Applied Intelligence (2021) 51:3615–3644
https://doi.org/10.1007/s10489-020-01935-6

An empirical study of ensemble techniques for software fault prediction

Santosh S. Rathore[1] · **Sandeep Kumar**[2]

Accepted: 9 September 2020 / Published online: 16 November 2020

Abstract

Previously, many researchers have performed analysis of various techniques for the software fault prediction (SFP). Oddly, the majority of such studies have shown the limited prediction capability and their performance for given software fault datasets was not persistent. In contrast to this, recently, ensemble techniques based SFP models have shown promising and improved results across different software fault datasets. However, many new as well as improved ensemble techniques have been introduced, which are not explored for SFP. Motivated by this, the paper performs an investigation on ensemble techniques for SFP. We empirically assess the performance of seven ensemble techniques namely, Dagging, Decorate, Grading, MultiBoostAB, RealAdaBoost, Rotation Forest, and Ensemble Selection. We believe that most of these ensemble techniques are not used before for SFP. We conduct a series of experiments on the benchmark fault datasets and use three distinct classification algorithms, namely, naive Bayes, logistic regression, and J48 (decision tree) as base learners to the ensemble techniques. Experimental analysis revealed that rotation forest with J48 as the base learner achieved the highest precision, recall, and G-mean 1 values of 0.995, 0.994, and 0.994, respectively and Decorate achieved the highest AUC value of 0.986. Further, results of statistical tests showed used ensemble techniques demonstrated a statistically significant difference in their performance among the used ones for SFP. Additionally, the cost-benefit analysis showed that SFP models based on used ensemble techniques might be helpful in saving software testing cost and effort for twenty out of twenty-eight used fault datasets.

Keywords Software fault prediction · Ensemble techniques · PROMISE data repository · Empirical analysis

1 Introduction

Current software systems are growing rapidly in complexity and size, thus, ensuring their reliability and quality are paramount important, which depends on software faults [1]. Software fault prediction (SFP) actively helps in the detection of faults by highlighting potential faulty areas of code in the software system [2]. This identification of areas of code liable to more faults can help the testing team to allot software quality assurance resources optimally and efficiently [3, 4]. SFP modeling has been examined widely by several researchers due to its inherent advantages in optimizing testing resources utilization and improving the quality of software projects [5–7].

For the last two decades, various learning techniques have been used greatly for SFP [8–12]. Naïve Bayes, regression techniques, k-nearest neighbors, decision trees, multilayer perceptron, rule-based learners, etc. are the few of them. However, analysis of these algorithms showed that most of the algorithms achieved an average prediction accuracy of 80%-85% with a higher misclassification rate [4, 13, 14]. Moreover, the performance of algorithms has not been consistent across different fault datasets [15–18]. In the case of the software system, it is observed that most of the faults are concentrated in the small area of code. Therefore, the evaluation of a classification algorithm using accuracy measures will not provide an accurate depiction of the model performance [19, 20].

✉ Santosh S. Rathore
santosh.srathore@gmail.com

Sandeep Kumar
sandeepkumargarg@gmail.com

[1] Department of Information Technology, ABV-Indian Institute of Information Technology and Management, Gwalior, India

[2] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India

Earlier research in the SFP domain revealed that individual classification and learning techniques have reached the verge of their performance threshold point and the performance of these techniques may not be further improved without applying external corrections in the fault datasets or model building process [2, 21, 22]. Some researchers have tried to break this performance ceiling by adapting different performance-improving strategies such as enriching the information content of the training datasets [21], by customizing the prediction model to the specific local business goals [2], or by combining multiple sets of software metrics [23]. The results of these performance-improving strategies showed positive conclusions to break the performance bottleneck of SFP models. Presently, ensemble techniques based SFP models have gained popularity in the software engineering research community [24–26]. Many research evidence showed that ensemble techniques can help to overcome the performance bottleneck of classification algorithms and can serve as a tool to develop improved fault prediction models [23]. Few researchers have analyzed ensemble techniques such as bagging, boosting, voting, and stacking for SFP [26–28]. However, these studies were limited to some fault datasets and analyzed one or two ensemble techniques only. Further, many new as well as improved ensemble techniques have been reported by the researchers, but their evaluation for the SFP has not been performed yet. This motivated us to undertake a study of these ensemble techniques and to establish their usefulness for the SFP.

This paper performs an extensive experimental study of seven ensemble techniques including Dagging, Decorate, Grading, MultiBoostAB, RealAdaBoost, Rotation Forest, and Ensemble Selection for the SFP. To the best of our knowledge, most of the ensemble techniques used in this study have not been investigated thoroughly before for the SFP. For the ensemble techniques, three different classification algorithms, namely, naive Bayes, logistic regression, and J48 (decision tree) are chosen to serve as base learners. The experimental study is performed for twenty-eight public-domain software fault datasets available in the PROMISE data repository [29]. Precision, recall, AUC (area under the ROC curve), specificity, and G-means (G-mean 1 and G-mean 2) measures are considered to evaluate the performance of ensemble techniques. The relative significance difference in the performance of seven ensemble techniques is evaluated by using Friedman' test and Wilcoxon signed-rank test. Additionally, a cost-benefit analysis is carried out to assess the cost-effectiveness of used ensemble techniques in terms of saving software testing cost and effort. Results and observations obtained from this empirical study can help practitioners in building effective SFP models.

1.1 Contributions

Since the last decade, various researchers have used different ensemble techniques for software fault prediction. However, recently many new as well as improved versions of existing ensemble techniques have been introduced in the machine learning domain, which are not explored for the SFP. This raises the need for a comprehensive evaluation of these techniques to benchmark their performance for the SFP. This could be very beneficial to the research community and the practitioners working in the SFP domain.

The contributions of the presented work are discussed as follow:

1. We provide a systematic literature review of the ensemble techniques used for the software fault prediction and reported the findings of the review.
2. We perform an extensive comparison of seven different ensemble techniques for the SFP, which to the best of our knowledge have not explored before.
3. We repeat experiments for the twenty-eight distinct fault datasets of different domains to establish the feasibility and usefulness of used ensemble techniques for the SFP.
4. Further, we perform a cost-benefit analysis of the used ensemble techniques to assess their economic viability for the SFP.

Following research questions have been framed to investigate in the presented experimental study:

RQ1: *Which ensemble technique shows overall best performance for software fault prediction?*

RQ2: *Is there any statistically significant performance difference between the chosen ensemble techniques?*

RQ3: *How do base learners affect the performance of ensemble techniques?*

RQ4: *For a given software system, how economically effective ensemble techniques are for software fault prediction?*

The structure of the paper is as follows. A discussion on earlier presented similar works is provided in Section 2. Section 3 provides a systematic review of the ensemble techniques based SFP. Section 4 includes the details of the software fault prediction process. Section 5 focuses on the overview of ensemble techniques used for SFP. Section 6 provides details of the empirical study including description of used software fault datasets, performance evaluation measures, experimental procedure, etc. Section 7 presents and discusses results of the study. The comparative study of used ensemble techniques is presented in Section 8.

Section 9 listed various threats to the validity to the presented study followed by the conclusions and future works in the final section.

2 Related work

Many works are available in the literature, which used ensemble techniques/methods for SFP [23, 25, 30, 31]. Tosun et al. [32] built an ensemble based fault prediction model that combines the learning of three different classifiers, naive Bayes, neural network, and voting feature intervals. Authors compared the performance of the presented ensemble model with naive Bayes and found that the presented model has achieved a considerably improved performance. However, authors focused on only one ensemble model and performed experiments for a few NASA datasets. In a similar study, J. Zheng [33] presented and evaluated three cost-sensitive boosting algorithms for SFP. The author used one threshold-updating and two weight-adjusting based algorithms and performed the analysis for four NASA datasets. Results of the study showed that the algorithm based on threshold-updating with the boosted neural network performed the best among the other techniques considered in the study for SFP. Wang et al. [26] presented a study for software defect prediction using some classifier ensembles. Authors assessed the capabilities of seven ensemble techniques such as Bagging, Boosting, Random trees, Random forest, Random subspace, Stacking, and Voting and used naive Bayes as the base learner among the ensemble techniques. Authors performed a series of experiments for several NASA datasets and found that voting and random forest performed better compared to other methods. Overall, authors suggested that ensemble methods produced better performance than a single classifier. B. Twala [34] built an ensemble technique based fault prediction model using three distinct techniques for a large space software system. The author showed that decision tree and apriori techniques based ensemble techniques outperformed other used ensemble techniques and yielded a better accuracy.

Aljamaan et al. [35] performed an investigation of bagging and boosting ensemble techniques for software defect prediction and compared their performance with other commonly used fault prediction techniques. Results found that ensemble based prediction models produced better accuracy values in comparison to most of the used fault prediction techniques. Recently, Siers and Islam [36] presented two ensemble methods, namely, CSForest and CSVoting using cost-sensitive analysis for SFP. The examined ensemble methods initially created a set of decision trees and later combined these trees to minimize the classification cost. Authors showed that presented ensemble methods were able to achieve superior performance compared to other used six classification algorithms.

In the presented work, we performed an extensive analysis of seven ensemble techniques, Dagging, Decorate, Grading, MultiBoostAB, RealAdaBoost, Rotation Forest, and Ensemble Selection for SFP. To the best of our knowledge, most of these ensemble techniques have not been explored and experimented for SFP till now. Further, we use three different classification algorithms as base learners to analyze the impact of base learners on the performance of ensemble techniques. The study was performed for twenty-eight software fault datasets, and a total of 532 fault prediction models have been generated. We believe that the analysis of ensemble techniques presented in this paper will help the research community to build more effective fault prediction models using ensemble techniques.

3 Systematic review of ensemble techniques based software fault prediction

To identify the papers related to the ensemble techniques for the software fault prediction, we have searched in the Google Scholar, IEEE Explorer, ScienceDirect, and Scopus databases and extracted papers published between January 2010 and April 2020. We have selected this timeline for article search, because most the works using ensemble techniques for the software fault/defect prediction published in last decade only. The query string used for the database search is *"(Software Fault OR Defect OR Bug Prediction) AND (Ensemble techniques OR Bagging OR Boosting OR Stacking)"*. The initial query run resulted into a large number of articles. We have applied the inclusion and exclusion criteria to filter out the articles and to select only the relevant articles/papers [37].

Inclusion Criteria

1. Paper must be written in the English language.
2. Full content of the paper must be available online.
3. Paper must be published between January 2010 and April 2020.
4. The study reported in the paper used on the software project datasets not the simulated one.
5. The paper applied at least one ensemble technique for the software fault/defect prediction.
6. Paper must be reported new experiments only.
7. Paper must be reported results using standard performance measures with sufficient details.

Table 1 listed the studies related to the ensemble techniques based software fault /defect prediction (SFP). The use of ensemble techniques for the SFP has expedited

Table 1 Analysis of ensemble techniques based software fault prediction literature

S. No.	Paper	Aim of the study	Used ensemble techniques	Used fault datasets	Used performance measures	Results
1.	[38]	Software defect prediction with class imbalance handling	Ensemble of decision trees (CSForest) and cost-sensitive voting (CSVoting)	MC2, PC1, KC1, PC3,MC1, and PC2 from NASA repository	Weighted precision and recall, Cyclomatic Complexity Density, Halstead metrics	Results showed that presented ensemble methods outperformed decision tree classifier.
2.	[39]	Analysis of combining feature selection and ensemble learning for the software defect classification	Forward selection and average probability ensemble (APE) with SVM	Ant 1.7, Camel 1.6, KC3, MC1, PC2, and PC4	Area under ROC curve (AUC), and G-means	Presented APE method performed correctly even with the poor features and APE combined with forward selection achieved improved AUC values.
3.	[40]	Explore the use of deep learning and two-stage ensemble (TSE) for the software defect prediction	Deep learning and stacking with TSE (SDAEsTSE)	12 NASA datasets, CM1, KC1, KC2, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and JM1	F-measure, AUC, and MCC	Presented SDAEsTSE method produced significantly higher performance compared to the Random Forest, Bagging, AdaBoost.
4.	[41]	Hybridize ensemble methods for the just-in-time defect prediction	Two-layer ensemble learning (TLEL) with decision tree and ensemble learning	Bugzilla, Columba, JDT, Mozilla, Platform, and PostgreSQL	Cost-effectiveness and F1-measure	Results showed that TLEL discovered over 70% bugs and used only 20% of lines of code. The F1-scores of TLEL was significantly higher than other methods.
5.	[42]	Analysis of Bug Prediction using Deep representation and Ensemble learning (BPDET) techniques	Ensemble learning (Staked denoising auto-encoders) and deep feature representation	12 NASA datasets	MCC, AUC, Precison-Recall curve, and f-measure	Presented BPDET performed better for the most of used datasets compared to the AdaBoost, Bagging, Random forest, and Logistic Boost.
6.	[43]	Software defect prediction using ensemble techniques	Weighted majority voting (WM), Randomized weighted majority voting (RWM), Cascading weighted majority voting (CWM), and Cascading randomized weighted majority voting (CRWM) techniques.	Apache Log4j Trunk, Apache Jmeter, Apache Ant, and Apache Tomcat	Accuracy, f-measure, and AUC	Results show that change metrics outperformed other considered metrics. Used ensembles performed better than considered base classifiers.
7.	[44]	Software fault prediction at various metrics levels using ensemble methods	Bagging, boosting, stacking, and voting	KC1 from NASA repository and Eclipse project dataset	Root mean square error, and AUC	Bagging outperformed other used ensemble methods for the used datasets.

Table 1 (continued)

S. No.	Paper	Aim of the study	Used ensemble techniques	Used fault datasets	Used performance measures	Results
8.	[45]	Evaluation of ensemble methods for software fault prediction	AdaboostM1, Vote and StackingC with Naïve Bayes, Logistic, J48, VotedPerceptron, and SMO as base learners	Ant-1.7, Camel-1.6, e-Learning, Forest-0.8, jEdit-4.3, Tomcat, Xalan 2.7, Xerces-1.4, Zuzel, Berek, pbean2, and Velocity-1.6	Accuracy and f-measure	StackingC outperformed other used ensemble methods as well as the base learners.
9.	[46]	Ensemble based software defect prediction using diversity selection	Weighted accuracy diversity (WAD) and stacking	Ant 1.5, Ant 1.6, jedit-4.1, jedit-4.2, Tomcat, Xalan-2.5, and Xalan-2.6	Precision, MCC, and diversity	Naïve Bayes and SMO provided better results with the presented WAD technique.
10.	[47]	Explore the use of ensemble learning for the software fault prediction	AdaBoost, Bagging, Randon forest, Random subspace, and Voting	CM1, JM1, KC1, KC2, and PC1	Precision, f-measure, and AUC	Random forest with the SMOTE techniques yielded the best performance.
11.	[48]	Present a three stage ensemble learning framework for the software fault prediction	Ensemble learning, feature selection, SMOTE, and Noise reduction techniques	AR1, AR3, AR4, AR5, AR6, CM1, JM1, KC2, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5	Accuracy, f-measure, Root mean square error	Presented ELA approach with feature selection and data balancing produced a higher performance.
12.	[49]	Software defect prediction using heterogeneous ensemble method	A hybrid ensemble approach using different classifiers	21 fault datasets from NASA and PROMISE data repository	Precision, Recall, and G-means	Presented ensemble methods outperformed bagging, Adaboost and other used base learning techniques for all the considered cases.
13.	[50]	A comparison of ensemble methods for the software fault prediction	Bagging, boosting, and rotation forest with 8 base learners	MC2, MW1, KC3, CM1, KC1, PC1, and PC4 from NASA repository	Accuracy and Recall	Results showed that rotation forest with the resampling yielded the best performance for the software fault prediction.
14.	[51]	Evaluation of sampling based ensembles on imbalanced data for software defect prediction	A ensemble method based on bagging mechanism and SMOTE	JM1, KC3, PC1, Ant-1.7, Camel-1.6, Ivy-2.0, Poi-2.0, Tomcat, Xalan-2.4, Synapse-1.2	Accuracy, Precision, Recall and f1-score	Results showed that combining ensemble learning with the sampling techniques increased prediction performance.
15.	[52]	Software fault prediction in the large space systems	Apriori, Decision tree, K-nearest neighbour, naïve Bayes, Support vector machine, a majority voting based ensemble method	CM1, JM1 and PC1 from NASA repository	Error rate	Results showed that Naive Bayes classifier and ensembles with a decision tree classifier achieved higher accuracy rates.

