.1	111	56.8%	3.7	245.9
Ivy-1.4	241	6.6%	1.12	246
Ivy-2.0	352	11.4%	1.4	249.3
Jedit-3.2	272	33.1%	4.24	473.8
Jedit-4.0	306	24.5%	3.01	473.2
Jedit-4.1	312	25.3%	2.75	490.7
Jedit-4.2	367	13.1%	2.21	465.1
Jedit-4.3	492	2.2%	1.09	411.3
Log4j-1.0	135	25.2%	1.79	159.6
Log4j-1.1	109	33.9%	2.32	182.9
Log4j-1.2	205	92.2%	2.63	186.3
Lucene-2.0	195	46.7%	2.95	259.5
Lucene-2.2	247	58.3%	2.88	257.4
Lucene-2.4	340	59.7%	3.11	302.5
Poi-1.5	237	59.5%	2.43	233.9
Poi-2.0	314	11.8%	1.05	296.7
Poi-2.5	385	64.4%	2.0	311.0
Poi-3.0	442	63.6%	1.78	292.6
Synapse-1.0	157	10.2%	1.31	183.5
Synapse-1.1	222	27%	1.65	190.5
Synapse-1.2	256	33.6%	1.69	209.0
Velocity-1.4	196	75%	1.43	263.8
Velocity-1.5	214	66.4%	2.33	248.3
Velocity-1.6	229	34.1%	2.44	249.0
Xalan-2.4	723	15.2%	1.42	311.3
Xalan-2.5	803	48.2%	1.37	379.7
Xalan-2.6	885	46.4%	1.52	465.2
Xalan-2.7	909	98.8%	1.35	471.5
Xerces-init	162	47.5%	2.17	560.0
Xerces-1.2	440	16.1%	1.62	361.9
Xerces-1.3	453	15.2%	2.8	368.9
Xerces-1.4	588	74.3%	3.65	240.1

number of defective software modules in the defect dataset. The higher Recall@20% value means more defective software modules can be found.

$$Recall@20\% = \frac{p}{P} \quad (2)$$

**Norm(*Popt*)** indicates the difference between the optimal EADP model and the predictive model. In the optimal model, all software modules are sorted in descending order of the actual defect density, while all software modules are ranked in descending order of the predicted defect density in the predictive model. In Figure 1, the horizontal axis represents the cumulative proportion of the inspected LOC, and the vertical axis represents the cumulative percentage of the found bugs. The red curve in the figure denotes the optimal model, the blue curve denotes the prediction model, and  $\Delta opt$  denotes the difference between the two models. Then, *Popt* is defined as follows:

$$Popt = 1 - \Delta opt \quad (3)$$

According to the research of Kamei et al. [44], since the minimum value of *Popt* is dependent on the total quantity of defects in the dataset, we also employ the normalised *Popt*, that is, Norm(*Popt*), as the evaluation metric. Obviously, the higher

**TABLE 2** The definition of the 20 software features in the datasets

No.	Feature	Name
1	wmc	Weighted methods per class
2	dit	Depth of inheritance tree
3	noc	Number of children
4	cbo	Coupling between object classes
5	rfe	Response for a class
6	lcom	Lack of cohesion in methods
7	ca	Afferent couplings
8	ce	Efferent couplings
9	npm	Number of public methods
10	lcom3	Another lack of cohesion in methods
11	loc	Line of code
12	dam	Data access metric
13	moa	Measure of aggregation
14	mfa	Measure of functional abstraction
15	cam	Cohesion among methods of class
16	ic	Inheritance coupling
17	cbm	Coupling between methods
18	amc	Average method complexity
19	max_cc	Max McCabe's cyclomatic complexity
20	avg_cc	Average McCabe's cyclomatic complexity

$\text{Norm}(P_{opt})$  value means that the predictive model is closer to the optimal model.

$$\text{Norm}(P_{opt}) = \frac{P_{opt} - \min(P_{opt})}{\max(P_{opt}) - \min(P_{opt})} \quad (4)$$

**Precision@20%** indicates the proportion of the number of inspected actual defective software modules to that of the software modules in the top 20% LOC. The higher Precision@20% value means better performance of the prediction model.

$$\text{Precision@20\%} = \frac{p}{m} \quad (5)$$

**PMI@20%** indicates the proportion of the number of inspected software modules to the total number of software modules in the dataset. The lower PMI@20% value means the software testing team is required to inspect fewer software modules.

$$\text{PMI@20\%} = \frac{m}{M} \quad (6)$$

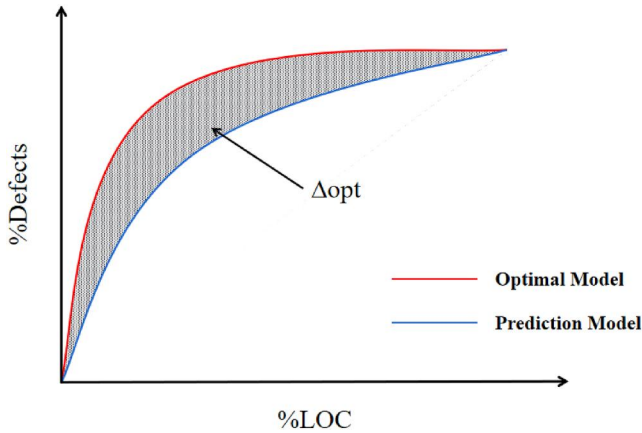


FIGURE 1 A cumulative lift chart

**IFA** indicates the number of inspected modules before the testing team finds the first defective module. The lower IFA value means better performance of the prediction model. When the IFA value is greater than 10, it is considered unacceptable [17].

In general, when the model obtains a higher Recall@20% value, it would achieve a lower Precision@20%. When the model obtains a higher PofB@20% value, it will achieve a higher PMI@20%.

### 3.3 | Experimental process

The overall experimental process is shown in Figure 2. The previous EADP studies [17, 20] usually use the ten-fold cross-validation method. However, the within-version validation that derives both training and testing data from a single version is very unrealistic in the actual software testing environment [45, 46]. In addition, we acknowledge the existence of cross-project validation that uses other projects (i.e., cross-projects) as the training data and predicts the ranking of the modules in the within-project. But we need to consider the data distribution difference between within-project and cross-project data. Therefore, it is more realistic to construct the predictive model by using the historically developed data and calculate the defect-proneness of the developing modules. Therefore, we use the cross-version validation method, that is, the prediction model is constructed by using the previous version of a software project and then applied to predict the software modules in the current version of the project. For example, we employ the Xerces-1.3 as the training data and the following version (Xerces-1.4) as the testing data. Therefore, we totally have 30 different training and testing datasets pairs.

For each pair, we first utilise the 24 feature selection methods to pick the representative features from the training datasets, so we can obtain the simplified training datasets. Then, we choose the same features from the testing datasets to get the simplified testing datasets. Therefore, we can construct the new simplified training and testing datasets pairs. Next, we utilise the new training datasets to train the EADP model and apply the model to the new testing datasets. Finally, we

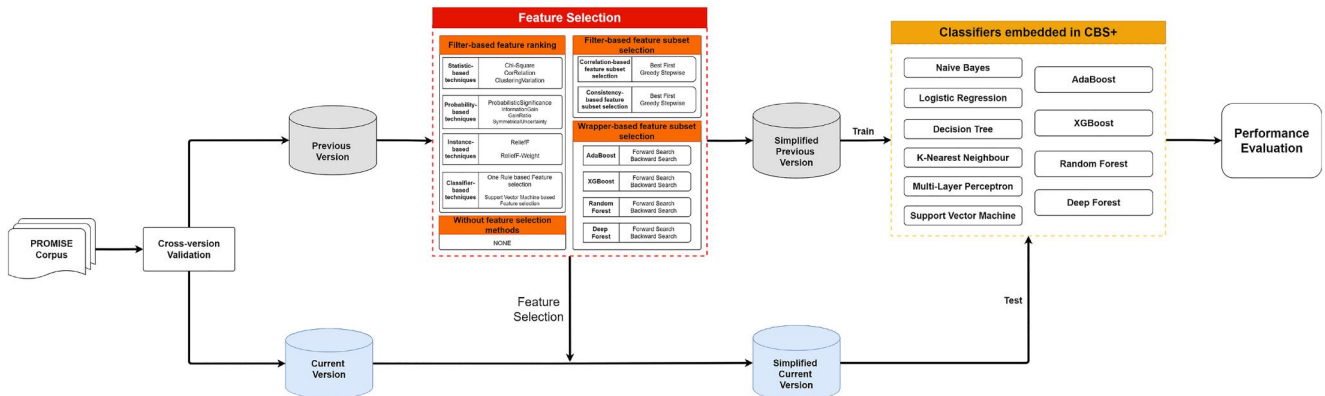


FIGURE 2 An overview of the experiment

calculate the corresponding performance measures and conduct the experimental evaluation.

### 3.4 | Classifiers

Huang et al. [17] deploy the LR as the base classifier in CBS+, and we want to investigate the impact of the feature selection methods with more classifiers embedded in CBS+ for EADP. Since we implement the CBS+ method by using the Python language, we prefer to choose the base classifiers embedded in CBS+ that can be implemented by using Python machine learning packages. Therefore, we construct the CBS+ model with 10 classifiers, as depicted in Figure 2. These classifiers fall into the six groups, including statistic-based (i.e., naive Bayes and logistic regression), decision tree-based (i.e., decision tree), nearest neighbour-based (i.e., K-nearest neighbour), neural network-based (i.e., multi-layer perceptron), support vector machine-based (i.e., support vector machine), and ensemble-based (i.e., AdaBoost, XGBoost, random forest, and deep forest).

- (1) **Naive Bayes (NB)** is based on the Bayes theorem and supposes that the software features are independent of each other. It greatly simplifies the complexity of Bayesian methods.
- (2) **Logistic Regression (LR)** adds a non-linear mapping (Sigmoid function) to linear regression, which makes it able to classify software modules into discrete outcomes.
- (3) **Decision Tree (DT)** is a tree function composed of multiple judgement nodes, where each non-leaf node is a feature attribute, each branch is the output of the feature attribute on a certain value range, and each leaf node is a class label.
- (4) **K-Nearest Neighbour (KNN)** decides the defect-proneness of the new software modules based on the defect-proneness of the nearest one or several software modules.
- (5) **Multi-Layer Perceptron (MLP)** is the popularisation of single-layer perceptron and contains an input layer, hidden layers, and an output layer. It trains the model with the back-propagation algorithm.
- (6) **Support Vector Machine (SVM)** is a generalised linear classifier that classifies data into binary categories with supervised learning. Its decision boundary is the hyper-plane with the maximum margin of the sample.
- (7) **AdaBoost (ADB)** employs an adaptive boosting that the weights of software modules misclassified by the previous basic classifier are increased, while the weights of software modules classified correctly are reduced, and the new weighted total instances are used again to train the next basic classifier.
- (8) **XGBoost (XGB)** is an optimised distributed gradient boosting algorithm and is robust to handle a variety of data types, relationships, and distributions.

- (9) **Random Forest (RF)** generates an ensemble model with basic decision trees. It randomly samples each instance to train different decision trees.
- (10) **Deep Forest (DF)** consists of non-differentiable decision trees. Its training process does not depend on the back-propagation algorithm and gradient calculation.

### 3.5 | Statistic test

We employ the Scott–Knott ESD test [30] as the statistic test method in our experiment, which is a multiple comparison technique for statistical analysis by using a hierarchical clustering algorithm. It divides the different feature selection methods into significantly different groups, where the feature selection methods in the same group have no significant difference, while the feature selection methods in different groups have a significant difference.

In our study, we use the double Scott–Knott ESD test to investigate the significant differences in the feature selection techniques. In the first phase, when we conduct the result analysis at the testing dataset level, we provide the performance measure values of the feature selection methods on 10 classifiers on each testing dataset to the first Scott–Knott ESD test and obtain the ranking of each feature selection method on each dataset; when we generate each feature selection method's ranking on each classifier, we provide the performance measure values of the feature selection methods on 30 testing datasets on each classifier for the first Scott–Knott ESD test.

In the second phase, we obtain the final rankings of the feature selection methods at the classifier level, with the 10 different Scott–Knott ESD rankings being the input to the second Scott–Knott ESD test; we input the 30 different Scott–Knott ESD rankings generated by the first test to the second Scott–Knott ESD test and generate the final rankings of the feature selection methods at the testing dataset level.

## 4 | EXPERIMENTAL RESULTS

### 4.1 | RQ1: Does CBS+ perform the best in the file-level EADP?

**Motivations:** Although CBS+ has been verified to have the best performance on some datasets (e.g., change-level projects [17], JavaScript projects [21], and Alibaba projects [22]), the performance of CBS+ on the file-level PROMISE datasets is still unknown. Therefore, we would like to figure out how CBS+ performs on our experimental datasets.

**Methods:** We compare CBS+ with the four regression models, including EALR [18], Ridge Regression (RR), Gradient Boosting Regression, and Random Forest Regression (RFR), and one unsupervised method called ManualUP [19], since Yu et al.'s [47] study showed the four regression models achieved better EADP performance and EALR and ManualUP were

widely used as the baseline methods in the previous EADP studies [17, 20, 22]. We first use the trained regression models to predict the bug numbers and then divide the bug numbers by LOC to obtain the predicted bug density. Table 3 shows the average values of the six evaluation metrics of the models. Figure 3 shows the performance distribution of the models more intuitively. The red, green, blue, yellow, and purple boxplots represent the first, second, third, fourth, and fifth Scott–Knott ESD rankings, respectively.

**Results:** According to Figure 3f, we can observe that the IFA values of the four regression methods and ManualUP are greater than 10, which is unacceptable in the EADP task. Previous studies have pointed out that software testers would not continue to inspect the predicted faulty modules if the first 10 modules were all false alarms [17]. Since software modules with fewer LOC tend to have higher defect density, the regression models tend to rank the modules with fewer LOC first, according to the predicted defect density. As an unsupervised method, ManualUP sorts the software modules in the ascending order of LOC. Because the modules with fewer LOC tend to be non-defective, the regression models and ManualUP result in more modules that need to be tested before the first bug is being found. In addition, we also find the PMI@20% values of the four regression methods and ManualUP are high, which indicates that software testers are required to put in more effort to inspect more modules. CBS+ with the 10 classifiers achieves low IFA values (less than 10), since it suggests that software testers inspect the predicted defective modules by the 10 classifiers first. The experiment results of the EADP models are consistent with those in the previous studies [17, 20, 22]. Therefore, we prefer CBS+ as our EADP model, since it obtains the acceptable IFA value and relatively high PofB@20%, Recall@20%, and Norm(*Popt*) value.

