e-Informatica Software Engineering Journal, Volume 16, Issue 1, 2022, pages: 220104, DOI 10.37190/e-Inf220104

Examining the Predictive Capability of Advanced Software Fault Prediction Models – An Experimental Investigation Using Combination Metrics

Pooja Sharma*, Amrit Lal Sangal*

**Dr. B R Ambedkar National Institute of Technology, Jalandhar, India*

poojanitjal@gmail.com, sangalal@nitj.ac.in

Abstract

Background: Fault prediction is a key problem in software engineering domain. In recent years, an increasing interest in exploiting machine learning techniques to make informed decisions to improve software quality based on available data has been observed.
Aim: The study aims to build and examine the predictive capability of advanced fault prediction models based on product and process metrics by using machine learning classifiers and ensemble design.
Method: Authors developed a methodological framework, consisting of three phases, i.e., (i) metrics identification (ii) experimentation using base ML classifiers and ensemble design (iii) evaluating performance and cost sensitiveness. The study has been conducted on 32 projects from the PROMISE, BUG, and JIRA repositories.
Result: The results shows that advanced fault prediction models built using ensemble methods show an overall median of F-score ranging between 76.50% and 87.34% and the ROC(AUC) between 77.09% and 84.05% with better predictive capability and cost sensitiveness. Also, non-parametric tests have been applied to test the statistical significance of the classifiers.
Conclusion: The proposed advanced models have performed impressively well for inter project fault prediction for projects from PROMISE, BUG, and JIRA repositories.

Keywords: product and process metrics, classifiers, ensemble design, software fault prediction, software quality

1. Introduction

Software fault prediction has been an important research topic in the software engineering field for more than three decades, increasingly catching the interest of researchers [1, 2]. According to IEEE terminology [3] the term fault is used to indicate an incorrect step, process, or data definition in a computer program (i.e., a BUG). In the literature, authors have addressed the software fault prediction (SFP) problem with two viewpoints, i.e., in the first viewpoint, they proposed new method or method combinations to increase fault prediction performance. In the second viewpoint, they used new parameters to present the most influential metrics for fault prediction. Based on first perspective many fault prediction approaches have been proposed in literature and most of these papers

Submitted: 27 Oct. 2020; Revised: 02 Dec. 2021; Accepted: 07 Dec. 2021; Available online: 14 Feb. 2022

categorize a software module faulty or non-faulty. Unfortunately, fault-proneness of software components classification remains a largely unsolved problem [2]. In order to address this issue, researchers have been increasingly using sophisticated techniques and we can say that the fault prediction is going towards novel and more attractive directions, like the use of machine learning, deep learning or unsupervised techniques [4–6]. The usage of machine learning algorithms has increased in the last decade and is still one of the most popular methods for defect prediction [4, 6–10]. According to Lessmann et al. [11] ““there is a need to develop more reliable research procedures before we can have confidence in the conclusion of comparative studies of software prediction models””. Thus, in the present study we aim to consider and evaluate the performance of different classifier models and not any particular classifier. Further, application of ensemble techniques has been reported by the researchers [4, 8, 12] for improving the accuracy of fault prediction. Moreover, the diversity of classifiers, while building the ensemble model, should also be investigated to improve the effectiveness of the ensemble designs [9]. This motivated us to design ensembles for improving predictive capability of classifiers.

As regards to the second viewpoint, considerable amount of the research has been undertaken in which authors have used software metrics extracted from the code to unveil whether a software component is fault prone or not. It has been observed that fault estimation models are designed mainly based on product metrics in literature [13–16], but the models which are build using a combination of product and process metrics are little known [17, 18]. Though some authors [19, 20] has emphasized about the usage of both product and process metrics in their works. Madeyski and Jureczko [18], in their research, determined that process metrics provide information for fault proneness. The usage of process metrics to ascertain the faults possibly results in superior outcomes than only with the product metrics. They emphasized the need to conduct further studies and establish evidence for developing such advanced models. Radjenovic et al. [19] in their SLR, stressed finding ways to measure and evaluate process-related information for fault proneness. Wan et al. [19] in their study on perceptions, expectations, and challenges in defect prediction, concluded that software practitioners prefer rational, interpretable, and actionable metrics for defect prediction. It is also observed from the literature studies that not only process metrics have been shown to be superior to product metrics, but also alternative features have been proposed on the basis of developer-related factors, code smells, etc. [21–24]. This calls for further studies to examine the association between metrics and fault proneness to provide meaningful insights for making informed decisions. To this effect, the authors in the present study aimed to develop advanced models for software fault prediction, which utilises combination metrics. After finding a suitable set of product metrics, advanced fault prediction models are created using process metrics one at a time approach.

Thus, to motivate the need for development of advanced models for fault prediction authors in the present study developed a research framework which consists of three phases. In Phase-I, the metrics were identified after performing pre-processing and feature extraction on the datasets. In phase-II, experimentation is carried out by training and testing various models using machine learning classifiers, i.e., Naive Bayes (NB), Decision Tree (DT), Multilayer Perceptron (MLP), Random Tree (RT), and Support Vector Machine (SVM). To estimate the performance of the advanced models, an assessment criterion based upon accuracy, root mean square error (RMSE), F-score, and the area under curve AUC(ROC) has been applied. In phase-III, rather than relying on the outcome of base classifiers, authors used the ensemble approach to combine multiple classifiers to further improve the performance, particularly fault-detection abilities. Also, the cost sensitiveness of the proposed best models is examined. The comparison of results confirms the predictive capability of proposed classifiers for developing advanced fault prediction models.

Thus, the significant contributions of the work are as follows:

1. Development of learning scheme consisting of both base and ensemble learning classifiers.
2. Building and examining the predictive capability of advanced fault prediction models.
3. Evaluating the cost sensitiveness of the proposed ensemble-based classifier using a cost evaluation framework.

The work presented in the study is reported as follows. Section 2 offers related research. Section 3 presents a description of the proposed framework, research questions, dataset selection, feature extraction, selection, normalization procedure, classifier selection, and performance measurement indices. Section 4 presents the experimental design and Section 5 presents the results. Section 6 presents threats to validity, and Section 7 presents the conclusions.

2. Related Work

Over the preceding two decades, software researchers have shown great prominence in fault prediction studies, as evident from work dealing with the development of fault prediction models. Table 1 presents the state of the art and proposed benchmark solutions. The contributions provided by the researchers in recent years are summarised based on the software metrics (product, process, change) and techniques used to tackle the software fault prediction problem. Malhotra and Jain [8] provided empirical comparison of software defect prediction models developed by using various boosting based ensemble methods on three open source JAVA projects. Ghotra et al. [25] studied the impact of classification techniques on the performance of defect prediction models. Yucalar et al. [26] conducted experiments using 15 software projects from the PROMISE repository to demonstrate that ensemble predictors might improve fault detection performance to some extent. Qiao et al. [16] proposed deep learning techniques to predict defects in a software system. The study by Malhotra [15] uses a logistic regression-based classifier on object-oriented metrics data set to predict the software fault proneness. Laradji et al. [27] demonstrated the positive effects of combining feature selection and ensemble learning on the performance of defect classification.

Comprehensive surveys on fault prediction were presented by Catal and Diri [28], Li Zhiqiang et al. [1]; Matloob et al. [9] and Radjenovic et al. [19] in the context of prediction models, modelling techniques and the metrics used. According to Radjenovic et al. [19], in the literature on fault prediction studies, process metrics account for 24%, source code accounts for 27%, and object-oriented accounts for 49%. Future studies shall apply the ways to measure and evaluate process-related information for fault proneness along with product metrics. Madeyski and Jureczko [17] performed an empirical study using industrial and open-source software datasets to ascertain the process metrics, which noticeably improved results. At the same time, they stressed upon replicating the study using machine learning approaches, as it is unclear whether the features that work fine in one method will also be useful in other approaches. Hence, experimentation can be conducted to investigate the usage of the product and process-related metrics. Khoshgoftaar et al. [29] build software quality models with majority voting using multiple training datasets. The work can be extended using data from various software project repositories and analyse the predictive capability of ensembles as compared to base classifiers for advanced models. Chen et al.

[30] in work investigated whether different cross–project defect prediction methods identify the same defective modules? The result can be extended by using learning approaches based on ensemble design to further improve cross–project defect prediction performance. In the study Zhang et al. [31], investigated the use of various algorithms that integrate ML predictors for cross-project defect prediction. However, for examining the predictive capability of advanced algorithms, additional experimentation is required.

Studying the presented works above, it is clear that using a pre-processing technique on the dataset significantly affected the performance of learning algorithm. Most of the studies lack the processing on a larger dataset so that the generalized model will be formed. Also, class imbalance problem, needs to be addressed to improve the performance of fault prediction [9]. The parameter combinations are often less investigated in literature studies. Hence, it is observed that the work can be replicated by including more datasets with focus on product and process software metrics and experimenting different scenarios or combinations of models (simple and advanced models) to achieve the reliability and robustness.

Further investigations shall include the use of more classifiers or classifier ensembles and the development of advanced defect prediction models with datasets from various projects written in different programming languages, and commercial projects from industry can also be considered for experimentation. In the proposed work, authors presented a three phase framework consisting of dataset pre-processing, feature extraction and selection; learning classifiers along with cost evaluation to predict the fault-prone components.

Table 1. Literature review

Authors	Metrics considered	Study outcomes and proposed benchmark solutions
Song et al. [2]	Product metrics	Authors proposed and evaluated a general framework for software defect prediction using different learning schemes for different data sets. The future work shall include process attributes for fault estimation. Experiments with the various available techniques can be undertaken for generalization.
Yang et al. [5]	Product metrics	Authors proposed a deep learning technique to predict defect-prone changes. The experiments can be replicated on more datasets using other classifiers to reduce the threats to external validity.
Yibiao et al. [6]	Change metrics	Authors investigated the predictive power of simple unsupervised models in effort-aware JIT defect prediction using commonly used change metrics. The work can be checked with closed-source software systems.
Yang et al. [4]	–	Authors hybridized various ensemble learning methods to examine performance of just-in-time defect prediction. Experiments on more datasets can be performed to reduce the threats due to external validity.
Matloob et al. [9]	–	This research provides a systematic literature review on the use of the ensemble learning approach for software defect prediction and stressed for further analysis and comparison of results.
Pascarella et al. [10]	Change metrics	Authors proposed a novel fine-grained just-in-time defect prediction model to predict the specific files, contained in a commit, that are defective. Future work can replicate the results on a larger set of systems in an industrial context by including other independent variables too.

Table 1 continued

Malhotra, Jain [8]	Product metrics	Authors provided empirical comparison of software defect prediction models developed by using various boosting based ensemble methods on three open source JAVA projects. The future work shall investigate more attributes for fault estimation with more datasets for replication.
Li et al. [14]	Code metrics	Authors summarised the defect prediction studies focusing on emerging topics, e.g., ML-based algorithms, data manipulation, and effort-aware prediction. They stressed overcoming the class imbalance problem and the development of models in defect prediction.
Ghotra et al. [25]	Product metrics	Authors studied the impact of classification techniques on the performance of defect prediction models using NASA dataset and the Promise dataset. Further experiments with the various available techniques can be undertaken for generalization.
Yucalar et al. [26]	Product metrics	The authors conducted experiments using 15 software projects from the Promise repository to demonstrate that ensemble predictors might improve fault detection performance to some extent. The future work shall investigate more attributes for fault estimation to provide help in successive releases.
Rathore and Kumar [12]	Product metrics	Authors performed an investigation on ensemble techniques for SFP by using 21 object-oriented software metrics. Future work can assess the ensemble techniques for the fault datasets from other software systems and shall include additional software metrics for generalization.
Qiao et al. [16]	Product metrics	The authors proposed deep learning techniques to predict defects in a software system. In future work, more investigations by including more projects are written in different programming languages, and commercial projects from industry can be carried out.
Malhotra [15]	Product metrics	The study uses a logistic regression-based classifier on object-oriented metrics data set to predict the software fault proneness. Future investigations shall include the use of more classifiers or classifier ensembles and the development of advanced defect prediction models with cross project defect prediction datasets from various projects.
Madeyski and Jureczko [18]	Product and Process	They performed an empirical study using industrial and open source software datasets to ascertain the process metrics, which noticeably improved results. At the same time, they stressed upon replicating the study using ML approaches, as it is unclear whether the features that work fine in one method will also be useful in other approaches. Hence, experimentation can be conducted to investigate the usage of the product and process-related metrics.
Radjenovic et al. [19]	Process and Product	According to the authors, in the literature on fault prediction studies, process metrics account for 24%, source code accounts for 27%, and object-oriented accounts for 49%. Future studies shall apply the ways to measure and evaluate process-related information for fault proneness along with product metrics.
Rahman, and Devanbu [24]	Product and process	Authors analysed the applicability and efficiency of process and code metrics. The future work shall replicate the findings with more data sets from several different perspectives.
Bird, Christian et al. [21]	Change metrics	Authors examined the relationship between different ownership measures and software failures in two large software projects: Windows Vista and Windows 7.

Table 1 continued

Nucci et al. [22]	Product and change metrics	Provided a developer centred bug prediction model. Work can be extended to analyse the role of developer related factors along with product metrics in the bug prediction field using different base line predictors.
Palomba et al. [23]	Process and Product	Authors evaluated the code smell intensity by adding it to existing bug prediction models based on both product and process metrics. Future work shall be devoted to the analysis of the contribution of smell-related information in the context of local-learning bug prediction models.
Laradji et al. [27]	Product metrics	Authors demonstrated the positive effects of combining feature selection and ensemble learning on the performance of defect classification. The work can be replicated by including more datasets with focus on product and process software metrics.
Lee et al. [32]	Process and Product	They proposed micro-interaction metrics to study developer"s interaction by experimenting with Mylyn dataset. More experiments need to be conducted to show that MIMs considerably improve software defect prediction.
Juneja [33]	Product	Author"s proposed Neuro-fuzzy framework to predict the fault in software system based on feature based evaluation of inter-project and intra-project modules. The effectiveness of models can be compared using process metrics.
Wang et al. [34]	Product metrics	The authors performed a study using seven classifiers ensemble methods on MDP datasets from real software projects of NASA. The use of classifiers ensemble on multiple datasets can be experimented.
Petric et al. [35]	Product metrics	They used explicit diversity technique with stacking ensemble to investigate improvement in defect prediction. The work can be extended and the experiments should be conducted using more classifiers and applying full parameter search in order to build models with superior performances.
Pecorelli and Nucci [36]	Product metrics	Authors compared the performance of seven ensemble techniques on 21 open-source software projects to verify how ensemble techniques perform in cross and local project settings. The work can be replicated using cross-project and within-project strategies in larger contexts, using a richer set of independent variables.
Nucci et al. [37]	Product metrics	An empirical study conducted on 30 software systems indicates that ASCI exhibits higher performances than five different classifiers used independently and combined with the majority voting ensemble method. Work can be extended to analyse how the proposed model works in the context of cross-project bug prediction.
Bowles et al. [38]	Product metrics	Authors investigated difference in the individual defects and prediction stability using RPart, SVM, Naive Bayes, and Random Forest classifiers. They used NASA, open-source, and commercial datasets. The work can be extended by developing advanced models using ensemble-based classifiers.
Abaei and Selamat [39]	Product metrics	They proposed fuzzy clustering and probabilistic neural network to study defect prediction accuracy. The use of machine learning approaches can be investigated to analyze advanced defect prediction models.
Erturk and Sezer [40]	CK Product metrics	In their work, the authors concluded that ANFIS outperforms NN and SVM approaches for predicting faults. The future work may include the process metrics or develop advanced defect prediction models.

