The impact of feature selection techniques on effort-aware defect prediction
An empirical study
Li, Fuyang; Lu, Wanpeng; Keung, Jacky Wai; Yu, Xiao; Gong, Lina; Li, Juan

Published in:
IET Software

Published: 01/04/2023

Document Version:
Final Published version, also known as Publisher's PDF, Publisher's Final version or Version of Record

License:
CC BY

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1049/sfw2.12099

Publication details:

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.
The impact of feature selection techniques on effort-aware defect prediction: An empirical study

Fuyang Li\textsuperscript{1} | Wappeng Lu\textsuperscript{1,2} | Jacky Wai Keung\textsuperscript{3} | Xiao Yu\textsuperscript{1,4,5} | Lina Gong\textsuperscript{6} | Juan Li\textsuperscript{7}

\textsuperscript{1}School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China
\textsuperscript{2}School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China
\textsuperscript{3}Department of Computer Science, City University of Hong Kong, Hong Kong, China
\textsuperscript{4}Sanya Science and Education Innovation Park of Wuhan University of Technology, Sanya, China
\textsuperscript{5}Wuhan University of Technology Chongqing Research Institute, Chongqing, China
\textsuperscript{6}School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
\textsuperscript{7}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

Lina Gong, School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.
Email: 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, 62022233; Youth Fund Project of Hainan Natural Science Foundation, Grant/Award Number: 620QN344; 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: WUT22310002, WUT22510001; 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, 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 \textit{noc} (number of children), \textit{ic} (inheritance coupling), \textit{cbo} (coupling between object classes), and \textit{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.
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 helps 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, P@20 (Proportion of Modules Inspected when inspecting the top 20% LOC), and PMI@20% (Proportion of Modules Inspected when inspecting the top 20% LOC) to measure the effects of the above-mentioned feature selection methods and classifiers. In addition, we also use Precision@20% and IFA (Initial False Alerts) 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 POB@20%, Recall@20%, and Norm(PoP) 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 POB@20%, Recall@20%, and Norm(PoP), 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 POB@20%, Recall@20%, and Norm(PoP) values on the testing datasets.

(4) The best-performing CorBF method in CBPD 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, cbo, 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 (ADBF), Deep Forest (DFF), Random Forest (RF), and XGBoost (XGBF) as the base classifiers embedded in CBS+ can achieve the best performance in terms of Precision@20%, Recall@20%, POB@20%, and Norm(PoP).

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 methods is to first identify 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 discriminates 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 Khoshgoftar 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 Forward Search, ADB with Backwards Search, XGB with Forward Search, XGB with Backwards Search, DF with Forward Search, DF with Backwards Search, RF with Forward 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.

\[
P_{\text{of B}@20\%} = \frac{q}{Q}
\]

(1)

- **Recall@20\%** indicates the proportion of the number of inspected actual defective software modules to the total
The number of defective software modules in the defect dataset. The higher Recall@20% value means more defective software modules can be found.

\[
\text{Recall@20\%} = \frac{\text{P}}{\hat{\text{P}}}
\]

\(\text{Norm}(\text{Popt})\) 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 \(\Delta\text{opt}\) denotes the difference between the two models. Then, \(\text{Popt}\) is defined as follows:

\[
\text{Popt} = 1 - \Delta\text{opt}
\]

According to the research of Kamet al. [44], since the minimum value of \(\text{Popt}\) is dependent on the total quantity of defects in the dataset, we also employ the normalised \(\text{Popt}\), that is, \(\text{Norm}(\text{Popt})\), as the evaluation metric. Obviously, the higher number of defective software modules in the defect dataset. The higher Recall@20\% value means more defective software modules can be found.