**TABLE 3** The average PofB@20%, Recall@20%, Norm(*Popt*), Precision@20%, PMI@20%, and IFA values of the different Effort-Aware Defect Prediction (EADP) models

Models		PofB@20%	Recall@20%	Norm( <i>Popt</i> )	Precision@20%	PMI@20%	IFA
CBS+	NB	0.235	0.241	0.493	0.507	0.170	6.100
	LR	0.273	0.280	0.613	0.502	0.245	3.567
	DT	0.295	0.309	0.574	0.444	0.240	7.633
	KNN	0.245	0.280	0.531	0.429	0.269	3.533
	MLP	0.296	0.331	0.607	0.448	0.316	3.500
	SVM	0.292	0.343	0.343	0.448	0.395	6.733
	ADB	0.300	0.320	0.593	0.427	0.265	4.300
	XGB	0.315	0.327	0.627	0.476	0.251	3.967
	RF	0.308	0.320	0.617	0.485	0.250	4.267
	DF	0.317	0.322	0.624	0.470	0.264	5.300
Regression	EALR	0.374	0.450	0.666	0.316	0.545	20.600
	RR	0.373	0.450	0.666	0.317	0.546	20.233
	GBR	0.371	0.442	0.680	0.317	0.546	16.967
	RFR	0.372	0.438	0.674	0.330	0.509	13.900
ManualUP		0.382	0.494	0.682	0.293	0.687	16.767

### Answer to RQ1

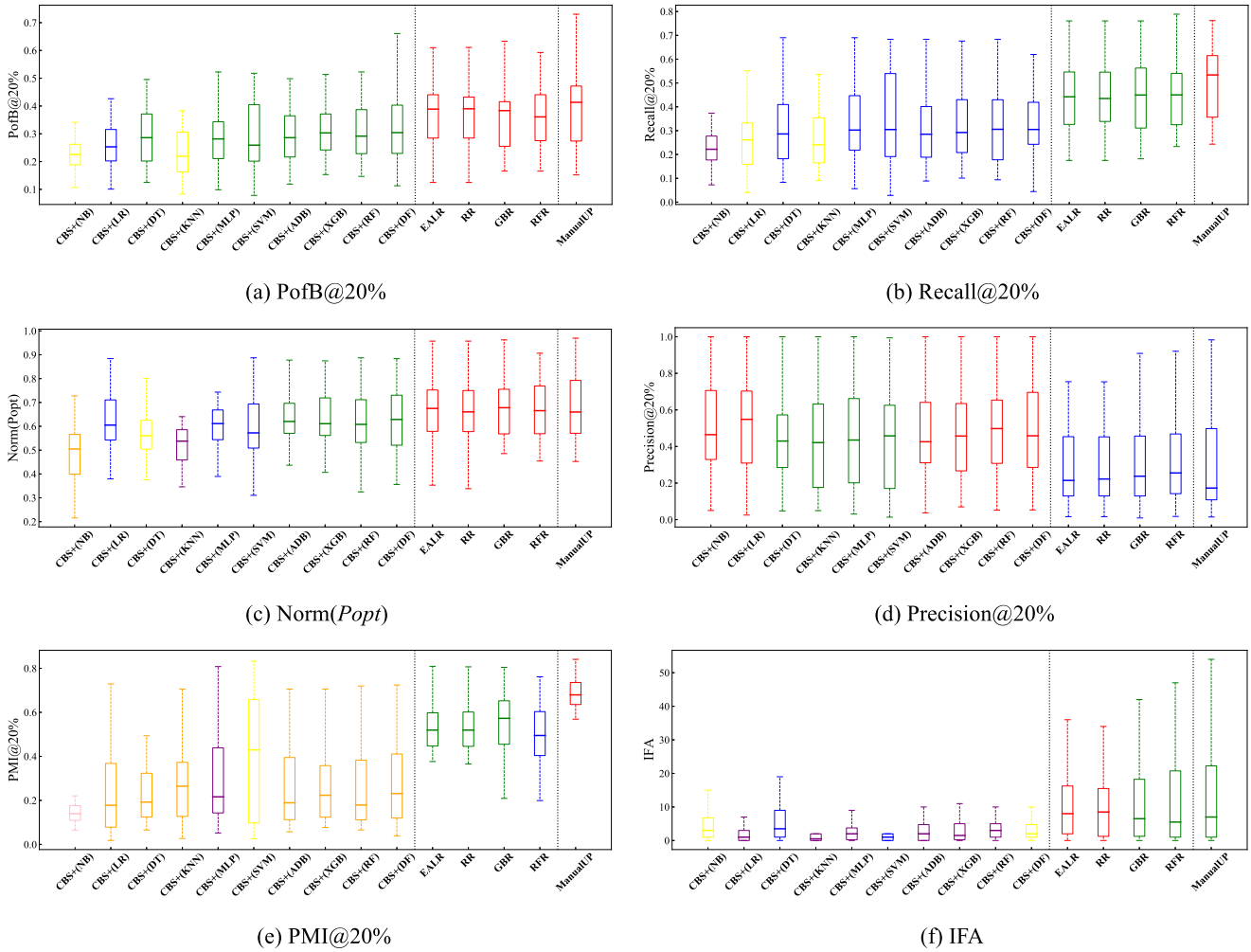
CBS+ performs better than the four regression models and ManualUP.

## 4.2 | RQ2: Do different feature selection methods affect the performance of different testing datasets for a given classifier?

**Motivations:** Previous researches have shown that irrelevant or redundant features often lead to poor classification performance of CBDP models. Some feature selection methods enhance the CBDP performance by filtering out the useless features, while others may have the opposite effect. It is observed that the CorBF usually performs the best for CBDP models according to Xu et al. [28] and Ghotra et al. [26]. However, it is still unknown whether the feature selection methods (including CorBF) can improve the EADP performance. Since we totally have 30 different training and testing datasets pairs through the cross-version validation method and there may be differences between different versions of datasets, it is necessary to explore the influence of feature selection methods on a large number of different datasets. Therefore, we conduct an experiment on the testing datasets level, discuss how different feature selection methods affect the performance of different testing datasets for each classifier, and finally find out which feature selection method(s) would perform the best.

**Methods:** We analyse the average value of the methods on each testing dataset across all 10 classifiers. We totally have 720





**FIGURE 3** The performance of different Effort-Aware Defect Prediction (EADP) models in terms of six evaluation metrics

(=24 feature selection methods  $\times$  30 testing datasets) average results across all classifiers in terms of each evaluation metric. Therefore, we employ the heat map to present the average result of each feature selection method on each testing dataset across all classifiers. Figures 4–6 depict the average value of different methods on each testing dataset across all classifiers in terms of six performance measures. It is noteworthy that the cell with darker colour represents the better performance in terms of PofB@20%, Recall@20%, Norm(*Popt*), and Precision@20%, while the cell with brighter colour indicates the better performance in terms of PMI@20% and IFA. Then, we conduct the Scott–Knott ESD test to investigate the statistically significant difference between each feature selection method. Figures 7–9 demonstrate the corresponding Scott–Knott ESD ranking values of different feature selection methods on each testing dataset in terms of six performance measures. The cell with darker colour represents the worse Scott–Knott ESD ranking result in terms of PofB@20%, Recall@20%, Norm(*Popt*), and Precision@20%, while the cell with brighter colour indicates the feature selection method obtains the better Scott–Knott ESD ranking in terms of PMI@20% and IFA. We further apply the second Scott–Knott

ESD test to get the final Scott–Knott ESD rankings of different feature selection techniques at the testing dataset level as shown in Figure 10.

**Results:** From these figures, it can be found that the four wrapper-based feature selection methods with forwards search, ADBF, DFF, RFF, and XGBF outperform the other families of feature selection methods. And among these four methods, RFF and XGBF perform the best. Considering the main objective of EADP, we first give the more detailed findings of the feature selection methods in terms of PofB@20%, Recall@20%, and Norm(*Popt*), then analyse the Precision@20%, PMI@20%, and IFA results.

- (1) As shown in Figure 4a, most of the feature selection methods obtain the high PofB@20% values on the two testing datasets (i.e., Velocity-1.5 and Xerces-1.2), and most of the feature selection methods perform worse on two testing datasets (i.e., Poi-2.5 and Xalan-2.5) in terms of **PofB@20%**. Additionally, the four wrapper-based feature selection methods with forwards search (including ADBF, DFF, RFF, and XGBF) obtain better performance on nearly all testing datasets. In particular, ADBF, DFF, RFF,

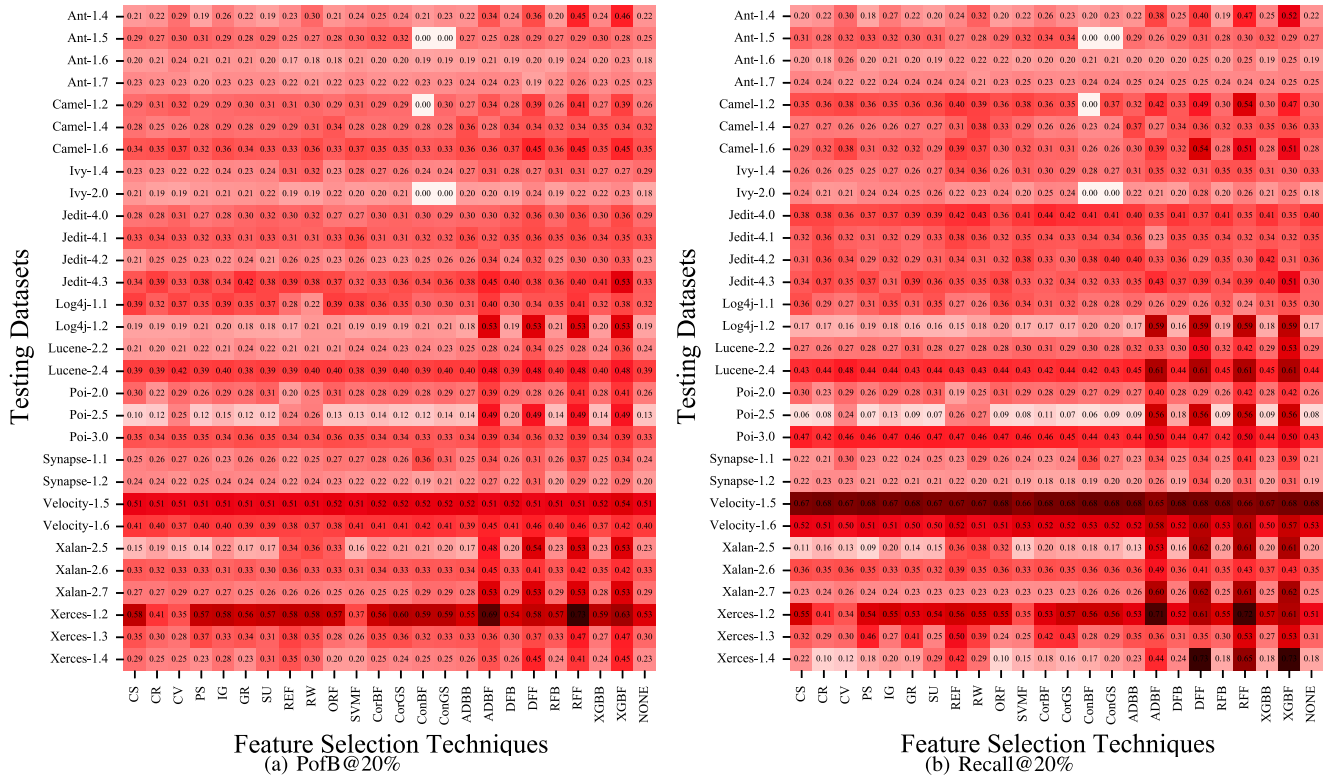


FIGURE 4 The average PofB@20% and Recall@20% values of these methods on each testing dataset across all classifiers

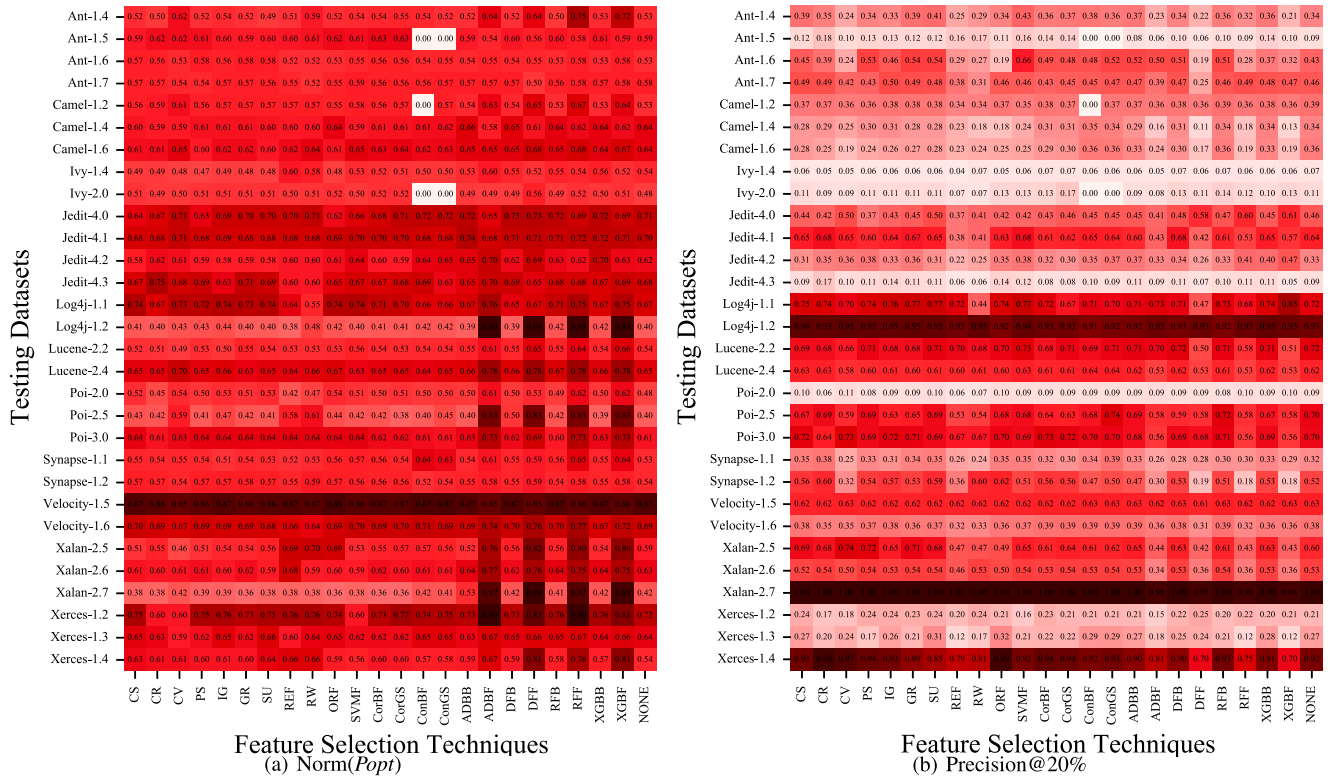


FIGURE 5 The average Norm(Popt) and Precision@20% values of these methods on each testing dataset across all classifiers