Table 1 continued

Zhang et al. [31]	Process and Product metrics	In the study authors, investigated the use of various algorithms that integrate ML predictors for cross-project defect prediction. However, for examining the predictive capability of advanced algorithms, additional experimentation is required.
Khoshgoftaar et al. [29]	Product and Process	Authors build software quality models using majority voting using multiple training datasets. The work be extended using data from various software project repositories and analyze the predictive capability of ensembles as compared to base classifiers for advanced models.
Yong Hu et al. [41]	Product (all CK metrics)	This study provides a research framework that combines cost-sensitive learning with the ensemble method. Future work can examine the use of ensembles trained on different datasets. Such solutions may not only enhance the prediction accuracy but also address the defect prediction problems.
Elish et al. [42]	product metrics	The authors used product metrics to investigate and empirically validate ensemble methods for software maintenance effort and change proneness. However, future studies shall use the proposed ensemble approaches to investigate defect prediction using combination metrics.
Chen et al. [30]	Process and Product	The authors in work investigated whether different cross project defect prediction methods identify the same defective modules. The result can be extended by using learning approaches based on ensemble design to further improve cross project defect prediction performance.
Peng He et al. [43]	Static code metrics	The authors provided guidelines for the selection of training data, classifier, and metric subset. They conducted an empirical study on software defect prediction with a simplified metric set. The guidelines can further be used to develop advanced models for defect prediction in different scenarios.
Wasiur R et al. [44]	Change metrics	Authors conducted an empirical study for defect prediction using software change metrics. The application of hybrid algorithms used in the task can be used to develop advanced models.
Kaur and Kaur [45]	Product metrics	Authors used statistical and machine learning techniques for predicting the quality of the software. For experimentation, they used five open source software projects. Further experiments can be conducted using product-process or combination metrics using cross project defect data.

3. Research Framework

The proposed framework consists of three phases, as shown in Figure 1. Phase-I deals with "dataset pre-processing, feature extraction and experimental setup;" Phase-II is "classification methods, ensemble design and performance measurement" and Phase-III is "cost evaluation framework". Briefly, the phases shown in Figure 1 are discussed as:
Phase-I deals with identifying the metrics suite from metric datasets available in PROMISE, BUG, and JIRA dataset repository. Further, various pre-processing methods such as feature ranking methods and feature subset selection methods and normalization have been applied to select a minimal subset of features from the original dataset so that the features are reduced based on a specific evaluation criterion. It also reduces the dimensionality of feature

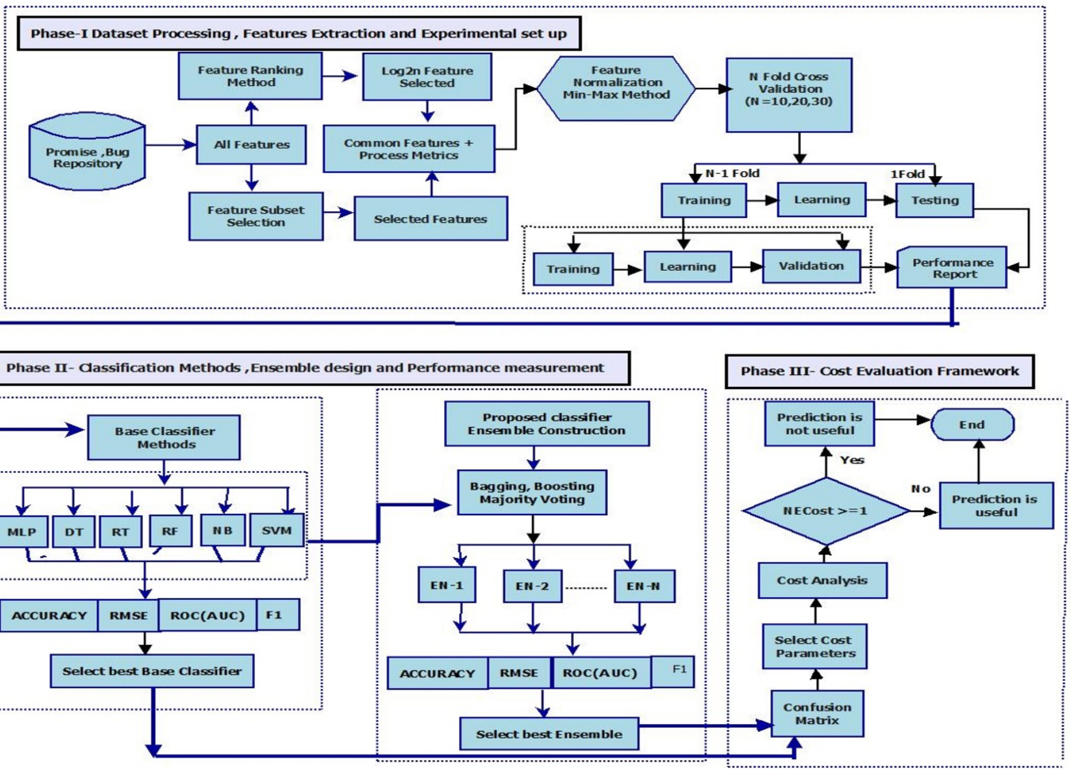

Figure 1. A framework of Proposed ensemble model with cost analysis

space, removes redundant, irrelevant information and improves the data quality, thereby improving the algorithm performance. An experimental design with N-fold cross-validation is used to train, test and replicate the experiment using various datasets.

Phase-II deals with the evaluation of simplified dataset representing different scenarios, i.e., scenario-1: simple model (product metrics); scenario-2: Advanced model-1 (Product metrics + NR process metric); scenario-3: Advanced model-2 (Product metrics + NDC process metric); scenario-4: Advanced model-3 (Product metrics + NML process metric); scenario-4: Advanced model-4 (Product metrics + NDPV process metric) using various base ML classifiers. The performances of proposed models are evaluated using performance indices, i.e., accuracy, AUC (ROC), RMSE, and F-score. Further, to improve base ML classifier"s performance, the classifier ensembles were designed by following Bagging, AdaBoostM1 (which is the most popular version of boosting), and Voting algorithms.

Phase-III deals with examining the cost sensitiveness of the proposed ensemble classifiers. It is achieved by developing a cost analysis framework to compare the best ensemble"s cost with the best base classifier by finding normalized fault removal cost.

Research Questions

Based on the literature studies and potential research gaps, the research questions framed are as follows:

RQ1: How does the advanced defect prediction models proposed in the study perform using various machine learning classifiers?

RQ2: How does the ensemble design improve classification performance when compared to individual machine learning classifiers?

RQ3: Whether there exist any statistically significant performance difference among the base classifiers and ensemble classifiers?

RQ4: For a given software system, whether the proposed ensembles are cost sensitive?
The rationale behind the selection of the research questions RQ1 and RQ2 is to investigate the effectiveness of advanced models representing different scenarios of combination of software product and process metrics. These models are trained using base learning and ensemble based classifiers. The model performances are tested with measures such as accuracy, RMSE, ROC(AUC) and F-score. The rationale behind the usage of statistical test was to find the empirical evidence regarding the performance of predictors, i.e., to answer RQ3. Cost-based evaluation framework has been adopted to examine the cost-sensitiveness of proposed predictors in RQ4.

3.1. Selection of Dataset

In software engineering, Tera-Promise [46], Bug Prediction Dataset [47], Promise [48] and NASA and repositories contain versioned datasets of different software projects that can be assessed for fault prediction. In the present study, authors examined versioned datasets of (i) Ant, Camel, J-edit, Lucene, Synapse, Xalan, Xerces projects from the Promise repository, (ii) Equinox, Eclipse-JDT, Eclipse-PDE, MYLYN projects from the Bug dataset and (iii) ActiveMQ 5.0.0, Derby-10.5.1.1, Groovy1_6_BETA_1, Hbase-0.94.0, Hive-0.9.0, Jruby-1.1, Wicket-1.3.0beta2 from Jira repository, respectively. Table 2 presents the data related to versions, total modules, faulty modules, and defect rates of different projects with their interpretations. To improve the quality of software datasets, we performed data pre-processing following the guidelines provided by Shepperd et al. [49] in order to remove noisy data. To make the training set uniform for the fault-prone and non-fault prone classes to handle data imbalance, in the study, we have applied the synthetic minority over-sampling technique proposed by Chawla et al. [50]. In literature, researchers too have considered class imbalance learning techniques to improve the predictors performance [8, 29, 51].

Table 2. Project dataset versions

	Project	Total modules	Faulty modules	Defect rate
Promise dataset	ant 1.4	178	40	22.47
	ant 1.5	293	32	10.92
	ant 1.6	351	92	26.21
	ant 1.7	745	166	22.28
	camel 1.2	608	216	35.53
	camel 1.4	872	145	16.63
	camel 1.6	965	188	19.48
	jedit 4.0	306	75	24.51
	jedit 4.1	312	79	25.32
	jedit 4.2	367	48	13.07
	jedit 4.3	492	11	2.24
	Lucene 2.2	247	144	58.3
	Lucene 2.4	340	203	59.7
	synapse 1.1	222	60	27.03
	synapse 1.2	256	86	33.59
	xalan 2.5	803	387	48.19
	xalan 2.6	885	411	46.44

Table 2 continued

Promise dataset	xalan 2.7	909	897	98.79
	xerecs 1.2	440	71	16.14
	xerecs 1.3	453	69	15.23
	xerecs 1.4	588	437	74.32
Projects from Bug repository	Equinox	324	129	39.81
	Eclipse-JDT	997	206	20.06
	Eclipse-PDE	1497	209	13.96
	MYLYN	1862	245	13.15
Projects from Jira repository	ActiveMQ 5.0.0	1884	293	15.55
	Derby-10.5.1.1	2705	383	14.15
	Groovy1_6_BETA_1	821	70	8.52
	Hbase-0.94.0	1059	218	20.58
	Hive-0.9.0	1416	283	19.98
	Jruby-1.1	731	87	11.9
	Wicket-1.3.0beta2	1763	130	7.5

3.2. Feature Extraction, Selection and Normalization

Feature selection is categorised as feature ranking and feature subset selection, or be classified as filters and wrappers. In filter based algorithms, a subset of features is selected without involving any learning algorithm and in wrapper based algorithms feedback from a classification learning algorithm is used to determine the feature(s) to be included in development of a classification model. The more refined a feature subset becomes, the more stable a feature selection algorithm is [42]. It reduces the dimensionality of feature space, removes redundant, irrelevant information and improves the quality of the data thereby improving the performance of the algorithm. In the literature [42, 52, 53] numerous methods have been proposed to discard features which are least important to improve defect prediction.

3.2.1. Feature Ranking Methods

It is the process of ordering the features based upon the value of some scoring function, which generally measures feature relevance. In this study, authors have used Information Gain (IG) attribute estimation which is the frequency driven observation in which the information explicit to a particular metric is considered on the class value. The available information is corresponding to the fault proneness of specific modules. Similar feature ranking methods has been applied by various authors in their work on software fault prediction [7, 19, 26, 31]. Gain Ratio (GR) attribute estimation is an alternative of IG and is used to rank the attributes present in the datasets to reduce its bias [19, 54]. Gain Ratio is used for the proliferation of nodes when data is evenly distributed and small while choosing an attribute when all data belong to one branch.

3.2.2. Feature Subset Selection Methods

Instead of using all metrics of the dataset, a subset of features is used as input in the study. These methods are used to generate a subset of attributes that jointly have excellent predictive ability. The classifier subset evaluation method uses a classifier method to estimate the "merit" of the possible subsets of features in the project. It evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them [50, 53]. Correlation-based feature selection (CFS) evaluates values of the subset of attributes according to correlation with the class label and individual features along with the degree of redundancy between them [55]. Filtered subset evaluation is a random subset of the evaluator made to run on a class through an arbitrary filter using data [50, 56]. These filters do not change the order, and the numbers of attributes entirely depend on training data. In literature CFS based feature selection technique has been applied by various authors [7, 45, 57].

3.2.3. Feature extraction and selection

In the study, feature ranking and feature subset selection techniques such as IG, GR attribute evaluation, Classifier subset evaluation, CFS subset evaluation and Filtered subset evaluation were used in the experiments. The common sets of features extracted are shown in Table 3, respectively. A total of 15 features are selected and used in the experiments. The simple defect prediction model is constructed using the 15 product metrics, and advanced defect prediction models are built using 15 product metrics and single process metric with one at a time approach, as discussed in Section 3. Table A1 in appendix provides the definitions for the selected features based on product and process metrics.