Table 1 (continued)

S. No.	Paper	Aim of the study	Used ensemble techniques	Used fault datasets	Used performance measures	Results
16.	[53]	Cross-project software defect prediction (CPDP)	A cost-sensitive boosting approach	Ant, ARC, Camel, e-learning, jedit, Log4j, Lucene, Poi, Prop-6, Redaktor, Synapse, Systemdata, Tomcat, Xalan, and Xerces from PROMISE repository	Probability of detection, probability of false alarm, G-mean, and Balance	Results showed that combining transfer learning and class imbalance improved the performance of CPDP.
17.	[54]	Cross-project software defect prediction	AdaBoost, Bagging, and SMOTE	CM1, MW1, PC1, PC3, and PC4 from NASA repository	Accuracy, True positive rate, False positive rate, and AUC	Results found that bagging method combined with the SMOTE performed the best.
18.	[55]	Ensemble learning for the software defect prediction	Multiple kernel ensemble learning (MKEL) technique	CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5 from NASA repository	F-measure, probability of detection, probability of false alarm	Presented MKEL technique produced improved results for all used 12 datasets and it also outperformed other considered state-of-the-art approaches.
19.	[56]	Prediction of software black-box defects	Stacked generalization approach (PMoSG)	45 industrial project datasets	Root mean square error and correlation coefficient	PMoSG approach achieved improved prediction performance than LibSVM, LR, IBK, Decision tree, and MLP.
20.	[57]	Use of a coding-based ensemble learning for the software defect prediction	Class imbalance technique, three coding-based ensemble methods	CM1, JM1, KC1, KC2, KC3, KC4, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5 from NASA repository	AUC and Ranking	Presented methods outperformed cost-sensitive learning, bagging, boosting, and four other base classifiers (random forest, C4.5, naïve Bayes, and Ripper).
21.	[58]	Exploration of ensemble methods for the fault prediction in the Eclipse projects	Bagging, Boosting, Random Subspace, Rotation forest, and stacking	Average absolute error and average relative error	AS_eclipse 2.0, AS_eclipse 2.1, and AS_eclipse 3.0	Random subspace outperformed other used ensemble methods. Bagging, boosting and rotation forest performed moderately.
22.	[59]	A comparative study of ensemble based learning for the software fault prediction	Bagging and AdaBoost.M1 with J48 and DT as base learners	Accuracy and AUC	AR1, AR4, JM1, KC2, MC1, MW1, PC3, and PC4	Results showed that ensemble learning combined with feature selection and data balancing achieved improved performance.
23.	[60]	Use of linear homogeneous ensemble for software fault prediction	Extreme Learning Machine Ensemble (DELME) and Non-differentiable Extreme Learning Machine Ensemble (NELME)	Average absolute error, average relative error, MoC, Prediction at *level l*	22 fault datasets from PROMISE data repository	Proposed methods performed consistently better for all the used datasets.
24.	[61]	Static and dynamic ensemble approaches for the software fault prediction	Over-Bagging and ensemble selection, Omni-Ensemble Learning (OEL)	Probability of detection, probability of false alarm, AUC, and G-mean	MC2, KC2, KC3, CM1, MW1, and PC1	OEL produced the best performance for all used performance measures.

Table 1 (continued)

S. No.	Paper	Aim of the study	Used ensemble techniques	Used fault datasets	Used performance measures	Results
25.	[62]	Analysis of heterogeneous ensembles for online failure prediction	Decision tree, bagging, random forest, gradient boosting, and stacking with soft voting	Confusion matrix parameters	Windows XP (SP3)	Results suggested that combination of weak learners often improved the overall prediction performance.
26.	[63]	Software fault prediction for the imbalanced datasets	Ensemble MultiBoost based on RIPPER classifier (EMR_SD), PCA, and ADASYN	Accuracy, Recall, Precision, F-measure, ROC, Balance, and probability of false alarm	15 fault datasets from NASA repository	Results showed presented EMR_SD yielded better performance than used DNC, CEL and other defect prediction techniques.
27.	[64]	Handling of imbalanced data using ensemble learning in software defect prediction	AdaBoost.Ml, AdaBoost.M2, RUSBoost, SMOTEBoost, MSMOTEBoost, DataBoost	AUC, G-mean, and Balance	Synapse-1.0, Camel-1.4, and Ant-1.7	Results showed that RUSBoost produced the best performance followed by MSMOTEBoost and SMOTEBoost ensemble methods.
28.	[65]	Software defect prediction with cost-sensitive boosting approach	Cost-sensitive boosting neural networks (CSBNN)	Misclassification rate, type I and type II errors	KC1, KC2, CM1, and PC1	CSBNN always achieved lowest error rate among the used models and it also obtained good performance.
29.	[66]	A source code metric and ensemble approach for the software fault prediction	Best Training Ensemble, Majority Voting Ensemble, Nonlinear Ensemble Decision Tree Forest with five base learners	Accuracy, F-measure, and cost-analysis	45 fault datasets from PROMISE data repository	Results showed that majority voting ensemble produced the best results and ensemble based prediction models are also cost-effective.
30.	[67]	Handling software defect prediction for imbalance data	AdaBoost and back-propagation neural network (BPNN)	Accuracy, Precision, Recall, Specificity, F-measure, and G-mean	CM1, KC3, PC4, and PC5	Results found that Adaboost outperformed BPNN for the imbalanced fault datasets.
31.	[68]	Analysis of meta-heuristic algorithm and ensemble methods for the software fault prediction	Particle swarm optimization (PSO) and bagging ensemble methods	Area under the ROC curve	CM1, KC1, KC3, MC2, MW1, PC1, PC2, PC3, and PC4	Combination of PSO and bagging methods with the feature selection outperformed other 11 used classifiers for the software fault prediction.
32.	[69]	Analysis of meta-heuristic algorithm and ensemble methods for the cross-project defect prediction	Harmony search-based optimization and cost-sensitive boosting	Probability of detection, probability of false alarm, G-means, and Balance	15 fault datasets from PROMISE data repository	Results showed that use of meta-heuristic algorithm and cost-sensitive boosting with the parameter tuning help in the performance improvement of the techniques.
33.	[70]	Explore the use of ensemble techniques for the bug assignment in the large industrial contexts	Stacked Generalization (SG) with five base learners	Accuracy	A large power and automated company and a large telecommunication company	SG consistently outperformed base learners. Additionally, results suggested that bug assignment must use at least 2,000 bug reports for training of SG.

Table 1 (continued)

S. No.	Paper	Aim of the study	Used ensemble techniques	Used fault datasets	Used performance measures	Results
34.	[71]	Heterogeneous software defect prediction	Two-Stage Ensemble Learning (TSEL), Ensemble Multiple Kernel Correlation Alignment (EMKCA), and RESample with replacement (RES) techniques	AUC, Precision, Recall, F-measure, and Balance	30 software fault datasets from various software repositories	Results showed that TSEL performed better than the baseline methods.
35.	[72]	An industrial case study of ensemble methods for locating software defects	Neural network, naïve Bayes, and voting feature intervals	Probability of false alarm (PF), Probability of detection (PD), and Balance		Proposed ensemble method reduced PF and increased precision with the combination of voting scheme.
36.	[73]	Cross-project software defect prediction (CPDP)	Value-cognitive boosting and support vector machine (VCB-SVM)	AUC, H-measure, Probability of detection and Probability of false alarm	PC1, KC1, KC2, KC3, CM1, MW1, MC2, AR3, AR4, and AR5	VCB-SVM provided higher prediction performance than existing CPDP technique and the existing class imbalance techniques.
37.	[74]	Cross-project software defect prediction (CPDP)	A cost-sensitive boosting approach (TCSBoost)	Probability of detection, Probability of false alarm, G-mean, and Balance	15 fault datasets from PROMISE data repository	The proposed TCSBoost resulted in the improved CPDP performance.
38.	[75]	A hybrid approach for the software defect prediction	Analytic Hierarchy Process (AHP) and ensemble methods (bagging, boosting, and stacking)	Accuracy, TPR, FPR, TNR, FNR, Precision, Recall, F-measure, AUC, Kappa, and MAE	CM1, JM1, KC3, KC4, MC1, MW1, PC2, PC3, and PC4	AdaBoost gives the best results and K-nearest neighbour was the best performing base learner.
39.	[76]	Use of combined classifier for cross-project defect prediction (CPDP)	Meta classification algorithm (CODEPLogistic) and 5 base learners	F-measure, Precision, Recall, Mean average precision, and cost effectiveness	Ant, Camel, Ivy, jedit, Log4j, Lucene, Poi, Prop, Tomcat, and Xalan from PROMISE data repository	Results showed that Bootstrap aggregation with J48 yielded the best performance and outperformed CODEPLogistic.
40.	[77]	A comparison of ensemble techniques with feature selection for the software fault prediction	17 different feature ranking techniques and their ensemble using stacking and Naïve Bayes algorithm	F-measure, Odd-ratio, Probability ratio, AUC, G-means, and Area under precision and recall curve	16 fault datasets from Eclipse and NASA repository	Results showed that no particular ensemble method outperformed other used methods. Further, ensembles of few rankers were than ensembles of many or all rankers.
41.	[78]	Ensemble of regression approaches for the cross-project fault prediction	Ensemble of regression	Area under the ROC curve	CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5 from NASA repository	Results showed that presented method performed better than regression approaches.
42.	[68]	Predicting number of defect in the software system	Swarm Ensemble Clustering Technique	Accuracy	KC1, CM1, and KC2	Results showed that PSO with Manhattan similarity measure performed as good as or better than the other algorithms.

Table 1 (continued)

S. No.	Paper	Aim of the study	Used ensemble techniques	Used fault datasets	Used performance measures	Results
43.	[79]	Heterogeneous defect prediction	Ensemble Multiple Kernel Correlation Alignment (EMKCA)	Area under the ROC curve	30 software project datasets	Presented EMKCA produced better performance than multiple kernel learning and ensemble learning.
44.	[80]	Heterogeneous defect prediction	Krnel spectral embedding transfer ensemble (KSETE)	Probability of detection, AUC, G-measure, and MCC	22 software project datasets	The experimental results showed that KSETE is effective in the both HDP and CPDP-CM scenarios.
45.	[26]	Classifier ensemble based software defect prediction	Bagging, Boosting, Random trees, Random forest, Random subspace, Stacking, and Voting	Area under the ROC curve	CM1, JM1, KC1, KC2, KC3, KC4, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5 from NASA repository	Results showed that majority voting yielded the best performance followed by random forest.
46.	[35]	Identification of faulty classes in object-oriented software	Bagging and Boosting ensemble methods, MLP, RBF, BBN, NB, SVM, and DT	Accuracy	KC1 from NASA data repository	The results indicated that used ensemble methods outperformed all used single classifiers. Whereas, among ensemble techniques, sometimes, bagging performed better and other times boosting performed better.
47.	[23]	Heterogeneous ensemble methods for the software fault prediction	Four ensemble methods (ERWT, LRWT, DTF, and GBR) with three base learners, DTR, MLP, and LR	Average absolute error, Average relative error, Prediction at level l, and measure of completeness	15 fault datasets from PROMISE data repository and 3 from Eclipse repository	Results showed that non-linear rule based (DTF and GBR) performed the best for the fault prediction.

since 2010 [38–41]. The review of the ensemble techniques showed that a large number of researchers have focused on the use of bagging, boosting, and stacking based ensemble techniques. Different studies have used different classifiers as the base learners to these ensemble techniques such as naïve Bayes, decision tree, multilayer perceptron, etc. Results of the analysis showed that these ensemble techniques produced higher or at least equal performance as compared to the base learners [23]. Some other researchers have explored different variations of the traditional ensemble techniques such as cost-sensitive neural network, cost-sensitive boosting, bagging with the oversampling, etc. Authors claimed that these variations of the ensemble techniques resulted in an improved performance as compared to the traditional ensemble techniques [65, 74]. A few researchers have used hybrid ensemble techniques such as ensemble techniques with the feature selection, ensemble techniques sampling, etc. These studies showed that the use of hybrid ensemble techniques could be useful in building accurate fault prediction models [55, 61]. However, over the last few years many new or improved ensemble techniques have been presented by the researchers. Although, a comprehensive evaluation of these newly available ensemble techniques is missing. Thus, in this work, we include the ensemble techniques, which are not explored before for the SFP.

4 Software fault prediction process: An overview

In this section, we have discussed a generic process used for the prediction of software faults. There are many works reported in the literature presenting various approaches for the software fault prediction. The aim of this section is to discuss the commonly used steps for the software fault prediciton based on various available works [81, 84–87]. These steps are also useful in building the ensemble models under study for software fault prediction discussed in the upcoming sections.

The aim of software fault prediction (SFP) is to identify the software modules having a higher probability of being faulty. The SFP process is based on the use of some underlying characteristics such as source code metrics, change and revision history, structural properties, etc. of the software project. The SFP model uses such software project datasets augmented with corresponding fault information for a known project as a training dataset, and subsequently uses the trained SFP model to predict faults for unknown projects. The working assumption of the SFP process is that if a software project developed in an environment that led to faults, then any subsequent software modules developing in a similar environment with similar underlying characteristics will end to be faulty [81, 82]. Let us say that the software fault dataset is defined as $D = \{X, Y\}$, where X represents a set of software metrics (features or attributes or independent variables) and it is a matrix of N X M size. N is the number of rows (software modules) and M is the number of features. Y represents fault information (dependent variable) and it is a vector of N size. $\{x_i, y_i\}$ is the i^{th} observation in the dataset. The dependent variable is (DV) $y_i \in [1, 0]$, where "1" stands for a faulty software module, and "0" stands for the non-faulty software module. The prediction models are built on the dataset D and aim to classify the unseen software modules in faulty or non-fault labels, yielding classifier results $y_i = (x_i)$. If we use a classification algorithm to build the SFP model, then it is often referred to as the classification model or binary classification model given its binary outcome.

Figure 1 depicts an overview of the software fault prediction process. The process shown in the figure and depicted as below is a generic process used for the prediction of software faults. The steps involve in the SFP model building and assessment are described as follows [83].