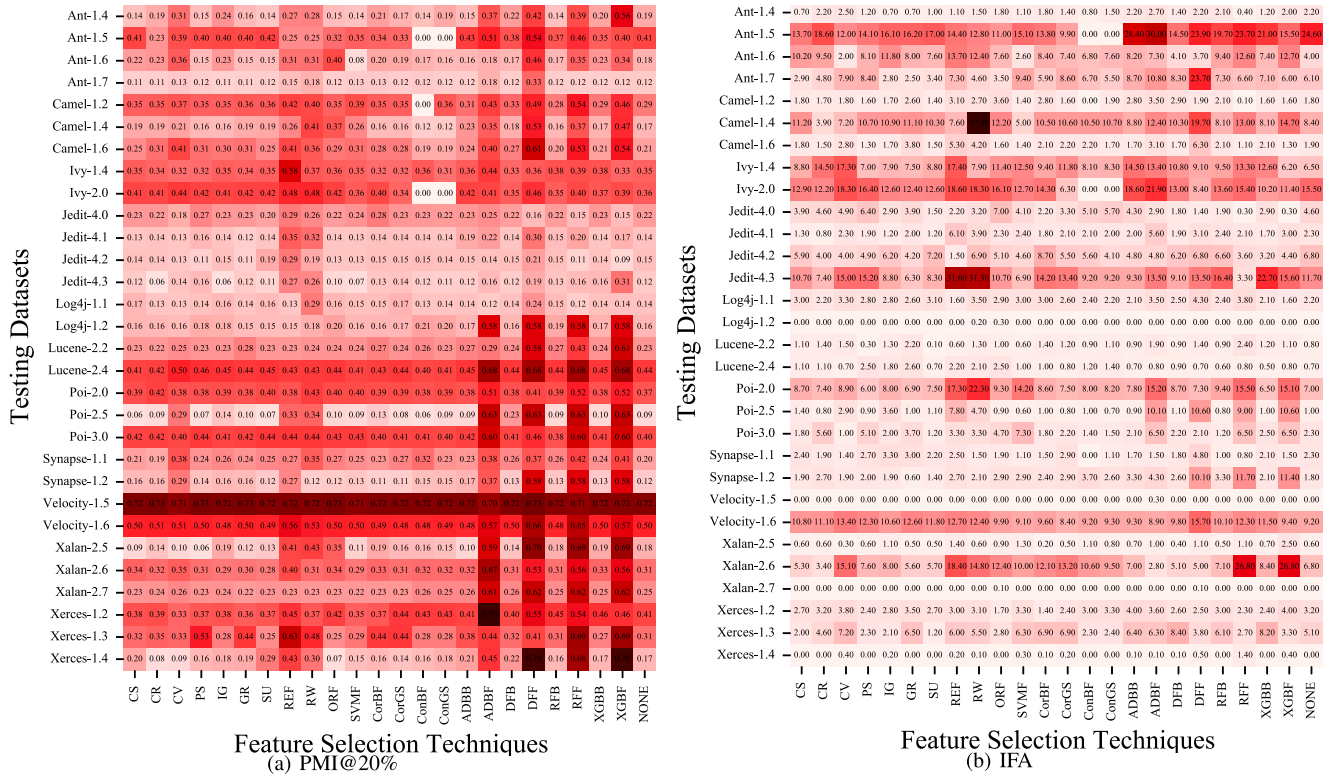


FIGURE 6 The average PMI@20% and IFA values of these methods on each testing dataset across all classifiers

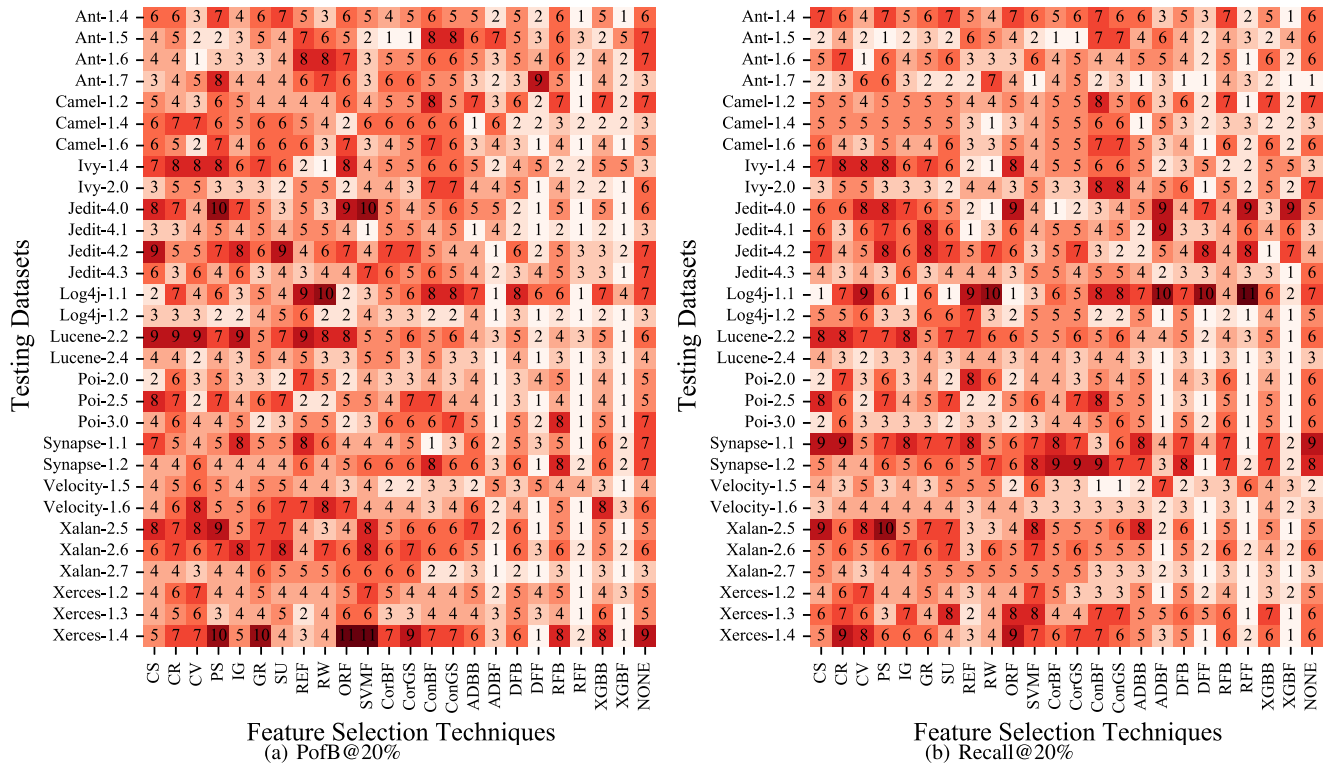


FIGURE 7 The Scott-Knott ESD ranking of the methods for each classifier across all testing datasets in terms of PofB@20% and Recall@20%.

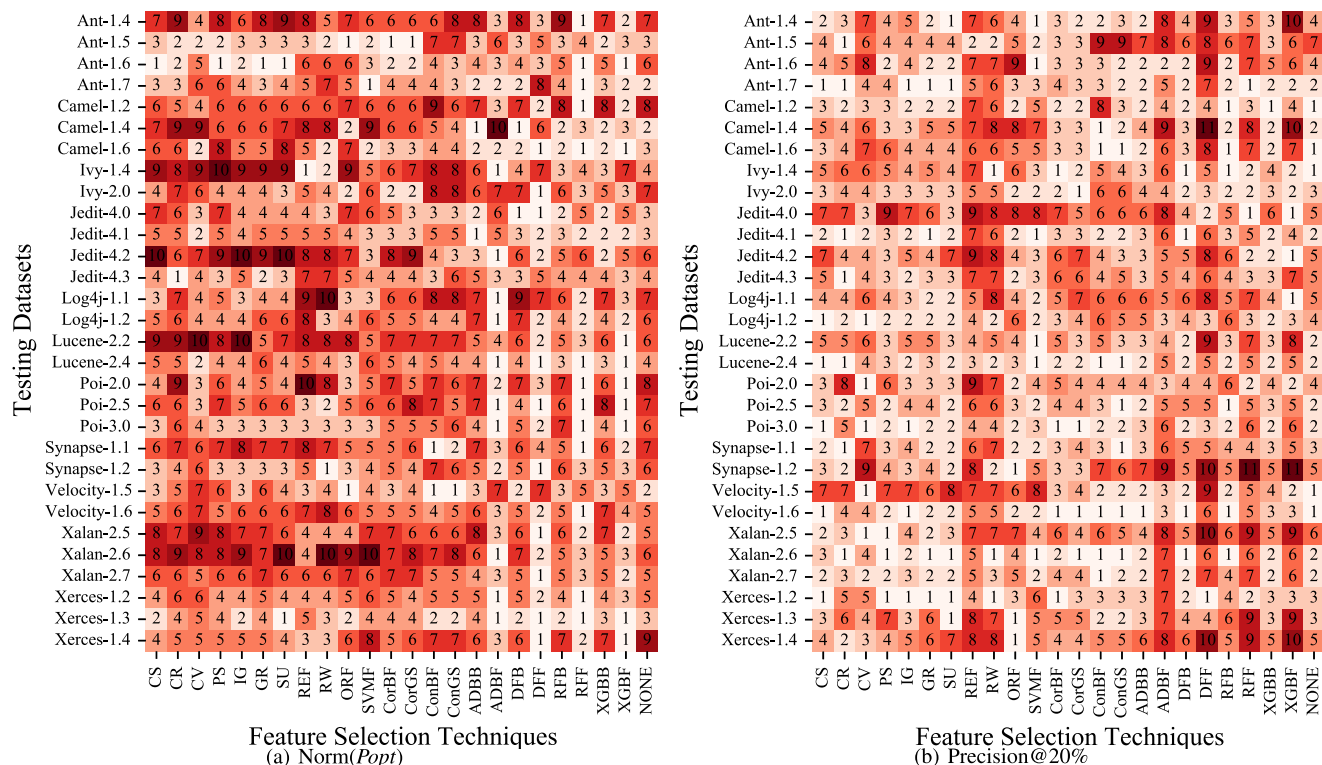


FIGURE 8 The Scott-Knott ESD ranking of the methods for each classifier across all testing datasets in terms of Norm( $P_{opt}$ ) and Precision@20%.

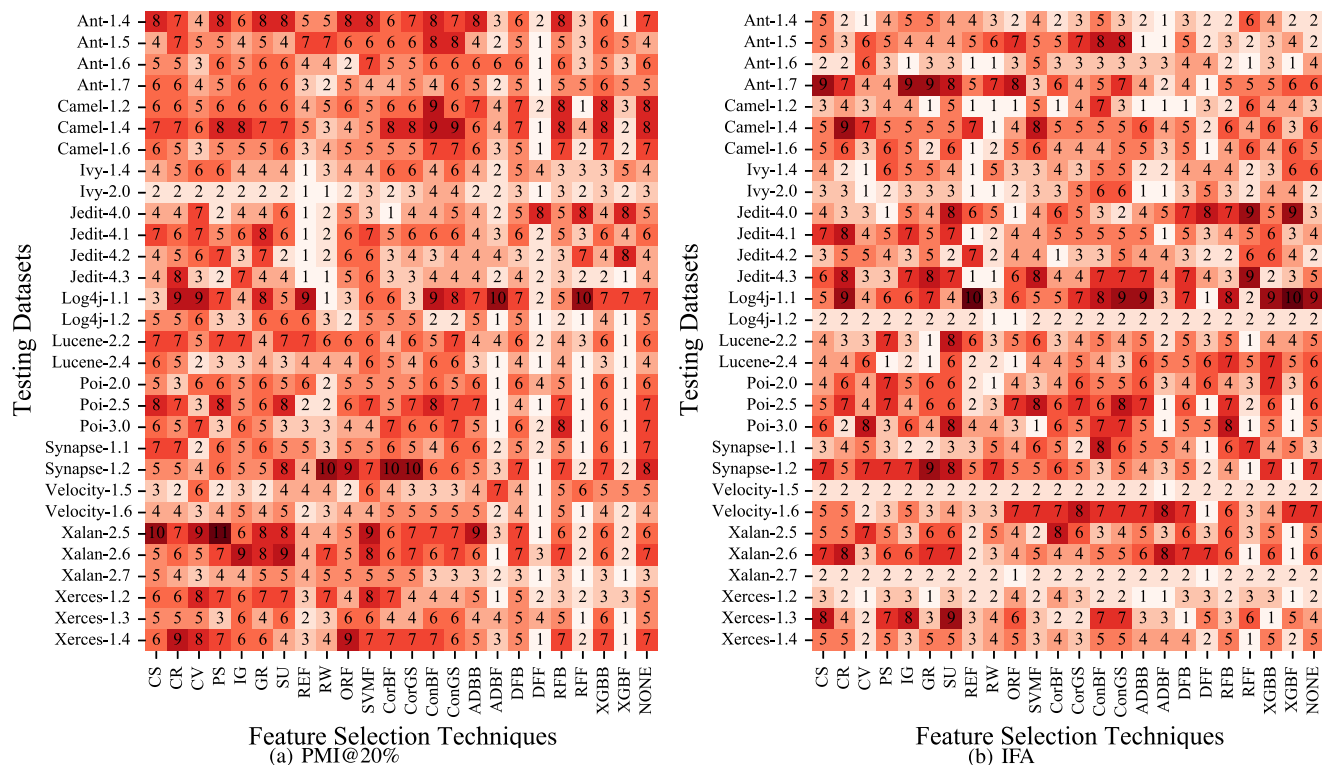
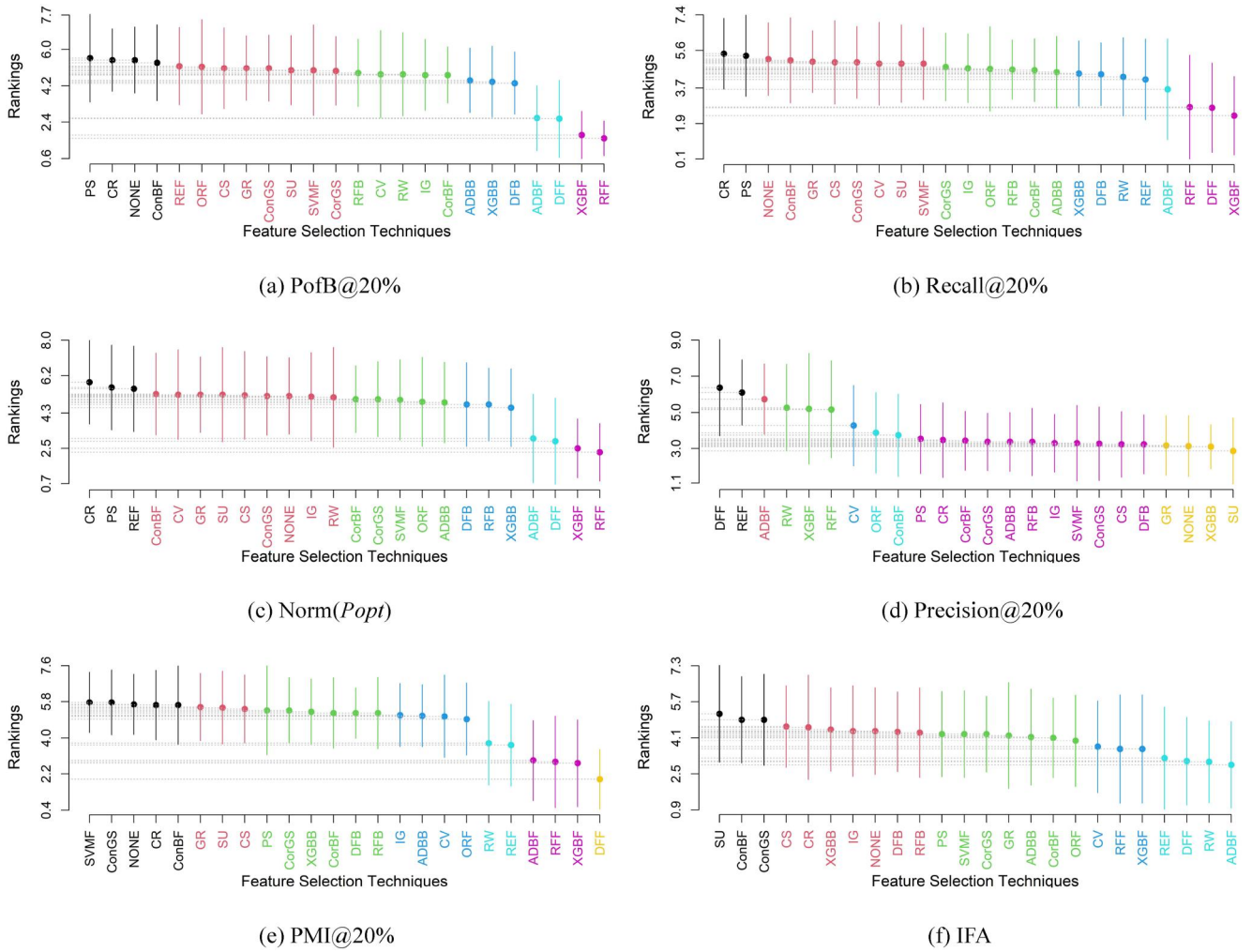