Table 3. Selection of metrics

Feature Ranking Methods	Selected Metrics
Information Gain (IG)	AMC, LOC, CAM, LCOM3, LOC, AVG-CC, RFC, MFA, WMC, CBO, DAM, NPM, CE, MAX-CC, MOA, CA, NOC, CBM, IC, DIT
Gain Ratio (GR)	AMC, LCOM3, LOC, LCOM, CAM, AVG-CC, DAM, MFA, MOA, RFC, WMC, MAX-CC, CE, CBO, NPM, NOC, CA, CBM, IC, DIT
Feature Subset Methods	**Selected Metrics**
Subset evaluation Classifier	AMC, LCOM3, LOC, LCOM, CAM, AVG-CC, DAM, MFA, MOA, RFC, WMC, MAX-CC, CE, CBO, NPM, NOC, CA, CBM, IC, DIT
CFS subset evaluation	MOA, DAM, MAX-CC, LCOM, NOC, LCOM3, CE, IC, NPM, CBO, WMC, DIT, CA, RFC, MFA, AMC, LOC
Filtered subset evaluation	WMC, DIT, NOC, LCOM, NPM, MOA, CA, RFC, CE, LOC, DAM, AMC, CBO, AVG-CC, MAX-CC
Common Selected Features	LCOM, CA, LOC, AMC, CBO, RFC, DAM, WMC, DIT, NOC, MOA, CAM, MAX-CC, CE, NPM
Process Metrics	NR, NDC/NAUTH, NML/NREF, NDPV

3.2.4. Normalization of selected features

The performance of prediction models can also be affected by the different levels of design complexity metrics [58–60]. Various software metrics values which are obtained from the dataset have different ranges or magnitude; to make the data in a similar series or format, we have applied data normalization. For the data normalization process, a simple min-max normalization method is used [61]. After the data is normalized, the values are transformed between intervals of 0–1.

3.3. Selection of classifiers

The main aim of the study is to demonstrate the predictive capability of advanced software defect perdition models. The well-known ML classifiers, i.e., Naive Bayes (NB); Decision tree (DT); Random tree (RT); Support Vector Machine (SVM) and Multilayer Perceptron (MLP) are used in the study to build defect prediction models. We used Catal et al. [28] review to determine frequency of base predictors in the software fault prediction literature. Authors performed comparative experimentation by taking one classifier from each category to achieve the balance between different classification models (statistical approaches, neural networks and tree-based methods) as proposed by various researchers [42–44, 61]. Also, to get an enhanced learning algorithm, classifiers ensembles have been designed. The names of the classifiers, classifiers ensembles and their references with brief description are presented in Table A2 in Appendix.

3.4. Performance measurement indices

For the assessment of defect predictors performance, various measures have been used in literature by researchers [11, 19, 27, 62, 63]. In the study, the performance indices, i.e., accuracy, RMSE, ROC (AUC) and F-score are used to measure the performance of fault prediction models. The brief details are presented in Table A3 in Appendix. Table A4 in Appendix presents the confusion matrix for fault prediction models, which is used to compute all the parameters. It contains actual and predicted classification information using various prediction techniques.

3.5. Framework for cost evaluation

Cost-based evaluation framework is necessary to assess the usability of designed fault prediction models. The analysis of cost evaluation is very important because misclassification of faulty prone (fp) modules is more costly as compared to the misclassification of non-faulty prone (nfp) modules. Some researchers [14, 41, 53] have adopted a cost evaluation criterion in their study. In this section, we discussed the cost evaluation framework, proposed by Wagner [64]. He has designed the cost-based evaluation framework based on certain constraints, as mentioned below:
(i) Different phases (unit, integration and testing phases) of testing account for different fault removal cost.
(ii) None of testing phase can detect 100% faults.
(iii) It is not practically feasible to perform unit testing on all modules, so a limited number of important logical paths should be selected to ensure proper working of the delivered software.
Since different projects are developed on varying platforms and in varying organization standards, the cost varies. The normalized fault removal cost for test techniques, i.e., unit, integration, system and field are presented in Table 4 with min, max and median values. The fault detection efficiency values for different test phases are taken from study by Jones [65]are summarized in Table 5. Wilde and Huitt [66] stated that more than fifty percent of

modules are usually very small in size; hence performing unit testing on these modules is not fruitful.

3.5.1. Estimated fault removal cost (E_{cost})

The estimated fault removal cost (E_{cost})is the sum of cost of unit testing, cost for integration test system test and the cost for field test. The number of faulty modules recognized by the predictor is the sum of true positive and false positive values. Hence, it is important to calculate testing and verification cost at the module level, which means that this cost is equal to the cost of unit testing ($Cost_{\text{unit}}$). Equation (1) shows the total unit testing cost.

$$Cost_{\text{unit}} = (TP + FP) * Cost_u \tag{1}$$

The fault removal cost for integration test ($Cost_{\text{integration}}$) is obtained as

$$Cost_{\text{integration}} = \delta_i * C_i * (FN + TP(1 - \delta_u)) \tag{2}$$

The left out faulty modules which are not predicted by integration testing are predicted by system test. Equation (3) gives the fault removal cost for system test

$$Cost_{\text{system}} = \delta_s * C_s * (1 - \delta_i) * (TP(1 - \delta_u) + FN) \tag{3}$$

For the left out faulty modules which were not predicted in system testing, field-testing is done. The fault removal cost for field test ($Cost_{\text{field}}$) is given by Eq. (4) as

$$Cost_{\text{field}} = (1 - \delta_s) * C_f * (1 - \delta_i)(TP(1 - \delta_u) + FN) \tag{4}$$

So, the value of the overall estimated fault removal cost can be determined by adding Eq. (1) to (4), as shown by Eq. (5)

$$E_{\text{cost}} = Cost_{\text{unit}} + Cost_{\text{integration}} + Cost_{\text{system}} + Cost_{\text{field}} \tag{5}$$

3.5.2. Estimated testing cost (T_{cost})

The steps followed to calculate estimated testing cost are:
The cost of unit testing on all the modules is given by Eq. (6)

$$Cost_{\text{unit}} = M_p * C_u * TM \tag{6}$$

The testing cost for faulty modules that are not detected during unit testing and may be detected in integration, system, and field testing are calculated as follows.

$$Cost_{\text{integration}} = \delta_i * C_i * (1 - \delta_u) * FM \tag{7}$$

$$Cost_{\text{system}} = \delta_s * C_s * (1 - \delta_i) * (1 - \delta_u) * FM \tag{8}$$

$$Cost_{\text{field}} = (1 - \delta_s) * (1 - \delta_i) * (1 - \delta_u) * FM \tag{9}$$

The overall value of estimated testing cost (T_{cost}) is given by adding the Eq. (6)) to (9), as represented by Eq. (10)

$$\begin{aligned} T_{\text{cost}} = (\{M_p * C_u * TM\} + \{\delta_i * C_{\text{i}} * (1 - \delta_u) * FM\} + \\ \{\delta_s * C_s * (1 - \delta_i) * (1 - \delta_u) * FM\} + \{(1 - \delta_s) * (1 - \delta_i) * (1 - \delta_u) * FM\} \end{aligned} \tag{10}$$

3.5.3. Normalized fault removal cost (NE_{cost})

The normalized fault removal cost is obtained as ratio of estimated fault removal cost to estimated testing cost, as shown by Eq. 11

$$NE_{\text{cost}} = \frac{E_{\text{cost}}}{T_{\text{cost}}} = \begin{cases} < 1 & \text{application of proposed fault prediction is useful} \\ \geq 1 & \text{application of testing methods is useful} \end{cases} \tag{11}$$

Where: E_{cost} and T_{cost} is the estimated fault removal cost of the software with and without using the fault prediction approach.

Table 4. Removal cost for test techniques (staff hours per defect)

Testing Type	Min	Max	Median
Unit (C_{u})	1.5	6	2.5
Integration (C_{i})	3.06	9.5	4.55
System (C_{s})	2.82	20	6.2
Field (C_{f})	3.9	66.6	27

Table 5. Fault identification efficiencies for different test phases

Testing Type	Min	Max	Median
Unit(δ_{u})	0.1	0.5	0.25
Integration(δ_{i})	0.25	0.60	0.45
System(δ_{s})	0.25	0.65	0.5

4. Experiment design

For conducting the experiments, we designed five scenarios, based on the research questions. In scenario1, we collected all the product metrics after data-processing and normalization. This is called "Simple model". The detail of selected metrics is shown in Table 3. In scenario-2: the "Advanced model-1" is constructed by using product metrics and one process metric, i.e., Product + NR metric. Similarly, in scenario-3: "Advanced model-2" is formed by using Product + NDC metric, scenario-4 is constructed by using "Advanced model-3" using Product + NML metric and in scenario-5 "Advanced model-4" is built by using with Product + NDPV metric. All the designed models are tested on various project datasets repositories, i.e., Promise, Bug, and Jira using different classifiers such as DT, MLP, SVM, RT, NB and classifiers ensembles, as discussed in Section 3.3, respectively. The performance of various models "Simple model;" "Advanced model-1;" "Advanced model-2;" "Advanced model-3," and "Advanced model-4" are measured using accuracy, RMSE, ROC(AUC), and F-score.

The metrics used in the base classifiers are obtained after performing feature selection and feature ranking. N-fold cross-validation technique [51, 52] is used to evaluate the performance of the base classifiers, which makes use of both training and testing. Cross-validation technique splits the dataset into N parts each of which contains an equivalent number of samples in the dataset. While conducting the experiments algorithm is made to run N times; and in each run, training is achieved through $(N - 1)$ parts, and the testing is performed with the leftover part. N fold are usually selected as 10, 20, 30, 40, 50, 60, 70, 80, and 90. Authors tried with 10 fold for the cross-validation. This approach is carried out on different versions of datasets for different base classifiers.

To answer RQ2, i.e., to evaluate and compare the performance of various ensemble methods presented in Section 3, the library of the said algorithms was installed using the pip Python installer, e.g., (*sudo pip install xgboost*) to conduct the experiments. The algorithm packets used in the study are Bagging, AdaBoostM1 (which is the most popular version of boosting), and Voting [67]. Heterogeneous classifier ensembles applied the majority voting method, whereas the homogenous ones applied both bagging and boosting methods. For ensembles with boosting and bootstrap aggregating, the weak learners selected in the study are Decision Stump and REPTree, as these are widely used in literature studies [67–70]. In AdaBoosting, a training set is modified by repeatedly applying a basic learning device, i.e., classifier, under a pre-specified number of iterations. Initially, the training samples are equal in weight, and the first base classifier is trained to test the training set. Thus, at each iteration, a weight is assigned to each instance of the training set, and the weights of misclassified instances are increased so that their chances to be correctly predicted by the new models get increased. The adjusted training set trains the second basic classifier, and this process is repeated until a good learning device is obtained. During bootstrap aggregating, in the training phase, m data sets of the same size are extracted by performing sampling with replacement (bootstrap) from the training set. Therefore, for each data set, a model is trained using a weak classifier. For each instance, the multiple classifiers utilize a majority voting to obtain the classification result in the test phase. Ensembles are designed using voting works by constructing two or more sub-models. Each sub-model gives a prediction, which is pooled either by taking the mean or the mode of the predictions, permitting each sub-model to vote on the possible outcome. The final output is the class label that attains the maximum number of votes from the predictors. Otherwise, the input is rejected, and the classifier ensembles make no prediction. In our case, the base learners for ensemble design chosen are the four best classifiers. From the pool of four base classifiers, all sets of classifiers of size three were chosen to design ensembles committee. This meant that there were a total of four classifier ensembles. The various constituent combinations, so obtained are defined as: VOT-E1 (DT + MLP + RT), VOT-E2 (DT + MLP + SVM), VOT-E3 (MLP + RT + SVM), and VOT-E4 (DT + RT + SVM). The ensembles performance is measured using the same metrics as used for base classifiers discussed in Section 4. Also, to check whether the ensemble design improves the classification performance compared to individual machine learning classifiers, the comparison of the best ensemble, i.e., VOT-E2, is made with other base classifiers in terms of AUC(ROC) values.

To answer RQ3, i.e., whether there exist any statistically significant performance difference among the base classifiers and ensemble classifiers? Author”s tested the following hypothesis using Friedman”s tests and Wilcoxon signed rank tests [71].
H_0*: There is no significant difference between base classifier performance and ensemble classifier performance.*
To answer RQ4, i.e., cost sensitiveness of proposed ensembles, the normalized fault removal cost approach has been used as discussed in Section 3.5. Further, to evaluate the cost sensitiveness of the best ensemble classifier for the misclassification of faults, we predicted

the fault removal cost of the best ensemble, i.e., VOT-E2 strategy, and compared its performance with the best base classifier, i.e., MLP.

5. Results and discussions

The section presents the experimental results and discussions to all the research questions. Results related to examining the predictive capability of advanced models are discussed in Section 5.1 followed by discussion on results based on ensemble design in Section 5.2. Section 5.3 discusses the results related to statistical difference among the base classifiers and ensemble classifiers and Section 5.4 discusses the results related to the cost sensitiveness of the proposed ensembles.

5.1. Results for predictive capability of advanced models

For examining the predictive capability of proposed advanced models, we evaluated the performance of simple model, advanced model-1, advanced model-2, advanced model-3 and advanced model-4 using various base classifiers. For the simple model, the values of accuracy so obtained are presented in Table 6 for all the datasets. Also, the results are compared with [33] for classifiers DT, MLP, RT and NB classifiers for the projects from Promise data set. Similarly, for all models, the values of accuracy are obtained. Table 7 shows the average accuracies of all the base classifiers for simple model, advanced model-1, advanced model-2, advanced model-3, and advanced model-4 with the standard deviation values after ten executions of the classifiers for all the datasets.

For promise dataset the average accuracy for the simple model in MLP is 75%, for Advanced model-1 is 80%, Advanced model-2 is 87%, Advanced model-3 is 85%, and Advanced model-4 is 79%. It is clear from the bar graph that average accuracy for MLP is higher for Advanced model-2, than for Advanced model-3, Advanced model-1 and simple model. From the bar graph Figure 2a it is observed that average accuracy is behaving well with advanced models as compared to simple models. The average accuracy for the DT"s simple model is 74%, for advanced model-1 is 81%, advanced model-2 is 87%, advanced model-3 is 83%, and advanced model-4 is 77%. So, it is clear from the bar graph that the average accuracy for DT is higher for advanced model-2 then for advanced model-3, advanced model-1, and simple model. The average accuracy results achieved for all projects from Promise data set by Decision tree, Random Tree, Naive Bayes and Multilevel Perceptron classifiers for advanced models is 82.4%, 78.25%, 74.75% and 81.75% as compared to 64.58%, 63.83%, 61.17% and 64.54%, reported by Juneja [33]. This shows that advanced models performed better.