1. Extraction of fault information: Each software project has source code and bug repositories such as SVN or CVS. The extraction of fault information involves data retrieval from the bug repository and linking it to its source. Based on the log contents and status of the bug, it is decided whether a commit is a bugfix or not. All such reported bugs are collected from the bug repository and mapped to their corresponding source code modules.
2. Collecting software metrics (features or attributes) and creating fault dataset: This step collects software metrics information from the source code of the software project or from the log contents of the projects. First, it is decided that what type of properties of the given software are required. Further, based on that source code or log files are parsed and corresponding software metrics are collected. Last, extracted fault information and collected software metrics are combined together to create the fault dataset that is used to train the SFP model.
3. Building SFP models: Usually, some classification algorithms or regression techniques such as decision tree, support vector machine, naïve Bayes, or linear regression are used to build the SFP model using fault set. Subsequently, the trained SFP model is then used to predict the faults in the unseen software modules.
4. Evaluation: To assess the SFP model's performance, generally a separate testing dataset is used besides the training dataset. This testing dataset is created by partitioning the fault dataset into training and testing

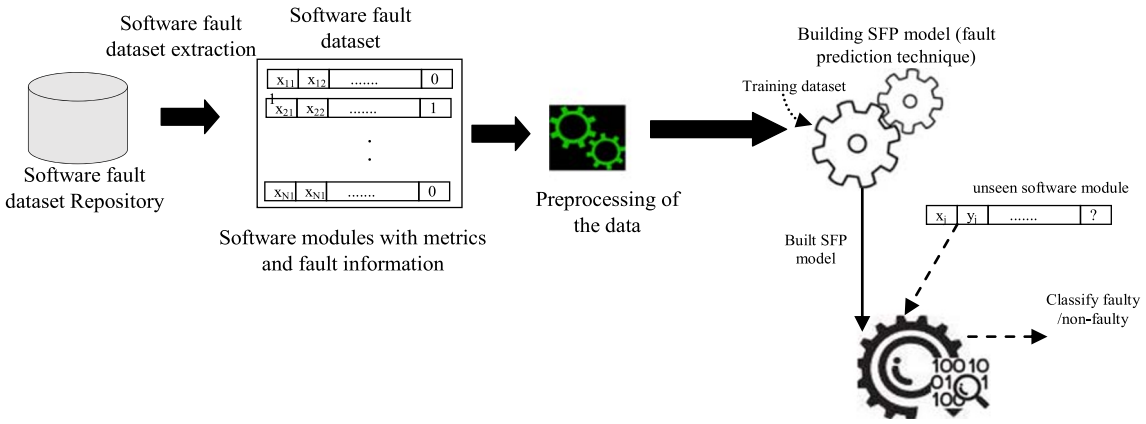

Fig. 1 Software fault prediction process

parts. The fault-proneness of software modules in the testing dataset is predicted. Then, the performance of the model is evaluated, by comparing the predicted value of faults and the corresponding actual value of faults.

A number of researchers have explored/presented different models for the software fault prediction. Most of these works focused on the binary class classification of faults (faulty or non-faulty) [84–87]. Some of the researchers have built prediction models for the number of faults in a software module prediction or the severity of the fault prediction [88–90]. The results of these studies showed that the average prediction accuracy of software fault prediction models was 80%-85% (approx.) with 30%-40% of the misclassification rate. Additionally, it has been found that no single learning technique (classifiers or regression techniques) always performed better than the other techniques across different software projects [91]. However, some learning techniques such as Naive Bayes, Logistic Regression, and random forest achieved better performance than techniques such as support vector machine (SVM) and multiplayer perceptron (MLP). Athough, in some cases SVM or MLP yielded better performance than other techniques [4]. A few researchers have performed a comparative analysis study or meta-analysis study of learning techniques for the software fault prediction [82, 92]. Recently, Li et al. [93] and Ning Li et al. [37] have reported benchmark studies for software fault prediction in the years 2019 and 2020, respectively. In 2019, Li et al. [93] reported an updated benchmark study, where authors evaluated various classifiers using new fault datasets and new evaluation metrics. The result analysis showed that techniques such as bagged MLP, ANN/MLP, decision tree, and random forest yielded better prediction performance as compared to the techniques such as CART, Logistic regression, SVM, Naïve Bayes, etc. Further authors stated that there is no single best classifier found for the SFP. Moreover, the authors suggested the use of simple classifiers over the complex ones for the SFP due to the problem of hyper-parameter tuning of the classifiers. In 2020, Ning et al. [37] reported a systematic review and meta-analysis of unsupervised learning techniques for software defect prediction. After, the thorough screening of the works published between 2000 and 2018, the authors included a total of 49 studies in their presented meta-analysis. The results of the meta-analysis showed that the performance of unsupervised learning techniques was comparable with supervised learning techniques for both within-project and cross-project prediction. Among the considered unsupervised learning techniques, Fuzzy CMeans (FCM) and Fuzzy SOMs (FSOMs) yielded the best performance. Further, the authors stated that factors such as dataset characteristics did not show any significant impact on the performance of unsupervised techniques.

5 Ensemble techniques for software fault prediction

Ensemble technique refers to the technique that generates several intermediate prediction models, which are integrated together to make an overall prediction [94]. The primary purpose of an ensemble technique is to overcome the performance ceiling problem of the single learning algorithm and to enhance the overall performance of prediction model. Several techniques are available in the literature to generate the intermediate prediction models for the ensemble techniques [95]. Ensemble techniques make an effective use of these intermediately generated prediction models to reduce the variance in the prediction performance without increasing any bias [96]. In this work, the SFP problem is defined as a classification task, where the aim is to categorize the given software modules into the faulty or non-faulty classes. The technique used for the prediction takes the form of a function f, which uses a vector of size $n+1$ of n software metrics $(A_1, A_2, ..., A_n)$ and one dependent variable (D, fault information) as input and outputs (Y) fault-proneness of the

given software modules. Each vector of software metrics and dependent variable describes a software module i.e., a class in object-oriented software systems or a file in other software systems. The calibration of f is done on the training dataset (TR) having several such vectors or examples. The dependent variable is faulty and non-faulty information of a software module.

Figure 2 shows the working of ensemble techniques for the SFP. The process of building a prediction model using ensemble technique is two-folded: (1) generation of intermediate prediction models to be used for the ensemble (ensemble generation), and (2) integration of generated prediction models for the ensemble to obtain the final prediction (ensemble integration) [95]. Ensemble techniques utilize multiple models (known as "weak learners") that are trained and combined to get improved results. The accurate working of ensemble techniques depends on the correctly combined weak learners. In ensemble theory, a weak learner is a model that does not perform so well alone either because it has a high bias or high variance. Ensemble techniques overcome this high bias-variance problem by combining several weak learners to reduce bias and variance of such weak learners. Most of the ensemble techniques rely on the use of a single base learning algorithm to generate multiple weak learners. However, each instance of the weak learner is trained differently. This setting is known as homogeneous ensemble techniques. However, some ensemble techniques use different learning algorithms to generate weak learners. It is known as heterogeneous ensemble techniques. The next step is the correct aggregation of weak learners. Different ensemble techniques combine weak learners differently. For example, in the bagging ensemble technique, weak learners are combined by using a deterministic averaging process. In boosting ensemble technique weak learners are generated adaptively and combined using a deterministic strategy. In the stacking ensemble technique, weak learners are combined using a meta-model that learns on the outputs of a weak learner and combined their outputs.

Despite the use of the type of ensemble techniques, every ensemble technique takes one or some learning algorithms as the input. Additionally, a training dataset is taken as input by ensemble techniques. Depend on the number of weak learners to be generated, the input training dataset is partition into several subsamples. One weak learner is trained on the one subsample of the training dataset. The output of this training phase is the several trained weak learners on the different subsamples. Next, based on the used combination strategy, weights of each weak learner are decided and their outputs are combined for the final prediction. There are several techniques proposed by researchers for the ensemble generation and ensemble integration [95]. In the presented work, we focus on the homogeneous ensemble generation techniques, where the same algorithm is used to generate intermediate prediction models. There are seven different homogeneous ensemble techniques used in the study and the description of these techniques is given as follows.

1. **Dagging:** In this ensemble technique, initially, several disjoint stratified subsets of the given original fault dataset is generated. Subsequently, the generated subsets are fed to the classification algorithm (base learner). The final prediction is made by using the majority voting scheme to combine the outcomes of the base learner for all the generated subsets [97]. It differs from the bagging in the sense that here disjoint subsets of given dataset are used to build the prediction models.

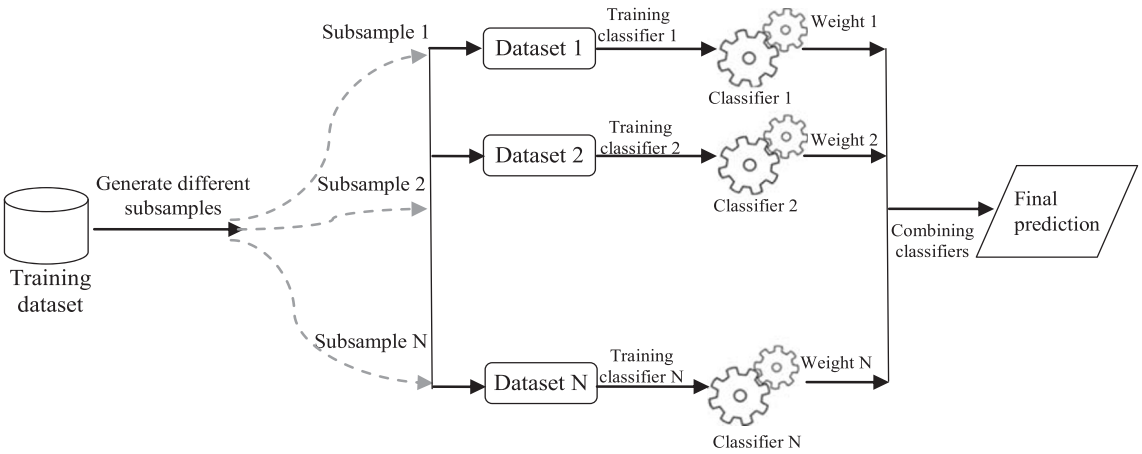

Fig. 2 Working of ensemble techniques for the SFP

2. **Decorate:** This ensemble technique generates diverse intermediate prediction models by using specially constructed artificial training examples. It follows an iterative ensemble generation process. In each iteration, an intermediate prediction model is generated and added to the current ensemble. The base learner is trained in each iteration for the training dataset augmented with some artificially generated data points. The population of artificial training data points is drawn from the original data distribution and it is specified as a fraction of the training dataset size [98]. The class of these artificial data points is maximally different from the current ensemble's predictions.
3. **Grading:** It is a meta-classification scheme, which uses graded predictions on the meta-level classes to make the final prediction [99]. For each base learner, a meta-classifier is learned whose task is to predict when the base learner will be incorrect. Graded prediction is a prediction that has been marked as correct or incorrect. The training dataset for meta-classifier is made up using the graded predictions of the corresponding base learners as new class labels for the original attributes. The final prediction is derived from the predictions of base learners that are predicted to be correct by the meta-classification schemes [100].
4. **MultiBoostAB:** This ensemble technique extends the working of AdaBoost ensemble technique. It combines the capabilities of AdaBoost with wagging techniques to reduce the prediction bias and variance in the final model [101]. The advantage of MultiBoost technique over AdaBoost is that in contrast to the AdaBoost, in this technique, intermediate models can learn in parallel, which speed up the training and model building process.
5. **RealAdaBoost:** RealAdaBoost is a modified version of AdaBoost ensemble technique that fits an additive logistic regression and produces a non-linear version of logistic regression [102]. It extends AdaBoost techniques and removes the need for a coefficient as the optimal coefficient is always 1. Additionally, it generates fewer trees than AdaBoost to reach the final prediction [103].
6. **Rotation Forest:** This ensemble technique makes use of a PCA (Principal Component Analysis) algorithm to choose features and instances of the training dataset when building decision trees [104]. First, the features of the training dataset are split into K non-overlapping subsets of equal size. Then, 25% of the training data examples are removed randomly by using a bootstrap method and PCA is used for the rest of 75% of data examples. These steps are repeated for each tree in rotation forest and the final prediction is based on the integrated outputs of each tree.
7. **Ensemble Selection:** Ensemble selection is a meta-classification ensemble technique. It uses a set of base learners to generate the final ensemble. Initially, technique starts with an empty ensemble. Iteratively, it adds a base learner to the ensemble library that maximizes the ensemble's performance. This process is repeated for a fixed number of rounds and the final ensemble based prediction model is the nested set of base learners that maximizes the prediction performance [105].

6 Empirical study

6.1 Experimental datasets

In this work, fault datasets were gathered from the PROMISE data repository for building and evaluating prediction models[1] [29]. A total of twenty-eight benchmarked software fault datasets have been gathered from the mentioned repository. Considered fault datasets include data of several open-source software systems such as Apache Camel, Apache Xerces, Apache Xalan, PROP, etc. The details of considered datasets are given in Table 2. The used datasets (described in Table 2) are same as ones used in our one previous paper [16]. All the used fault datasets are having 300 or more software modules. We have drooped all the smaller size datasets below the given threshold limit of 300 modules. Each of the used dataset contained twenty-one object-oriented software metrics and number of faults found in each software module. Since, aim of the presented study is to classify software modules into faulty or non-faulty modules, therefore, we performed data transformation on these datasets and categorized given number of faults information into faulty and non-faulty classes. Software modules with one or more faults have been marked as faulty, other modules with zero faults have been marked as non-faulty. We apply the same data transformation scheme on all twenty-eight datasets. The considered dependent variable is faulty and non-faulty labels of the software modules.

6.2 Experimental procedure

Figure 3 depict the procedure used for the experimental study presented in the paper.

[1] https://sites.google.com/site/santoshiiitmdj/software-fault-datasets?authuser=0

Experimental Procedure:

Input: S: A set of twenty-eight software fault datasets having object-oriented software metrics and fault information
L: A set of seven ensemble techniques,
L = {Dagging, Decoarte, Grading, MultiBoostAB, RealAdaBoost, Rotation Forest, and Ensemble Selection}
B: A set of three base learners, B={Naive Bayes, Logistic Regression, and J48}

Output: Performance of used ensemble techniques for precision, recall, AUC, specificity, and G- means measures
Results of used statistical tests

Preprocessing: Transform the number of faults value information into faulty and non-faulty classes by labeling examples with one or more faults as faulty and modules with zero fault as non-faulty

Begin:

1. *for* each software fault dataset $s \epsilon S$
2. *for* each ensemble technique $l \epsilon L$
3. *for* each base learner $b \epsilon B$
4. build model with the ensemble technique l and base learner b over s and by doing a 10-fold cross-
5. validation calculate the value of confusion matrix parameter (TP, FP, TN,and FN) // where TP:True // positive, FP : False positive, // FN : False negative, TN = // True negative
6. end *for*
7. end *for*
8. end *for* // Performance values of all used ensemble // techniques in terms of TP, FP, FN, and TN will be //populated after this step
9. calculate values of performance measures, precision, recall, AUC, specificity, and G-means
10. perform the statistical tests for each pair of ensemble techniques and calculate results

End

The experimental procedure mainly consists of three steps. In initial step, training and testing subsets are generated from the original fault dataset by splitting it into multiple partitions. A ten-folds cross-validation scheme is used to build prediction model and evaluate the performance of ensemble techniques. This scheme partitions the original fault dataset into ten disjoint folds. For each iteration, nine folds are served as training dataset used to train the ensemble techniques and remaining one is served as testing dataset used to evaluate the performance of ensemble techniques. This process is repeated for ten folds. The second step is the building of the ensemble techniques. The selected training dataset is used to build the prediction model. Three different classification algorithms are used as base learners to the ensemble techniques. Each time a different classification algorithm is fed to the ensemble technique. This process is repeated for all the base learners. The final step is the evaluation of built ensemble based fault prediction models for the testing dataset. Various performance measures are used to evaluate the performance of built models. Further, Friedman's test and Wilcoxon signed rank test are used to evaluate the statistically significant performance difference among the chosen ensemble techniques. The experimental procedure is described as follows.

6.3 Base Learners

Three different classification algorithms namely, naive Bayes, logistic regression, and J48 (decision tree) have been used as base learners. Previous research showed that these algorithms produced better performance compared to other classification algorithms for the SFP [4]. For this reason, we have selected these algorithms as base learners to feed into ensemble techniques. A brief description of these algorithms is given as follow.

