An Empirical Analysis on Effective Fault Prediction Model Developed using Ensemble Methods

Lov Kumar
NIT Rourkela, India
lovkumar505@gmail.com

Santanu Rath
NIT Rourkela, India
skrath@nitrkl.ac.in

Ashish Sureka
ABB Corporate Research, India
ashish.sureka@in.abb.com

***Abstract*—Software fault prediction model are employed to optimize testing resource allocation by identifying fault-prone classes before testing phases. Several researchers' have validated the use of different classification techniques to develop predictive models for fault prediction. The performance of the statistical models are proven to be influenced by the training and testing dataset. Ensemble method learning algorithms have been widely used because it combines the capabilities of its constituent models towards a dataset to come up with a potentially higher performance as compared to individual models (improves generalizability). In the study presented in this paper, three different ensemble methods have been applied to develop a model for predicting fault proneness. The efficacy and usefulness of a fault prediction model also depends on the source code metrics which are considered as the input for the model.**

In this paper, we propose a framework to validate the source code metrics and select the right set of metrics with the objective to improve the performance of the fault prediction model. The fault prediction models are then validated using a cost evaluation framework. We conduct a series of experiments on 45 open source project dataset. Key conclusions from our experiments are: (1) Majority Voting Ensemble (MVE) methods outperformed other methods; (2) selected set of source code metrics using the suggested source code metrics using validation framework as the input achieves better results compared to all other metrics; (3) fault prediction method is effective for software projects with a percentage of faulty classes lower than the threshold value (low - 54.82%, medium - 41.04%, high - 28.10%)

***Index Terms*—Software fault prediction, machine learning, predictive modeling, source code metrics, ensemble methods**

I. Introduction

Early identification of source code regions, classes or modules where faults are likely to occur can help in optimizing and guiding testing efforts resulting in improvement of software quality. Fault prediction is an important and challenging problem in the area of software engineering and hence attracted the attention of several researchers. Many fault prediction techniques have been proposed and their performance have been evaluated on different dataset. Hall et al. [11] and Catal et al. [5] conduct a systematic literature review in the area of fault prediction. Hall et al. analyze 208 fault prediction studies and conclude that the methodology used to build models seems to be influential to predictive performance. One of the conclusions of their study is that more studies are needed that use a reliable methodology and which report their context, methodology, and performance comprehensively [11]. Arisholm et al. conduct a comprehensive examination of methods to develop and evaluate the performance of fault prediction models [1]. Their study is performed in an industrial setting in the context of large Java legacy system development project [1].

We believe that there are several research gaps and scope for more studied in the area of fault prediction. One of the research gaps is in modeling technique. The application of homogeneous and heterogeneous ensemble methods is relative unexplored. One of the novel contributions of our work in context to existing work is the application of various base learners such as logistic regression analysis, artificial neural network and radial basis function neural network as constituents of ensemble methods. Another research contribution of the work presented in this paper is an in-depth study of the generalizability of fault prediction model as we conduct our experiments on dataset belonging to 45 open source projects. We propose a cost evaluation model which takes into account the economics of software quality assurance [14]. The application of estimated fault removal cost, testing cost and the normalized fault removal cost in the fault prediction framework based on ensemble methods is a novel and unique contribution of our work.

II. Background

In this section, we define the dependent and independent variables of the models and provide the experimental dataset description.

A. Dependent and Independent Variables

In our study, the objective is to develop a fault prediction model using source code metrics as input variables or predictors. Hence, the class or category "bugs" is a dependent variable and various sets of metrics based on source code are independent variables. The independent and dependent variables for the fault prediction model is shown in Table I.

TABLE I: Predictors and Target Class

Dependent Variable	Independent Variable
Fault	DIT, WMC, RFC, CBO, LCOM, NOC, Ce, Ca, LCOM3, NPM, DAM, MOA, LOC, CAM, MFA, CBM, AVG-CC, MAX-CC, AMC, IC
Fault	Reduced feature attributes using proposed framework

B. Experimental Dataset

Based on our analysis of previour work, we observe that authors apply several different datasets to validate their proposed prediction models. We observe that several authors have used only a limited number of software systems to investigate the relationships between fault proneness and object-oriented metrics. Hence we beleive that due to experiments being conducted on limited dataset, it may not be clear whether a particular conclusion could be generalized to other software systems. However, in our study presented this paper, we have considered 45 real-life datasets from the PROMISE repository [1]. PROMIS repository is one if the largest repositories for software engineering research data. The dataset on PROMISE repository is publicly available and it is very easy to find research dataset on the repository. By conducting experiments on dataset form PPROMISE repository, we make our experiments easily replicable for other researchers. Table II shows the percentage of faulty classes in each project in the PROMISE repository used in our experiments.