As shown in the graph Figure 2b, the average accuracy is behaving well with advanced models compared to a simple model. The average accuracy is high in Camel projects and low for Xerces projects. The average accuracies for various classifiers like SVM, RT, and NB are calculated as shown in Figure 2c to e. The results show that in the advanced model-2, average accuracy for SVM, RT, and NB is 76%, 82%, and 81%, respectively. The model is behaving significantly good as the average accuracy is higher than 0.5. So, from Table 7 and Figures 2a–e, it is clear that the advanced model-2 (Product + NDC metric) is performing better as compared to other models.

For the projects from Bug repository, the results of average accuracy in the case of advanced model-2, for MLP is 86%, for DT is 85% , for SVM is 85%, for RT is 81%and

Table 6. Simple model accuracy with ten-fold for various classifiers

Projects	DT	DT [33]	MLP	MLP [33]	RT	RT [33]	NB	NB [33]	SVM
ant 1.4	77.53%	76.40%	77.99%	77.52%	75.45%	73.3%	67.97%	67.1%	73.23%
ant 1.5	93.88%	94.88%	94.93%	95.90%	90.88%	100	80.54%	80.45%	90.98%
ant 1.6	73.79%	72.93%	78.89%	73.21%	71.11%	69.76%	59.50%	58.4%	70.43%
ant 1.7	77.72%	75.83%	81.02%	75.97%	76.65%	74.56%	61.98%	61.07%	73.69%
camel 1.2	64.30%	64.30%	68.87%	64.43%	66.60%	65.65%	63.99%	62.7%	65.95%
camel 1.4	82.45%	82.45%	87.76%	87.02%	76.43%	79.85%	79.84%	79.9%	79.41%
camel 1.6	79.68%	79.66%	81.05%	80.51%	80.01%	76.70%	74.66%	74.65%	78.56%
jedit 4.0	74.67%	74.12%	74.67%	74.78%	70.00%	73.1%	67.00%	67.3%	73.89%
jedit 4.1	75.32%	75.33%	75.99%	75.85%	69.95%	69%	69.00%	69.5%	71.87%
jedit 4.2	86.10%	86.10%	87.93%	85.98%	81.90%	80%	74.00%	73.3%	81.14%
jedit 4.3	95.12%	95.12%	96.13%	95.73%	89.95%	95.5%	80.00%	80.1%	89.44%
Lucene 2.2	66.98%		69.77%		68.42%		55.00%		67.87%
Lucene 2.4	69.04%		73.27%		70.27%		78.92%		65.63%
synapse 1.1	72.52%	72.53%	72.87%	72.07%	66.12%	66.8%	69.98%	69.8%	64.98%
synapse 1.2	66.40%	66.1%	66.63%	65.62%	64.92%	66.5%	65.89%	66%	68.04%
xalan 2.5	51.43%	51.42%	54.76%	51.76%	48.89%	51%	54.00%	54.87%	50.98%
xalan 2.6	53.89%	53.9%	60.43%	62.43%	53.34%	53%	61.00%	60.09%	50.76%
xalan 2.7	71.29%	71%	71.24%	70%	62.14%	64.2%	55.00%	57%	58.42%
xerecs 1.2	82.41%	83.40%	83.41%	83.4%	79.41%	80.21%	73.45%	73.14%	78.34%
xerecs 1.3	84.55%	84.54%	84.59%	84.5%	82.98%	83%	76.99%	77%	80.88%
xerecs 1.4	57.36%	28.84%	61.52%	61.12%	90.01%	94%	78.92%	78.5%	91.91%
Equinox	74.07%		73.15%		71.91%		71.60%		73.46%
Eclipse-JDT	82.65%		84.35%		81.44%		83.95%		85.06%
Eclipse-PDE	89.3%		85.64%		79.89%		82.77%		84.05%
MYLYN	84.91%		86.36%		81.68%		83.94%		86.84%
ActiveMQ 5.0.0	86.46%		88.09%		82.95%		85.03%		85.56%
derby-10.5.1.1	85.80%		87.88%		83.25%		83.84%		84.02%
Groovy-1	91.47%		91.01%		92.73%		86.84%		91.59%
Hbase-0.94.0	82.43%		88.35%		77.71%		80.07%		81.11%
Hive-0.9.0	80.01%		86.81%		80.15%		82.52%		81.64%
Jruby-1.1	85.49%		90.02%		88.46%		89.09%		84.95%
Wicket-1	95.03%		95.98%		93.12%		93.42%		83.55%

for NB is 80%, respectively. The model is behaving significantly well as average accuracy is higher than 0.5. So, it is clear that the advanced model-2 (Product + NDC metric) performs better than other models.

For the projects from Jira repository, the results of average accuracy for advanced model-2, for MLP is 91%, for DT is 89% , for SVM is 86%, for RT is 87%, NB is 88%, respectively. The model is behaving significantly well as average accuracy is higher than 0.5. So, it is clear that the Advance model-2 (Product + NDC metric) performs better than other models for Jira projects.

After presenting the accuracy-based evaluation, further analysis is conducted to examine the root mean square error for Promise, Bug and Jira dataset repositories. The average RMSE values for the Promise dataset in proposed advanced model-1, advanced model-2, advanced model-3 and advanced model-4 is low as compared to the simple model. Table 8 presents the details of the average RMSE with standard deviation. The Advanced model-2 has the error ratio 0.13, 0.12, 0.18, 0.19, and 0.16 for DT, MLP, RT, NB and SVM which is significantly lower than the simple model.

Table 7. Average accuracy for various models with standard deviation on different classifiers

	Projects	DT	MLP	RT	NB	SVM
PROMISE	Simple	$74 \pm 0.67\%$	$75 \pm 0.99\%$	$71 \pm 0.11\%$	$60 \pm 0.01\%$	$71 \pm 0.72\%$
	Advanced model-1	$81 \pm 0.09\%$	$80 \pm 0.74\%$	$75 \pm 0.37\%$	$73 \pm 0.07\%$	$76 \pm 0.94\%$
	Advanced model-2	$87 \pm 0.01\%$	$87 \pm 0.08\%$	$82 \pm 0.06\%$	$81 \pm 0.30\%$	$76 \pm 0.45\%$
	Advanced model-3	$83 \pm 0.03\%$	$85 \pm 0.16\%$	$79 \pm 0.08\%$	$72 \pm 0.42\%$	$74 \pm 0.36\%$
	Advanced model-4	$77 \pm 0.04\%$	$79 \pm 0.18\%$	$77 \pm 0.05\%$	$73 \pm 0.55\%$	$72 \pm 0.16\%$
BUG Dataset	Simple	$82 \pm 0.28\%$	$82 \pm 0.66\%$	$79 \pm 0.18\%$	$79 \pm 0.86\%$	$82 \pm 0.76\%$
	Advanced model-1	$83 \pm 0.90\%$	$84 \pm 0.70\%$	$80 \pm 0.66\%$	$80 \pm 0.85\%$	$83 \pm 0.59\%$
	Advanced model-2	$85 \pm 0.63\%$	$86 \pm 0.92\%$	$81 \pm 0.95\%$	$80 \pm 0.19\%$	$85 \pm 0.27\%$
	Advanced model-3	$84 \pm 0.14\%$	$84 \pm 0.36\%$	$81 \pm 0.04\%$	$79 \pm 0.59\%$	$84 \pm 0.38\%$
	Advanced model-4	$81 \pm 0.96\%$	$82 \pm 0.76\%$	$79 \pm 0.73\%$	$79 \pm 0.47\%$	$82 \pm 0.96\%$
JIRA	Simple	$86 \pm 0.53\%$	$89 \pm 0.59\%$	$85 \pm 0.34\%$	$85 \pm 0.69\%$	$84 \pm 0.54\%$
	Advanced model-1	$88 \pm 0.15\%$	$91 \pm 0.07\%$	$87 \pm 0.39\%$	$87 \pm 0.64\%$	$85 \pm 0.45\%$
	Advanced model-2	$89 \pm 0.76\%$	$91 \pm 0.85\%$	$87 \pm 0.32\%$	$88 \pm 0.32\%$	$86 \pm 0.18\%$
	Advanced model-3	$88 \pm 0.75\%$	$90 \pm 0.65\%$	$87 \pm 0.44\%$	$87 \pm 0.72\%$	$86 \pm 0.05\%$
	Advanced model-4	$86 \pm 0.89\%$	$89 \pm 0.72\%$	$85 \pm 0.97\%$	$86 \pm 0.30\%$	$84 \pm 0.81\%$

For Bug dataset the average RMSE values for proposed advanced model-2, and advanced model-3 are significantly lower than the advanced model-4, Advance model-1 and simple model. The average RMSE values for DT, MLP, RT, NB and SVM are 0.18, 0.15, 0.16, 0.17 and 0.13, respectively for advanced model-2.

For Jira dataset the average RMSE values for proposed advance model-2 are significantly lower than the advanced model-3, advanced model-4, advanced model-1 and simple model. The RMSE values for advanced model-1 and advanced model-3 are almost similar for DT, MLP and SVM. The average RMSE values for DT, MLP, RT, NB and SVM are 0.18, 0.12, 0.15, 0.15 and 0.14, respectively for advanced model-2.

AUC is the other performance measure considered in the study. Greater the AUC value better is the model performance [7, 20]. The ROC curves provide the trade-off between the TPR and FPR for a predictive model using different probability thresholds. These measures are good performance indicator for the classification of an imbalanced dataset.

Table 9 shows the aggregative AUC of all the base classifiers for simple, advanced model-1, advanced model-2, advanced model-3 and advanced model-4 with the standard deviation values after ten executions of the classifiers. The aggregative average AUC for Promise dataset achieved in advanced model-2 are 76%, 79%, 70%, 65% and 75% for DT, MLP, RT, NB and SVM respectively. As evident from literature studies [7] that the AUC value lying between 0.7 and 1 is considered significantly high and the accuracy value lying

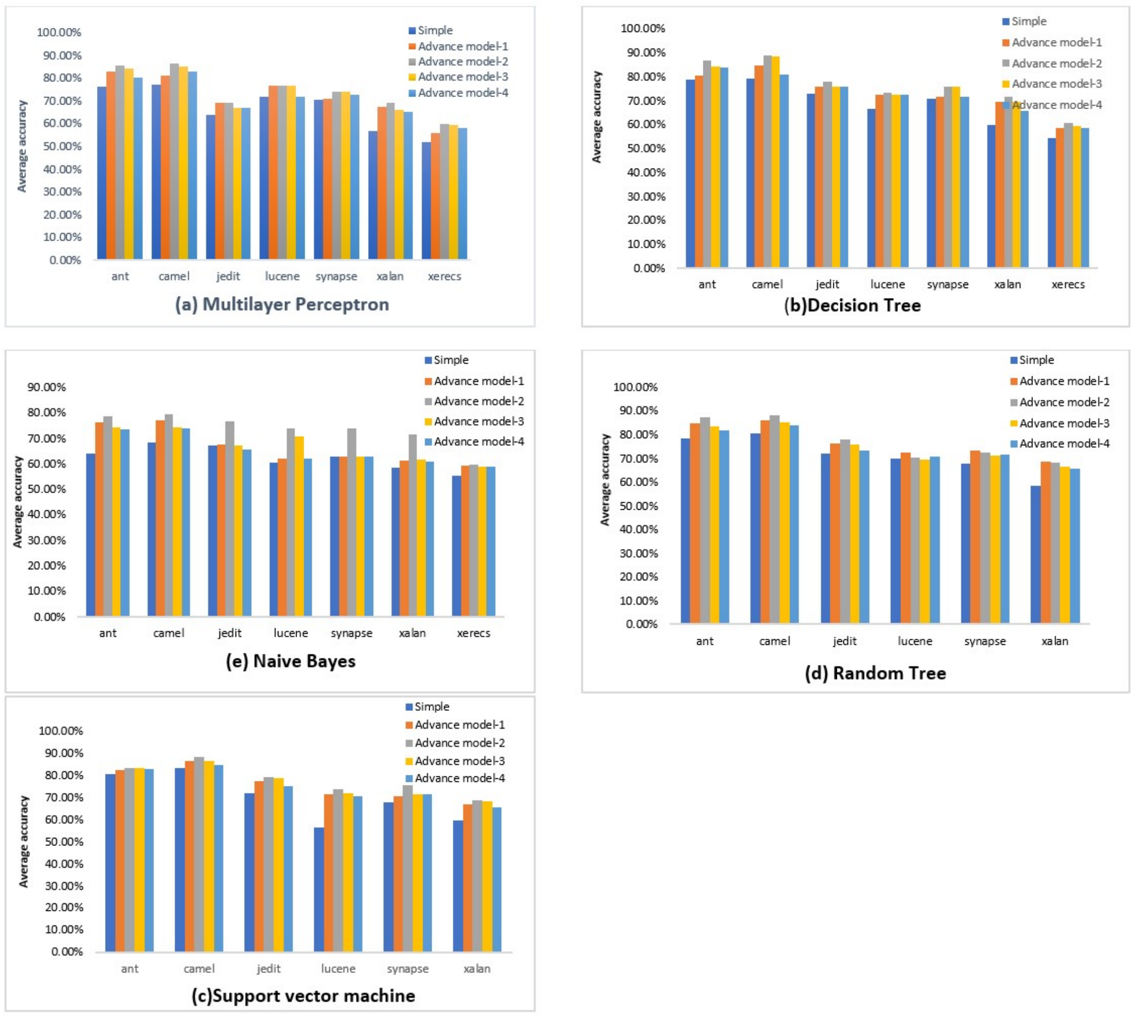

Figure 2. Average Accuracy for MLP, DT, SVM, RT, and NB using PROMISE data

between 0.6 and 0.7 is considered significantly good. It is evident from the Table 9 that the advanced model-2 achieves and maintains high accuracy with respect to all classifiers. The advanced model-3 is followed by advanced model-1and advanced model-4.

For the Bug dataset the average ROC values for the advanced model-2 and advanced model-3 are significantly higher and for advanced model-1and advanced model-4 the average ROC values are good as compared to simple model as shown in Table 9. The average accuracy of advanced model-2 for DT, MLP, RT, NB, and SVM are 79%, 83%, 74%, 70%, and 78%, respectively. It shows that the proposed advanced models has performed impressively well for inter project fault prediction.