FIGURE 9 The Scott-Knott ESD ranking of the methods for each classifier across all testing datasets in terms of PMI@20% and IFA





**FIGURE 10** The final Scott-Knott ESD rankings of different feature selection techniques at the testing dataset level in terms of six evaluation metrics

- and XGBF achieve the highest PofB@20% values on Xerces-1.2 (0.69, 0.58, 0.73, and 0.63, respectively). From Figure 7a, we can find that ADBF, DFF, RFF, and XGBF fall into the top-3 Scott-Knott ESD ranking on 77%, 73%, 97%, and 90% of the testing datasets, respectively.
- (2) As shown in Figure 4b, most methods achieve the better performance on one testing dataset (i.e., Velocity-1.5), while most of the feature selection methods perform worse on two testing datasets (i.e., Poi-2.5 and Xalan-2.5) in terms of **Recall@20%**. Additionally, the four wrapper-based feature selection methods with forwards search (including ADBF, DFF, RFF, and XGBF) obtain better performance on nearly all testing datasets. In particular, ADBF and RFF obtain the highest Recall@20% values on Xerces-1.2 (0.71 and 0.72, respectively), and DFF and XGBF obtain the same highest Recall@20% values on Xerces-1.4 (0.73). From Figure 7b, we can also find that ADBF, DFF, RFF, and XGBF fall into the top-3 Scott-Knott ESD ranking on 60%, 80%, 83%, and 83% of the testing datasets, respectively.
- (3) As shown in Figure 5a, most of the feature selection methods obtain the higher performance values on one

- testing dataset (i.e., Velocity-1.5), but most of the feature selection methods perform worse on three testing datasets (including Log4j-1.2, Poi-2.5, and Xalan-2.7) in terms of **Norm(*Popt*)**. Additionally, the four wrapper-based feature selection methods with forwards search (including ADBF, DFF, RFF, and XGBF) obtain better performance on nearly all testing datasets. In particular, ADBF and RFF achieve the highest Norm(*Popt*) value on Xerces-1.2 (0.90), DFF achieves the highest Norm(*Popt*) value on Log4j-1.2 and Xalan-2.7 (0.89), and XGBF achieves the highest Norm(*Popt*) value on Log4j-1.2 (0.89). As shown in Figure 8a, ADBF, DFF, RFF, and XGBF fall into the top-3 Scott-Knott ESD ranking on 77%, 73%, 80%, and 83% of the testing datasets, respectively.
- (4) As shown in Figure 5b, most of the feature selection methods achieve better performance on three testing datasets (including Log4j-1.2, Xalan-2.7, and Xerces-1.4), while most of the feature selection methods perform worse on five testing datasets (including Ant-1.5, Ivy-1.4, Ivy-2.0, Jedit-4.3, and Poi-2.0) in terms of **Precision@20%**. Additionally, we noticed that even though the four wrapper-based feature selection methods with

forwards search (including ADBF, DFF, RFF, and XGBF) cannot perform the best on all testing datasets, they still obtain the acceptable Precision@20% values compared with other feature selection methods. As shown in Figure 8b, most feature selection methods fall into the top-3 Scott–Knott ESD ranking on over half of the testing datasets.

- (5) As shown in Figure 6a, most methods achieve better performance on over half of the testing datasets, while most of the feature selection methods perform worse on one testing dataset (i.e., Velocity-1.5) in terms of PMI@20%. Additionally, we noticed that even though the four wrapper-based feature selection methods with forwards search (including ADBF, DFF, RFF, and XGBF) cannot perform the best on all testing datasets, their PMI@20% values are still acceptable. From Figure 9a, we can find that ADBF, DFF, RFF, and XGBF fall into the top-3 Scott–Knott ESD ranking on 73%, 87%, 80%, and 73% of the testing datasets, respectively. In other words, using ADBF, DFF, RFF, and XGBF requires software testers to inspect more modules. But these feature selection methods can suggest that the software testing team finds more defects.
- (6) As shown in Figure 6b, almost all testing datasets (except for Ant-1.5, Camel-1.4, Ivy-1.4, Ivy-2.0, Jedit-4.3, Poi-2.0, Velocity-1.6, and Xalan-2.6) obtain better IFA values (which are less than 10) on nearly all feature selection methods. From Figure 9b, most of the feature selection methods fall into the top-3 Scott–Knott ESD ranking on over half of the testing datasets.
- (7) From Figure 7a–c, we can observe that compared with the None method (i.e., retaining all original features), most of the feature selection methods can achieve the similar or even better performance on most of the testing datasets across all classifiers in terms of PofB@20%, Recall@20%, and Norm(Popt). This shows the validity of the most feature selection methods across all classifiers, when they are applied to most testing datasets for the EADP task.
- (8) It is worth noting that the best-performing CorBF method in CBDP does not perform well on the EADP task. CorBF belongs to the first Scott–Knott ESD ranking in the best cases and the last ranking in the worse cases in terms of PofB20%, Recall@20%, and Norm(Popt). In particular, CorBF falls into the last-3 Scott–Knott ESD ranking on 33%, 43%, and 37% of the studied datasets in terms of the three metrics. In addition, CorBF can only achieve a similar performance with None on most testing datasets across all classifiers in terms of the three metrics. Obviously, the four wrapper-based feature selection methods with forwards search ADBF, DFF, RFF, and XGBF significantly outperform CorBF on the EADP task.
- (9) In summary, the four wrapper-based methods feature selection methods with forwards search (i.e., ADBF, DFF, RFF, and XGBF) outperform other feature selection methods. Furthermore, from Figure 10a,b we can observe that both RFF and XGBF rank the first in terms of PofB@20% and Recall@20%, and from Figure 10c we can

notice that RFF rank first in terms of Norm(Popt) among all the feature selection methods. Therefore, we conclude that the two feature selection methods (i.e., RFF and XGBF) outperform the others and we recommend employing RFF and XGBF as the feature selection methods on most testing datasets in EADP tasks when considering the primary objective of EADP.

#### Answer to RQ2

The two wrapper-based feature subset selection methods (i.e., RFF and XGBF) perform the best and both obtain high rankings in terms of PofB@20%, Recall@20%, and Norm(Popt).

### 4.3 | RQ3: Do different feature selection methods affect the performance of different classifiers for a given testing dataset?

**Motivations:** Similar to RQ2, how different feature selection methods affect the performance of EADP models with different classifiers for each testing dataset is still unknown. Therefore, we conduct an experiment on the classifier level and finally find out which feature selection method(s) perform the best.

**Methods:** We analyse the average values of these methods for each classifier across all 30 testing datasets. We totally have 240 (=24 feature selection methods × 10 classifiers) mean results across all testing datasets in terms of each evaluation metric. Therefore, we employ the heat map to present the average result of the methods for each classifier across all testing datasets. Figure 11 depicts the average value of different methods for each classifier across all testing datasets in terms of six evaluation metrics. Then, we conduct the Scott–Knott ESD test to investigate the statistically significant difference between each feature selection method. Figure 12 demonstrates the corresponding Scott–Knott ESD ranking values of different methods for each classifier in terms of six performance measures. Similar to RQ2, we further apply the second Scott–Knott ESD test to get the final Scott–Knott ESD rankings of different feature selection techniques at the classifier level as shown in Figure 13.

**Results:** From these figures, we can find that the four wrapper-based feature selection methods with forwards search, ADBF, DFF, RFF, and XGBF perform the best. Similar to RQ2, we also first analyse the PofB@20%, Recall@20%, and Norm(Popt) results and then provide the findings in terms of Precision@20%, PMI@20%, and IFA.

- (1) As shown in Figure 11a, MLP achieves better performance, while NB and KNN perform worse than other classifiers on most feature selection methods in terms of PofB@20%. Additionally, RFF and XGBF obtain better

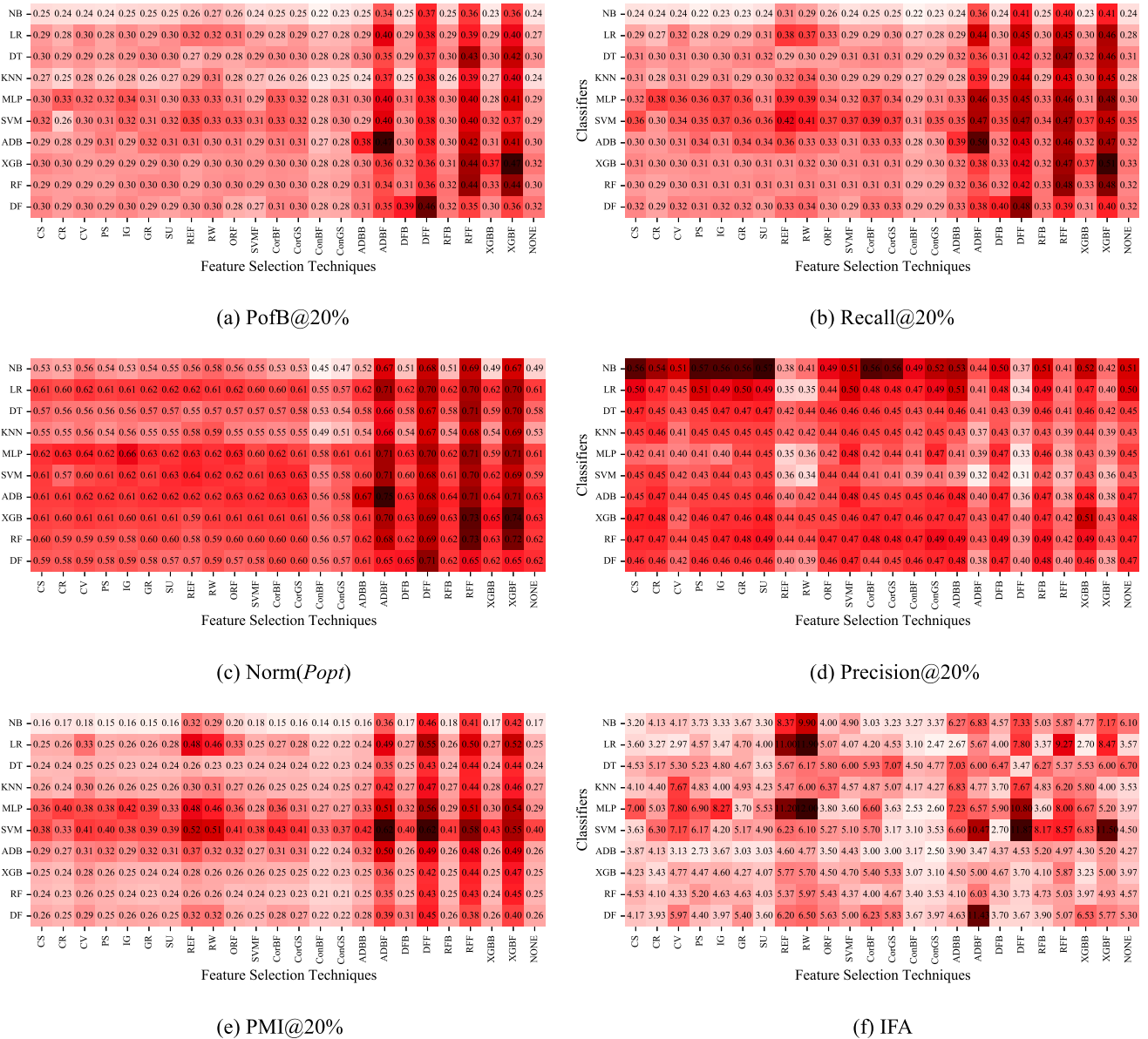
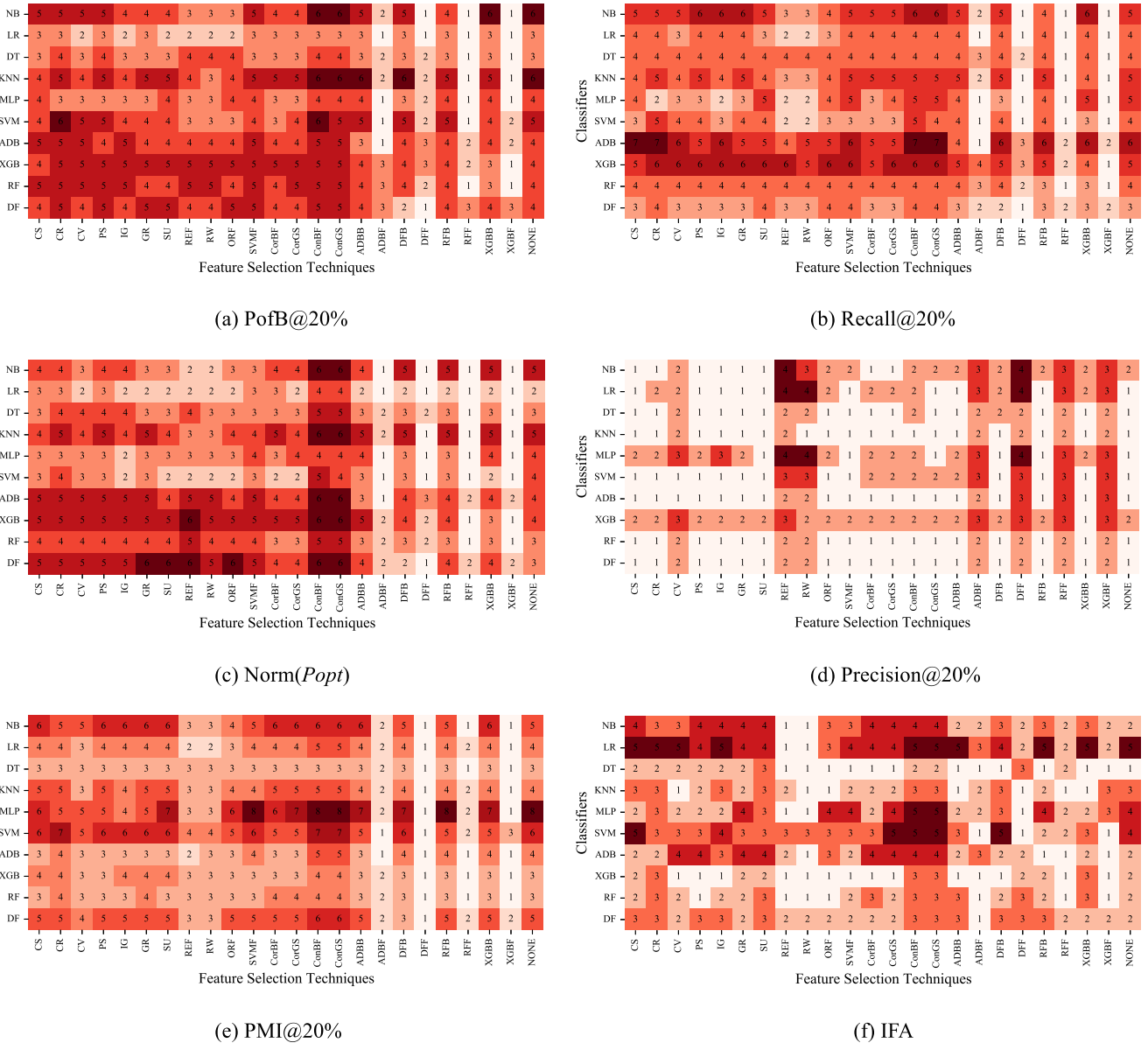