C. Research Questions

Based on our analysis of several techniques on fault prediction as well as literature review studies, we frame the following research questions as gaps and contribution to the body of knowledge on software fault prediction:

RQ1 *Do source code metrics predict faulty or non faulty classes?*

The objective of this question is to test the relationship between each source code metric and fault proneness. In this research question, t-test analysis is used to test the statistical significance between faulty and non-faulty group metrics. T-test statistical significance testing is done by analyzing the means and distribution of faulty and non-faulty group metrics.

RQ2 *Can the selected set of source code metrics better predict whether a class is faulty or not?*

In this research question, our aim to evaluate the performance of the selected sets of metrics. In this study, two steps (i.e. t-test and forward stepwise selection procedure) are considered to identify subsets of object-oriented software metrics (from) that are more capable of predicting whether class is faulty or not.

RQ3 *Which fault prediction technique is a most suitable one for this purpose among all?*

This question helps to investigate the performance of different types of fault-prediction techniques. To address this question, three ensemble methods are considered for developing a fault prediction models in order to achieve the best performance.

RQ4 *For any given software product, is performing fault prediction analyses economically effective?*

This question investigates the effectiveness of different fault prediction techniques. To address this issue, a cost evaluation framework is proposed which performs cost based analysis for mis-classification of faults.

[1] $http://openscience.us/repo/defect/$

TABLE II: List of projects used in our experiments, number of classes and percentage of faulty classes

Id	Project	No. of class	No. of Faulty class	Faulty (%)
D1	ant-1.3	125	20	16
D2	ant-1.4	178	40	22.47
D3	ant-1.5	293	32	10.92
D4	ant-1.6	351	92	26.21
D5	ant-1.7	745	166	22.28
D6	arc	234	27	11.54
D7	berek	43	16	37.21
D8	camel-1.0	339	13	3.83
D9	camel-1.2	608	216	35.53
D10	camel-1.4	872	145	16.63
D11	camel-1.6	965	188	19.48
D12	e-learning	64	5	7.81
D13	ivy-1.1	111	63	56.76
D14	ivy-1.4	241	16	6.64
D15	ivy-2.0	352	40	11.36
D16	jedit-3.2	272	90	33.09
D17	jedit-4.0	306	75	24.51
D18	jedit-4.1	312	79	25.32
D19	jedit-4.2	367	48	13.08
D20	kalkulator	27	6	22.22
D21	log4j-1.0	135	34	25.19
D22	log4j-1.1	109	37	33.94
D23	log4j-1.2	205	189	92.2
D24	lucene-2.0	195	91	46.67
D25	lucene-2.2	247	144	58.3
D26	lucene-2.4	340	203	59.71
D27	pdftranslator	33	15	45.45
D28	prop-1	18471	2738	14.82
D29	prop-2	23014	2431	10.56
D30	prop-3	10274	1180	11.49
D31	prop-4	8718	840	9.64
D32	prop-5	8516	1299	15.25
D33	prop-6	660	66	10
D34	redaktor	176	27	15.34
D35	serapion	45	9	20
D36	synapse-1.0	157	16	10.19
D37	synapse-1.1	222	60	27.03
D38	synapse-1.2	256	86	33.59
D39	termoproject	42	13	30.95
D40	velocity-1.5	214	142	66.36
D41	velocity-1.6	229	78	34.06
D42	xerces-1.2	440	71	16.14
D43	xerces-1.3	453	69	15.23
D44	xerces-1.4	588	437	74.32
D45	xerces-init	162	77	47.53

III. Source Code Metrics Validation Framework

Researchers' uses different set of source code metrics as input to develop a model for predicting whether class is faulty or not [6] [13][2]. This shows that the performance of fault prediction model depends on the software metrics which have

been considered as inputs to develop a model. Selection of a suitable set of features is an important data pre-processing task in different applications of data mining and machine learning [9][10][8]. In this paper, a source code metrics validation framework has been proposed for validating the metrics and identifying suitable set of source code metrics with an aim to reduce irrelevant metrics and improve the performance of the fault prediction model. Irrelevant source code metrics are those features with very low predictive or discriminatory power with respect to the target class. The proposed framework is applied to 45 real-life datasets taken from the PROMISE repository [2]. Finally, we have validated the framework by comparing the performance of the models developed using a selected set of source code metrics with the performance of those developed using original dataset.