We also calculated the performance of simple and advanced models in terms of F-score. F-score values can range from (0–1) and accepted to be better as it approaches to one [41, 60]. Table 10 presents the average F-score for simple and advanced models for Promise, Bug, and Jira datasets on various classifiers. It is observed from Table 10; in the Promise dataset the advanced model-2 for MLP classifier has the highest $F1$-score value, i.e., 0.83 as compared to the advanced model-3 (0.82), advanced model-2 (0.80), and the simple model (0.79). The F-score value for DT in simple model (0.76), advanced model-1 (0.77), advanced model-2 (0.81), the advanced model-3 (0.80) and the advanced model-4 is (0.76).

Table 8. Average RMSE for various models with standard deviation on different classifiers

	Projects	DT	MLP	RT	NB	SVM
PROMISE	Simple	0.21 ± 0.0057	0.19 ± 0.006	0.21 ± 0.0101	0.20 ± 0.0059	0.17 ± 0.0089
	Advanced model-1	0.16 ± 0.0067	0.14 ± 0.007	0.19 ± 0.0090	0.18 ± 0.006	0.17 ± 0.0067
	Advanced model-2	0.13 ± 0.007	0.12 ± 0.005	0.18 ± 0.0398	0.19 ± 0.046	0.16 ± 0.009
	Advanced model-3	0.15 ± 0.007	0.14 ± 0.008	0.19 ± 0.0256	0.18 ± 0.025	0.15 ± 0.011
	Advanced model-4	0.16 ± 0.006	0.16 ± 0.005	0.19 ± 0.0006	0.19 ± 0.005	0.17 ± 0.012
BUG Dataset	Simple	0.22 ± 0.084	0.20 ± 0.0127	0.21 ± 0.0317	0.21 ± 0.0997	0.17 ± 0.038
	Advanced model-1	0.21 ± 0.067	0.18 ± 0.055	0.19 ± 0.0672	0.21 ± 0.0302	0.15 ± 0.037
	Advanced model-2	0.18 ± 0.09	0.15 ± 0.0545	0.16 ± 0.0995	0.17 ± 0.0997	0.13 ± 0.035
	Advanced model-3	0.18 ± 0.075	0.16 ± 0.0902	0.19 ± 0.0215	0.19 ± 0.0615	0.13 ± 0.068
	Advanced model-4	0.20 ± 0.0387	0.18 ± 0.0857	0.20 ± 0.0727	0.20 ± 0.0382	0.16 ± 0.052
JIRA	Simple	0.20 ± 0.099	0.15 ± 0.0395	0.19 ± 0.0265	0.18 ± 0.0951	0.17 ± 0.048
	Advanced model-1	0.19 ± 0.088	0.13 ± 0.0757	0.17 ± 0.0742	0.17 ± 0.048	0.15 ± 0.097
	Advanced model-2	0.18 ± 0.037	0.12 ± 0.0108	0.15 ± 0.085	0.15 ± 0.085	0.14 ± 0.019
	Advanced model-3	0.19 ± 0.014	0.13 ± 0.0982	0.16 ± 0.0334	0.16 ± 0.095	0.15 ± 0.067
	Advanced model-4	0.20 ± 0.038	0.14 ± 0.0721	0.18 ± 0.0295	0.17 ± 0.0938	0.16 ± 0.0308

It is observed from the table that the advanced model-2 is behaving significantly well in all the classifiers. The MLP is behaving well than DT, DT is better than RT, RT is better than SVM, and SVM is better than NB.

Similarly, for the Bug dataset the advanced model-2 and the advanced model-3 have almost similar values for MLP and NB classifiers. The advanced model-2 having F-score MLP (0.88), DT (0.84), RT (0.84), NB (0.81) and SVM (0.85). For the simple model the F-score values are 0.82, 0.86, 0.81, 0.79, 0.78 for DT, MLP, RT, NB, and SVM classifiers, respectively. Similarly, for advanced model-1 the DT, MLP, RT, NB and SVM the values for F-score are 0.82, 0.87, 0.82, 0.80 and 0.82, respectively and; for the advanced model-3 and for the advanced model-4 the F-score values are 0.83, 0.88, 0.84, 0.81, 0.85 and 0.81, 0.87, 0.81, 0.79, 0.79, respectively.

For the Jira dataset, the average F-score has the highest values for advanced model-2 compared to advanced model-1, advanced model-3, advanced model-4, and simple model. The advanced model-2 has F-score values for MLP (0.89), DT (0.83), RT (0.82), NB (0.80) and SVM (0.83), respectively. The advanced model-2 with MLP has the highest F-score values as compared to other classifiers. Thus, it is observed that MLP performs best with values 0.83, 0.88, and 0.89 with advanced model-2 for Promise, Bug, and Jira datasets. Comparing the overall performance, the advanced model-2 with MLP performs best followed by advanced model-3.

So, it is concluded for the RQ1 that Advanced model-2 with MLP classifier having high predictive capability as compared to other models and classifiers. The advanced model-2 with MLP has high accuracy, ROC (AUC) and *F*-score, and small RMSE values.

5.2. Experiment results based on ensemble design

In this section, we have summarised the results and discussed the main findings of various ensemble methods. Table 11 presents the results for average Accuracy, average RMSE, average ROC(AUC), and average *F*-score. Diagrammatically, the results of the performance measures are shown with the help of Box plots. Figures 3a–l shows the box plot analysis results for proposed ensembles with respect to the average Accuracy, average AUC(ROC), average *F*-score, and average RMSE. The different regions of box plots in the figures present the maximum, median minimum, first quartile, and third quartile values of the dataset. The middle line of the box indicates the median value of the dataset. With respect to average accuracy, average AUC(ROC), average *F*-score, and average RMSE, the proposed ensembles VOT-E1 and VOT-E2 have high median value and high maximum value followed by AdaBoost and Random Forest with features based on Product + NDC metric data set for projects not only from Promise repository but also from Bug and Jira dataset repositories, which validates the results and makes the approach more reliable.

Table 9. Average ROC(AUC) for various Models with standard deviation on different classifiers

	Projects	DT	MLP	RT	NB	SVM
PROMISE	Simple	71 ± 0.009	74 ± 0.006	63 ± 0.014	61 ± 0.02	70 ± 0.014
	Advanced model-1	75 ± 0.001	78 ± 0.006	67 ± 0.017	63 ± 0.09	74 ± 0.004
	Advanced model-2	76 ± 0.002	79 ± 0.008	70 ± 0.001	65 ± 0.98	75 ± 0.009
	Advanced model-3	77 ± 0.012	78 ± 0.012	69 ± 0.019	62 ± 0.06	73 ± 0.009
	Advanced model-4	70 ± 0.013	77 ± 0.002	65 ± 0.016	60 ± 0.07	69 ± 0.008
BUG Dataset	Simple	73 ± 0.09025	77 ± 0.5775	66 ± 0.05	63 ± 0.955	74 ± 0.2425
	Advanced model-1	76 ± 0.058	81 ± 0.04	71 ± 0.37	67 ± 0.845	77 ± 0.0725
	Advanced model-2	79 ± 0.03375	83 ± 0.14	74 ± 0.7375	70 ± 0.5325	78 ± 0.1825
	Advanced model-3	78 ± 0.069	81 ± 0.0785	73 ± 0.515	72 ± 0.4375	76 ± 0.7725
	Advanced model-4	74 ± 0.079	77 ± 0.0665	67 ± 0.2975	66 ± 0.2475	75 ± 0.2325
JIRA	Simple	76 ± 0.0628	82 ± 0.11	72 ± 0.62857	81 ± 0.181	74 ± 0.7142
	Advanced model-1	78 ± 0.0732	83 ± 0.051571	74 ± 0.39429	83 ± 0.11	76 ± 0.5485
	Advanced model-2	79 ± 0.01429	84 ± 0.046	76 ± 0.27143	83 ± 0.82	77 ± 0.9171
	Advanced model-3	78 ± 0.0986	83 ± 0.0351	74 ± 0.92429	82 ± 0.75571	76 ± 0.8457
	Advanced model-4	77 ± 0.06	82 ± 0.0514	73 ± 0.88571	81 ± 0.97143	75 ± 0.7271

From Figure 3a, it is observed that projects from Promise repository, VOT-E1 have the highest accuracy, i.e., 0.8922, and high median value, i.e., 0.8906.

Similarly, the box plot for VOT-E2 shows the median value of 0.8797 and a maximum value of 0.8814. Similar trends are observed for projects from Bug and Jira repositories, as shown in Figure 3b and Figure 3c, respectively. Thus, VOT-E1 performs better in terms of accuracy as compared to other ensembles.

The average AUC(ROC) values are shown in Figures 3d–f. From Figure 3d, it is observed that for projects from Promise repository, VOT-E2 have the highest AUC(ROC), i.e., 0.8392, and high median value, i.e., 0. 8402 followed by VOT-E1 with median value 0.7972 and a maximum value 0.8011. The values of ensembles constructed with Boosting and Bagging are 0.7851 and 0.7781, respectively. Similar trends are observed for Bug and Jira repositories projects as shown in Figures 3e and 3f. VOT-E2 performs better in terms of accuracy as compared to other ensembles. We also found that the difference in AUC(ROC) for the best performing ensembles, i.e., VOT-E1 and VOT-E2 is very minimal, ranging from 1% to 3%. Moreover, the shape of the box-plot for the projects is nipped, which signifies that the performance of the ensemble method is the same among all the releases of different projects from Promise, Jira and Bug repositories.

The performance for the average F-score is shown in Figure 3g–i, respectively. From Figure 3g, it is observed that for projects from Promise repository, VOT-E2 have the highest

Table 10. Average F-score for various models with standard deviation on different classifiers

	Projects	**DT**	**MLP**	**RT**	**NB**	**SVM**
PROMISE	Simple	0.76 ± 0.012	0.79 ± 0.012	0.79 ± 0.03	0.73 ± 0.06	0.75 ± 0.05
	Advanced model-1	0.77 ± 0.04	0.80 ± 0.18	0.77 ± 0.16	0.74 ± 0.017	0.76 ± 0.04
	Advanced model-2	0.81 ± 0.05	0.83 ± 0.05	0.79 ± 0.07	0.78 ± 0.09	0.79 ± 0.09
	Advanced model-3	0.80 ± 0.03	0.82 ± 0.16	0.78 ± 0.078	0.79 ± 0.02	0.78 ± 0.06
	Advanced model-4	0.76 ± 0.05	0.79 ± 0.06	0.75 ± 0.04	0.74 ± 0.053	0.74 ± 0.08
BUG Dataset	Simple	0.82 ± 0.13	0.86 ± 0.07	0.81 ± 0.024	0.79 ± 0.01	0.78 ± 0.05
	Advanced model-1	0.82 ± 0.045	0.87 ± 0.1	0.82 ± 0.05	0.80 ± 0.023	0.82 ± 0.09
	Advanced model-2	0.84 ± 0.063	0.88 ± 0.12	0.84 ± 0.063	0.81 ± 0.05	0.85 ± 0.02
	Advanced model-3	0.83 ± 0.05	0.88 ± 0.05	0.83 ± 0.045	0.82 ± 0.06	0.85 ± 0.07
	Advanced model-4	0.81 ± 0.09	0.87 ± 0.03	0.81 ± 0.098	0.79 ± 0.07	0.79 ± 0.04
JIRA	Simple	0.81 ± 0.045	0.83 ± 0.061	0.79 ± 0.07	0.79 ± 0.09	0.79 ± 0.06
	Advanced model-1	0.82 ± 0.08	0.84 ± 0.07	0.80 ± 0.023	0.79 ± 0.06	0.80 ± 0.078
	Advanced model-2	0.83 ± 0.045	0.89 ± 0.01	0.82 ± 0.06	0.80 ± 0.05	0.83 ± 0.087
	Advanced model-3	0.83 ± 0.063	0.88 ± 0.08	0.81 ± 0.05	0.80 ± 0.16	0.83 ± 0.04
	Advanced model-4	0.81 ± 0.092	0.78 ± 0.065	0.79 ± 0.07	0.76 ± 0.01	0.81 ± 0.045

F-score, i.e., 0.8329 followed by VOT-E3 0.8201, AdaBoost 0.8068, and Random Forest 0.7864. The box plot for VOT-E3 shows the median value of 0.8134 and the maximum value of 0.8201. Similar tendency is observed for F-score values for projects from Bug and Jira repositories as shown in Figures 3h and 3i, where VOT-E2 has the highest F-score value followed by VOT-E1. The overall median of F-score ranges between 85% (VOT-E2) and 77.71% (Bagging). So, VOT-E2 performs better in terms of F-score as compared to other ensembles.

The average root mean square error values are shown in Figure 3j–l. From Figure 3j, it is observed that for the projects from Promise repository, the average RMSE is least for the ensembles VOT-E1, i.e., 0.1693 and VOT-E2, i.e., 0.1724 and high for VOT-E4 0.199, AdaBoost 0.1785, and Random Forest 0.1966. The VOT-E1 and VOT-E2 performed better, and it has the narrow box compared to the ensembles with bagging, adaboost, and RF, respectively. The overall difference of RMSE ranges between 0.1703 (VOT- E1) and 0.1998 (VOT- E4).