1. **Naive Bayes (NB):** Naive Bayes algorithm belongs to the Bayesian classifier family. Its working is based on the use of Bayes equation to categories the given testing module into one of the classes [106]. Initially, naive Bayes calculates the posterior probability of each class using the attribute values (software metrics) of the given module. Further, the testing module is classified with the label the same as the class label of the highest probability class. Parameter estimation process of naive Bayes classifier involves a simple estimation of the probability of attribute values within each class from the training modules. A comprehensive description of naive Bayes can be referred from [107].
2. **Logistic Regression (LR):** LR is a type of regression technique used when response variable is of categorical type. It calculates the probability of a binary response variable using one or more independent variables (software metrics) [108]. The simple logistic model only predicts the probabilities of outcomes in terms of input values. To use it as classifier, we need to select a cutoff value (threshold), which classifies values greater than cutoff into one class and values lower than cutoff into another class.The more details of logistic regression is given in [109].
3. **J48 (decision tree):** As the name implies, decision tree form a tree type of structure to make the decisions. Building the decision tree involves selection of tree nodes and splitting criteria along with the knowing when to stop [110]. Initially, it selects the most promising node as the root node of the tree and continues with the tree construction with intermediate

Table 2 Details of considered software fault datasets [16]

"S. No.	Dataset	Release	# non- commented -LOC	Total number of modules	Total number of faulty modules	% of faulty modules
1.	Ant	Ant-1.7	208KLOC	746	166	22.25%
2.	Camel	Camel-1.0	33KLOC	340	13	3.82%
3.		Camel-1.2	66KLOC	609	216	35.47%
4.		Camel-1.4	98KLOC	873	145	16.61%
5.		Camel-1.6	113KLOC	966	188	19.46%
6.	Ivy	Ivy-2.0	87KLOC	353	40	11.33%
7.	Jedit	Jedit-4.0	144KLOC	307	75	24.43%
8.		Jedit-4.1	153KLOC	313	79	25.24%
9.		Jedit-4.2	170KLOC	368	48	13.04%
10.		Jedit-4.3	202KLOC	493	11	2.23%
11.	Lucene	Lucene-2.4	102KLOC	341	203	59.53%
12.	Poi	Poi-2.0	93KLOC	315	37	11.75%
13.		Poi-2.5	119KLOC	386	248	64.25%
14.		Poi-3.0	129KLOC	443	281	63.43%
15.	Prop	Prop-1	3816KLOC	18472	2738	14.82%
16.		Prop-2	3748KLOC	23015	2431	10.56%
17.		Prop-3	1604KLOC	10275	1180	11.48%
18.		Prop-4	1508KLOC	8719	840	9.63%
19.		Prop-5	1081KLOC	8517	1299	15.25%
20.		Prop-6	97KLOC	661	66	9.98%
21.	Tomcat	-	300KLOC	859	77	8.96%
22.	Xalan	Xalan-2.4	225KLOC	724	111	15.33%
23.		Xalan-2.5	304KLOC	804	387	48.13%
24.		Xalan-2.6	411KLOC	886	411	46.39%
25.		Xalan-2.7	428KLOC	910	898	98.68%
26.	Xerces	Xerces-1.2	159KLOC	441	71	16.10%
27.		Xerces-1.3	167KLOC	454	69	15.20%
28.		Xerces-1.4	141KLOC	589	437	74.19%"

promising nodes. Typically, information gain (Infogain) or Gain Ratio is used as the splitting criteria [111]. We used J48 algorithm is the present study, which is an implementation of decision tree in the Weka machine learning tool [112].

6.4 Implementation details

The implementation of all ensemble techniques has been performed using Weka machine learning tool [113]. The parameter values of different used ensemble techniques and

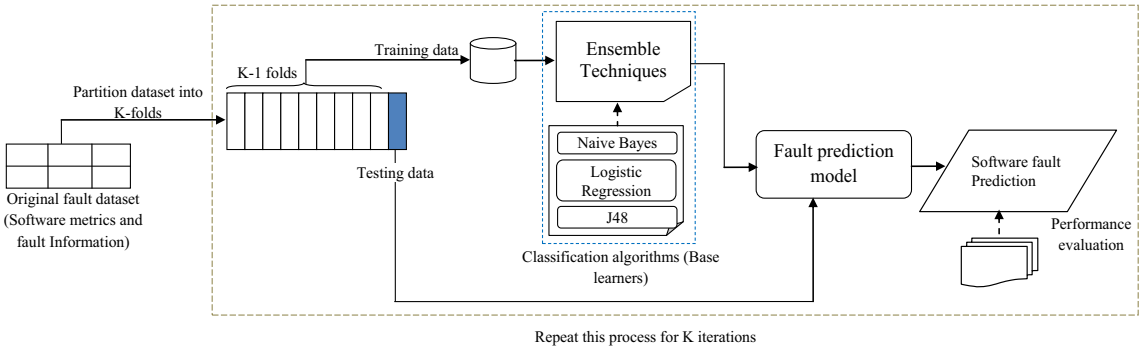

Fig. 3 Overview of the experimental procedure

base learners are given in Appendix. Each used ensemble technique receives the training dataset having software metrics and corresponding fault information as input. The training dataset is used to train the SFP model based on the internal working of the ensemble technique. After the training, a separate testing dataset is fed as input to the trained SFP model and a prediction is made for the software modules of the testing dataset. Each ensemble technique output the faulty or non-faulty labels of the given software modules. We have used seven different ensemble techniques and different classification algorithms for software fault prediction. So, a total of nineteen fault prediction models have created for a fault dataset. We replicated the experiments for twenty-eight fault datasets. Therefore, 532 total fault prediction models have been created.

6.5 Performance evaluation measures

Five different performance measures namely, precision, recall, AUC (area under ROC curve), specificity, and G-means have been used to evaluate the performance of all seven ensemble techniques [23, 114]. It was reported in previous studies that accuracy measure does not provide a complete evaluation of the model performance due to the imbalance in the fault datasets. For this reason, we have excluded it from the study. We have selected those measures, which can provide complete model evaluation despite the imbalance in the fault datasets [115]. An explanation of these performance measures is as follows[2].

(i) Precision: It is used to identify the portion of the correctly predicted faulty modules out of all modules predicted faulty. It is defined by Equation 1.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

(ii) Recall: It is used to identify how many correct faulty modules are predicted. It is defined by Equation 2.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

(iii) AUC: It stands for area under the receiver operating characteristic curve. It is a graphical plot that depicts the diagnostic capabilities of a prediction model under different threshold values. It plots the true positive rate in the y-axis and false positive rate in the x-axis. Area under the curve shows the probability that a classifier will classify a randomly chosen positive module higher than a randomly chosen negative module.

[2]TP = True positive, FP = False positive, FN = False negative, TN = True negative, N = Negative

(iv) Specificity: It is used to identify the portion of negative modules that are actually predicted correctly by a model. Specificity therefore quantifies the avoiding false positive. It is defined by Equation 3.

$$Specificity = \frac{TN}{N} \tag{3}$$

A high value of specificity shows that the prediction model has a low false positive rate and thus helps in a significant reduction in the resource consumption to the false alarm cases. However, a low value of specificity signifies a higher false positive rate and thus a high consumption of resources on the false alarm cases.

(v) G-means: It stands for geometric means. Two measures, G-mean 1 and G-mean 2 are generally used together.

G-mean 1 is calculated as the square root of the precision and recall. G-mean 2 is calculated as the square root of the product of recall and specificity. They are defined by (4) and (5), respectively.

$$G - mean1 = \sqrt{Precision * Recall} \tag{4}$$

$$G - mean2 = \sqrt{Specificity * Recall} \tag{5}$$

(vi) Statistical tests: We perform Friedman's test and Wilcoxon signed rank sum test to identify the difference in performance of the used ensemble techniques [116]. Both the used tests are nonparametric in nature, so they do not make any assumptions related to the normality of the data points. In this test, significance level (α) is set to 0.05, which shows 95% probability of not accepting the null hypothesis when it is true. For these tests, the framed null hypothesis (H_0) and alternative hypothesis (H_a) are as follow:

H_0: There is no significant performance difference among the used ensemble techniques at the given significance level.

H_a: There is a significant performance difference among the used ensemble techniques at the given significance level.

(vii) Cost-benefit Analysis: The cost-benefit analysis of used ensemble techniques is performed to assess the cost-effectiveness of SFP models. Wagner initially proposed the concept of cost-benefit analysis in the context of SFP [117]. This analysis estimates the amount of testing efforts and cost that can be saved by using the results of SFP models along with the software testing process in the software development life cycle. The analysis model considers fault removal cost and the fault identification efficiency of different testing phases derived from the case studies of different software organization to estimate the fault

removal cost of specific fault prediction model. Kumar et al. [118] explored the use of cost-benefit analysis in SFP. We have used that presented model in our work for cost effectiveness analysis of built SFP models. Certain assumptions have been made in designing the cost-benefit model, as specified below:

(a) Each of the testing phase such as unit testing, integration testing, system testing has different fault removal cost.
(b) None of the software testing phase is able to detect 100% of software faults.
(c) Unit testing of all software modules is not practically feasible.

Equation (6) shows the estimated fault removal cost (Ecost) that can occur when results of fault prediction are used along with the software testing process. Equation (7) shows the minimum fault removal cost (Tcost) that can occur without the use of fault prediction results in the software testing process. Equation 8 shows the normalized fault removal cost and its interpretation.

$$\begin{aligned} Ecost &= C_{ini} + C_u * (FP + TP) \\ &+\delta_i * C_i * (FN + (1-\delta_u) * TP)) \\ &+\delta_s * C_s * (1-\delta_i) * (FN + (1-\delta_u) * TP \\ &+(1-\delta_s) * C_f * ((1-\delta_u) * FN \\ &+(1-\delta_u) * TP)) \end{aligned} \tag{6}$$

$$\begin{aligned} Tcost &= M_p * C_u * (TM) + \delta_i * C_i * (1-\delta_u) * FM \\ &+\delta_s * C_s * (1-\delta_i) * (1-\delta_u) * FM \\ &+(1-\delta_s) * C_f * ((1-\delta_i) * (1-\delta_u) * FM \end{aligned} \tag{7}$$

$$Ncost = \frac{Ecost}{Tcost} \begin{cases} < 1 & \text{Fault prediction is useful} \\ => 1 & \text{Unit testing is useful} \end{cases} \tag{8}$$

The meanings of used notations are same as described in one of the study by [118].

Where,

"Ecost: Estimated fault removal cost of the software with the use of software fault prediction results
Tcost: Total fault removal cost of the software without the use of software fault prediction results
Ncost: Normalized fault removal cost of the software when software fault prediction is used

C_{ini}: Initial setup cost for using software fault-prediction model (C_{ini} = 0)
C_u: Normalized fault removal cost in unit testing
C_s: Normalized fault removal cost in system testing
C_f: Normalized fault removal cost in field testing
C_i: Normalized fault removal cost in integration testing
M_p: Percentage of modules unit tested
FP: Number of false positives
FN: Number of false negatives
TP: Number of true positives
TM: Total modules
FM: Total number of faulty modules
δ_u: Fault identification efficiency of unit testing
δ_s: Fault identification efficiency of system testing
δ_i: Fault identification efficiency of integration testing"

The fault identification efficiency of different testing phases is defined as staff hour per fault and is borrowed from the study performed by Jones [119]. We have considered median of the fault identification efficiency values maintained by Jones in our study. The used values are, δ_u = 0.25, δ_s= 0.5, and δ_i=0.45. The normalized fault removal cost is defined as staff hour per fault and is borrowed from the Wagner's work [117]. Again, we have considered median of these values. The used values are, C_f =27, C_s=6.2, C_u=2.5, and C_i=4.55. M_p shows the fraction of modules unit tested. Its value is taken from the study performed by [120] and is M_p=0.5. A detailed description of used cost-benefit analysis model is given in [118].

7 Results and analysis

This section reports the results of used ensemble techniques for various performance measures. Further, an analysis of results is performed to draw observations about the ensemble techniques' performance. The experimental procedure discussed in Section 6 has been used to build and evaluate prediction models. Later, this section discusses the results of the used statistical tests.

7.1 Results for precision, recall, AUC, specificity, and G-means

Tables 3, 4, 5, 6, 7 show the summarized results of ensemble techniques for various used datasets. Each table depicts the results of one performance measure. The table contains min, max, and means values of each ensemble technique calculated from all datasets. We have reported the summarized results due to the space constraint. Following observations are drawn from tables.

– With respect to the precision measure, Rotation Forest with J48 as base learner achieved highest max value and highest mean value. Whereas, MultiBoostAB with NB as base learner yielded the lowest min value.
– With respect to the recall measure, again Rotation Forest with J48 as base learner achieved highest max value and highest mean value. Whereas, Rotation Forest with NB as base learner yielded the lowest min value.

Table 3 Summarized results of ensemble techniques for the used fault datasets with respect to precision measure

Technique	Base Leaner	Precision		
		Min	Max	Mean±std
Dagging	NB	0.589	0.976	0.806±0.085
	LR	0.619	0.976	0.808±0.085
	J48	0.601	0.976	0.82 ±0.087
Decorate	NB	0.581	0.978	0.798±0.09
	LR	0.609	0.986	0.816±0.082
	J48	0.676	0.99	0.826±0.073
Grading	NB	0.629	0.987	0.807±0.083
	LR	0.651	0.987	0.816±0.079
	J48	0.665	0.988	0.822±0.076
MultiBoostAB	NB	0.583	0.986	0.80±0.089
	LR	0.622	0.987	0.812±0.082
	J48	0.673	0.993	0.823±0.077
RealAdaBoost	NB	0.535	0.983	0.791±0.102
	LR	0.623	0.988	0.813±0.083
	J48	0.674	0.993	0.819±0.073
Rotation Forest	NB	0.595	0.983	0.797±0.092
	LR	0.622	0.987	0.816±0.081
	J48	0.677	0.995	0.829±0.074
Ensemble Selection		0.594	0.983	0.804±0.086

Table 4 Summarized results of ensemble techniques for the used fault datasets with respect to recall measure

Technique	Base Leaner	Recall		
		Min	Max	Mean±std
Dagging	NB	0.575	0.988	0.811±0.093
	LR	0.619	0.982	0.831±0.09
	J48	0.639	0.988	0.845±0.087
Decorate	NB	0.541	0.976	0.783±0.111
	LR	0.608	0.981	0.838±0.084
	J48	0.676	0.99	0.846±0.07
Grading	NB	0.629	0.983	0.834±0.085
	LR	0.646	0.985	0.838±0.082
	J48	0.663	0.988	0.841±0.08
MultiBoostAB	NB	0.567	0.943	0.784±0.099
	LR	0.623	0.982	0.835±0.084
	J48	0.679	0.993	0.839±0.079
RealAdaBoost	NB	0.563	0.977	0.785±0.105
	LR	0.623	0.986	0.835±0.084
	J48	0.675	0.993	0.832±0.074
Rotation Forest	NB	0.507	0.943	0.776±0.115
	LR	0.621	0.983	0.838±0.085
	J48	0.677	0.994	0.851±0.078
Ensemble Selection		0.575	0.951	0.797±0.095

Table 5 Summarized results of ensemble techniques for the used fault datasets with respect to AUC measure