Figure 1 shows the detail steps of the proposed software metrics validation framework. Our proposed approach is a multi-step process. The objective of this framework is to first investigate whether these source code metrics are significant predictors of fault proneness without involving any learning algorithm. After identifying all significant source code metrics, the wrapper approach is employed for identifying the right set of source code metrics. It uses the performance of the chosen learning algorithm to evaluate each candidate feature subset. In this experiment, linear discriminant analysis is considered as a classification algorithm.

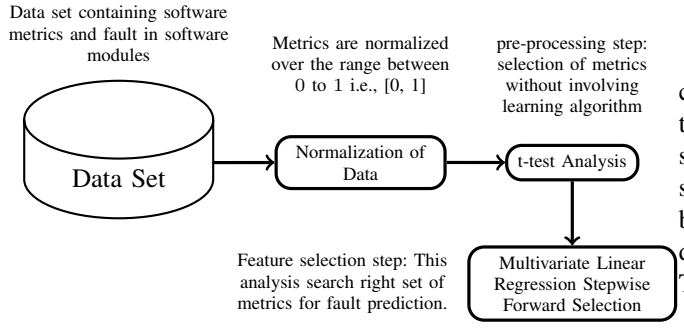


Fig. 1: Proposed Framework of Software Metrics Validation

i. *Data Set:* The requisite fault data of 45 projects listed in Table II are taken from tera-PROMISE Repository. The data set containing fault information and twenty software source code metrics are also included.
ii. *Normalization of Data:* All source code metrics are normalized over the range between 0 to 1 using Min-Max normalization technique. The normalization of source code metrics is required to adjust the defined range of metrics. Hence, before applying the machine learning algorithm and subsequent t-test analysis, the input metrics are normalized or standardized using min-max scaling.
iii. *Filter Approach:* Filter approach is usually used as a pre-processing step to remove insignificant features. We have considered the characteristics of features to select significant sets of features without involving a learning algorithm. T-test has been employed to remove insignificant features. The objective of this step is to test the relationship between each source code metric and fault proneness. In this study, t-test analysis is used to test the statistical significance between faulty and non-faulty group metrics. In 2-class problems (faulty class and non faulty classes), test of null hypothesis ($H0$) means that the two populations are not equal; on other words, there is a significant difference between their mean value and both features are different. It further implies that the metrics affect the fault prediction result. Hence, these metrics have been considered and those having no significant difference between their mean value are rejected. Therefore, it is necessary to accept the null hypothesis ($H0$) and reject the alternate hypothesis. Here, a $t-test$ on each metric is applied and compared with their corresponding $P-value$ for each metric as a measure of how effectively it separates the groups. The metrics having $P-value$ smaller than 0.05 have strong discrimination powers.
iv. *Wrapper Approach:* The above filter approach does not consider the interaction between source code metrics; hence, it may be possible that the selected metrics using filter approach also contain redundant information. In this work, multivariate linear regression stepwise forward selection is considered in a wrapper fashion to compute an optimal set of source code metrics.

[2]http://openscience.us/repo/

IV. Cost Analysis Model

In this section, we describe the construction of our proposed cost evaluation model. The cost evaluation model accounts for the realistic costs incurred to remove a fault or defect in the system and computes the estimated fault removal cost for a specific fault prediction technique based on the ideas proposed by Wagner et al. ([14]). We make certain assumptions and define the constraints in designing this cost evaluation model. The assumptions and constraints are listed as follows:

i. Different testing phases account for varying fault removal cost.
ii. It is not practically possible to completely detect all faults within one testing phase.
iii. It is not practically possible to perform unit testing on all classes in a system.