5.3. Results for examining the performance difference among the base classifiers and ensemble classifiers

The results shown in Table 12 in bold indicate the best performance. The proposed VOT-E2 produced the best results with advanced model-2, advanced model-3 and advanced model-1, while it gave second best results for advanced model-4 and simple model datasets. The values of ROC(AUC) for base classifiers were taken from Table 9. For comparison of classifiers, the average ranks were computed. The ranks for each classifier for each dataset were ascertained and later on summed up to get average ranks by dividing the average values by the number of datasets. The lower the average ranking value; the better is the performance of the model. The proposed ensemble-1 has a lower average rank of 1.2,

Table 11. Ensemble results for average Accuracy, average RMSE, average ROC(AUC), and F-score

Datasets	Bag	ADA	RF	VOT-E1	VOT-E2	VOT-E3	VOT-E4
Average accuracy							
PROMISE Dataset	87.35%	88.07%	87.41%	89.22%	88.14%	86.30%	82.94%
BUG Dataset	87.69%	88.02%	86.79%	89.15%	88.70%	87.37%	85.08%
JIRA Dataset	90.19%	90.26%	88.89%	91.25%	90.47%	89.15%	88.45%
Average ROC (AUC)							
PROMISE Dataset	78.51%	77.81%	79.32%	80.11%	83.92%	79.58%	77.17%
BUG Dataset	77.09%	79.23%	80.07%	80.53%	81.64%	79.25%	78.97%
JIRA Dataset	78.18%	79.01%	78.19%	81.60%	84.05%	79.82%	78.19%
Average F1 score							
PROMISE Dataset	77.91%	80.68%	78.64%	80.37%	83.29%	82.01%	78.54%
BUG Dataset	78.62%	81.12%	78.86%	85.09%	87.34%	84.10%	78.72%
JIRA Dataset	76.50%	80.09%	76.94%	85.26%	87.24%	81.06%	79.80%
Average RMSE							
PROMISE Dataset	0.1845	0.1785	0.1966	0.1693	0.1724	0.1843	0.199
BUG Dataset	0.1855	0.1765	0.20195	0.181615	0.18525	0.2036	0.2014
JIRA Dataset	0.170	0.1785	0.1966	0.1601	0.1679	0.1748	0.1992

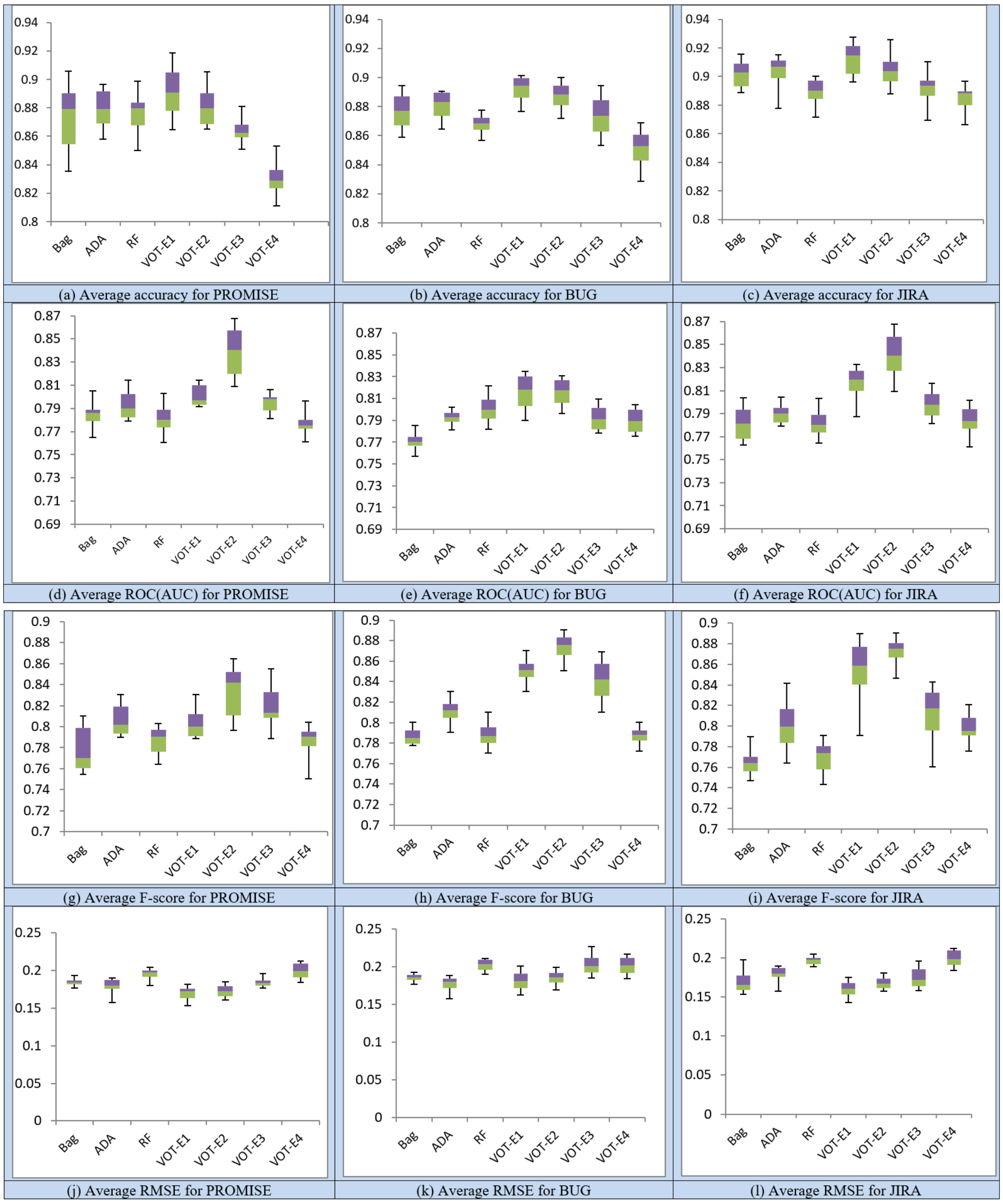

Figure 3. Box plots for Ensemble Results for average accuracy, average RMSE, average ROC(AUC) and average *F*-score

followed by the classifier MLP with a rank 1.6. All other classifiers have ranks between 1.2 and 5.8 as shown in Table 13.

Thus, based upon the results, we can say that ensemble learning (i.e., AdaBoosting, Bagging, Random Forests, and Voting) works best as compared to base predictors. The ensemble algorithms combine signals from base classifiers in the committee to produce an enhanced fault prediction algorithm. While experimenting, we have noticed that ensemble

Table 12. Comparisons of different classifiers in terms of ROC(AUC)

Classifiers	Simple	Advanced model-1	Advanced model-2	Advanced model-3	Advanced model-4	Avg Rank
DT	0.7338	0.7637	0.7801	0.7772	0.7371	3
MLP	0.7789	**0.8069**	**0.8206**	**0.8071**	**0.7871**	1.8
SVM	0.7029	0.7587	0.7703	0.7554	0.7332	4.0
RT	0.6723	0.7092	0.7366	0.7248	0.6873	5.8
NB	0.6871	0.7135	**0.740**	0.7241	0.6942	5.2
VOT-E2	**0.7801**	**0.8182**	**0.8320**	**0.8156**	0.7780	1.2

Table 13. Friedman test comparison

	J (base classifiers)	Pair wise differences
VOT- E2	DT	1.6 ($P < 0.01$)
	MLP	0.4
	SVM	2.6 ($P < 0.001$)
	RT	3.5 ($P < 0.001$)
	NB	4.6 ($P < 0.001$)

techniques performed better among all the advanced models. With respect to average accuracy, average AUC(ROC), average F-score, and average RMSE, the proposed ensembles VOT-E1 and VOT-E2 have high median value and high maximum value followed by AdaBoost and Random Forest with features based on (Product + NDC metric data set) for projects not only from Promise repository but also from Bug and Jira dataset repositories, which validates the results and makes the approach more reliable. In general, the ensemble methods show an overall median of F1 score ranging between 76.50% and 87.34% and the ROC (AUC) between 77.09% and 84.05%. Base classifiers instead, reach an overall average F-score ranging between 73% (simple model) and 83% (Advanced model-2) for Promise data set and the ROC(AUC) between 60% (Advanced model-4) and 79% (Advanced model-2). Thus, we can say that the ensemble design enhances the strengths of multiple predictors and supplements to state of art in fault prediction problem [35, 72].

Furthermore, to examine whether the measured average ranks are significantly different from the mean rank 3.5, the Friedman test has been applied. The results of the test show below the significance level ($p < 0.01$), which means that at least two of the predictors are significantly different from each other. When the scores differ significantly, the researchers in the literature recommended follow-up pair-wise comparisons [68, 70]. For pair-wise comparisons, Wilcoxon signed ranks test had been applied. The results of pair-wise comparisons are presented in Table 13. It is observed that the performance of ensemble classifier is considerably dissimilar than other classifiers, apart from the MLP based classifier. Thus, the null hypothesis is rejected, which states that there is no significant difference between base classifiers performance and ensemble classifier performance. The results of Friedman"s tests and Wilcoxon signed rank tests illustrate that the ensemble method exhibits statistically significant performance differences.

5.4. Results for examining the cost sensitiveness

Table 14 presents the predicted values of estimated fault removal cost (E_{cost}) for various projects for both the best ensemble and the best base classifier. The unit, integration, system, and field-testing values are used to obtain estimated values for fault removal cost using Equations (1) to (4) presented in Section 3.5.1. Table 15 shows the estimated testing cost (T_{cost}) values for various projects. These cost values are obtained using equations (6) to (9) presented in Section 3.5.1. The values of fault removal cost (staff hours per defect) for test techniques are shown in Table 4, and values of testing phase efficiencies are presented in Table 5 in Section 3.5. Further, normalized cost values (N_{cost}) of VOT-E1 ensemble and best base classifier for projects from datasets (Promise, Bug and Jira)

Table 14. Estimated Fault Removal cost (E_{cost}) for best base classifier and best ensemble classifier

Projects	Best base classifier (MLP)			VOT-E2 ensemble classifier		
	Min	Max	Median	Min	Max	Median
ant	1042	3862	1852	935	3712	2320
camel	534	2144	1167	654	2356	1367
Jedit	102	147	276	69	155	287
lucene	854	2393	3000	844	2382	2976
synapse	400	1018	1650	444	968	1601
xalan	3506	9803	11653	3498	9800	11650
xerecs	777	3912	1821	662	2461	1641
Equinox	596.06	2296.8	1502.42	562	2283.43	1407.7
Eclipse-JDT	921.753	4037.02	2419.48	918.45	4041.12	2389.04
Eclipse-PDE	1726.65	6565.25	3711.75	1710.12	6500.98	3645
MYLYN	2124.93	7548.55	4361.03	2023.34	7481.16	4243.03
ActiveMQ 5.0.0	1390.19	6510.64	3776.14	1390	6511	3768
derby-10.5.1.1	2041.09	9432.96	5607.88	2100	9456	5704
Groovy1_6_BETA_1	336.57	1309.85	857.90	321.34	1296	823
Hbase-0.94.0	1039.84	4907.87	2835.09	1109	4987	2965
Hive-0.9.0	1286.61	5661.904	3392.029	1261	5673	3753
Jruby-1.1	376.61	1547.42	962.66	376	1654	997
Wicket-1.3.0-beta2	1129.77	5998.59	3519.73	1127	5974	3678

Table 15. Estimated Testing cost (T_{cost}) for various projects

Projects from PROMISE repository										
	Unit	Integ.	System	Ant	Camel	Jedit	Lucene	Synapse	Xalan	Xerecs
Min	0.1	0.25	0.25	1079.78	1313.84	403.53	892.17	461.93	3500.35	1812.63
Max	0.5	0.6	0.65	3913.59	4796.06	1587.23	3072.74	1637.63	11807.5	6128.94
Median	0.25	0.45	0.5	2322.85	2782.28	707.21	2126.17	1040.95	8664.3	4398.43
Projects from BUG repository										
	Unit	Integ.	System	Equinox	Eclipse-JDT	Eclipse-PDE	MYLYN			
Min	0.1	0.25	0.25	647.89	1394.33	1778.74	2165.49			
Max	0.5	0.6	0.65	2276.44	5074.07	6604.41	8063.44			
Median	0.25	0.45	0.5	1486.42	2973.17	3623.32	4381.36			
Projects from Zira repository										
	Unit	Integ.	System	ActiveMQ	derby	Groovy1	Hbase	Hive	Jruby	*Wicket*
Min	0.1	0.25	0.25	2332.65	3230.89	835.46	1478.49	1950.26	821.32	1730.28
Max	0.5	0.6	0.65	8614.81	11987.9	3170.84	5381.41	7109.69	3072.74	6603.56
Median	0.25	0.45	0.5	4811.25	6591.98	1613.06	3151.27	4142.42	1643.08	3293.55

Table 16. Normalized fault removal cost (NE_{cost})

Projects	Best base classifier (MLP)			VOT-E2 ensemble classifier		
	Min	Max	Median	Min	Max	Median
ant	0.96	0.98	0.79	0.86	0.84	0.99
camel	0.40	0.44	0.42	0.49	0.50	0.49
jedit	0.25	0.21	0.17	0.17	0.21	0.18
lucene	0.95	1.12	0.97	0.94	1.12	0.97
synapse	0.87	0.98	1.0	0.96	0.93	0.98
xalan	1.0	1.13	0.98	0.99	1.13	0.98
xerecs	0.72	0.99	0.78	0.61	0.62	0.70
Equinox	0.75	0.82	0.79	0.86	1.00	0.94
Eclipse-JDT	0.66	0.79	0.81	0.65	0.79	0.80
Eclipse-PDE	0.56	0.60	0.71	0.96	0.98	1.0
MYLYN	0.62	0.72	0.83	0.93	0.92	0.96
ActiveMQ 5.0.0	0.59	0.75	0.78	0.59	0.75	0.78
derby-10.5.1.1	0.63	0.78	0.85	0.64	0.78	0.86
Groovy-1_6_BETA_1	0.40	0.41	0.53	0.38	0.40	0.51
Hbase-0.94.0	0.70	0.91	0.89	0.75	0.92	0.94
Hive-0.9.0	0.65	0.79	0.81	0.64	0.79	0.90
Jruby-1.1	0.45	0.50	0.58	0.45	0.53	0.60
Wicket-1.3.0beta2	0.65	0.90	1.06	0.65	0.90	1.11

are obtained using Equation (11). The values so obtained are presented in Table 16. The values > 1.0 show that the proposed best ensemble, i.e., VOT-E1 is cost-effective. It entails that if the results of fault prediction are used with software testing, then overall testing cost and effort can be saved. At the same time, values greater than 1.0 demonstrate that the results of fault prediction do not help save overall testing cost and effort, and thus, it is suggested not to use fault prediction models in such cases. From the results presented in Table 16, it can be observed that for almost all projects, i.e., Ant, Camel, Jedit, Synapse, Xerecs, Equinox, Eclipse JDT, EclipsePDE, MYLYN, ActiveMQ 5.0.0, derby 10.5.1.1, Groovy-1_6_BETA1, Hbase-0.94.0, Hive-0.9.0 and Jruby-1.1 from the Promise, Bug and Jira repositories, N_{cost} values are lower or equal to the threshold value, i.e., 1.0 for both the proposed ensemble VOT-E2 and best base classifier MLP except in few cases, i.e., Xalan, Lucene and Wicket-1.3.0beta2 the normalized cost values are more than threshold value. Therefore, as observed from the results, it may not be beneficial to make use of SFP based on the suggested best ensemble and best base classifier. Thus, it is advisable to test all the modules at the unit level in place of using predictor for defect prediction for such projects. For all other datasets, the values of N_{cost} are lower than the threshold value, i.e., 1 and thus it is advantageous to utilize fault prediction approaches proposed in the study.