FIGURE 11 The six average metrics of these methods for each classifier across all testing datasets

performance on almost all classifiers. Except for the four wrapper-based feature selection methods with forwards search (including ADBF, DFF, RFF, and XGBF), the other methods do not perform well on almost all classifiers. In addition, ADBF, DFF, RFF, and XGBF achieve comparatively high PofB@20% values with their corresponding classifiers, that is, ADBF with the AdaBoost classifier (0.47), DFF with the deep forest classifier (0.46), RFF with the random forest classifier (0.44), and XGBF with the XGBoost classifier (0.47). From Figure 12a, we can find that most feature selection methods fall into the top-4 Scott–Knott ESD ranking across over half of the classifiers. ADBF, DFF, RFF, and XGBF fall into the top-3 Scott–Knott ESD ranking for all classifiers. In particular, RFF and XGBF rank first on seven classifiers.

- (2) From Figure 11b, except for NB, the rest of the classifiers perform better on most of the feature selection methods in terms of **Recall@20%**. RFF and XGBF perform better on almost all classifiers. In addition, ADBF with ADB, DFF with DF, RFF with RF, and XGBF with XGB achieve comparatively high Recall@20% values (0.50, 0.48, 0.48, and 0.51, respectively). As shown in Figure 12b, most feature selection methods fall into the top-4 Scott–Knott ESD ranking across over half of the classifiers, in which DFF, RFF, and XGBF fall into the top-3 Scott–Knott ESD ranking for all classifiers. Especially, RFF and XGBF rank first on seven and eight classifiers, respectively.
- (3) As shown in Figure 11c, except for NB and KNN, the rest of the classifiers perform better on most of the feature selection methods in terms of **Norm( $P_{opt}$ )**. Additionally,



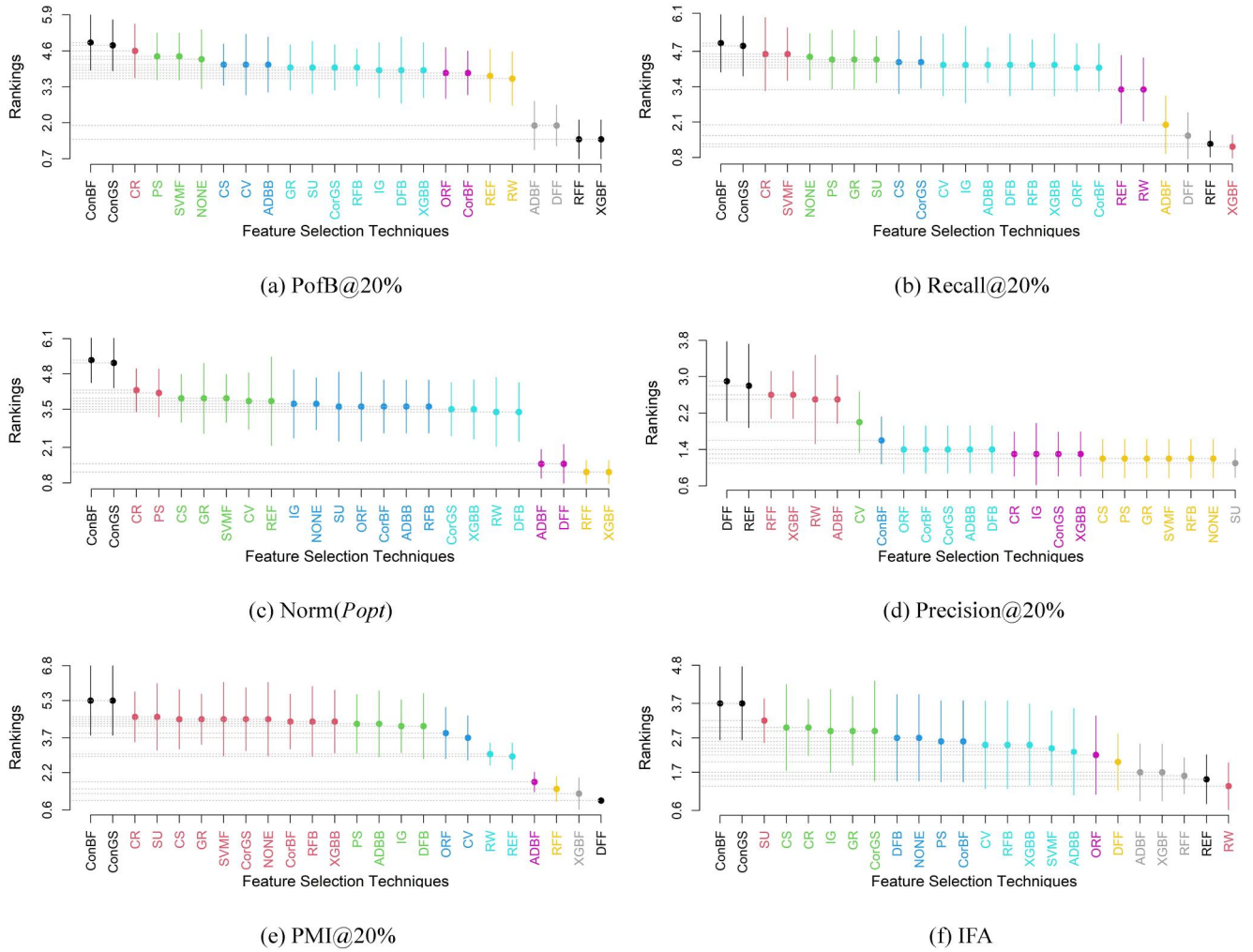
**FIGURE 12** The Scott-Knott ESD ranking of the methods for each classifier across all testing datasets in terms of six evaluation metrics

the four wrapper-based feature selection methods with forwards search (including ADBF, DFF, XGBF, and RFF) obtain the better performance on nearly all classifiers and achieve the highest Norm( $P_{opt}$ ) values with their corresponding classifiers (0.75, 0.71, 0.73, and 0.74, respectively.) From Figure 12c, most feature selection methods rank in the top-4 across over half of the classifiers. In particular, ADBF, DFF, RFF, and XGBF fall into the top-3 Scott-Knott ESD ranking for all classifiers, and RFF and XGBF fall into the first Scott-Knott ESD ranking on eight classifiers.

- (4) From Figure 11d, we can find that NB outperforms other classifiers when employing most methods in terms of **Precision@20%**. As shown in Figure 12d, nearly all methods fall into the top-3 Scott-Knott ESD ranking across nearly all classifiers.

- (5) From Figure 11e, we can observe that nearly all of the classifiers perform better when using most of the feature selection methods, in which NB performs the best in terms of **PMI@20%**. As shown in Figure 12e, most of the feature selection methods fall into the top-3 Scott-Knott ESD ranking across over half of the classifiers in which ADBF, DFF, RFF, and XGBF fall into the top-3 Scott-Knott ESD ranking for all classifiers. In other words, using ADBF, DFF, RFF, and XGBF requires software testers to inspect more modules.
- (6) From Figure 11f, almost all classifiers obtain better **IFA** values (which are less than 10) on nearly all feature selection methods. In addition, ADBF with the ADB classifier, DFF with the DF classifier, RFF with the RF classifier, and XGBF with the XGB classifier achieve low IFA values (3.47, 3.67, 5.03, and 5.00, respectively). From





**FIGURE 13** The final Scott-Knott ESD rankings of different feature selection techniques at the classifier level in terms of six evaluation metrics

Figure 12f, we can find that most methods fall into the top-3 Scott-Knott ESD ranking on over half of the classifiers.

- (7) From Figure 12a–c, it is observed that compared to the None method (i.e., retaining all original features), except for ConBF and ConGS, most of the feature selection methods can achieve the similar or better performance on most of the classifiers across all testing datasets in terms of PofB@20%, Recall@20%, and Norm(*Popt*). In other words, using fewer software features selected by most feature selection methods can have similar performance or enhance the overall performance of most classifiers.
- (8) Similar to RQ2, the best-performing CorBF method in CBDP does not perform well on the EADP task. CorBF belongs to the third Scott-Knott ESD ranking in the best cases and the last ranking in the worse cases in terms of PofB@20%, Recall@20%, and Norm(*Popt*). In particular, CorBF falls into the last two Scott-Knott ESD ranking on 80%, 70%, and 50% of the studied classifiers in terms of the three metrics. Additionally, we can observe that compared with the None method, CorBF can only achieve similar performance on most classifiers across all testing

datasets in terms of the three metrics. Obviously, ADBF, DFF, RFF, and XGBF significantly outperform CorBF on the EADP task.

- (9) In summary, the four wrapper-based methods (i.e., ADBF, DFF, RFF, and XGBF) outperform other feature selection methods. Furthermore, from Figure 13a and c, we can observe that both RFF and XGBF rank first in terms of PofB@20% and Norm(*Popt*), and from Figure 13b, we can notice that XGBF rank first in terms of Recall@20% among all the feature selection methods. Based on this, Figure 11a–c show that XGBF with XGBoost as the embedded classifier in CBS+ obtains the highest average PofB@20%, Recall@20%, and Norm(*Popt*) on the testing datasets. In addition, XGBF with XGBoost as the embedded classifier belongs to the top-1 ranking in terms of PofB@20%, Recall@20%, and Norm(*Popt*) from Figure 12a–c. Therefore, since the primary objective of EADP is to find more bugs, more defective modules, or obtain a more correct global ranking of software modules based on the defect density, we recommend to employ XGBF with its corresponding classifier (i.e., XGBoost) as the feature selection method.