The normalized fault removal cost approach suggested by Wagner *et al.* ([14]) is applied in our work to formulate the proposed cost evaluation model. The fault removal cost varies because various projects are developed on different platforms following organization standards and practises which varies across projects. The normalized fault removal costs are summarized in Table III. The fault identification efficiencies for different testing phases used in our work are inspired from the study by Jones et al. ([4]). The efficiencies of testing phases are summarized in Table IV. Wilde *et al.* ([12]) stated that more than 50% of the classes are usually very small in size, and hence performing unit testing on these classes may not be very much helpful.

TABLE III: Removal costs of test techniques (in staff hour per defects)

Type	Min	Max	Mean	Median
Unit (C_u)	1.5	6	3.46	2.5
Integration (C_i)	3.06	9.5	5.42	4.55
System (C_s)	2.82	20	8.37	6.2
Field (C_f)	3.9	66.6	27.24	27

TABLE IV: Fault identification efficiencies - test phases

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

The formulations of E_{cost}, T_{cost} and the NE_{cost} of the proposed cost based evaluation framework are presented int he below section. The mathematical notations used in our framework are described below:

i. C_i: Initial setup cost of used fault-prediction technique, C_u, C_i, C_s, and C_f are the normalized fault removal cost in unit, integration, system, and field testing respectively. M_p: percentage of classes unit tested.
ii. δ_u, δ_i and δ_s are the fault identification efficiency of unit, integration, and system testing respectively.
iii. FC and TC are the number of faulty modules and total number of modules in software projects respectively. TN, FN, FP, and TP are the value of true negative, false negative, false positive, and true positive respectively.

Estimated fault removal cost (E_{cost}): The series of steps involved in computing the estimated fault removal cost of the software system when fault prediction is performed (E_{cost}) are defined as follows:

i. Total number of faulty classes identified by the predictor are equal to the summation of true positive (TP) and false positive (FP) values. Hence, it is necessary to compute testing and verification cost at class level of granularity which indicates that the value of cost is equal to the cost of unit testing (C_u). The total cost on unit testing of software system is defined as:

$$Cost_{unit} = (TP + FP) * C_u \quad (1)$$

ii. Since it impractical to detect all fault within a specific testing phase, there is a possibility that some of the correctly predicted faulty classes remain undetected in unit testing. Furthermore, there is a possibility that these faulty classes which were predicted as non-faulty classes (number of false negative (FN)), are identified by the predictor in the later phases of testing, such as integration(C_i), system, and field testing. The fault removal cost in integration, system, and field testing is computed as follows:

$$Cost_{Integration} = C_i * \delta_i * (FN + TP* (1 - \delta_u)) \quad (2)$$

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

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

iii. The estimated overall fault removal cost can be determined as:

$$Ecost = C_i + C_u * (FP + TP) + \delta_i * C_i * (TP* (1 - \delta_u) + FN) + \delta_s * C_s * ((1 - \delta_i) * (TP * (1 - \delta_u)+ FN)) + (1-\delta_s) * C_f * ((1-\delta_i) * (TP * (1-\delta_u) + FN)) \quad (5)$$

Estimated testing cost (T_{cost}): The list of steps to compute the estimated fault removal cost of the software system without using fault prediction approach (T_{cost}) is defined as follows:

i. In testing phase, if fault prediction analysis is not conducted, then the testing team often performs unit testing on all the classes. Therefore, total unit testing may be computed as:

$$Cost_{unit} = M_p * C_u * TC \quad (6)$$

ii. Furthermore, there is a likelihood that some of the faulty classes that remain undetected in unit testing may later be identified in integration, system, and field testing phases. The total integration, system, and field testing cost is calculated as:

$$Cost_{integration} = \delta_i * C_i * (1 - \delta_u) * FC \quad (7)$$

$$Cost_{system} = \delta_s * C_s * ((1 - \delta_i) * (1 - \delta_u) * FC) \quad (8)$$

$$Cost_{field} = (1-\delta_s) * C_f * ((1-\delta_i) * (1-\delta_u) * FC) \quad (9)$$

iii. The estimated overall fault removal cost without the use of fault prediction can be determined by using following equation as:

$$Tcost = M_p * C_u * TC + \delta_i * C_i * (1 - \delta_u) * FC + \delta_s * C_s * ((1 - \delta_i) * (1 - \delta_u) * FC + (1 - \delta_s) C_f * ((1 - \delta_i) * (1 - \delta_u) * FC) \quad (10)$$