### TABLE 1
The details of the experimental datasets

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

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

<table>
<thead>
<tr>
<th>No.</th>
<th>Feature</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>wmc</td>
<td>Weighted methods per class</td>
</tr>
<tr>
<td>2</td>
<td>dit</td>
<td>Depth of inheritance tree</td>
</tr>
<tr>
<td>3</td>
<td>noc</td>
<td>Number of children</td>
</tr>
<tr>
<td>4</td>
<td>cbo</td>
<td>Coupling between object classes</td>
</tr>
<tr>
<td>5</td>
<td>rfc</td>
<td>Response for a class</td>
</tr>
<tr>
<td>6</td>
<td>lcom</td>
<td>Lack of cohesion in methods</td>
</tr>
<tr>
<td>7</td>
<td>ca</td>
<td>Afferent couplings</td>
</tr>
<tr>
<td>8</td>
<td>ce</td>
<td>Efferent couplings</td>
</tr>
<tr>
<td>9</td>
<td>rpm</td>
<td>Number of public methods</td>
</tr>
<tr>
<td>10</td>
<td>lcom3</td>
<td>Another lack of cohesion in methods</td>
</tr>
<tr>
<td>11</td>
<td>loc</td>
<td>Line of code</td>
</tr>
<tr>
<td>12</td>
<td>dam</td>
<td>Data access metric</td>
</tr>
<tr>
<td>13</td>
<td>moa</td>
<td>Measure of aggregation</td>
</tr>
<tr>
<td>14</td>
<td>mfa</td>
<td>Measure of functional abstraction</td>
</tr>
<tr>
<td>15</td>
<td>cam</td>
<td>Cohesion among methods of class</td>
</tr>
<tr>
<td>16</td>
<td>ic</td>
<td>Inheritance coupling</td>
</tr>
<tr>
<td>17</td>
<td>ebm</td>
<td>Coupling between methods</td>
</tr>
<tr>
<td>18</td>
<td>amc</td>
<td>Average method complexity</td>
</tr>
<tr>
<td>19</td>
<td>max_cc</td>
<td>Max McCabes cyclomatic complexity</td>
</tr>
<tr>
<td>20</td>
<td>avg_cc</td>
<td>Average McCabes cyclomatic complexity</td>
</tr>
</tbody>
</table>
Norm\( (P_{opt}) \) value means that the predictive model is closer to the optimal model.

\[
\text{Norm}(P_{opt}) = \frac{P_{opt} - \min(P_{opt})}{\max(P_{opt}) - \min(P_{opt})}
\]  

\( \text{Precision@20\%} \) 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 \( \text{Precision@20\%} \) value means better performance of the prediction model.

\[
\text{Precision@20\%} = \frac{P}{m}
\]

\( \text{PMI@20\%} \) indicates the proportion of the number of inspected software modules to the total number of software modules in the dataset. The lower \( \text{PMI@20\%} \) value means the software testing team is required to inspect fewer software modules.

\[
\text{PMI@20\%} = \frac{m}{M}
\]

![FIGURE 1](image1.png) **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 \( \text{Recall@20\%} \) value, it would achieve a lower \( \text{Precision@20\%} \). When the model obtains a higher \( \text{PoF@20\%} \) value, it will achieve a higher \( \text{PMI@20\%} \).

### 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
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., naïve 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. **Naïve 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 hyperplane 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.


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

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

**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...
In total, 24 feature selection methods were compared across 30 testing datasets, with the aim of evaluating their performance in terms of each evaluation metric. To present the average result of each feature selection method on each testing dataset across all classifiers, we employ the heat map. 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 a darker color represents the better performance in terms of \( \text{PofB@20\%} \), \( \text{Recall@20\%} \), \( \text{Norm(\text{Popt})} \), and \( \text{Precision@20\%} \), while the cell with a brighter color indicates the better performance in terms of \( \text{PMI@20\%} \) 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 color represents the worse Scott–Knott ESD ranking result in terms of \( \text{PofB@20\%} \), \( \text{Recall@20\%} \), \( \text{Norm(\text{Popt})} \), and \( \text{Precision@20\%} \), while the cell with brighter color indicates the feature selection method obtains the better Scott–Knott ESD ranking in terms of \( \text{PMI@20\%} \) 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 \( \text{PofB@20\%} \), \( \text{Recall@20\%} \), and \( \text{Norm(\text{Popt})} \), then analyse the \( \text{Precision@20\%} \), \( \text{PMI@20\%} \), and IFA results.

1. As shown in Figure 4a, most of the feature selection methods obtain the high \( \text{PofB@20\%} \) 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 \( \text{PofB@20\%} \). 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 P@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 PoB@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.
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). 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 achieve 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.