Technique	Base Leaner	AUC		
		Min	Max	Mean±std
Dagging	NB	0.491	0.862	0.781±0.084
	LR	0.617	0.912	0.748±0.062
	J48	0.458	0.934	0.719±0.116
Decorate	NB	0.568	0.986	0.708±0.086
	LR	0.558	0.918	0.721±0.091
	J48	0.641	0.965	0.769±0.081
Grading	NB	0.496	0.851	0.608±0.096
	LR	0.492	0.874	0.619±0.097
	J48	0.494	0.885	0.629±0.099
MultiBoostAB	NB	0.523	0.843	0.685±0.07
	LR	0.586	0.921	0.701±0.072
	J48	0.632	0.956	0.771±0.072
RealAdaBoost	NB	0.572	0.87	0.719±0.084
	LR	0.602	0.904	0.737±0.073
	J48	0.532	0.947	0.752±0.078
Rotation Forest	NB	0.561	0.847	0.717±0.077
	LR	0.566	0.925	0.742±0.084
	J48	0.417	0.953	0.752±0.117
Ensemble Selection		0.637	0.912	0.747±0.062

Table 6 Summarized results of ensemble techniques for the used fault datasets with respect to specificity measure

Technique	Base Leaner	Specificity		
		Min	Max	Mean±std
Dagging	NB	0.523	0.998	0.821±0.137
	LR	0.578	0.977	0.831±0.101
	J48	0.641	1	0.846±0.093
Decorate	NB	0.076	0.984	0.772±0.218
	LR	0.33	0.977	0.816±0.136
	J48	0.6	0.977	0.84±0.093
Grading	NB	0.375	0.977	0.815±0.129
	LR	0.4	0.977	0.817 ±0.129
	J48	0.5	0.977	0.827±0.116
MultiBoostAB	NB	0.065	0.985	0.776±0.214
	LR	0.352	0.977	0.815±0.133
	J48	0.609	0.977	0.852±0.084
RealAdaBoost	NB	0.222	0.984	0.773±0.208
	LR	0.428	0.977	0.815±0.128
	J48	0.669	0.977	0.850±0.083
Rotation Forest	NB	0.048	0.982	0.769±0.225
	LR	0.375	0.977	0.815±0.131
	J48	0.666	1	0.851±0.089
Ensemble Selection		0.068	0.985	0.78±0.208

Table 7 Summarized results of ensemble techniques for the used fault datasets with respect to G-mean 1 and G-mean 2 measures

Technique	Base Leaner	G-mean 1			G-mean 2		
		Min	Max	Mean±std	Min	Max	Mean±std
Dagging	NB	0.581	0.981	0.808±0.089	0.56	0.99	0.815±0.111
	LR	0.619	0.978	0.819±0.087	0.00	0.975	0.801±0.18
	J48	0.62	0.981	0.832±0.086	0.64	0.99	0.846±0.090
Decorate	NB	0.572	0.976	0.790±0.21	0.572	0.976	0.790±0.098
	LR	0.608	0.983	0.827±0.083	0.571	0.972	0.824±0.100
	J48	0.676	0.99	0.836±0.07	0.680	0.977	0.842±0.075
Grading	NB	0.629	0.984	0.820±0.083	0.607	0.976	0.821±0.096
	LR	0.648	0.985	0.827±0.080	0.627	0.972	0.825±0.095
	J48	0.663	0.988	0.831±0.078	0.634	0.973	0.831±0.090
MultiBoostAB	NB	0.574	0.954	0.791±0.091	0.237	0.963	0.771±0.167
	LR	0.625	0.984	0.823±0.082	0.588	0.969	0.822±0.098
	J48	0.677	0.993	0.831±0.070	0.643	0.976	0.845±0.078
RealAdaBoost	NB	0.568	0.977	0.788±0.102	0.453	0.960	0.773±0.152
	LR	0.623	0.986	0.824±0.083	0.621	0.966	0.822±0.096
	J48	0.675	0.993	0.825±0.073	0.679	0.972	0.840±0.074
Rotation Forest	NB	0.583	0.953	0.786±0.100	0.202	0.962	0.763±0.182
	LR	0.621	0.984	0.827± 0.082	0.607	0.971	0.824±0.098
	J48	0.677	0.994	0.840±0.075	0.682	0.966	0.851±0.083
Ensemble Selection		0.584	0.958	0.801±0.088	0.239	0.967	0.779±0.161

- With respect to the AUC measure, Decorate with NB as base learner produced highest max value and Dagging with NB as base learner produced highest mean value. RealAdaBoost with J48 as base learner produced the lowest min value.
- With respect to the specificity measure, MultiBoostAB with J48 as base learner produced highest mean value and Rotation Forest with J48 and Dagging with J48 as base learner produced highest max value. Rotation Forest with NB produced the lowest min value.
- With respect to the G-mean measures, Rotation Forest with J48 as base learner produced highest max value and highest mean value for G-mean 1, and Dagging with NB as base learner produced highest max value and Dagging with J48 as base learner produced highest mean value for G-mean 2. RealAdaBoost with NB as base learner produced the lowest min value for G-mean 1 and Dagging with LR produced lowest min value for G-mean 2.
- Overall, it is found that Rotation Forest outperformed other used ensemble techniques and yielded better performance. In case of base learners, J48 achieved better performance among the used base learners.
- From tables, it can be observed that for all the considered performance measures, used ensemble techniques produced mean values greater than 0.7, except for the grading ensemble technique in terms of AUC measure. The standard deviation values (std) of all ensemble techniques are below 0.10 for most of the cases for all performance measures, except for the specificity measure. For specificity measure, all ensemble techniques produced std values above 0.10 with the highest value of 0.223. This high variation in the model's performance signifies a low true negative rate and thus it shows that prediction models missed some true negative cases and classified them as false positives. This will increase the consumption of software testing to test false positive cases. However, in their work, Bohem et al. [121] argued that the verification/testing efforts saved by a fault prediction model of correct identification of one fault are higher than the cost of misclassification of a hundred fault-free modules as fault-prone. Therefore, the high std values of specificity measure would result in the marginal increase in the testing cost but overall software testing cost would be saved.

Figure 4 shows box-plots for comparing the degree of dispersion, inter-quartile range, outliers and skewness in term of precision, recall, AUC, specificity, and G-means values for all ensemble techniques across all fault datasets. Each box-plot is corresponding to one ensemble technique and for one performance measure. The middle line in box-plots shows midpoints of the data (median values). Following observations have been obtained from the figure.

- It is depicted in the figure that for the AUC measure, all ensemble techniques performed relatively poor as compared to other used performance measures.
- Additionally, it is observed that the inter-quartile range (the difference between the first quartile and third quartile) for AUC measure is more than other used performance measures.
- The box-plots of specificity measure are relatively wider than other box-plots and hence it shows the variation in the specificity values across dataset. The upper and lower whiskers of box-plots corresponding to specificity measure in the figure show that many values are deviated largely from the median value.
- For other performance measures such as precision, recall, and G-means, it is observed that there are

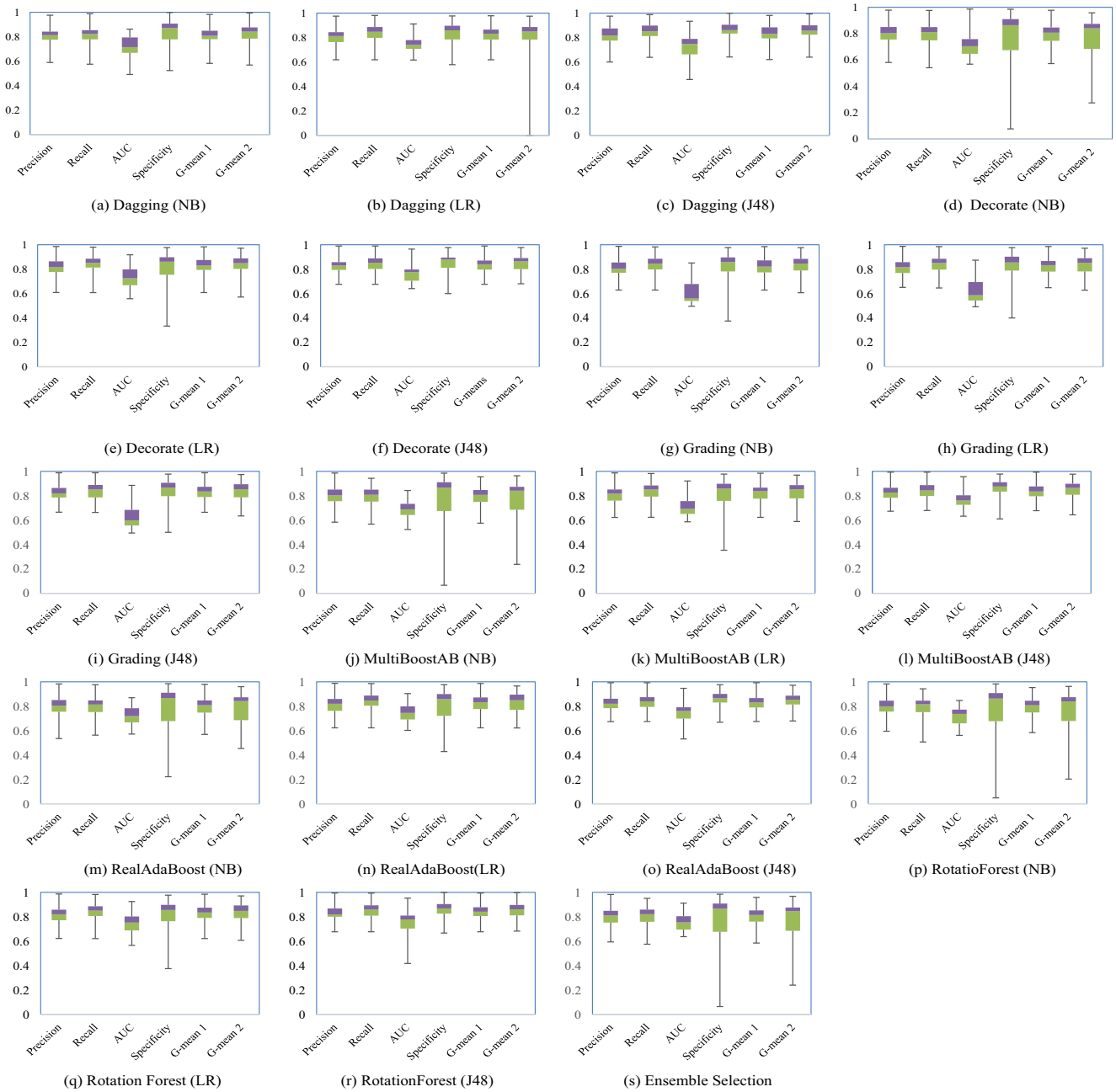

Fig. 4 Boxplot diagrams showing the degree of dispersion, interquartile range, outliers and skewness for all the used performance measures

Table 8 Results of statistical comparisons of Friedman's tests among the used ensemble techniques for all five performance measures

Friedman's Test for Precision		Friedman's Test for Recall	
H-stat	78.62	H-stat	286.7949
DF	18	DF	18
P-value	1.49E-09	P-value	2.48E-50
Alpha	0.05	Alpha	0.05
Significant	Yes	Significant	Yes
Friedman's Test for AUC		Friedman's Test for Specificity	
H-stat	242.15	H-stat	56.19
DF	18	DF	18
P-value	3.2E-41	P-value	8.29E-06
Alpha	0.05	Alpha	0.05
Significant	Yes	Significant	Yes
Friedman's Test for G-mean 1		Friedman's Test for G-mean 2	
H-stat	240.64	H-stat	206.5365
DF	18	DF	18
P-value	6.49E-41	P-value	4.92E-34
Alpha	0.05	Alpha	0.05
Significant	Yes	Significant	Yes

not many variations in the values. For these three performance measures, all the ensemble techniques achieved relatively better performance.

7.2 Results of statistical tests

Table 8 shows results of the Friedman's tests of all used ensemble techniques for all five performance measures. It is observed from the table that a statistically significant difference in the performance of at least one pair of ensemble techniques has been found for all used performance evaluation measures. P-values are lower than the considered significant values (α=0.05) for all cases. These results showed that different ensemble techniques performed differently for at least one pair of techniques for the given software fault datasets. Further, Wilcoxon signed rank sum test is performed to calculate the within pair difference among the used ensemble techniques.

Table 9 shows results of the Wilcoxon signed rank test for all five performance measures of all used ensemble techniques. Each sub-table is for one performance measure. Due to the space constraint, we used the abbreviated ID's for technique names. The full name of each ID is provided in the table caption. A black filled circle shows the significant performance difference in the pair of ensemble techniques at $\alpha = 0.05$ and thus rejecting the null hypothesis. A hollow circle shows no significant performance difference at $\alpha = 0.05$ and thus accepting the null hypothesis. There are a total of 171 pair-wise comparisons of seven ensemble techniques is reported in Table 9 for each performance measure. The summarized results of Wilcoxon signed rank sum test are given below.

- For the precision measure, a total of 106 pairs have shown a statistical significant difference in performance and other 65 pairs have not shown any statistical significant difference in performance.
- In case of recall performance measure, a total of 138 pairs have shown a statistical significant difference in performance and other 33 pairs have not shown any statistical significant difference in performance.
- For the AUC measure, a total of 133 pairs have shown a statistical significant difference in performance and other 38 pairs have not shown any statistical significant difference in performance.
- In case of specificity measure, a total of 128 pairs have shown a statistical significant difference in performance and other 43 pairs have not shown any statistical significant difference in performance.
- For the G-mean 1 measure, a total of 54 pairs have shown a statistical significant difference in performance and other 117 pairs have not shown any statistical significant difference in performance.
- For the G-mean 2 measure, a total of 128 pairs have shown a statistical significant difference in performance and other 43 pairs have not shown any statistical significant difference in performance.

These results showed that performance of ensemble techniques differs statistically significantly from one to other. Except the G-mean 1 performance measure, for all

other used performance measures, cases where statistically significant performance difference have been found are more than the cases where no statistically significant performance different have been found.

7.3 Results of cost-benefit analysis

Table 10 shows normalized cost values (Ncost) of each ensemble techniques for all the used software fault datasets. For each dataset, Ncost value is reported in the table and values less than 1.0 show the cost-effectiveness of the ensemble techniques. It implies that if the results of SFP are used with software testing than overall testing cost and effort can be saved. On the other hand, values higher than 1.0 show that SFP is not helpful in saving testing cost and effort and it is suggested not to use SFP models in those cases. From the table, it can be seen that for datasets such as Lucene-2.4, Poi-2.5, Poi-3.0, Xalan-2.5, Xalan-2.6, Xalan-2.7, Xerces-1.3, and Xerces-1.4, Ncost values are higher than the threshold value (1.0) for all the used ensemble techniques. Therefore, as estimated from this study, it may not be beneficial to use software fault prediction based on used ensemble techniques along with the software testing for these fault datasets. For all other 20 datasets, Ncost values are lower than the threshold value and thus it is beneficial to use software fault prediction based on used ensemble techniques.

7.4 Answer to the research questions

Based on the results reported in Tables 3–9, the answers of research questions are discussed as follow:

RQ1: *Which ensemble technique shows overall best performance for software fault prediction?*

Results reported in Table 3–7 showed that for most of the cases Rotation Forest yielded better performance compared to other used ensemble techniques. MultiBoostAB, Decorate, and Dagging produced better performance in some cases. Other ensemble techniques performed relatively poor.

RQ2: *Is there any statistically significant performance difference between the chosen ensemble techniques?*

Results of Friedman's tests and Wilcoxon signed rank tests reported in Tables 8 and 9 showed that for majority of the cases pairs of ensemble techniques showed statistically significant performance difference. This pattern has been found for all the used performance measures except G-means measure.