**TABLE 4** The selected features by AdaBoost with Forwards Search (ADBF), Deep Forest with Forwards Search (DFF), Random Forest with Forwards Search (RFF), and XGBoost with Forwards Search (XGBF) on each dataset

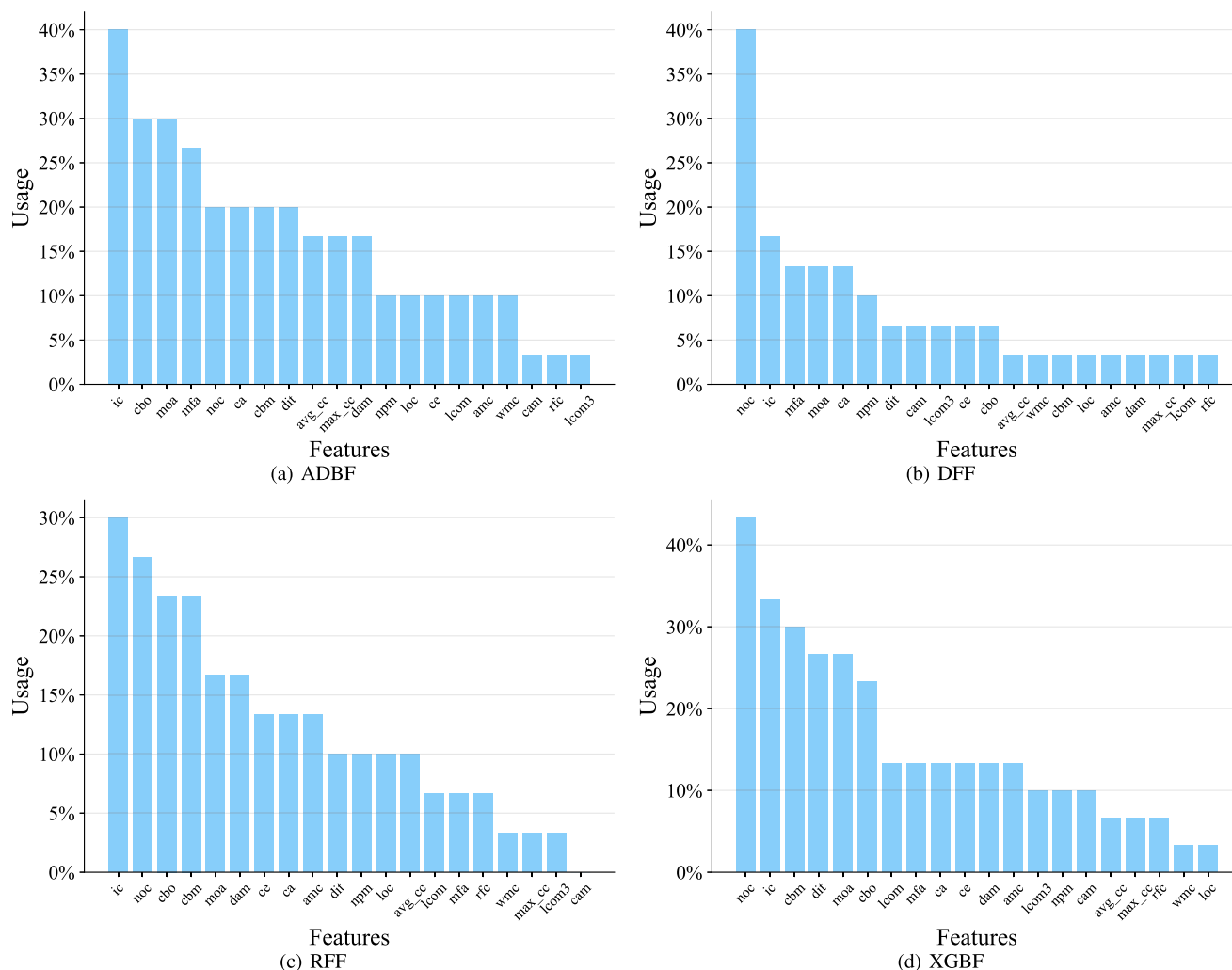
Datasets	ADBF	DFF	RFF	XGBF
Ant-1.4	ic, mfa, cbo, avg_cc	mfa, avg_cc	ce	wmc
Ant-1.5	loc, amc	npm, noc, dam	loc, ic	noc, cbo, ce, dit, dam, lcom, ic, cam
Ant-1.6	loc, rfc, lcom	npm	cbo, amc, noc	noc, cbo, amc
Ant-1.7	ic, mfa, amc, ce, dit, dam, cbm	lcom3	cbo, amc, ce, dam, avg_cc	rfc, ce, cbm, lcom3, moa
Camel-1.2	cbo	ca, noc	noc, dam	noc, dit
Camel-1.4	max_cc	moa	ca, cbm	ca
Camel-1.6	moa, avg_cc, mfa, dam	moa	moa, noc, dam	moa, noc, dam
Ivy-1.4	ca, max_cc, avg_cc, cbm, ic, moa, mfa, dam, dit	ce, cbo, dit	ca, avg_cc, lcom, ic, dam, mfa, lcom3	ca, avg_cc, mfa, max_cc, cbm, ic, moa, cbo, dam, amc, lcom3, noc, dit, cam, rfc, ce, lcom
Ivy-2.0	cbo, ic, mfa, cbm, moa, dam, amc, dit	moa, cbo	loc, ic, ce, rfc, dit, moa, cbm	npm, lcom, mfa, cbm, loc, moa, ic, dit, cam
Jedit-4.0	cbo, avg_cc	moa, wmc	moa	moa
Jedit-4.1	cbo, noc	dit, mfa, cam, ic, cbm	dit, ic, cbo, lcom, noc	dit, ic, cbo, lcom, lcom3
Jedit-4.2	ca, moa, mfa	ca, lcom3	npm, ic	npm, cbm
Jedit-4.3	npm, loc, max_cc, ic, moa, ce	cam, loc, ic, amc	amc	mfa, moa, ic, noc, dit
Log4j-1.1	cbo, ic	ca, mfa	cbo	wmc, noc
Log4j-1.2	noc	noc	noc	noc
Lucene-2.2	cbo, max_cc, lcom, npm	noc	cbm	noc
Lucene-2.4	ic	ic	ic	ic
Poi-2.0	moa, ca, noc	noc, npm	moa, ca	moa, ca
Poi-2.5	dit	dit	dit	dit
Poi-3.0	moa	max_cc	moa	moa
Synapse-1.1	ce, wmc, ca, ic, mfa, cbm, moa, cbo, noc, lcom, dam, dit	lcom, ca, ic	cbo, mfa, noc	cbo, mfa, cbm, ic, noc, dit
Synapse-1.2	cam, dit	noc	ic	ic
Velocity-1.5	wmc, npm, max_cc	ce	amc, rfc, npm, loc, ca, wmc, dam	amc, cbo, cbm
Velocity-1.6	cbo, cbm, moa, ic, ca	mfa	cbo, cbm	ca, avg_cc, noc
Xalan-2.5	cbm, ic	noc	ic	ic
Xalan-2.6	noc	ic	cbm	cbm
Xalan-2.7	ic, cbm	noc	cbm	cbm
Xerces-1.2	ic	noc	noc, ic, max_cc, amc, cbo, avg_cc, cbm	ic, max_cc, amc, dam, cbo, cbm, ce
Xerces-1.3	wmc, avg_cc	noc, rfc	npm	npm
Xerces-1.4	mfa, dit, lcom3	noc	dit	noc

### Answer to RQ3

XGBF with XGBoost as the embedded classifier in CBS+ performs the best and obtains the highest average PofB@20%, Recall@20%, and Norm(Popt) values on the testing datasets.

### 4.4 | RQ4: Which features are frequently selected by the four wrapper-based methods?

**Motivations:** Generally, some features are frequently selected by different feature selection methods in different datasets. It means that these features make a greater contribution to the construction of EADP models. Therefore, to explore which features help construct the EADP models more effectively, we



**FIGURE 14** The usage of each software feature among 30 testing datasets when applying four feature selection methods

conduct an experiment and discuss which features are frequently selected by the four wrapper-based feature selection methods with forwards search.

**Methods:** Table 4 shows the selected features after applying the four feature selection methods (i.e., ADBF, DFF, RFF, and XGBF) on each dataset. Figure 14a–d present the usage percentage of the 20 software features among the 30 studied testing datasets by the four feature selection methods, respectively. Among these figures, a 100% for one feature means it is selected in all 30 testing datasets, while a 0% for one feature indicates none of the feature selection methods chooses the feature.

**Results:** (1) The selected features are different by the four feature selection methods on the same dataset. The potential reason is that the four wrapper-based methods employ different classifiers to find the best feature subsets. (2) The selected features by the same feature selection method vary with different versions of the same project. For example, ADBF selects *cbo* on Camel-1.2, *max\_cc* on Camel-1.4, and *moa*, *avg\_cc*, *mfa*, and *dam* on Camel-1.6. (3) Almost all features (except for *cam*) are selected by the four feature selection methods, which indicates that they have played important roles in the construction of

EADP models. In addition, the most frequently selected features are *noc*, *ic*, *cbo*, and *cbm*. In detail, the feature *noc* is selected in 20%, 40%, 26.67%, and 43.33% of testing datasets when applying ADBF, DFF, RFF, and XGBF, respectively; The feature *ic* is selected in 40%, 30%, and 33.33% of testing datasets when applying ADBF, RFF, and XGBF, respectively; The feature *cbo* is selected in 30%, 23.33%, and 23.33% of testing datasets applying ADBF, RFF, and XGBF, respectively; The feature *cbm* is selected in 20%, 23.33%, and 30% of testing datasets when applying ADBF, RFF, and XGBF, respectively. In other words, these features are more useful when building EADP models.

#### Answer to RQ4

The selected features vary with different feature selection methods and different datasets, and *noc*, *ic*, *cbo*, and *cbm* are the frequently selected features by the four wrapper-based feature selection methods with forwards search.

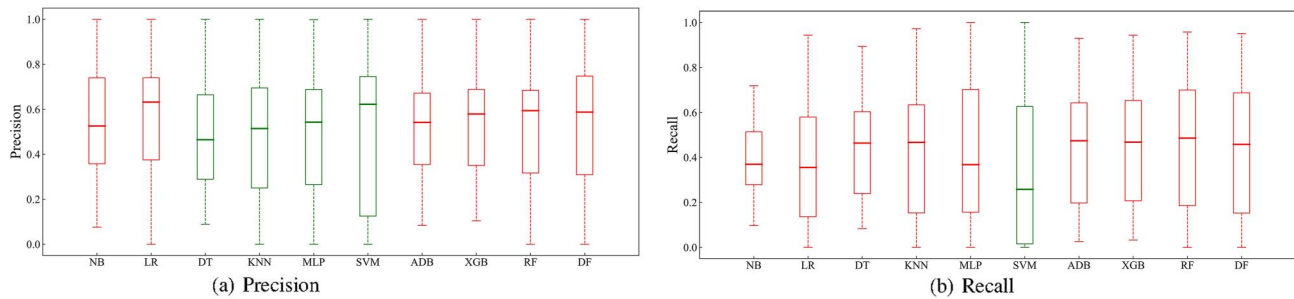


FIGURE 15 The performance of the different classifiers in terms of the two classification metrics

#### 4.5 | RQ5: Which is the best classifier embedded in CBS+?

**Motivation (1):** The original CBS+ uses LR as the base classifier, while the original Effort-Aware Supervised Cross-project defect prediction (EASC) employs NB. The previous studies [48–50] show the superiority of ensemble learning algorithms for CBDP. Therefore, to explore which classifier is the most suitable for constructing EADP models, we utilise six machine learning techniques, that is, NB, LR, DT, KNN, MLP, and SVM, and four ensemble-based methods, that is, ADB, RF and the two advanced deep ensemble learning algorithms (i.e., DF and XGB) as the base classifiers.

**Method (1):** We first use Figure 15 to show the classification performance (i.e., the Precision and Recall values) of the classifiers using all original features across all testing datasets. Then, Figure 3 shows the distribution of the PofB@20%, Recall@20%, Norm(*Popt*), Precision@20%, PMI@20%, and IFA values of CBS+ embedding the different classifiers using all original features across all testing datasets. The red, green, blue, yellow, purple, and orange boxplots represent the first, second, third, fourth, fifth, and sixth Scott–Knott ESD rankings, respectively.

**Result (1):** ADB, DF, RF, and XGB perform the best in terms of Precision and Recall. It indicates that the four ensemble learning algorithms indeed outperform other classification algorithms for CBDP. Except for the regression models and ManualUP in Figure 3, NB belongs to the third Scott–Knott ESD ranking in terms of PofB@20%, the fourth ranking in terms of Recall@20%, and the fifth ranking in terms of Norm(*Popt*); LR belongs to the second Scott–Knott ESD ranking in terms of PofB@20% and Norm(*Popt*), and the third in terms of Recall@20%; ADB, DF, RF, and XGB perform the best, since they fall into the first Scott–Knott ESD ranking in terms of Precision@20%, Recall@20%, and PofB@20%.

In addition, the four classifiers also achieve the acceptable PMI@20% and IFA values. Therefore, we suggest that researchers employ ADB, DF, RF, and XGB as the base classifiers embedded in CBS+.

**Motivation (2):** Based on the above-mentioned results, we can conclude that the four ensemble learning classifiers can better construct the EADP model. So in order to further explore the relationship between these four methods in the classification performance and the effort-aware performance, we conduct an

experiment and discuss the correlation among the performance of the four ensemble learning classifiers embedded in CBS+.

**Method (2):** In addition, we make a correlation analysis to investigate the relationship between the classification performance and the effort-aware performance more intuitively. To answer RQ5, we totally consider eight performance measures, including six effort-aware evaluation metrics and two classification-based evaluation metrics. We calculate the Pearson correlation coefficient (i.e.,  $r$ ) and make a correlation analysis among the performance of the four ensemble learning classifiers embedded in CBS+ on all datasets. We employ the heat map to present the Pearson correlation coefficient. Different colours represent different degrees of correlation between each two evaluation metrics. According to Hinkle et al. [51], the correlation 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 ( $0.9 \leq |r| \leq 1$ ).

**Result (2):** As shown in Figure 16a–d, we can observe that: (1) Recall has a high or moderate correlation with both PofB@20% (0.69, 0.58, 0.73, and 0.68, respectively) and Norm(*Popt*) (0.52, 0.52, 0.64, and 0.60, respectively). It is obvious that more defective modules are ranked first by an EADP model, the higher PofB@20% and Norm(*Popt*) values of the model has. (2) PofB@20% has a moderate correlation with PMI@20% (0.55, 0.53, 0.60, and 0.69, respectively), since more software modules have been inspected, more defects are likely to be discovered. (3) Precision@20% has a moderate correlation with IFA (−0.45, −0.65, −0.57, and −0.56, respectively), since if there are many false alarms in the top 20% LOC, the IFA will be more likely to reach a high value. (4) Precision has a very high correlation with Precision@20% on these four classifiers (0.97, 0.87, 0.98, and 0.96, respectively) and Recall has a high correlation with Recall@20% (0.74, 0.80, 0.87, and 0.85, respectively). It indicates that the better classification performance of the algorithms can contribute to the superiority of CBS+. Therefore, we suggest that researchers employ the more superior classifiers (e.g., ADB, DF, RF, and XGB) as the base classifiers embedded in CBS+.

#### Answer to RQ5

ADB, DF, RF, and XGB embedded in CBS+ perform the best.