**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
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 PoF@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(Popt). Additionally,
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(Popt) 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 X GB classifier achieve low IFA values (3.47, 3.67, 5.03, and 5.00, respectively). From

**Figure 12** The Scott-Knott ESD ranking of the methods for each classifier across all testing datasets 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.

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.

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.

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

<table>
<thead>
<tr>
<th>Datasets</th>
<th>ADBF</th>
<th>DFF</th>
<th>RFF</th>
<th>XGBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant-1.4</td>
<td>ic, mfa, cbo, avg_cc</td>
<td>mfa, avg_cc</td>
<td>ce</td>
<td>wmc</td>
</tr>
<tr>
<td>Ant-1.5</td>
<td>loc, ame</td>
<td>npm, noc, dam</td>
<td>loc, ic</td>
<td>noc, cbo, ce, dit, dam, lcom, ic, cam</td>
</tr>
<tr>
<td>Ant-1.6</td>
<td>loc, rfc, leom</td>
<td>npm</td>
<td>ebo, ame, noc</td>
<td>noc, ebo, ame</td>
</tr>
<tr>
<td>Ant-1.7</td>
<td>ic, mfa, ame, ce, dit, dam, cbm</td>
<td>leom3</td>
<td>ebo, ame, ce, dam, avg_cc</td>
<td>rfc, ce, cbm, leom3, moa</td>
</tr>
<tr>
<td>Camel-1.2</td>
<td>ebo</td>
<td>ca, noc</td>
<td>noc, dam</td>
<td>noc, dit</td>
</tr>
<tr>
<td>Camel-1.4</td>
<td>max_cc</td>
<td>moa</td>
<td>ce, cbm</td>
<td>ca</td>
</tr>
<tr>
<td>Camel-1.6</td>
<td>moa, avg_cc, mfa, dam</td>
<td>moa</td>
<td>ce, ebo, dit</td>
<td>ca, avg_cc, leom, ic, dam, mfa, leom3</td>
</tr>
<tr>
<td>Ivory-1.4</td>
<td>ce, ebo, dit</td>
<td>moa, ebo</td>
<td>moa, ebo, lcom, ic, ce, rfc, dit, moa, cbm</td>
<td>npm, leom, cbo, cbm, lcom, ic, ce, rfc, dit, cam</td>
</tr>
<tr>
<td>Ivory-2.0</td>
<td>ebo, ic, mfa, cbo, moa, dam, ame, dit</td>
<td>moa, ebo</td>
<td>moa, ebo</td>
<td>moa</td>
</tr>
<tr>
<td>Jedit-4.0</td>
<td>ebo, avg_cc</td>
<td>moa, wmc</td>
<td>moa</td>
<td>moa</td>
</tr>
<tr>
<td>Jedit-4.1</td>
<td>ebo, noc</td>
<td>dit, mfa, cam, ic, cbm</td>
<td>dit, ic, ebo, leom, noc</td>
<td>dit, ic, ebo, leom, cbm</td>
</tr>
<tr>
<td>Jedit-4.2</td>
<td>ca, moa, mfa</td>
<td>ca, leom3</td>
<td>npm, ic</td>
<td>npm, cbm</td>
</tr>
<tr>
<td>Jedit-4.3</td>
<td>npm, loc, max_cc, ic, moa, ce</td>
<td>cam, loc, ic, ame</td>
<td>ame</td>
<td>mfa, moa, ic, noc, dit</td>
</tr>
<tr>
<td>Log4j-1.1</td>
<td>ebo, ic</td>
<td>ca, mfa</td>
<td>ebo</td>
<td>wmc, noc</td>
</tr>
<tr>
<td>Log4j-1.2</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
<td>nc</td>
</tr>
<tr>
<td>Lucene-2.2</td>
<td>ebo, max_cc, leom, npm</td>
<td>nc</td>
<td>cbm</td>
<td>nc</td>
</tr>
<tr>
<td>Lucene-2.4</td>
<td>ic</td>
<td>ic</td>
<td>ic</td>
<td>ic</td>
</tr>
<tr>
<td>Poi-2.0</td>
<td>moa, ca, noc</td>
<td>noc, npm</td>
<td>moa, ca</td>
<td>moa, ca</td>
</tr>
<tr>
<td>Poi-2.5</td>
<td>dit</td>
<td>dit</td>
<td>dit</td>
<td>dit</td>
</tr>
<tr>
<td>Poi-3.0</td>
<td>moa</td>
<td>max_cc</td>
<td>moa</td>
<td>moa</td>
</tr>
<tr>
<td>Synapse-1.1</td>
<td>ce, wmc, ca, ic, mfa, cbo, moa, ebo, leom, dam, dit</td>
<td>leom, ca, ic</td>
<td>ebo, mfa, noc</td>
<td>cbo, mfa, cbm, ic, noc, dit</td>
</tr>
<tr>
<td>Synapse-1.2</td>
<td>cam, dit</td>
<td>noe</td>
<td>ic</td>
<td>ic</td>
</tr>
<tr>
<td>Velocity-1.5</td>
<td>wmc, npm, max_cc</td>
<td>ce</td>
<td>ame, rfc, npm, loc, ca, wmc, dam</td>
<td>ame, cbo, cbm</td>
</tr>
<tr>
<td>Velocity-1.6</td>
<td>ebo, cbm, moa, ic, ca</td>
<td>mfa</td>
<td>cbo, cbm</td>
<td>ca, avg_cc, noe</td>
</tr>
<tr>
<td>Xalan-2.5</td>
<td>cbm, ic</td>
<td>noe</td>
<td>ic</td>
<td>ic</td>
</tr>
<tr>
<td>Xalan-2.6</td>
<td>noe</td>
<td>ic</td>
<td>cbm</td>
<td>cbm</td>
</tr>
<tr>
<td>Xalan-2.7</td>
<td>ic, cbm</td>
<td>noe</td>
<td>cbm</td>
<td>cbm</td>
</tr>
<tr>
<td>Xerces-1.2</td>
<td>noe</td>
<td>noe</td>
<td>noe, ic, max_cc, ame, cbo, avg_cc, cbm</td>
<td>ic, max_cc, ame, dam, cbo, cbm, ce</td>
</tr>
<tr>
<td>Xerces-1.3</td>
<td>wmc, avg_cc</td>
<td>noe, rfc</td>
<td>npm</td>
<td>npm</td>
</tr>
<tr>
<td>Xerces-1.4</td>
<td>mfa, dit, leom3</td>
<td>noe</td>
<td>dit</td>
<td>noe</td>
</tr>
</tbody>
</table>