RQ3: *How do base learners affect the performance of ensemble techniques?*

The evidence obtained from the experimental results discussed in Section 7 showed that the performance of ensemble techniques varies with the use of the base learner. Overall, J48 as a base learner helped in achieving improved prediction performance. NB as a base learner generally resulted in the inferior performance of the ensemble techniques.

RQ4: *For a given software system, how economically effective ensemble techniques are for software fault prediction?*

The evidence obtained from Table 10 shows that for twenty out of twenty-eight fault datasets, SFP models based on the used ensemble techniques helped in saving software testing cost and effort. For only eight fault datasets, used ensemble techniques have not been helped in saving software testing cost and effort. From the results, it can be recommended to use SFP models based on the used ensemble techniques to reduce the software testing cost.

In this paper, we have explored the use of seven different ensemble techniques for the software fault prediction. Three different classification algorithms have bene used as base learners in the used ensemble techniques. The observations drawn from the experimental results and main advantages of the presented work are summarized as follow.

- The analysis of used ensemble techniques showed that no single ensemble technique always provides the best performance across all the fault datasets, and the use of a particular ensemble technique for SFP depends on the properties of the fault dataset in-hands.
- However, among the used ensemble techniques, Rotation Forest yielded better prediction performance than others. J48 as a base learner outperformed other used base learners. Thus, from this study, it may be recommended to use Rotation Forest and J48 to build the SFP models for better prediction performance.
- The cost-benefit analysis showed that the SFP models based on the ensemble techniques under consideration can help in reducing the software testing cost and can help in optimizing the testing resources.

8 Comparison analysis

There few efforts have been reported earlier regarding the evaluation of ensemble techniques based on fault prediction models. A comparison of reported study with these works on various attributes has tabulated in Table 11. A majority of previous works listed in Table 11 included the contextual information of the fault prediction model, model building information, used software fault datasets, and prediction modeling techniques with the experimental findings in

Table 9 Results of the statistical comparison of Wilcoxon signed rank test among the used ensemble techniques for all five performance measures. A filled circle shows the significance difference and a hollow circle shows no significance difference. (ID1: Dagging(NB), ID2: Dagging(LR), ID3: Dagging(J48), ID4: Decorate(NB), ID5: Decorate(LR), ID6: Decorate(J48), ID7: Grading(NB), ID8: Grading(LR), ID9: Grading(J48), ID10: MultiBoostAB(NB), ID11: MultiBoostAB(LR), ID12: Multi-BoostAB(J48), ID13: RealAdaBoost(NB), ID14: RealAdaBoost(LR), ID15: RealAdaBoost(J48), ID16: RotationForest(NB), ID17: RotationForest(LR), ID18: RotationForest(J48), ID19: Ensemble Selection)

(a) For Precision measure

	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	ID9	ID10	ID11	ID12	ID13	ID14	ID15	ID16	ID17	ID18	ID19
ID1	-	○	●	○	●	●	○	●	●	○	○	●	○	●	●	○	●	●	○
ID2		-	●	○	●	●	○	●	●	○	●	●	○	○	●	○	●	●	○
ID3			-	●	○	○	●	○	○	●	○	○	●	○	○	●	○	●	●
ID4				-	●	●	○	●	●	○	●	●	○	●	●	○	●	●	●
ID5					-	○	●	○	○	●	○	○	●	○	○	●	○	●	●
ID6						-	●	●	●	●	●	○	●	●	●	●	○	○	●
ID7							-	●	●	○	○	●	○	○	●	○	●	●	○
ID8								-	●	●	●	○	●	○	○	●	○	●	●
ID9									-	●	●	○	●	○	○	●	○	●	●
ID10										-	●	●	○	●	●	○	●	●	●
ID11											-	●	●	○	○	●	●	●	○
ID12												-	●	○	○	●	○	●	●
ID13													-	●	●	○	●	●	●
ID14														-	○	●	○	●	●
ID15															-	●	○	●	●
ID16																-	●	●	●
ID17																	-	●	●
ID18																		-	●
ID19																			-

(b) For Recall measure

	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	ID9	ID10	ID11	ID12	ID13	ID14	ID15	ID16	ID17	ID18	ID19
ID1	-	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
ID2		-	●	●	●	●	○	●	●	●	●	○	●	○	○	●	●	●	●
ID3			-	●	●	○	●	●	○	●	●	●	●	●	●	●	●	●	●
ID4				-	●	●	●	●	○	●	●	○	●	○	●	○	●	●	●
ID5					-	○	●	●	○	●	●	●	●	●	●	●	○	●	●
ID6						-	●	●	●	●	●	●	●	●	●	●	○	○	●
ID7							-	●	●	●	○	○	●	○	○	●	●	●	●
ID8								-	●	●	●	○	●	●	○	●	○	●	●
ID9									-	●	●	○	●	○	●	●	○	●	●
ID10										-	●	●	○	●	●	○	●	●	●
ID11											-	○	●	○	○	●	●	●	●
ID12												-	●	○	●	●	○	●	●
ID13													-	●	●	○	●	●	●
ID14														-	○	●	●	●	●
ID15															-	●	●	●	●
ID16																-	●	●	●
ID17																	-	●	●
ID18																		-	●
ID19																			-

(c) For AUC measure

	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	ID9	ID10	ID11	ID12	ID13	ID14	ID15	ID16	ID17	ID18	ID19
ID1	-	●	○	●	○	●	●	●	●	●	●	●	○	●	●	●	●	●	●
ID2		-	○	●	●	●	●	●	●	●	●	●	●	○	○	●	○	●	○
ID3			-	●	○	●	●	●	●	●	●	●	○	○	●	●	○	●	○
ID4				-	●	●	●	●	●	●	○	●	○	●	●	○	●	●	●
ID5					-	●	●	●	●	●	●	●	○	●	●	○	●	●	●
ID6						-	●	●	●	●	●	○	●	●	○	●	○	○	○
ID7							-	●	●	●	●	●	○	●	●	○	●	●	●
ID8								-	●	●	●	●	●	●	●	●	●	●	●
ID9									-	●	●	●	●	●	●	●	●	●	●
ID10										-	○	●	●	●	●	●	●	●	●
ID11											-	●	○	●	●	○	●	●	●
ID12												-	●	●	●	●	●	○	●
ID13													-	●	●	○	●	●	●
ID14														-	○	●	○	○	○
ID15															-	●	○	○	○
ID16																-	●	●	●
ID17																	-	○	○
ID18																		-	○
ID19																			-

(d) For Specificity measure

	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	ID9	ID10	ID11	ID12	ID13	ID14	ID15	ID16	ID17	ID18	ID19
ID1	-	○	○	●	○	○	○	○	○	●	○	○	●	○	○	●	○	○	●
ID2		-	○	●	○	●	○	○	○	○	○	●	○	○	●	○	●	○	○
ID3			-	○	○	○	○	○	○	○	○	●	○	○	●	○	○	○	○
ID4				-	○	○	○	○	○	○	○	●	○	○	●	○	○	○	●
ID5					-	●	○	○	○	○	○	●	○	○	●	○	○	○	○
ID6						-	●	●	●	○	●	●	○	●	●	○	●	○	○
ID7							-	○	●	○	○	●	○	○	●	○	●	○	●
ID8								-	●	○	○	●	○	○	●	○	○	○	○
ID9									-	○	●	○	●	○	○	●	○	○	○
ID10										-	○	●	○	○	●	○	○	○	○
ID11											-	●	○	○	●	○	○	○	○
ID12												-	●	●	○	●	●	●	●
ID13													-	○	●	○	○	○	○
ID14														-	●	○	○	○	○
ID15															-	●	●	●	●
ID16																-	○	○	○
ID17																	-	○	○
ID18																		-	○
ID19																			-

(e) For G-mean 1 measure

	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	ID9	ID10	ID11	ID12	ID13	ID14	ID15	ID16	ID17	ID18	ID19
ID1	-	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	○
ID2		-	●	●	●	●	○	●	●	●	●	●	●	○	○	●	●	●	●
ID3			-	●	○	○	●	○	○	●	●	○	●	●	○	●	○	●	●
ID4				-	●	●	●	●	●	○	●	●	○	●	●	○	●	●	●
ID5					-	○	●	○	○	●	○	○	●	○	○	●	○	●	●
ID6						-	●	●	●	●	●	○	●	●	●	●	●	○	●
ID7							-	●	●	○	○	●	○	○	●	○	●	●	●
ID8								-	●	●	●	○	●	○	○	●	○	●	●
ID9									-	●	○	●	○	●	●	○	●	●	●
ID10										-	●	○	●	○	●	○	●	●	●
ID11											-	○	○	●	○	●	○	●	●
ID12												-	●	○	●	○	●	●	●
ID13													-	●	●	○	●	●	●
ID14														-	○	●	○	●	●
ID15															-	●	○	●	●
ID16																-	●	●	●
ID17																	-	●	●
ID18																		-	●
ID19																			-

(f) For G-mean 2 measure

	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	ID9	ID10	ID11	ID12	ID13	ID14	ID15	ID16	ID17	ID18	ID19
ID1	-	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
ID2		-	●	●	○	●	○	●	●	●	○	●	○	○	○	●	○	●	●
ID3			-	●	○	○	●	●	○	●	●	○	●	●	○	●	●	○	●
ID4				-	●	●	●	●	●	○	●	●	○	●	●	●	●	●	●
ID5					-	●	○	○	○	●	○	○	●	○	○	●	○	●	●
ID6						-	●	●	●	●	●	○	●	●	●	●	●	○	●
ID7							-	●	●	○	●	●	○	●	●	○	●	●	●
ID8								-	●	●	●	●	○	○	●	○	●	●	●
ID9									-	●	○	●	○	○	●	○	●	●	●
ID10										-	●	○	●	●	○	●	○	●	●
ID11											-	●	●	○	○	●	○	●	●
ID12												-	●	●	●	○	●	●	●
ID13													-	●	●	○	●	●	●
ID14														-	○	●	○	●	●
ID15															-	●	○	●	●
ID16																-	●	●	●
ID17																	-	●	●
ID18																		-	●
ID19																			-

their studies. It can also be observed from the table that generally available works focused on the use of a limited set of prediction modeling techniques in their studies. In comparison to this, the reported work examined seven ensemble techniques, which have not been explored earlier. Moreover, most of the earlier works used only a few datasets to perform the experiments. In the reported study, a total of 28 different fault datasets have been considered to generalize the findings of the work. Further, we have performed a cost-benefit analysis to assess the economic viability of the used ensemble techniques for the SFP, which has not been done in previous studies. When comparing the results of the presented empirical study with the results reported by [31] for SFP, it has been found that ensemble techniques used in this study performed better. The highest AUC value achieved is 0.986 by Decorate with NB as a base learner in comparison to 0.96 value achieved by stacking in the Wang et al.'s study. In the presented study, the mean

Table 10 Results of the cost-benefit analysis (Ncost) of ensemble techniques for all used software fault datasets [ID1: Dagging(NB), ID2: Dagging(LR), ID3: Dagging(J48), ID4: Decorate(NB), ID5: Decorate(LR), ID6: Decorate(J48), ID7: Grading(NB), ID8: Grading(LR), ID9: Grading(J48), ID10: MultiBoostAB(NB), ID11: MultiBoostAB(LR), ID12: Multi-BoostAB(J48), ID13: RealAdaBoost(NB), ID14: RealAdaBoost(LR), ID15: RealAdaBoost(J48), ID16: RotationForest(NB), ID17: RotationForest(LR), ID18: RotationForest(J48), ID19: Ensemble Selection]

Techniques	Ant	Camel				Ivy	Jedit			Lucene	Poi		Prop						Tomcat	Xalan				Xerces		
	Ant-1.7	Camel-1.0	Camel-1.2	Camel-1.4	Camel-1.6	Ivy-2.0	Jedit-4.0	Jedit-4.1	Jedit-4.2	Jedit-4.3	Lucene-2.4	Poi-2.0	Poi-2.5	Poi-3.0	Prop-1	Prop-2	Prop-3	Prop-4	Prop-5	Prop-6	Tomcat	Xalan-2.4	Xalan-2.5	Xalan-2.6	Xalan-2.7	Xerces-1.2	Xerces-1.3	Xerces-1.4
ID1	0.8621	0.0766	0.9768	0.7549	0.7552	0.5582	0.8938	0.8993	0.7007	0.0447	1.1013	0.4156	1.1022	1.0942	0.6973	0.6138	0.6082	0.5344	0.6039	0.6452	0.9891	0.7375	1.0654	1.0200	1.1301	0.6130	0.735	1.1177
ID2	0.797	0.1010	0.9890	0.6397	0.6819	0.4766	0.8756	0.8488	0.5626	0.0789	1.1158	0.5221	1.0990	1.1048	0.4268	0.2645	0.2808	0.3078	0.3852	0.4180	0.4120	0.6451	1.0912	1.0433	1.1307	0.6481	0.6796	1.1023
ID3	0.7751	0.0766	0.9234	0.4504	0.5280	0.4150	0.8125	0.8377	0.5144	0.0447	1.1078	0.3474	1.1058	1.0986	0.4227	0.2547	0.2352	0.2703	0.3439	0.2	0.2839	0.4703	1.0821	1.0391	1.1301	0.3620	0.5424	1.0959
ID4	0.8439	0.4268	0.9439	0.7457	0.7171	0.6992	0.8404	0.8328	0.6750	0.4147	1.1025	0.6152	1.1214	1.1320	0.6941	0.6092	0.5633	0.5225	0.6001	0.7864	0.6552	0.7426	1.0597	1.0054	1.1314	0.7270	0.7425	1.1346
ID5	0.7972	0.1838	0.9112	0.5446	0.5925	0.4979	0.8197	0.8495	0.5518	0.1269	1.1036	0.4834	1.1075	1.0952	0.4254	0.2591	0.2829	0.2936	0.3782	0.3045	0.4151	0.5430	1.0834	1.0374	1.1304	0.4357	0.6539	1.1037
ID6	0.8004	0.1010	0.9830	0.5987	0.7306	0.4516	0.8618	0.8173	0.6063	0.0447	1.1014	0.4750	1.0875	1.0938	0.5517	0.3822	0.2994	0.3068	0.3825	0.4396	0.4080	0.5663	1.0759	1.0315	1.1294	0.6734	0.5882	1.0964
ID7	0.8201	0.1408	0.9464	0.5439	0.6370	0.4979	0.7853	0.8099	0.5679	0.0620	1.1059	0.4773	1.1035	1.0994	0.5030	0.4302	0.2773	0.2970	0.4768	0.3717	0.3197	0.5157	1.0900	1.0405	1.1302	0.5025	0.6538	1.1023
ID8	0.7946	0.1898	0.9211	0.5420	0.6252	0.5469	0.8281	0.8246	0.5186	0.1269	1.0998	0.4554	1.1044	1.0950	0.4771	0.4229	0.2924	0.3153	0.3907	0.3904	0.3946	0.5693	1.0667	1.0255	1.1300	0.5232	0.6290	1.0997
ID9	0.7995	0.1688	0.9353	0.5841	0.6893	0.5368	0.8172	0.8345	0.5830	0.1113	1.0998	0.4609	1.0984	1.0913	0.5258	0.3638	0.2934	0.3123	0.3926	0.3904	0.3751	0.5843	1.0683	1.0235	1.1297	0.6208	0.6786	1.0974
ID10	0.8553	0.3920	0.9326	0.7095	0.7259	0.7050	0.8486	0.8532	0.6912	0.3586	1.1063	0.6247	1.1060	1.1245	0.6922	0.6092	0.5999	0.5225	0.6028	0.7855	0.6537	0.7574	1.0607	1.0054	1.1464	0.7060	0.7262	1.1339
ID11	0.7560	0.2610	0.9547	0.5369	0.6145	0.4886	0.8725	0.8478	0.5639	0.1714	1.1074	0.5170	1.1052	1.1022	0.4305	0.2645	0.2781	0.3077	0.3879	0.3322	0.4023	0.5736	1.0855	1.0387	1.1303	0.5145	0.6509	1.1019
ID12	0.8213	0.1688	0.9872	0.6599	0.7613	0.5664	0.8668	0.8586	0.6084	0.0620	1.1123	0.5214	1.920	1.0957	0.6329	0.4853	0.4240	0.4101	0.5873	0.5273	0.4080	0.6373	1.0738	1.0462	1.1290	0.6293	0.7058	1.0979
ID13	0.8535	0.4315	0.9407	0.7282	0.7234	0.6992	0.8352	0.8294	0.6683	0.3875	1.1025	0.6247	1.1438	1.1256	0.6922	0.6092	0.5999	0.5221	0.6028	0.7829	0.6426	0.7340	1.0628	1.0063	1.1311	0.7060	0.7165	1.1339
ID14	0.7603	0.3389	0.9344	0.5441	0.5726	0.4979	0.8317	0.8495	0.5713	0.2127	1.1028	0.4642	1.1081	1.1052	0.4524	0.2862	0.2898	0.3183	0.3923	0.3352	0.4115	0.5671	1.0826	1.0329	1.1299	0.5243	0.6597	1.1072
ID15	0.8228	0.2484	1.0039	0.6958	0.7989	0.5525	0.8509	0.8868	0.5899	0.1269	1.0958	0.5919	1.0983	1.0966	0.7082	0.5954	0.4457	0.4984	0.6402	0.6519	0.4318	0.6590	1.0764	1.0512	1.1290	0.7128	0.6821	1.1009
ID16	0.8495	0.3960	0.9473	0.7314	0.7365	0.7176	0.8369	0.8378	0.6784	0.3419	1.1014	0.6432	1.1463	1.1448	0.6851	0.5996	0.5503	0.5236	0.5579	0.7382	0.6463	0.7509	1.0558	1.0126	1.1483	0.6886	0.7412	1.1352
ID17	0.7521	0.1838	0.9300	0.5125	0.5621	0.4594	0.8211	0.8329	0.5712	0.1421	1.1041	0.4554	1.1056	1.1006	0.4268	0.2568	0.2750	0.2941	0.3787	0.3108	0.3650	0.5267	1.0835	1.0328	1.1302	0.4795	0.6351	1.1024
ID18	0.7569	0.1010	0.8656	0.4847	0.5432	0.3926	0.7920	0.8442	0.5349	0.0447	1.1039	0.3155	1.0961	1.0923	0.4461	0.3080	0.2470	0.2562	0.3468	0.2651	0.2749	0.4150	1.0764	1.0307	1.1289	0.5959	0.5818	1.0958
ID19	0.8426	0.3960	0.9445	0.7005	0.7191	0.6441	0.8316	0.8244	0.6508	0.3173	1.0937	0.5852	1.1032	1.1163	0.6562	0.5528	0.5529	0.4908	0.5628	0.7250	0.6311	0.7194	1.0586	1.0089	1.1408	0.6919	0.7206	1.1251