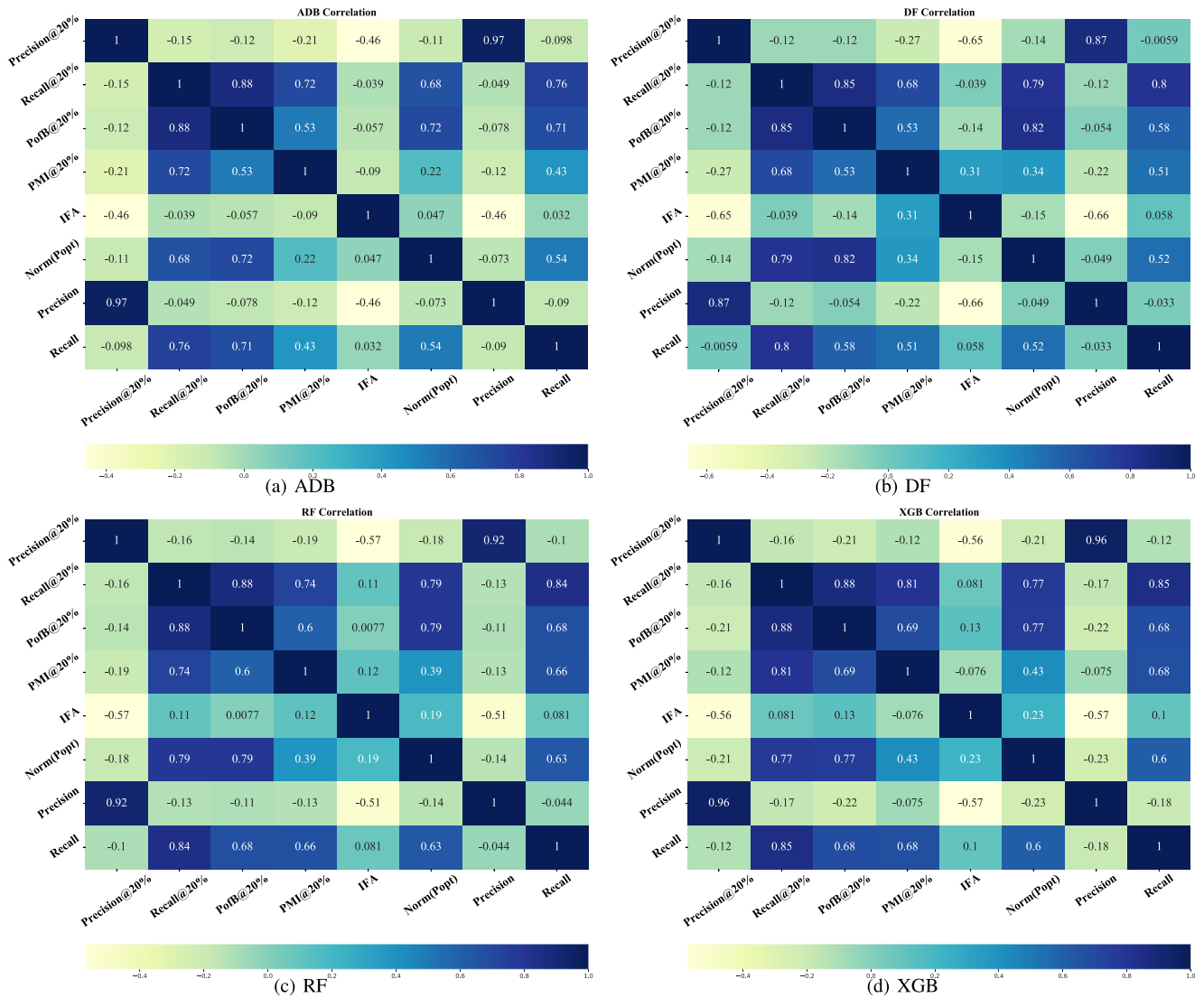


FIGURE 16 The correlation between evaluation metrics in terms of four classifiers

## 5 | DISCUSSION

### 5.1 | Performance interpretation

Sections 4.2 and 4.3 demonstrate that the four wrapper-based feature selection methods (i.e., ADBF, DFF, RFF, and XGBF) outperform others. The superiority of ADB, DF, RF, and XGB in Section 4.5 can also contribute to explaining the reason why the four wrapper-based feature subset selection methods perform the best. The four wrapper-based methods employ PoFB@20% as the evaluation criterion and find the best subset. In these methods, the EADP model is employed as a black box to evaluate the performance of the best feature subset found in the search process. Due to the application of the EADP model built with the best-performing base classifiers (i.e., ADB, DF,

RF, XGB) for the feature subset, the four wrapper methods can provide better performance.

The experimental results in Section 4.3 indicate that the two feature selection methods, (i.e., ConBF and ConGS) achieve even worse performance compared with None (i.e., without feature selection). ConBF and ConGS select the feature subset whose consistency is equal to the consistency of all original features with the BF strategy and GS strategy, respectively. In other words, both of them ignore the correlation between features and the class label in the process of feature selection. Additionally, except for the two methods, most of the feature selection methods we investigated can outperform None. It is obviously demonstrated that the performance of EADP models can be improved by eliminating redundant or irrelevant features.

## 5.2 | Implications

We summarise the main suggestions according to our results for the future EADP study.

(1) **Researchers and practitioners should consider employing XGBF with XGBoost as the embedded classifier in CBS+ to enhance the EADP performance.** From the results shown in Section 4.2 and Section 3, we find that XGBF with XGBoost as the embedded classifier in CBS+ performs the best in terms of PofB@20%, Recall@20%, and Norm(*Popt*) on both the classifier level and dataset level. We investigate almost the same feature selection methods as Ghotra et al.'s [26] and Xu et al.'s [28], but they found that the CorBF method performs the best for CBDP. In our EADP situation, CorBF does not perform well on the EADP task and only achieves similar performance on most classifiers and testing datasets compared with the None method. Therefore, the CorBF method is not recommended for pre-processing the defect datasets for EADP performance improvement. In addition, we observe that not all feature selection techniques can enhance the EADP performance. For instance, ConBF and ConGS achieve similar or even worse performances with None on most classifiers and testing datasets. The finding is similar to Ghotra et al.'s [26] and Xu et al.'s [28] studies. They also found that not all methods can improve the CBDP performance. Therefore, we suggest researchers and practitioners ought to carefully choose the appropriate feature selection techniques (e.g., XGBF) to find more bugs (higher PofB@20% value) and defective modules (higher Recall@20% value) and obtain a more accurate global ranking of software modules (higher Norm(*Popt*) value).

(2) **Future EADP research ought to consider exploring whether more advanced binary classification algorithms embedded in CBS+ can further improve the EADP performance.** The previous studies [48–50] have shown that employing some more advanced binary classification algorithms (e.g., ADB, DF, RF, and XGB) can build more accurate CBDP models. Therefore, we investigate whether utilising these algorithms as the base classifiers embedded in CBS+ can also improve the EADP performance, and the results in Section 4.5 show that these algorithms indeed enhance the performance in terms of Precision@20%, Recall@20%, PofB@20%, and Norm(*Popt*). In addition, the correlation analysis in Section 4.5 also indicates that the better classification performance of the algorithms can contribute to the superiority of CBS+. For example, Recall has a high or moderate correlation with Recall@20% and PofB@20%. The main reason is that CBS+ employs the built classification model to calculate the probability of new modules being defective and ranks new modules according to the ratio between the defect probability and LOC. In other words, the ranking performance of CBS+ depends on the accurate prediction of defect probability to some extent. Such a result shows that more advanced binary classification algorithms have the potential to enhance the EADP performance. Therefore, we encourage future research to investigate introducing higher-performing classification algorithms into EADP to rank software modules more accurately.

(3) **Researchers and practitioners should consider extracting the four features (i.e., *noc*, *ic*, *cho*, and *cbm*) and carefully select different features to train EADP models for different datasets.** The results in Section 4.3 show that the selected features vary with different feature selection methods, but the four features (i.e., *noc*, *ic*, *cho*, and *cbm*) appear more frequently than others. It indicates these four features help construct the EADP models more effectively. In addition, even for the same project, the same feature selection method chooses the different features for different versions. Therefore, the best feature subset of EADP models should be selected carefully for different datasets.

## 6 | THREATS TO VALIDITY

**Internal validity** lies in the investigated methods and technologies used in our experiments, that is, classification models and feature selection methods. (1) We apply four families of feature selection methods and six families of classifiers, which makes our research objects diverse and representative and helps to enhance the generalisation of the experimental results. The feature selection methods are widely used in prior studies [26, 28, 31–33]. The adoption of other feature selection methods not studied in our work is left for future study. To alleviate the technical errors in our experiments, we implement the investigated feature selection methods and classifiers provided by the third-party libraries, that is, Weka<sup>†</sup> and Scikit-learn<sup>‡</sup>. Specifically, we implement the feature selection methods based on Weka; the machine learning classifiers (i.e., DT, KNN, LR, NB, and MLP) and some of the ensemble-based classifiers (i.e., ADB and RF) are implemented based on Scikit-learn; The other ensemble-based classifiers are based on their own third-party libraries (i.e., XGB<sup>§</sup> and DF<sup>§</sup>). (2) We directly employ the default hyper-parameters of classifiers provided by Scikit-learn rather than tune these hyper-parameters. The main reason is that we employ several performance measures to evaluate EADP models comprehensively. If we directly optimise PofB@20%, it will increase the PMI@20% and IFA values of models. This results in that tuning hyper-parameters is a multi-objective optimisation problem. We will conduct a follow-up study to investigate the issue.

**External validity** in our work is mainly concerned with the datasets we studied. (1) Since we conduct the more realistic cross-version validation and need the information on the bug numbers, the experimental datasets are 11 public projects from PROMISE, each of which contains three or more versions of datasets. The datasets have been widely utilised and validated by a number of previous SDP studies [20, 52–54]. Additionally, all software projects used in our study from the PROMISE corpus were developed by the open-source community.

<sup>†</sup><https://git.cms.waikato.ac.nz/weka/weka>

<sup>‡</sup><https://github.com/scikit-learn/scikit-learn>

<sup>§</sup><https://github.com/dmlc/xgboost>

<sup>§</sup><https://github.com/kingfengji/gcforest>

Therefore, it is unclear whether we can extend our conclusions to other software projects in other fields with other software characteristics or programming languages, especially commercial projects. In future work, we plan to collect more defect datasets to verify the generalisation of our conclusions. (2) We do not consider the data imbalance problem, which may have some impact on our results. Therefore, we plan to conduct a follow-up study focussing on the effect of different methods dealing with data imbalance problems for the EADP tasks.

**Construct validity** mainly comes from the effort-aware evaluation metrics we used to measure the performance of EADP models. We use the six evaluation metrics (including Precision@20%, Recall@20%, PofB@20%, PMI@20%, IFA, and Norm(*Popt*)) to evaluate the experimental results comprehensively. Since EADP models aim to find more defective modules and bugs and obtain a more accurate ranking based on the predicted defect density, we employ Recall@20%, PofB@20%, and Norm(*Popt*). Since Precision@20% and Recall@20% are usually paired, Precision@20% is used. In addition, inspecting too many software modules also leads to an additional effort, so we employ PMI@20%. Then, the IFA is used, since the previous studies [17] showed that the software testing team would not use the EADP model when the IFA value is too high.

**Conclusion validity** mainly refers to the statistical analysis method used in our work. We apply the Scott–Knott ESD test to analyse the significant differences among the feature selection methods, which helps to enhance the rigour of our experiments. Furthermore, for a more intuitive observation, we rank the feature selection methods with the double Scott–Knott ESD test and cluster them into non-overlapping groups with statistically significant differences.

## 7 | RELATED WORK

### 7.1 | Effort-aware defect prediction

Mende et al. [55] incorporated the concept of “effort-aware” into the SDP study and proposed two strategies for evaluating EADP models. Kamei et al. [44] found that the process metrics outperformed the product metrics on EADP models. Kamei et al. [18] proposed an EALR model and demonstrated that EALR could find 35% of defective code changes by checking only 20% of all changes. Yang et al. [56] verified the capability of the slice-based cohesion metrics for EADP. Bennin et al. [45, 57] explored the best-performing EADP algorithms and studied the practical benefits of data resampling techniques for EADP. Yang et al. [58] found that the unsupervised method (i.e., ManualUp [19]) generally outperformed several simple supervised models for change-level EADP. Fu et al. [59] proposed the OneWay method to utilise the training dataset to select the best software feature for ManualUp automatically. Chen et al. [52] employed a multi-objective optimisation method to build the change-level EADP model. Yu et al. [47] explored the best EADP algorithms and found that the RR algorithm achieved the best performance. Yan

et al. [60] observed that the findings of Yang et al. [58] are consistent for within-project file-level EADP, but are inconsistent in the cross-project scenario. Qu et al. [61] proposed a top-core equation to help rearrange the likely defective modules for EADP. Qu et al. [62] proposed integrating developer information into EADP to enhance performance. Carka et al. [63] proposed to assess the EADP performance using the normalised PofB, which sorted software modules according to the predicted defect densities. Huang et al. [17] proposed the CBS+ algorithm for EADP. The results showed that CBS+ could find more defective changes than EALR; Compared with ManualUp, CBS+ could find a similar number of defective changes, but required to inspect fewer changes and significantly reduced the IFA value. Subsequently, Ni et al. [20] proposed the EASC algorithm for cross-project EADP. EASC has the same algorithm flow as CBS+ and employs NB as the base classification model. The results showed that EASC outperformed some cross-project defect prediction algorithms. Yan et al. [22] validated the effectiveness of CBS+ on Alibaba projects. Ni et al. [21] investigated the change-level EADP models on JavaScript projects and found that CBS+ statistically significantly outperformed EALR, OneWay, and ManualUp. Therefore, we employ CBS+ as the EADP model in our study.

It is worth mentioning that some researchers proposed to use the bug numbers to sort software modules and aim to find more bugs while inspecting a certain number of modules. For example, Santosh et al. [64, 65] proposed some regression algorithms predict bug numbers. Yang et al. [66] proposed a learning-to-rank method to directly sort modules based on the bug numbers. But the works do not belong to the category of EADP, so we do not discuss them in detail.

### 7.2 | Feature selection for Classification-Based Defect Prediction

Many researchers have performed empirical investigations to validate the effect of feature selection methods for CBDP. He et al. [67] investigated the feasibility of some classifiers trained on a simplified feature subset. Shivaji et al. [68] verified the effectiveness of IG, chi-square, PS probabilistic significance, ReliefF, and the wrapper-based methods for change-level CBDP. Muthukumaran et al. [69] studied 10 feature selection methods for CBDP and showed that the methods could help improve the accuracy of classifiers, and subset-based feature selection methods outperformed other methods on NASA and AEEEM datasets. Gao et al. [31] explored the effect of four subset selections and seven feature ranking methods on a software system with five classifiers, and indicated that the performance of the CBDP model was enhanced when eliminating more than 85% of features. Wang et al. [70] performed an empirical study with an ensemble of 17 feature ranking methods and found that an ensemble of fewer feature ranking methods could achieve better performance. Wang et al. [71] designed a novel ensemble technique that combines six filter-based ranking methods and concluded that the ensemble

technique outperformed any single filter-based method. Rathore et al. [32] studied 15 different feature selection methods and found that InfoGain and Principal Component Analysis corbf outperformed other feature ranking methods, while LR and ClassifierSubsetEval outperformed other feature subset selection methods. Xu et al. [28] investigated 32 feature selection methods on the CDBP model built with the random forest algorithm on NASA and AEEEM dataset, and found that the feature subset selection methods and wrapper-based methods could usually perform better. Gogra et al. [26] studied the effect of 30 feature selection methods on 21 classifiers for CDBP and observed that a correlation-based subset selection approach was better than other approaches. Kondo et al. [27] investigated eight feature reduction methods on five supervised learning and five unsupervised CDBP models. The results on three publicly available datasets demonstrated that the model using feature selection methods achieved better performance than the model built with all features. Blowgun et al. [25] studied 14 feature subset selections and four feature ranking methods by using four different classification models. The experimental analysis on five NASA datasets demonstrated that the effect of the methods varied due to the datasets and the classifiers. Jiarapakdee et al. [72] systematically investigated the interpretation of feature selection methods for CDBP. The results indicated that the features selected by these different methods were mostly inconsistent. Balogun et al. [24] addressed the biases in the existing feature selection empirical studies and investigated 46 feature selection methods for the CDBP task by using two classifiers. The experimental analysis on 25 datasets indicated that none of the feature selection methods could obtain the best performance, since their respective performance depended on the selection of the classification models, evaluation metrics, and different datasets. Kabir [46] studied the robustness of 11 feature selection methods to concept drift for CDBP.