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

Answer to RQ3

XGBF with XGBoost as the embedded classifier in CBS+ performs the best and obtains the highest average P@20%, Recall@20%, and Norm(Popt) values on the testing datasets.
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.
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 PoB@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 PoB@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 PoB@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 PoB@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 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 ≤ |r| < 0.5), moderate (0.5 ≤ |r| < 0.7), high (0.7 ≤ |r| < 0.9), and very high (0.9 ≤ |r| ≤ 1).

**Result (2):** As shown in Figure 16a–d, we can observe that: (1) Recall has a high or moderate correlation with both PoB@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 PoB@20% and Norm(Popt) values of the model has. (2) PoB@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.
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 P@B@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 PoF@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 preprocessing 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 PoF@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%, PoF@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 PoF@20%. The main reason is that CBS+ employs the bulk 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, cbo, 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, cbo, 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* and Scikit-learn§. 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* and DF§). (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 PoF@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 this 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.

1https://git.cms.waikato.ac.nz/weka/weka
2https://github.com/kingfengji/gcforest
3https://github.com/scikit-learn/scikit-learn
4https://github.com/dmlc/xgboost
5https://github.com/scikit-learn/scikit-learn
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%, PoB@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%, PoB@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 PoB, 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 to 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 CBDP 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 CBDP 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 CBDP 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. Jiarpakdee et al. [72] systematically investigated the interpretation of feature selection methods for CBDP. 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 CBDP 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 CBDP.

However, the above-mentioned empirical studies mainly focus on the CBDP 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 PoB@c%, Recall@c%, Norm(PoP), Precision@c%, PMI@c%, 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, ie, cbo, 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 to facilitate the replication of our work and conduct further study.

ACKNOWLEDGEMENTS
This work was in part supported by the Project of Sanya Yazhou Bay Science and Technology City (SCK-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

https://github.com/AllForSys/FSTonEADP