Table 17 presents the summary of research questions. Also, comparison of few related studies in literature with the proposed study is provided in Table 18.

6. Threats to validity

The section presents discussion on possible validity threats of the work presented in the paper along with possible measures how we mitigated them.

Construct validity: These types of threats are concerned with the relationship between theory and observations. In this work, we built advanced models for defect prediction

Table 17. Summary of research questions

Research question	Discussion
RQ1: How does the advanced defect prediction models proposed in the study perform using various machine learning classifiers?	For conducting the experiments, we designed five scenarios, i.e., "Simple model"; "Advanced model-1" (Product + NR metric); "Advanced model-2" (Product + NDC metric); "Advanced model-3" (Product + NML metric); and "Advanced model-4" (Product + NDPV metric), respectively. All the designed models are tested on various project datasets repositories, i.e., PROMISE, BUG and JIRA using different classifiers such as DT, MLP, SVM, RT, NB. The advanced model-2 with MLP classifier having high predictive capability followed by advanced model-3. Results discussed in Section 5.1 shows that the proposed advanced models have performed impressively well for inter project fault prediction.
RQ2: How does the ensemble design improve classification performance when compared to individual machine learning classifiers?	In general, the ensemble methods show an overall median of *F*-score ranging between 76.50% and 87.34% and the ROC (AUC) between 77.09% and84.05%. Base classifiers instead, reach an overall average *F*-score ranging between 73% (simple model) and 83% (Advanced model-2) for PROMISE data set and the ROC (AUC) between 60% (Advanced model-4) and 79% (Advanced model-2). Thus, we can say that the ensemble design enhances the strengths of multiple predictors and supplements to state of art in fault prediction problem. While experimenting, we have noticed that ensemble techniques (i.e., AdaBoosting, Bagging, Random Forests, and Voting) performed better among all the advanced models as discussed in Section 5.2. With respect to average accuracy, average AUC (ROC), average *F*-score, and average RMSE, the proposed ensembles VOT-E1 and VOT-E2 have high median value and high maximum value followed by AdaBoost and Random Forest with features based on (Product + NDC) metric data set for projects not only from PROMISE repository but also from BUG and JIRA dataset repositories, which validates the results and makes the approach more reliable.
RQ3: Whether there exist any statistically significant performance difference among the base classifiers and ensemble classifiers?	For pair-wise comparisons, Wilcoxon signed ranks test had been applied. It is observed from the results (Table 13 in Section 5.3) that the performance of ensemble classifier is considerably dissimilar than other classifiers, apart from the MLP based classifier. The results of Friedman"s tests and Wilcoxon signed rank tests illustrate that the ensemble method exhibits statistically significant performance differences.
RQ4: For a given software system, whether the proposed ensembles are cost sensitive?	From the results presented in Table 16 Section 5.4, it is observed that for almost all projects, i.e., Ant, Camel, Jedit, Synapse, Xerecs, Equinox, Eclipse JDT, Eclipse PDE, MYLYN, ActiveMQ5.0.0, derby 10.5.1.1, Groovy-1 _6 _BETA _1, Hbase-0.94.0, Hive-0.9.0 and Jruby-1.1 from the PROMISE, BUG and Zira repositories, Ncost values are lower or equal to the threshold value, i.e. 1.0 for both the proposed ensemble VOT-E2 and best base classifier MLP except in few cases, i.e. Xalan, Lucene and Wicket-1.3.0beta2 the normalized cost values are more than threshold value. Thus, for a given software system, the proposed ensembles are cost sensitive.

Table 18. Comparision summary of research techniques

Authors	Study objective	Software fault data sets used/ repository	Fault prediction techniques	Results
Song et al. [2]	Proposed and evaluated a general framework for software defect prediction	NASA, MDP and AR Data Sets	Three learning algorithms. Naive Bayes (NB), J48, one R and 12 learning schemes	Naive Bayes performs much better than J48, and J48 is better than OneR No learning scheme dominates, i.e., always outperforms the others for all 17 data sets.
Rathore and Kumar [26]	Empirical study of ensemble techniques for software fault prediction	28 software fault datasets (PROMISE repository)	7 ensemble techniques, i.e., Dagging, Decorate, Grading, MultiBoostAB, RealAdaBoost, Rotation Forest, and Ensemble Selection	Precision = 0.995 (Rotation Forest) Recall = 0.994 (Rotation Forest) AUC = 0.986 (Decorate) Cost-sensitiveness: proposed ensemble techniques saved software testing cost and effort for 20 out of 28 fault datasets.
Laradji et al. [27]	To demonstrate the positive effects of combining feature selection and ensemble learning on the performance of defect classification.	06 datasets: Ant-1.7, Camel-1.6, KC3, MC1, PC2, and PC4	Average probability ensemble (APE) consisting of 7 classifiers, i.e., random forests, gradient boosting, stochastic gradient descent, W-SVMs, logistic regression, multinomial naive Bayes, and Bernoulli naive Bayes	Higher AUC measures for each dataset, which were close to 1.0 in the case of PC2, PC4 and MC1 datasets.
Proposed approach	Study aims to develop advanced models for software defect prediction which uses both product and process metrics.	32 projects from PROMISE, BUG, and JIRA dataset repository.	5 Base learners and ensemble methods (i.e., AdaBoosting, Bagging, Random Forests, and Voting)	Ensemble methods: F-score (76.50% −87.34%) and the ROC (AUC) (77.09% −84.05%) for Product + NDC metric data for all data sets. Cost-sensitiveness: VOT-E2 ensemble saved software testing cost and effort for 29 out of 32 fault datasets.

using product and process metrics. To improve the quality of software datasets, we applied dimensional reduction, which is achieved by using feature ranking and feature subset selection techniques [7]. In this way, we obtained a reduced set of 15 features for defect prediction. Thus, data preprocessing helps to avoid the creation of an unstable model [30]. The study results are replicated with various datasets from Promise, Bug and Jira repositories which makes the study reliable. We have used N fold cross-validation while conducting experiments to avoid bias due to sampling. To obtain the experimentation results, authors in the study used F-score and ROC(AUC) metrics which are considered to be more consistent measures for evaluation of classification algorithms [26].

Conclusion validity: It denotes the relation between treatment and outcome. During the study, our objective was to examine the overall predictive capability of proposed advanced models using various machine learning classifiers. We also examined whether the ensemble design improves classification performance as compared to base machine learning classifiers. For this, we designed ensembles using bagging, boosting and voting to examine the improvement in the performance of proposed defect prediction models. The performance of ensembles is measured by constructing box plots for accuracy, precision and AUC(ROC). From the results, it is observed that fault prediction capability is increased using ensemble-based learning [29, 63]. Furthermore, we validated the statistical significance of performance differences among the base classifiers and the best ensemble classifier, using the non-parametric Friedman test and performed pair wise comparisons using the Wilcoxon Rank-Sum test to test the worst-performing classifiers.

External validity: It deals with the generalization of the results of our study to other settings. In this study, we considered thirty two releases from various datasets used for different application domains. To minimize the effect of a particular tool/technology, in our normalized dataset we have taken applications developed using different version control systems (CVS and SVN) and diverse bug tracking tools (Ckjm, Bug Info, Quality Spy) [17, 63]. We preprocessed the data related to the metrics and obtained a reduced set of thirteen features for defect prediction so that the generalized prediction models are formed. The comparative assessment performed using base and ensemble classifiers verified the significance of proposed advanced models in fault prediction and helps to minimize the threat due to external validity.

7. Conclusions

The work presents advanced models for software fault prediction, in which authors have used information related to product and process metrics. The models for investigation were built based on five different scenarios as discussed in Section 4. Scenario-1: simple model (consists of product metrics only); scenario-2: advanced model-1 (product metrics + NR process metric); scenario-3: advanced model-2 (product metrics + NDC process metric); scenario-4: advanced model-3 (product metrics + NML process metric); scenario-4: advanced model-4 (product metrics + NDPV process metric). The various base classifiers used to predict the performance of proposed models are DT, MLP, RT, SVM and NB. The study has been conducted on thirty-two open-source code projects extracted from the Promise, Jira and Bug repositories. The results show that among base classifiers the MLP based base classifier captures high average accuracy (87%), average ROC(AUC) (79%), average F-score (83%) and least RMSE error (0.12) for advanced model-2 constructed using (product + NDC metrics) from Promise repository as compared to other advanced models, i.e., advanced model-1, advanced model-3 and advanced model-4, respectively. Similar trend is observed for projects from Jira and Bug repositories too.

Furthermore, to examine whether the ensemble design improves classification performance as compared to individual machine learning classifiers we used the ensemble approach based on bagging, boosting and voting to combine multiple classifiers and conducted replication experiments. The comparison of results using average accuracy, average RMSE, average ROC(AUC) and average F-score confirms the predictive capability of proposed classifiers for developing advanced defect prediction models. The VOT-E2 (DT + MLP + SVM) ensemble produced the best results with advanced model-2, advanced model-3 and advanced model-1 followed by VOT-E1 classifiers (DT + MLP + RT), in terms of ROC(AUC) and F-measure. Further to validate the statistical significance of performance differences among the base classifiers and classifier ensemble, we also tested the hypothesis H_0, that there is no significant difference between base classifier performance and ensemble classifier performance using a non-parametric test. We also evaluated the fault removal estimation cost for the proposed ensemble and best base classifier. The normalized fault removal cost is obtained for different projects from Promise, Jira and Bug repositories by calculating the ratio of estimated fault removal cost to estimated testing cost, which is below the threshold value, i.e., less than one.

Our results shows that the advanced fault estimation models constructed with a normalized and minimum subset of software metrics, which includes product metrics and one process metric at a time, provide satisfactory performance as compared to simple models constructed using product metrics alone. The proposed approach based on combination models may prove useful to software engineers for their new projects. Though in the study, authors have conducted experiments using projects from Promise, Jira and Bug repositories, still, to establish evidence and improve generalization of results, the investigations shall be replicated using more open-source and cross-project data sets. Several defect prediction models have been developed which use heterogeneous metric data from other projects [31, 63]. The investigation using more number projects would not only increase the variety of examined data but also helps to improve the external validity of the research outcomes.

References

[1] Z. Li, X.Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *Iet Software*, Vol. 12, No. 3, 2018, pp. 161–175.

[2] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE transactions on software engineering*, Vol. 37, No. 3, 2010, pp. 356–370.

[3] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990, 1990. [Online]. https://ieeexplore.ieee.org/document/159342

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

[5] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 17–26.

[6] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu et al., "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," in *Proceedings of the 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 157–168.

[7] Ö.F. Arar and K. Ayan, "Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies," *Expert Systems with Applications*, Vol. 61, 2016, pp. 106–121.

[8] R. Malhotra and J. Jain, "Handling imbalanced data using ensemble learning in software defect prediction," in *10th International Conference on Cloud Computing, Data Science and Engineering (Confluence)*. IEEE, 2020, pp. 300–304.

[9] F. Matloob, T.M. Ghazal, N. Taleb, S. Aftab, M. Ahmad et al., "Software defect prediction using ensemble learning: A systematic literature review," *IEEE Access*, 2021.

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

[11] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, Vol. 34, No. 4, 2008, pp. 485–496.

[12] S.S. Rathore and S. Kumar, "An empirical study of ensemble techniques for software fault prediction," *Applied Intelligence*, Vol. 51, No. 6, 2021, pp. 3615–3644.

[13] R. Jabangwe, J. Börstler, D. Šmite, and C. Wohlin, "Empirical evidence on the link between object-oriented measures and external quality attributes: A systematic literature review," *Empirical Software Engineering*, Vol. 20, No. 3, 2015, pp. 640–693.

[14] Z. Li, X.Y. Jing, and X. Zhu, "Heterogeneous fault prediction with cost-sensitive domain adaptation," *Software Testing, Verification and Reliability*, Vol. 28, No. 2, 2018, p. e1658.

[15] R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with Android software," *Applied Soft Computing*, Vol. 49, 2016, pp. 1034–1050.

[16] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, Vol. 385, 2020, pp. 100–110.

[17] I. Kiris, S. Kapan, A. Kılbas, N. Yılmaz, I. Altuntaş et al., "The protective effect of erythropoietin on renal injury induced by abdominal aortic-ischemia-reperfusion in rats," *Journal of Surgical Research*, Vol. 149, No. 2, 2008, pp. 206–213.

[18] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study," *Software Quality Journal*, Vol. 23, No. 3, 2015, pp. 393–422.

[19] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and software technology*, Vol. 55, No. 8, 2013, pp. 1397–1418.

[20] Y. Wu, Y. Yang, Y. Zhao, H. Lu, Y. Zhou et al., "The influence of developer quality on software fault-proneness prediction," in *EIghth International Conference on Software Security and Reliability (SERE)*. IEEE, 2014, pp. 11–19.

[21] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! Examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 4–14.

[22] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto et al., "A developer centered bug prediction model," *IEEE Transactions on Software Engineering*, Vol. 44, No. 1, 2017, pp. 5–24.

[23] F. Palomba, M. Zanoni, F.A. Fontana, A. De Lucia, and R. Oliveto, "Toward a smell-aware bug prediction model," *IEEE Transactions on Software Engineering*, Vol. 45, No. 2, 2017, pp. 194–218.

[24] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 432–441.

[25] B. Ghotra, S. McIntosh, and A.E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 2015, pp. 789–800.

[26] F. Yucalar, A. Ozcift, E. Borandag, and D. Kilinc, "Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability," *Engineering Science and Technology, an International Journal*, Vol. 23, No. 4, 2020, pp. 938–950.

[27] I.H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, Vol. 58, 2015, pp. 388–402.

[28] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, Vol. 179, No. 8, 2009, pp. 1040–1058.

[29] T.M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *22nd IEEE International conference on tools with artificial intelligence*, Vol. 1. IEEE, 2010, pp. 137–144.

[30] X. Chen, Y. Mu, Y. Qu, C. Ni, M. Liu et al., "Do different cross-project defect prediction methods identify the same defective modules?" *Journal of Software: Evolution and Process*, Vol. 32, No. 5, 2020, p. e2234.