However, the above-mentioned empirical studies mainly focus on the CDBP task. Which methods can enhance the EADP performance has been unclear. Therefore, we, for the first time, study 24 feature selection methods with eight classifiers to explore the topic.

## 8 | CONCLUSION

This study assesses the practical benefits of 24 feature selection methods for EADP on 41 datasets from the PROMISE repository. We employ 10 classifiers embedded in the state-of-the-art CBS+ algorithm to build the EADP model. We use PofB@20%, Recall@20%, Norm(*Popt*), Precision@20%, PMI@20%, and IFA to evaluate the performance comprehensively and apply the Scott–Knott ESD test to analyse the experimental results from both classifiers and testing datasets level. We observe that the impact of the feature selection methods varies in classifiers embedded in CBS+ and testing datasets, and the four wrapper-based feature subset selection methods (i.e., ADBF, DFF, RFF, and XGBF) perform the best among the 10 classification models and on most testing datasets.

Hence, we suggest researchers and practitioners to employ the four wrapper-based feature selection methods with forward search to select the optimal feature subsets, and we prefer to recommend XGBF with XGBoost as the embedded classifier in CBS+ to enhance the performance of EADP models. We further find that the selected features vary with different feature selection methods and different datasets, and *noc*, *ic*, *cho*, and *cbm* are the most frequently selected features by the four methods. Therefore, researchers and practitioners should carefully select different features to train EADP models for different datasets. Finally, we observe that ADB, DF, RF, and XGB are the best-performing classifiers embedded in CBS+. We open-source the source code, experimental datasets, and detailed results<sup>1</sup> to facilitate the replication of our work and conduct further study.

## AUTHOR CONTRIBUTION

**Fuyang Li:** Data curation, Formal analysis, Methodology, Writing – original draft. **Wanpeng Lu:** Data curation, Methodology, Software, Writing – original draft. **Jacky Wai Keung:** Project administration, Software, Validation. **Xiao Yu:** Formal analysis, Methodology, Supervision, Writing – review & editing. **Lina Gong:** Resources, Software, Supervision, Writing – review & editing. **Juan Li:** Investigation, Project administration, Visualisation.

## ACKNOWLEDGEMENTS

This work was in part supported by the Project of Sanya Yazhou Bay Science and Technology City (SCKJ-JYRC-2022-17), NSFC (62102292, 62101393, 62202223), the Youth Fund Project of Hainan Natural Science Foundation (622QN344), Sanya Science and Education Innovation Park of Wuhan University of Technology (2021KF0031), the Fundamental Research Funds for the Central Universities (WUT223110002, WUT223110001), the Natural Science Foundation of Chongqing (cstc2021jcyj-msxmX1115, cstc2021jcyj-msxmX1148), the Natural Science Foundation of Jiangsu Province (BK20220881), and the Open Project of Wuhan University of Technology Chongqing Research Institute (ZL2021-6).

## CONFLICTS OF INTEREST

There are no Conflicts of Interests to declare.

## 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

- 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>

<sup>1</sup><https://github.com/AIForSys/FSTonEADP>



2. Lin, H., et al.: Gen-fl: quality prediction-based filter for automated issue title generation. *J. Syst. Software* 195, 111513 (2023). <https://doi.org/10.1016/j.jss.2022.111513>
3. Yang, Z., et al.: A multi-modal transformer-based code summarization approach for smart contracts. In: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), pp. 1–12. IEEE (2021)
4. Zheng, W., et al.: Interpretability application of the just-in-time software defect prediction model. *J. Syst. Software* 188, 111245 (2022). <https://doi.org/10.1016/j.jss.2022.111245>
5. 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>
6. 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>
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. Chen, X., et al.: Do different cross-project defect prediction methods identify the same defective modules? *J. Softw.: Evolution and Process* 32(5), e2234 (2020). <https://doi.org/10.1002/smr.2234>
9. Feng, S., et al.: Coste: complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction. *Inf. Software Technol.* 129, 106432 (2021). <https://doi.org/10.1016/j.infsof.2020.106432>
10. Zhang, N., et al.: Software defect prediction based on stacked sparse denoising autoencoders and enhanced extreme learning machine. *IET Softw.* 16(1), 29–47 (2022). <https://doi.org/10.1049/sfw2.12029>
11. 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>
12. Chen, X., et al.: Revisiting heterogeneous defect prediction methods: how far are we? *Inf. Software Technol.* 130, 106441 (2021). <https://doi.org/10.1016/j.infsof.2020.106441>
13. Manchala, P., Bisi, M.: Diversity Based Imbalance Learning Approach for Software Fault Prediction Using Machine Learning Models. *Applied Soft Computing* (2022).109069
14. 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>
15. Zhao, K., et al.: A Compositional Model for Effort-Aware Just-In-Time Defect Prediction on Android Apps. *IET Software* (2021)
16. 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. *Information and Software Technology* (2022).106980
17. 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>
18. 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>
19. Menzies, T., et al.: Defect prediction from static code features: current results, limitations, new approaches. *Autom. Software Eng.* 17(4), 375–407 (2010). <https://doi.org/10.1007/s10515-010-0069-5>
20. Ni, C., et al.: Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. *IEEE Trans. Software Eng.* 48(3), 786–802 (2020). <https://doi.org/10.1109/tse.2020.3001739>
21. Ni, C., et al.: Just-in-time defect prediction on javascript projects: a replication study. *ACM Trans. Software Eng. Methodol.* 31(4), 1–38 (2022). <https://doi.org/10.1145/3508479>
22. Yan, M., et al.: Effort-aware just-in-time defect identification in practice: a case study at alibaba. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1308–1319 (2020)
23. Wan, Z., et al.: Perceptions, expectations, and challenges in defect prediction. *IEEE Trans. Software Eng.* 46(11), 1241–1266 (2018). <https://doi.org/10.1109/tse.2018.2877678>
24. Balogun, A.O., et al.: Impact of feature selection methods on the predictive performance of software defect prediction models: an extensive empirical study. *Symmetry* 12(7), 1147 (2020). <https://doi.org/10.3390/sym12071147>
25. Balogun, A.O., et al.: Performance analysis of feature selection methods in software defect prediction: a search method approach. *Appl. Sci.* 9(13), 2764 (2019). <https://doi.org/10.3390/app9132764>
26. Ghotra, B., McIntosh, S., Hassan, A.E.: A large-scale study of the impact of feature selection techniques on defect classification models. In: González-Barahona, J.M., Hindle, A., Tan, L. (eds.) Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20–28, 2017, pp. 146–157. IEEE Computer Society (2017)
27. Kondo, M., et al.: The impact of feature reduction techniques on defect prediction models. *Empir. Software Eng.* 24(4), 1925–1963 (2019). <https://doi.org/10.1007/s10664-018-9679-5>
28. Xu, Z., et al.: The impact of feature selection on defect prediction performance: an empirical comparison. In: 27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23–27, 2016, pp. 309–320. IEEE Computer Society (2016)
29. Yu, Q., et al.: An empirical study on the effectiveness of feature selection for cross-project defect prediction. *IEEE Access* 7, 35710–35718 (2019). <https://doi.org/10.1109/access.2019.2895614>
30. Tantithamthavorn, C., et al.: An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans. Software Eng.* 43, 1–18 (2017). <https://doi.org/10.1109/tse.2016.2584050>
31. Gao, K., et al.: Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software Pract. Ex.* 41(5), 579–606 (2011). <https://doi.org/10.1002/spe.1043>
32. R'athore, S.S., Gupta, A.: A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction. In: Janakiram, D., Sen, K., Kulkarni, V. (eds.) 7th India Software Engineering Conference, Chennai, ISEC '14, Chennai, India - February 19 - 21, 2014, pp. 1–7. ACM (2014)
33. Rodríguez, D., et al.: Attribute selection in software engineering datasets for detecting fault modules. In: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2007), August 28–31, 2007, Lübeck, Germany, pp. 418–423. IEEE Computer Society (2007)
34. Khoshgoftaar, T.M., Golawala, M., Hulse, J.V.: An empirical study of learning from imbalanced data using random forest. In: 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), October 29–31, 2007, Patras, Greece, Volume 2, pp. 310–317. IEEE Computer Society (2007)
35. Menzies, T., et al.: The Promise Repository of Empirical Software Engineering Data. West Virginia University, Department of Computer Science (2012)
36. Chen, Y., Lu, X.: Deep category-level and regularized hashing with global semantic similarity learning. *IEEE Trans. Cybern.* 51(12), 6240–6252 (2020). <https://doi.org/10.1109/tcyb.2020.2964993>
37. 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>
38. 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>
39. He, C., Wu, J., Zhang, Q.: Research leadership flow determinants and the role of proximity in research collaborations. *J. Assoc. Inform. Sci. Technol.* 71(11), 1341–1356 (2020). <https://doi.org/10.1002/asi.24331>
40. 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>
41. He, C., Wu, J., Zhang, Q.: Proximity-aware research leadership recommendation in research collaboration via deep neural networks. *J. Assoc.*

- Inform. Sci. Technol. 73(1), 70–89 (2022). <https://doi.org/10.1002/asi.24546>
42. 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>
  43. 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>
  44. Kamei, Y., et al.: Revisiting common bug prediction findings using effort-aware models. In: Marinescu, R., Lanza, M., Marcus, A. (eds.) 26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12–18, 2010, Timisoara, Romania, pp. 1–10. IEEE Computer Society (2010)
  45. 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, QRS 2016, Vienna, Austria, August 1–3, 2016, pp. 214–221. IEEE (2016)
  46. 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>
  47. 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 (SANER), pp. 298–309. IEEE (2019)
  48. Matloob, F., et al.: Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review. *IEEE Access* (2021)
  49. Yang, H., Li, M.: Software defect prediction based on smote-tomek and xgboost. In: International Conference on Bio-Inspired Computing: Theories and Applications, pp. 12–31. Springer (2021)
  50. Zhou, T., et al.: Improving defect prediction with deep forest. *Inf. Software Technol.* 114, 204–216 (2019). <https://doi.org/10.1016/j.infsof.2019.07.003>
  51. Hinkle, D.E., Wiersma, W., Jurs, S.G.: Applied Statistics for the Behavioral Sciences. Volume 663. Houghton Mifflin College Division (2003)
  52. 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>
  53. 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>
  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 (ESEM), pp. 344–353. IEEE (2017)
  55. Mende, T., Koschke, R.: Effort-aware defect prediction models. In: 2010 14th European Conference on Software Maintenance and Reengineering, pp. 107–116. IEEE (2010)
  56. Yang, Y., et al.: Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study. *IEEE Trans. Software Eng.* 41(4), 331–357 (2015). <https://doi.org/10.1109/tse.2014.2370048>
  57. Bennin, K.E., et al.: Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models. In: 40th IEEE Annual Computer Software and Applications Conference, COMPSAC 2016, Atlanta, GA, USA, June 10–14, 2016, pp. 154–163. IEEE Computer Society (2016)
  58. Yang, Y., et al.: Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In: Zimmermann, T., Cleland-Huang, J., Su, Z. (eds.) Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13–18, 2016, pp. 157–168. ACM (2016)
  59. 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. ACM (2017)
  60. Yan, M., et al.: File-level defect prediction: unsupervised vs. supervised models. In: Bener, A., Turhan, B., Biffl, S. (eds.) 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, Toronto, ON, Canada, November 9–10, 2017, pp. 344–353. IEEE Computer Society (2017)
  61. 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 (2021). <https://doi.org/10.1109/tse.2019.2892959>
  62. Qu, Y., Chi, J., Yin, H.: Leveraging developer information for efficient effort-aware bug prediction. *Inf. Software Technol.* 137, 106605 (2021). <https://doi.org/10.1016/j.infsof.2021.106605>
  63. 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>
  64. Rathore, S.S.: An exploratory analysis of regression methods for predicting faults in software systems. *Soft Comput.* 25(23), 14841–14872 (2021). <https://doi.org/10.1007/s00500-021-06048-x>
  65. Rathore, S.S., Kumar, S.: An approach for the prediction of number of software faults based on the dynamic selection of learning techniques. *IEEE Trans. Reliab.* 68(1), 216–236 (2018). <https://doi.org/10.1109/tr.2018.2864206>
  66. Yang, X., Tang, K., Yao, X.: A learning-to-rank approach to software defect prediction. *IEEE Trans. Reliab.* 64(1), 234–246 (2014). <https://doi.org/10.1109/tr.2014.2370891>
  67. He, P., et al.: An empirical study on software defect prediction with a simplified metric set. *Inf. Software Technol.* 59, 170–190 (2015). <https://doi.org/10.1016/j.infsof.2014.11.006>
  68. Shivaji, S., et al.: Reducing features to improve code change-based bug prediction. *IEEE Trans. Software Eng.* 39(4), 552–569 (2013). <https://doi.org/10.1109/tse.2012.43>
  69. Kasinathan, M., Rallapalli, A., Neti, L.B.M.: Impact of feature selection techniques on bug prediction models. In: Padmanabhuni, S., et al. (eds.) Proceedings of the 8th India Software Engineering Conference, ISEC 2015, Bangalore, India, February 18–20, 2015, pp. 120–129. ACM (2015)
  70. Wang, H., Khoshgoftaar, T.M., Napolitano, A.: A comparative study of ensemble feature selection techniques for software defect prediction. In: Draghici, S., et al. (eds.) The Ninth International Conference on Machine Learning and Applications, ICMLA 2010, Washington, DC, USA, 12–14 December 2010, pp. 135–140. IEEE Computer Society (2010)
  71. Wang, H., et al.: Metric selection for software defect prediction. *Int. J. Software Eng. Knowl. Eng.* 21(02), 237–257 (2011). <https://doi.org/10.1142/s0218194011005256>
  72. Jiarpakdee, J., Tantithamthavorn, C., Treude, C.: The impact of automated feature selection techniques on the interpretation of defect models. *Empir. Software Eng.* 25(5), 3590–3638 (2020). <https://doi.org/10.1007/s10664-020-09848-1>

**How to cite this article:** Li, F., et al.: The impact of feature selection techniques on effort-aware defect prediction: An empirical study. *IET Soft.* 17(2), 168–193 (2023). <https://doi.org/10.1049/sfw2.12099>