AUC value is 0.781, and minimum AUC value is 0.532, which is comparable with the values reported by Wang et al.'s work.

9 Threats to the validity

The empirical analysis reported in this paper can be suffered by some threats to the validity, which are discussed as follow.

Construct Validity This validity threat concerns with the accuracy of the used software fault datasets. We gathered and used fault datasets reported in the PROMISE data repository, which is available in the public domain. The fault datasets in this repository are corresponding to various contributors. It is the primary repository used for building and evaluating software fault prediction model. This made us believe that fault datasets used in the study are accurate, consistent, and free from any inconsistency.

Internal Validity This validity threat concerns with the selection of base learners. The presented study included three different classification algorithms as base learners. The rationale behind the selection of these three algorithms is that previous research found that these algorithms performed better in comparison to other ones for software fault prediction. However, the selection of base learners is orthogonal to the intended contribution. We have used faulty or non-faulty information for a given software module as dependent variable due to the nature of the designed experimental study. Other dependent variables such as the quantity of faults in a software module, severity of a fault, etc. could also be used.

External Validity This validity threat is related to the used statistical tests. We have used Friedman's test and Wilcoxon signed rank sum test to evaluate the performance difference of the considered ensemble techniques. These tests are the non-parametric tests, which do not specify any conditions about the drawn population sample of the data. Selection of these tests were made according to the data sample available in hand in the presented study. However, any other statistical test can be used based on the given data. We have used

Table 11 Summary of comparative analysis

Author(s)	Aim of the study	Contextual information reported	Model building information reported	Software fault datasets used	Reported fault prediction techniques/models	Results
Mısırlı et al. [32]	Defect prediction model based on an ensemble of Classifiers for locating software defects	Yes	Yes	CM1, PC1, PC3 and PC4 from NASA data repository and 4 fault datasets from industry	A ensemble classifier (Ens2) that used three classifiers as base learners	For industrial datasets PD= 0.69, PF=0.17, Precision= 0.72, Bal =0.74 For NASA datasets PD=0.84, PF=0.15, Precision=0.63,Bal=0.75 *All results are average values
J. Zheng [33]	Three cost sensitive boosting algorithms based on boost neural networks for software defect prediction	Yes	Yes	KC1, KC2, CM1, and PC1 from NASA data repository	Three cost-sensitive neural network boosting algorithms (CSBNN-WU1 CSBNN-WU2, and CSBNN-TM)	CSBNN-WU2 algorithms was performed the best among the used techniques
Wang et al. [26]	A comparative study of various ensemble methods for software defect prediction	Yes	Yes	14 fault datasets from the NASA data repository	Bagging, AdaBoostM1 Random Forest, Random Tree, Random Subspace, Stacking, Naive Bayes, and Vote	Accuracy: 88.48% (for vote) AUC: 96% (for vote and stacking) *results show the mean value found highest among all used datasets
Siers and Islam [36]	A cost sensitive decision forest and voting for the class imbalance problem	Yes	Yes	MC2, PC1, KC1, PC3, MC1, and PC2 from NASA data repository	A cost-sensitive classification technique CSForest and a cost-sensitive voting technique CSVoting	Precision =0.85(SysFor+voting1) Recall=0.375 (CSForest+CSVoting)
B Twala [34]	Prediction of software faults in large space systems using ensemble techniques	Yes	Yes	CM1, JM1, and PC1 from NASA data repository and 1 dataset from NASA's Jet Propulsion Laboratory	Majority voting based ensemble method	Ensemble of aprior, decision tree and KNN produced best error rate of 19.0% approx.
Our work	Empirical study of ensemble techniques for software fault prediction	Yes	Yes	28 software fault datasets from the PROMISE data repository	Seven different ensemble techniques with three different classifiers as base learners	Precision= 0.995 (Rotation Forest) Recall= 0.994 (Rotation Forest) AUC= 0.986 (Decorate) Specificity= 1 (Dagging and Rotation Forest) G-mean 1= 0.994 (Rotation Forest) G-mean 2= 0.99 (Dagging)) Cost-benefit analysis: for 20 out of 28 used fault datasets, software fault prediction models based ensemble techniques helped in saving the software testing cost and effort.

datasets corresponding to different software projects to generalize the conclusions drawn in the presented study.

10 Conclusions and future work

In this work, an extensive experimental analysis of ensemble techniques for the SFP has been carried out. The study presented an evaluation of seven ensemble techniques by using three different classification algorithms as base learners on twenty-eight software fault datasets. In total 532 prediction models have been built and evaluated for fault prediction in the presented study. Overall, we found that ensemble techniques are useful modeling techniques and thus can be considered to build effective fault prediction models. Out of the used ensemble techniques, rotation forest yielded better prediction performance compared to the other ensemble techniques and J48 as the base learner has worked most effectively. Further, we found that used ensemble techniques have shown a statistically significant performance difference for the used performance measures. The work reported in this paper can be helpful to the research community in the modeling of accurate fault prediction models by selecting appropriate ensemble technique.

In the future, we aim to develop a hybrid ensemble technique based fault prediction model based on the findings of the reported study. Additionally, future work includes the assessment of ensemble techniques for the fault datasets drawn from other software systems and having different software metrics to generalize the findings of the work.

Acknowledgments We are thankful to the editor and the anonymous reviewers for their valuable comments that helped in improvement of the paper.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

Informed Consent This article does not contain any studies with human participants.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Appendix

In this study, we have used Weka implementation of used ensemble techniques and base learners. Following parameter values have been set for these three base learning algorithms amd seven ensemble techniques.

Techniques	Parameter values
Dagging	batchSize= 100, debug= False, doNotCheckCapabilities= False,verbose= False, numFolds= 10, numDecimalPlaces= 2, seed= 1
Decorate	seed = 1, desiredSize= 15, numDecimalPlaces= 2, batchSize= 100, numIterations= 50, artificialSize= 1.0, debug= False, doNotCheckCapabilities= Fasle
Grading	Seed= 1, numFolds= 10, numExecutionSlots= 1, numDecimalPlaces= 2, batchSize= 100
MultiBoostAB	Seed= 1, weightThreshold= 100, numSubCmtys= 3, numDecimalPlaces= 2, batchSize= 100, numIterations= 10, debug= False, doNotCheckCapabilities= False
RealAdaBoost	Seed= 1, weightThreshold= 100, numDecimalPlaces= 2, batchSize= 100, numIterations=10, debug= False, doNotCheckCapabilities= False, useResampling= False, shrinkage= 1.0
RotationForest	Seed= 1, numExecutionSlots= 1, numDecimalPlaces= 2, batchSize= 100, minGroup= 3, numberOfGroups= False, numIterations= 10, debug= False, removedPercentage= 50, maxGroup= 3, minGroup= 3, doNotCheckCapabilities= False, projectionFilter= PCA
Ensemble Selection	Seed= 1, sortInitializationRatio= 1.0, modelRatio= 0.5, replacement= True, numFolds= 10, numDecimalPlaces= 2, hillclimbMetric= Optimize with RME, batchSize= 100, verboseOutput= False, algorithm= Forward selection, debug= False, numModelBags= 10, validationRatio= 0.25, doNotCheckCapabilities= False, greedySortInitialization= True, hillclimbIterations= 100
Naive Bayes	useKernelEstimator= False, numDecimalPlaces= 2, batchSize= 100, debug= False, displayModelInOldFormat= False, useSupervisedDiscretization= False
Logistic Regression	numDecimalPlaces= 4, batchSize= 100, debug= False, ridge= 1.0E-08, useConjugateGradientDescent= False, maxIts= -1, doNotCheckCapabilities= False
J48	Seed= 1, unpruned= False, confidenceFactor= 0.25, numFolds= 10, numDecimalPlaces= 2, batchSize= 100, reducedErrorPruning= False, useLaplace= False, doNotMakeSplitPointActualValue= False, debug= False, subtreeRaising= true, saveInstanceData= False, binarySplits= False, doNotCheckCapabilities= False, minNumObj= 2, useMDLcorrection= True, collapseTree= True

References

1. Chen C, Alfayez R, Srisopha K, Boehm B, Shi L (2017) Why is it important to measure maintainability, and what are the best ways to do it? In: Proceedings of the 39th International conference on software engineering companion. IEEE Press, pp 377–378
2. Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A (2010) Defect prediction from static code features: Current results, limitations, new approaches. Automated Software Engineering Journal 17(4):375–407
3. Fenton NE, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(5):675–689
4. Hall T, Beecham S, Bowes D, Gray D, Counsell S (2012) A systematic literature review on fault prediction performance in software engineering. IEEE Trans Softw Eng 38(6):1276–1304
5. Kamei Y, Shihab E (2016) Defect prediction: Accomplishments and future challenges. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), vol 5. IEEE, pp 33–45
6. Jiang Y, Cukic B, Ma Y (2008) Techniques for evaluating fault prediction models. Empir Softw Eng 13(5):561–595
7. Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng 33(1):2–13
8. Tosun A, Bener AB, Akbarinasaji S (2017) A systematic literature review on the applications of bayesian networks to predict software quality. Softw Qual J 25(1):273–305
9. Hall T, Bowes D (2012) The state of machine learning methodology in software fault prediction. In: Proceedings of the 11th International conference on machine learning and applications, vol 2, pp 308–313
10. Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans Softw Eng 34(4):485–496
11. Challagulla VUB, Bastani FB, Ling I, Paul RA (2005) Empirical assessment of machine learning based software defect prediction techniques. Int J Artif Intell Tools 17(02):389–400
12. Chatterjee S, Nigam S, Singh JB, Upadhyaya LN (2012) Software fault prediction using nonlinear autoregressive with exogenous inputs (narx) network. Appl Intell 37(1):121–129
13. Rathore SS, Kumar S (2017) A study on software fault prediction techniques. Artif Intell Rev 1–73
14. Chatterjee S, Maji B (2018) A bayesian belief network based model for predicting software faults in early phase of software development process. Appl Intell 48(8):2214–2228
15. Madeyski L, Jureczko M (2015) Which process metrics can significantly improve defect prediction models? an empirical study. Softw Qual J 23(3):393–422
16. Rathore SS, Kumar S (2016) An empirical study of some software fault prediction techniques for the number of faults prediction. Soft Comput 1–18
17. Mendes-Moreira J, Jorge A, Soares C, de Sousa JF (2009) Ensemble learning: A study on different variants of the dynamic selection approach, pp 191–205
18. Bowes D, Hall T, Petrić J (2017) Software defect prediction: do different classifiers find the same defects? Softw Qual J, 1–28
19. Huizinga D, Kolawa A (2007) Automated defect prevention: best practices in software management. Wiley, Hoboken
20. Zhu X, Cao C, Zhang J (2017) Vulnerability severity prediction and risk metric modeling for software. Appl Intell 47(3):828–836
21. Menzies T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y (2008) Implications of ceiling effects in defect predictors. In: Proceedings of the 4th international workshop on Predictor models in software engineering, pp 47–54
22. Zhang H, Nelson A, Menzies T (2010) On the value of learning from defect dense components for software defect prediction. In: Proceedings of the 6th International conference on predictive models in software engineering. ACM, p 14
23. Rathore SS, Kumar S (2017) Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems. Knowl.-Based Syst 119:232–256
24. Yohannese CW, Li T, Bashir K (2018) A three-stage based ensemble learning for improved software fault prediction: An empirical comparative study. Int J Comput Intell Sys 11(1):1229–1247
25. Bal PR, Kumar S (2018) Cross project software defect prediction using extreme learning machine: An ensemble based study
26. Wang T, Li W, Shi H, Liu Z (2011) Software defect prediction based on classifiers ensemble. J Info Comput Sci 8(16):4241–4254
27. Laradji IH, Alshayeb M, Ghouti L (2015) Software defect prediction using ensemble learning on selected features. Inf Softw Technol 58:388–402
28. Aljamaan H, Elish MO et al (2009) An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software. In: Proceedings of the symposium on computational intelligence and data mining, pp 187–194
29. (2015) The PROMISE repository of empirical software engineering data, http://openscience.us/repo
30. Rathore SS, Kumar S (2017) Towards an ensemble based system for predicting the number of software faults. Expert Syst Appl 82:357–382
31. Wang S, Yao X (2013) Using class imbalance learning for software defect prediction. IEEE Trans Reliab 62(2):434–443
32. Mısırlı AT, Bener A, Turhan B (2011) An industrial case study of classifier ensembles for locating software defects. Softw Qual J 19(3):515–536
33. Zheng J (2010) Cost-sensitive boosting neural networks for software defect prediction. Expert Syst Appl 37(6):4537–4543
34. Twala B (2011) Predicting software faults in large space systems using machine learning techniques. Def Sci J 61(4):306–316
35. Aljamaan HI, Elish MO (2009) An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software. In: 2009 IEEE Symposium on computational intelligence and data mining. IEEE, pp 187–194
36. Siers MJ, Md ZI (2014) Cost sensitive decision forest and voting for software defect prediction. In: Pacific rim international conference on artificial intelligence. Springer, pp 929–936
37. Li N, Shepperd M, Guo Y (2020) A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology, p 106287
38. Siers MJ, Md ZI (2015) Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem. Inf Syst 51:62–71
39. Laradji IH, Alshayeb M, Ghouti L (2015) Software defect prediction using ensemble learning on selected features. Inf Softw Technol 58:388–402
40. Tong H, Liu B, Wang S (2018) Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Inf Softw Technol 96:94–111
41. Yang X, Lo D, Xia X, Sun J (2017) Tlel: A two-layer ensemble learning approach for just-in-time defect prediction. Inf Softw Technol 87:206–220
42. Pandey SK, Mishra RB, Tripathi AK (2020) Bpdet: An effective software bug prediction model using deep representation and ensemble learning techniques. Expert Syst Appl 144:113085
43. Moustafa S, ElNainay MY, El Makky N, Abougabal MS (2018) Software bug prediction using weighted majority voting techniques. Alexandria Eng J 57(4):2763–2774