[31] Y. Zhang, D. Lo, X. Xia, and J. Sun, "Combined classifier for cross-project defect prediction: An extended empirical study," *Frontiers of Computer Science*, Vol. 12, No. 2, 2018, p. 280.

[32] T. Lee, J. Nam, D. Han, S. Kim, and H.P. In, "Micro interaction metrics for defect prediction," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 311–321.

[33] K. Juneja, “A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation,” *Applied Soft Computing*, Vol. 77, 2019, pp. 696–713.

[34] H. Wang, T.M. Khoshgoftaar, and A. Napolitano, “A comparative study of ensemble feature selection techniques for software defect prediction,” in *Ninth International Conference on Machine Learning and Applications*. IEEE, 2010, pp. 135–140.

[35] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, “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*, 2016, pp. 1–10.

[36] F. Pecorelli and D. Di Nucci, “Adaptive selection of classifiers for bug prediction: A large-scale empirical analysis of its performances and a benchmark study,” *Science of Computer Programming*, Vol. 205, 2021, p. 102611.

[37] D. Di Nucci, F. Palomba, R. Oliveto, and A. De Lucia, “Dynamic selection of classifiers in bug prediction: An adaptive method,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, Vol. 1, No. 3, 2017, pp. 202–212.

[38] D. Bowes, T. Hall, and J. Petrić, “Software defect prediction: do different classifiers find the same defects?” *Software Quality Journal*, Vol. 26, No. 2, 2018, pp. 525–552.

[39] G. Abaei, A. Selamat, and H. Fujita, “An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction,” *Knowledge-Based Systems*, Vol. 74, 2015, pp. 28–39.

[40] E. Erturk and E.A. Sezer, “A comparison of some soft computing methods for software fault prediction,” *Expert systems with applications*, Vol. 42, No. 4, 2015, pp. 1872–1879.

[41] Y. Hu, B. Feng, X. Mo, X. Zhang, E. Ngai et al., “Cost-sensitive and ensemble-based prediction model for outsourced software project risk prediction,” *Decision Support Systems*, Vol. 72, 2015, pp. 11–23.

[42] M.O. Elish, H. Aljamaan, and I. Ahmad, “Three empirical studies on predicting software maintainability using ensemble methods,” *Soft Computing*, Vol. 19, No. 9, 2015, pp. 2511–2524.

[43] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, “An empirical study on software defect prediction with a simplified metric set,” *Information and Software Technology*, Vol. 59, 2015, pp. 170–190.

[44] W. Rhmann, B. Pandey, G. Ansari, and D.K. Pandey, “Software fault prediction based on change metrics using hybrid algorithms: An empirical study,” *Journal of King Saud University-Computer and Information Sciences*, Vol. 32, No. 4, 2020, pp. 419–424.

[45] A. Kaur and I. Kaur, “An empirical evaluation of classification algorithms for fault prediction in open source projects,” *Journal of King Saud University-Computer and Information Sciences*, Vol. 30, No. 1, 2018, pp. 2–17.

[46] D. Cotroneo, A.K. Iannillo, R. Natella, R. Pietrantuono, and S. Russo, “The software aging and rejuvenation repository: http://openscience.us/repo/software-aging,” in *International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2015, pp. 108–113.

[47] M. D’Ambros, M. Lanza, and R. Robbes, “An extensive comparison of bug prediction approaches,” in *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)*. IEEE CS Press, 2010, pp. 31 – 41.

[48] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters et al., “The promise repository of empirical software engineering data,” *West Virginia University, Department of Computer Science*, 2012.

[49] M. Shepperd, Q. Song, Z. Sun, and C. Mair, “Data quality: Some comments on the NASA software defect datasets,” *IEEE Transactions on Software Engineering*, Vol. 39, No. 9, 2013, pp. 1208–1215.

[50] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, Vol. 16, 2002, pp. 321–357.

[51] S. Wang and X. Yao, “Using class imbalance learning for software defect prediction,” *IEEE Transactions on Reliability*, Vol. 62, No. 2, 2013, pp. 434–443.

[52] K. Gao, T.M. Khoshgoftaar, H. Wang, and N. Seliya, “Choosing software metrics for defect prediction: an investigation on feature selection techniques,” *Software: Practice and Experience*, Vol. 41, No. 5, 2011, pp. 579–606.

[53] Z.H. Zhou and X.Y. Liu, “Training cost-sensitive neural networks with methods addressing the class imbalance problem,” *IEEE Transactions on knowledge and data engineering*, Vol. 18, No. 1, 2005, pp. 63–77.

[54] L. Kumar, S. Misra, and S.K. Rath, “An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes,” *Computer Standards and Interfaces*, Vol. 53, 2017, pp. 1–32.

[55] M. Dash and H. Liu, “Consistency-based search in feature selection,” *Artificial intelligence*, Vol. 151, No. 1-2, 2003, pp. 155–176.

[56] S.S. Rathore and S. Kumar, “Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems,” *Knowledge-Based Systems*, Vol. 119, 2017, pp. 232–256.

[57] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE transactions on software engineering*, Vol. 33, No. 1, 2006, pp. 2–13.

[58] A.E.C. Cruz and K. Ochimizu, “Towards logistic regression models for predicting fault-prone code across software projects,” in *3rd international symposium on empirical software engineering and measurement*. IEEE, 2009, pp. 460–463.

[59] J. Li, D.M. Witten, I.M. Johnstone, and R. Tibshirani, “Normalization, testing, and false discovery rate estimation for RNA-sequencing data,” *Biostatistics*, Vol. 13, No. 3, 2012, pp. 523–538.

[60] J. Nam, S.J. Pan, and S. Kim, “Transfer defect learning,” in *35th international conference on software engineering (ICSE)*. IEEE, 2013, pp. 382–391.

[61] S. Matsumoto, Y. Kamei, A. Monden, K.i. Matsumoto, and M. Nakamura, “An analysis of developer metrics for fault prediction,” in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–9.

[62] Y. Jiang, B. Cukic, and Y. Ma, “Techniques for evaluating fault prediction models,” *Empirical Software Engineering*, Vol. 13, No. 5, 2008, pp. 561–595.

[63] X. Xuan, D. Lo, X. Xia, and Y. Tian, “Evaluating defect prediction approaches using a massive set of metrics: An empirical study,” in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1644–1647.

[64] S. Wagner, “A literature survey of the quality economics of defect-detection techniques,” in *Proceedings of the ACM/IEEE international symposium on Empirical software engineering*, 2006, pp. 194–203.

[65] C. Jones and O. Bonsignour, *The economics of software quality*. Addison-Wesley Professional, 2011.

[66] N. Wilde and R. Huitt, “Maintenance support for object-oriented programs,” *IEEE Transactions on Software Engineering*, Vol. 18, No. 12, 1992, p. 1038.

[67] T. Wang, W. Li, H. Shi, and Z. Liu, “Software defect prediction based on classifiers ensemble,” *Journal of Information and Computational Science*, Vol. 8, No. 16, 2011, pp. 4241–4254.

[68] K. Bańczyk, O. Kempa, T. Lasota, and B. Trawiński, “Empirical comparison of bagging ensembles created using weak learners for a regression problem,” in *Asian Conference on Intelligent Information and Database Systems*. Springer, 2011, pp. 312–322.

[69] G. Catolino and F. Ferrucci, “An extensive evaluation of ensemble techniques for software change prediction,” *Journal of Software: Evolution and Process*, Vol. 31, No. 9, 2019, p. e2156.

[70] L. Reyzin and R.E. Schapire, “How boosting the margin can also boost classifier complexity,” in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 753–760.

[71] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, “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*, 2016, pp. 1–10.

[72] A.T. Mısırlı, A.B. Bener, and B. Turhan, “An industrial case study of classifier ensembles for locating software defects,” *Software Quality Journal*, Vol. 19, No. 3, 2011, pp. 515–536.

[73] J. Bansiya and C.G. Davis, “A hierarchical model for object-oriented design quality assessment,” *IEEE Transactions on software engineering*, Vol. 28, No. 1, 2002, pp. 4–17.

[74] E. Shihab, Z.M. Jiang, W.M. Ibrahim, B. Adams, and A.E. Hassan, “Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project,” in

Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2010, pp. 1–10.

[75] R. Martin, "OO design quality metrics," *An analysis of dependencies*, Vol. 12, No. 1, 1994, pp. 151–170.

A. Appendix

Table A1. Description of selected features

Product metrics	Description	References
WMC	It is the weighted sum of methods implemented within a class.	[73]
NOC	It is defined as the number of instant sub classes (children) subordinated to a class (parent) in the class hierarchy.	[73]
CBO	It defines the number of other classes that are tied to a given class during method call or function call, abstraction etc.	[73]
RFC	It is a count of methods in a class or methods directly called by these.	[73]
LCOM	It is a number of private methods in a class which don"t connect the class fields.	[73]
Ca	It is used to measure the number of classes that depends on a given class.	[74]
Ce	It is used to measure the number of classes on which a given class depends.	[74]
NPM	It is a number of public methods in a given class.	[58]
LOC	It is a number of lines of code in a given class.	[59]
DAM	It is the ratio of the number of private/protected attributes to the total number of attributes in a given class.	[65]
MOA	It is a number of classes whose declaration is user defined.	[65]
AMC	It is the average size of methods in a given class.	[75]
Max$-CC$	It is the maximum McCabe"s CC score for methods in a given class.	[75]
Avg$-CC$	It is the arithmetic mean of McCabe"s CC score for methods in a given class.	[75]
Process metrics	**Description**	**References**
NR	It represents the number of revisions of a given class because of bug/or some enhancements in a specific revision or version of a software system.	[17, 24, 44, 74]
NDC/NAUTH	It counts the number of different programmers /developers/authors who committed their changes in the given class during the improvement of the specific revision of the software.	[17, 24, 44, 74]
NML/NREF	It is the sum of all number of lines that are added or altered or number of times a file has been refactored.	[17, 24, 44, 74]
NDPV	The metric counts the number of defects in the previous version being corrected in the respective class while developing the previous releases or versions.	[24, 44, 74]

Table A2. Details of base classifiers and classifier ensembles

Base Classifiers	Description
Naive Bayes (NB) [29]	It is a probabilistic classification technique based on Baye"s Theorem with an assumption, that each pair of features being classified is independent of each other.
Decision Tree (DT) [29]	The simplest supervised learning method which creates tree structure to consider target values as discrete set or decision rules known as classification tree and nodes denotes class labels.
Random Tree (RT) [29]	It is a collection of multiple trees which are relatively uncorrelated, operating as a committee, split out a class with the most votes for model"s prediction.
Multilayer Perceptron (MLP) [29]	A supervised feed-forward artificial neural network model which maps input data onto a set of appropriate outputs and in between these two, an arbitrary number of hidden layers which work as a computational engine of the MLP.
Support vector machines (SVMs) [41]	SVMs are a group of supervised learning methods which makes use of statistical learning theory for classification. The methods are proposed by Cortes and [69]. The basic idea of SVM is to identify a similarity distance between two entities (classes) by considering a distance metric between them. It could also be used to handle unbalanced classes.
Ensemble classifiers	**Description**
Random Forest (RF) [29]	It is an ensemble-based method used in classification which constitute multiple decision trees on randomly selected data at training time and get prediction from each tree and choose best solution by "voting".
Boosting [61]	The method is proposed by Freund [71]. It modifies a training set by repeatedly applying a basic learning device (i.e., classifier) under a pre-specified number of iterations. Adaptive Boosting (AdaBoost) is a well-known Boosting technique.
Bootstrap aggregating [29]	Bagging is a bootstrap method proposed by Breiman [72] that mainly extracts a training sample from a training set by returning them to each extraction. It allocates equal weight to developed models, thereby reduces the variance related with classification, which in turn improves the classification process.
Voting [29]	It represents the simplest ensemble algorithm used for classification or regression problems. Each sub model in the algorithm makes use of "votes" or "algebraic combinations" (mean or the mode) of heterogeneous predictors to make predictions.

Table A3. Details of performance measures

Measures	Defined as	Description
Pd/Recall/TP	$TP/(TP+FN)$	It is defined as the ratio of the number of defective instances that are correctly classified as defective to the total number of defective instances.
Pf	$FP/(FP+TN)$	It is defined as the ratio of the number of non-defective instances that are wrongly classified as defective to the total number of non-defective instances.
Precision	$TP/(TP+FP)$	Precision is defined as the number of correctly identified positive results to the total number of all positive outcomes, including those not recognized correctly.
$F1$-score	$(2 \times \mathit{precision} \times Pd)/(\mathit{precision}+Pd)$	It is a measure for harmonic mean of Pd and *precision*.
Accuracy	$(TP+TN)/(TP+TN+FP+FN) \times 100$	It denotes the percentage of correctly predicted instances.
Root mean square error (RMSE)	$\sqrt{1/N \sum_{i=1}^{N}(A_i - P_i)^2}$	It defines the square root of the mean of squared differences of actual and expected predictions.
ROC(AUC)	Receiver operating Characteristics (Area under curve)	ROC(AUC) measures the performance of the classification problems at various thresholds in the imbalanced data-set. ROC is a probability curve, and AUC represents the measure of separability. If AUC value is high, the model is predicting better. It ranges from 0 to 1
False-positive rate (FPR)	$FPR = FP/(TP+TN)$	The expectancy of the false-positive ratio to the total of actual negative.
False-negative rate (FNR)	$FNR = FN/(TP+FN)$	The ratio of the individuals with an identified positive instance for which the classified test result is negative.

Table A4. Confusion matrix for classifying data as faulty or non-faulty

		Actual	
		Positive	Negative
Predicted	Positive	True-positive (TP)	False-positive (FP)
	Negative	False-negative (FN)	True-negative (TN)

Under the cost evaluation framework, the notations used to formulate various costs are:

- E_{cost}: Estimated fault removal cost of the software when software fault prediction results are used
- T_{cost}: Estimated fault removal cost of the software without the use of software fault prediction results
- NE_{cost}: Normalized fault removal cost of the software when software fault prediction is used
- C_u: Normalized value of fault removal cost when unit testing is done
- C_i: Normalized value of fault removal cost when integration testing is done
- C_s: Normalized value of fault removal cost when system testing is done
- C_f: Normalized value of fault removal cost when field testing is done
- M_p: Percentage of modules when unit tested
- FM: Total number of faulty modules, and
- TM: Total number of modules in software projects
- FP, FN: Number of false positives, number of false negatives
- TP: Number of true positives