44. Shanthini A (2014) Effect of ensemble methods for software fault prediction at various metrics level
45. Hussain S, Keung J, Khan AA, Bennin KE (2015) Performance evaluation of ensemble methods for software fault prediction: An experiment. In: Proceedings of the ASWEC 2015 24th Australasian software engineering conference, pp 91–95
46. Petrić J, Bowes D, Hall T, Christianson B, Baddoo N (2016) Building an ensemble for software defect prediction based on diversity selection. In: Proceedings of the 10th ACM/IEEE International symposium on empirical software engineering and measurement, pp 1–10
47. Li R, Zhou L, Zhang S, Liu H, Huang X, Sun Z (2019) Software defect prediction based on ensemble learning. In: Proceedings of the 2019 2nd International conference on data science and information technology, pp 1–6
48. Yohannese CW, Li T, Bashir K (2018) A three-stage based ensemble learning for improved software fault prediction: an empirical comparative study. Int J Comput Intell Sys 11(1):1229–1247
49. Alsawalqah H, Hijazi N, Eshtay M, Faris H, Radaideh AA, Aljarah I, Alshamaileh Y (2020) Software defect prediction using heterogeneous ensemble classification based on segmented patterns. Appl Sci 10(5):1745
50. Abdou AS, Darwish NR (2018) Early prediction of software defect using ensemble learning: A comparative study. Int J Comput Appl 179(46)
51. Khuat TT, Le MH (2020) Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems. SN Computer Science 1:1–16
52. Twala B (2011) Predicting software faults in large space systems using machine learning techniques
53. Ryu D, Jang Jong-In, Baik J (2017) A transfer cost-sensitive boosting approach for cross-project defect prediction. Softw Qual J 25(1):235–272
54. Saifudin A, Hendric SWHL, Soewito B, Gaol FL, Abdurachman E, Heryadi Y (2019) Tackling imbalanced class on cross-project defect prediction using ensemble smote. In: IOP conference series: Materials science and engineering, vol 662. IOP Publishing
55. Wang T, Zhang Z, Jing X, Zhang L (2016) Multiple kernel ensemble learning for software defect prediction. Autom Softw Eng 23(4):569–590
56. Li N, Li Z, Nie Y, Sun X, Li X (2011) Predicting software black-box defects using stacked generalization. In: 2011 Sixth International conference on digital information management. IEEE, pp 294–299
57. Sun Z, Song Q, Zhu X (2012) Using coding-based ensemble learning to improve software defect prediction. IEEE Trans Sys Man Cybern Part C (Applications and Reviews) 42(6):1806–1817
58. Rathore SS, Kumar S (2016) Ensemble methods for the prediction of number of faults A study on eclipse project. In: 2016 11th International Conference on Industrial and Information Systems (ICIIS). IEEE, pp 540–545
59. Yohannese CW, Li T, Simfukwe M, Khurshid F (2017) Ensembles based combined learning for improved software fault prediction: A comparative study. In 2017 12th International conference on intelligent systems and knowledge engineering (ISKE). IEEE, pp 1–6
60. Bal PR, Kumar S (2018) Extreme learning machine based linear homogeneous ensemble for software fault prediction. In: ICSOFT, pp 103–112
61. Mousavi R, Eftekhari M, Rahdari F (2018) Omni-ensemble learning (oel): Utilizing over-bagging, static and dynamic ensemble selection approaches for software defect prediction. Int J Artif Intell Tools 27(06):1850024
62. Campos JR, Costa E, Vieira M (2019) Improving failure prediction by ensembling the decisions of machine learning models: A case study. IEEE Access 7:177661–177674
63. He H, Zhang X, Wang Q, Ren J, Liu J, Zhao X, Cheng Y (2019) Ensemble multiboost based on ripper classifier for prediction of imbalanced software defect data. IEEE Access 7:110333–110343
64. Malhotra R, Jain J (2020) Handling imbalanced data using ensemble learning in software defect prediction. In: 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, pp 300–304
65. Zheng J (2010) Cost-sensitive boosting neural networks for software defect prediction. Expert Syst Appl 37(6):4537–4543
66. Kumar L, Rath S, Sureka A (2017) Using source code metrics and ensemble methods for fault proneness prediction. arXiv:1704.04383
67. Gao Y, Yang C (2016) Software defect prediction based on adaboost algorithm under imbalance distribution. In: 2016 4th International Conference on Sensors, Mechatronics and Automation (ICSMA 2016). Atlantis Press
68. Coelho RA, dos RN Guimarães F, Esmin AAA (2014) Applying swarm ensemble clustering technique for fault prediction using software metrics. In: 2014 13th International conference on machine learning and applications. IEEE, pp 356–361
69. Ryu D, Baik J (2018) Effective harmony search-based optimization of cost-sensitive boosting for improving the performance of cross-project defect prediction. KIPS Trans Softw Data Eng 7(3):77–90
70. Jonsson L, Borg M, Broman D, Sandahl K, Eldh S, Runeson P (2016) Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. Empir Softw Eng 21(4):1533–1578
71. Li Z, Jing X-Y, Zhu X, Zhang H, Xu B, Ying S (2019) Heterogeneous defect prediction with two-stage ensemble learning. Autom Softw Eng 26(3):599–651
72. Mısırlı AT, Bener A, Turhan B (2011) An industrial case study of classifier ensembles for locating software defects. Softw Qual J 19(3):515–536
73. Ryu D, Choi O, Baik J (2016) Value-cognitive boosting with a support vector machine for cross-project defect prediction. Empir Softw Eng 21(1):43–71
74. Ryu D, Jang J-I, Baik J (2017) A transfer cost-sensitive boosting approach for cross-project defect prediction. Softw Qual J 25(1):235–272
75. Yi P, Kou G, Wang G, Wu W, Shi Y (2011) Ensemble of software defect predictors: an ahp-based evaluation method. International Journal of Information Technology & Decision Making 10(01):187–206
76. Zhang Y, Lo D, Xia X, Sun J (2018) Combined classifier for cross-project defect prediction: an extended empirical study. Frontiers of Computer Science 12(2):280–296
77. Wang H, Khoshgoftaar TM, Napolitano A (2010) A comparative study of ensemble feature selection techniques for software defect prediction. In: 2010 Ninth international conference on machine learning and applications. IEEE, pp 135–140
78. Uchigaki S, Uchida S, Toda K, Monden A (2012) An ensemble approach of simple regression models to cross-project fault prediction. In: 2012 13th ACIS International conference on software engineering, artificial intelligence, networking and parallel/distributed computing. IEEE, pp 476–481
79. Li Z, Jing Xiao-Yuan, Zhu X, Zhang H (2017) Heterogeneous defect prediction through multiple kernel learning and ensemble

learning. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp 91–102
80. Tong H, Liu B, Wang S (2019) Kernel spectral embedding transfer ensemble for heterogeneous defect prediction. IEEE Transactions on Software Engineering
81. Jiang Y, Cukic B, Ma Y (2008) Techniques for evaluating fault prediction models. Empir Softw Eng 13(5):561–595
82. Catal C, Diri B (2009) A systematic review of software fault prediction studies. Expert Systems with Applications 36(4):7346–7354
83. Kim S, Whitehead Jr JE, Zhang Y (2008) Classifying software changes clean or buggy? IEEE Trans Softw Eng 34(2):181–196
84. Chatterjee S, Nigam S, Singh JB, Upadhyaya LN (2012) Software fault prediction using nonlinear autoregressive with exogenous inputs (narx) network. Appl Intell 37(1):121–129
85. Malhotra R (2014) Comparative analysis of statistical and machine learning methods for predicting faulty modules. Appl Soft Comput 21(1):286–297
86. Bishnu PS, Bhattacherjee V (2011) Software fault prediction using quad tree-based k-means clustering algorithm. IEEE Trans Knowl Data Eng 24(6):1146–1150
87. Caglayan B, Misirli AT, Bener AB, Miranskyy A (2015) Predicting defective modules in different test phases. Softw Qual J 23(2):205–227
88. Rathore SS, Kumar S (2017) An empirical study of some software fault prediction techniques for the number of faults prediction. Soft Comput 21(24):7417–7434
89. Yang C-Z, Hou C-C, Kao W-C, Chen X (2012) An empirical study on improving severity prediction of defect reports using feature selection. In: 2012 19th Asia-Pacific software engineering conference, vol 1. IEEE, pp 240–249
90. Yang X, Ke T, Yao X (2014) A learning-to-rank approach to software defect prediction. IEEE Trans Reliab 64(1):234–246
91. Rathore SS, Kumar S (2019) A study on software fault prediction techniques. Artif Intell Rev 51(2):255–327
92. Tantithamthavorn C, Hassan AE (2018) An experience report on defect modelling in practice: Pitfalls and challenges. In: Proceedings of the 40th International conference on software engineering: Software engineering in practice, pp 286–295
93. Li L, Lessmann S, Baesens B (2019) Evaluating software defect prediction performance: an updated benchmarking study. arXiv:1901.01726
94. Dietterich TG (2000) Ensemble methods in machine learning. In: Proceedings of the International workshop on multiple classifier systems, pp 1–15
95. Mendes-Moreira J, Soares C, Jorge A, Sousa JFD (2012) Ensemble approaches for regression: A survey. ACM Computing Surveys 45(1):1–40
96. Ho TK (2002) Multiple classifier combination: Lessons and next steps. Series in Machine Perception and Artificial Intelligence 47:171–198
97. Ting KM, Witten IH (1997) Stacking bagged and dagged models
98. Melville P, Mooney RJ (2003) Constructing diverse classifier ensembles using artificial training examples. In: IJCAI, vol 3, pp 505–510
99. Seewald AK, Fürnkranz J (2001) An evaluation of grading classifiers. In: International symposium on intelligent data analysis. Springer, pp 115–124
100. Seewald AK (2003) Towards a theoretical framework for ensemble classification. In: IJCAI, vol 3. Citeseer, pp 1443–1444
101. Webb GI (2000) Multiboosting: A technique for combining boosting and wagging. Machine Learning 40(2):159–196
102. Friedman J, Hastie T, Tibshirani R et al (2000) Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). Annals Stat 28(2):337–407
103. Lin W-C, Oakes M, Tait J (2008) Real adaboost for large vocabulary image classification. In: 2008 International workshop on content-based multimedia indexing. IEEE, pp 192–199
104. Mauša G, Bogunović N, Grbac TG, Bašić BD (2015) Rotation forest in software defect prediction. In: Proceedings of the 4th Workshop on software quality analysis, monitoring, improvement, and applications, pp 35–44
105. Aldave R, Dussault J-P (2014) Systematic ensemble learning for regression. arXiv:1403.7267
106. Zhang H (2004) The optimality of naive bayes. AA 1(2):3
107. Turhan B, Bener A (2009) Analysis of naive bayes' assumptions on software fault data: An empirical study. Data & Knowledge Engineering 68(2):278–290
108. Kleinbaum DG, Dietz K, Gail M, Klein M, Klein M (2002) Logistic regression. Springer, Berlin
109. Gyimothy T, Ferenc R, Siket I (2005) Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans Softw Eng 31(10):897–910
110. Quinlan JR (1986) Induction of decision trees. Machine Learning 1(1):81–106
111. Quinlan JR (1987) Simplifying decision trees. International Journal of Man-Machine Studies 27(3):221–234
112. Rathore SS, Kumar S (2016) A decision tree logic based recommendation system to select software fault prediction techniques. Computing 1–31
113. Witten IH, Frank E (2005) Data practical machine learning tools and techniques. Morgan Kaufmann, Burlington
114. Jiang Y, Cuki B, Menzies T, Bartlow N (2008) Comparing design and code metrics for software quality prediction. In: Proceedings of the 4th international workshop on Predictor models in software engineering. ACM, pp 11–18
115. Arisholm E, Briand LC, Johannessen EB (2010) A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J Syst Softw 83(1):2–17
116. Cohen P, West SG, Aiken LS (2014) Applied multiple regression/correlation analysis for the behavioral sciences. Psychology Press
117. Wagner S (2006) A literature survey of the quality economics of defect-detection techniques. In: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering. ACM, pp 194–203
118. Kumar L, Misra S, Rath SK (2017) An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. Computer Standards & Interfaces 53:1–32
119. Jones C, Bonsignour O (2011) The economics of software quality. Addison-Wesley Professional
120. Wilde N, Huitt R (1991) Maintenance support for object oriented programs. In: Proceedings. Conference on Software Maintenance 1991. IEEE, pp 162–170
121. Boehm B, Papaccio PN (1988) Understanding and controlling software costs. IEEE Trans Softw Eng 14(10):1462–1477

Santosh Singh Rathore has obtained his PhD degree from Indian Institute of Technology (IIT) Roorkee-India. Currently, he is working as an Assistant Professor with the Department of Information Technology, Indian Institute of Information Technology and Management Gwalior (ABV-IIITM Gwalior), Gwalior, India. He has supervised 9 Master's dissertations, about 20 Undergraduate projects, and is currently supervising 6 Master's students. He has published more than 15 research papers in international/national journals and conferences and has also authored/co-authored two books with Springer. His research interests are Software Fault Prediction, Software Quality Assurance, Empirical Software Engineering, Object-Oriented Software Development, and ObjectOriented Metrics. Dr. Rathore was the recipient of the SIGSE travel award 2012, Jenesys award 2012, and others.

Sandeep Kumar (SMIEEE' 17) is currently working as an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Roorkee, India. He has supervised four PhD thesis, sixty five master dissertations, about fifteen undergraduate projects, and is currently supervising four PhD students. He has published more than seventy five research papers in international/national journals and conferences and has also authored/co-authored three books with Springer. He has also filed two patents for his work done along with his students. Dr. Sandeep is the member of board of examiners and board of studies of various universities and institutions. He has collaborations in industry and academia. He is currently handling multiple national and international research/consultancy projects. He has received Young Faculty Research Fellowship award from MeitY, Govt. of India, NSF/TCPP early adopter award-2014, 2015, ITS Travel Award 2011 and 2013 and others. He is member of ACM and senior member of IEEE. His name has also been enlisted in major directories such as Marquis Whos Who, IBC and others. His areas of interest include Semantic Web, Web Services, Software Engineering, and Machine Learning.