Normalized fault removal cost (NE_{cost}) : Normalized fault removal cost ($\frac{E_{cost}}{T_{cost}}$) and its interpretation can be modeled as:

$$\text{If the value of } NEcost = \begin{cases} < 1, \text{then application of fault prediction is useful} \\ => 1, \text{then application of testing methodlogies may be helpful} \end{cases} \quad (11)$$

V. Experimental setup

In this section, we present the experimental setup of our developed fault prediction models. Figure 2 shows the work flow of the proposed work. In this study, we perform the following steps to develop fault prediction models:

Metrics Selection: Selection of suitable set of source code metrics using proposed source code metrics validation framework.
Prediction Model: Development of prediction model by considering source code metrics as input to predict fault proneness.
Performance Measures: Selection of performance measures that can used to evaluate the predictive capability of fault prediction models.
Validation Methods: Use of efficient validation methods to determine the true predictive applicability of the developed models.
Statistical Tests: Selection of appropriate statistical tests to determine the superiority of one prediction technique over the other prediction techniques and also determine the superiority of one set of source code metrics over the other sets.
Experimental Validation: Validation of developed fault prediction models using proposed cost analysis framework.

A. Selection of source code metrics

In this work, 20 source code metrics have been used for fault prediction. It is very essential to remove irrelevant and unimportant source code metrics out of these source code metrics so that only uncorrelated and relevant source code metrics are included in the construction of fault prediction models. In order to achieve this objective, we have proposed source code metrics validation framework as described in section III. This proposed framework is used for removing irrelevant source code metrics and select right set of metrics for fault prediction.

B. Classification Techniques

In this work, we have carefully selected three different ensemble methods. In ensemble of classification models, we have considered the outputs of all its individual constituent classification models where base learners are assigned a certain priority level in the each classification model and the final output is computed with the help of some combination rules. There are two types of ensemble methods,

- Homogeneous ensemble method: In this method, all considered base learners, i.e. classification models, are of the same types, but each one has a randomly generated training set.
- Heterogeneous ensemble method: In this method, all considered base learners, i.e. classification models, are of different types.

The ensemble methods can be further categorized into two different groups based on combination rules. These categories are:

- Linear ensemble method: In this method, the arbitrator combines the outputs of the base learners, i.e. classification models, in a linear fashion such as averaging, best in training, weighted averaging, etc.
- Nonlinear ensemble method: In this method, the output of the considered base learners, i.e. classification models, are fed into an arbitrator, which is a nonlinear prediction model such as neural network, Decision tree forest (DTF) etc.

In the present work, we have considered a heterogeneous ensemble method with three different combination rules (2 Linear, and 1 nonlinear). A detailed description of the ensemble methods used in this work are tabulated in Table V.

TABLE V: Ensembles of Classification Models

Base Learners	Combination Rules
LOGR, ANN, RBFN-RAN, RBFN-FCM, RBFM-KCM	Linear (best in training)
LOGR, ANN, RBFN-RAN, RBFN-FCM, RBFM-KCM	Linear (majority voting)
LOGR, ANN, RBFN-RAN, RBFN-FCM, RBFM-KCM	Non-linear (DTF)

C. Base learners

In this section, we briefly describe each base learners that were used in ensemble methods.

1) Logistic regression analysis (LOGR): Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that predicting the outcomes of dependent variable [7]. The logistic regression model is based on the following equation:

$$\pi(x) = \frac{e^{\alpha_0 + \sum_{i=1}^{p} \alpha_i * X_i}}{1 + e^{\alpha_0 + \sum_{i=1}^{p} \alpha_i * X_i}} \tag{12}$$

P represents the number of independent variables. π represents the probability of fault in the class during validation.

2) Artificial Neural Network (ANN) model: In the present work, ANN is considered for developing fault prediction models. The neural network can be represented as:

$$O' = f(W, I) \tag{13}$$

where I and $O^{'}$ are the input and desired output vectors. W is the weight vector, whose W value is updated in every iteration with an aim to reduce the value of mean square error (MSE). MSE is computed using the following equation:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (O_i' - O_i)^2 \tag{14}$$

where O, and $O^{'}$ are the actual and desired output values. In the present work, Gradient Descent method is considered for training the ANN model. The Gradient Descent (GD) method is used for updating the weights to minimize the output error [3]. GD method uses the 1st order derivative of the total error function to find the minima in error space. It is represented using the following equation:

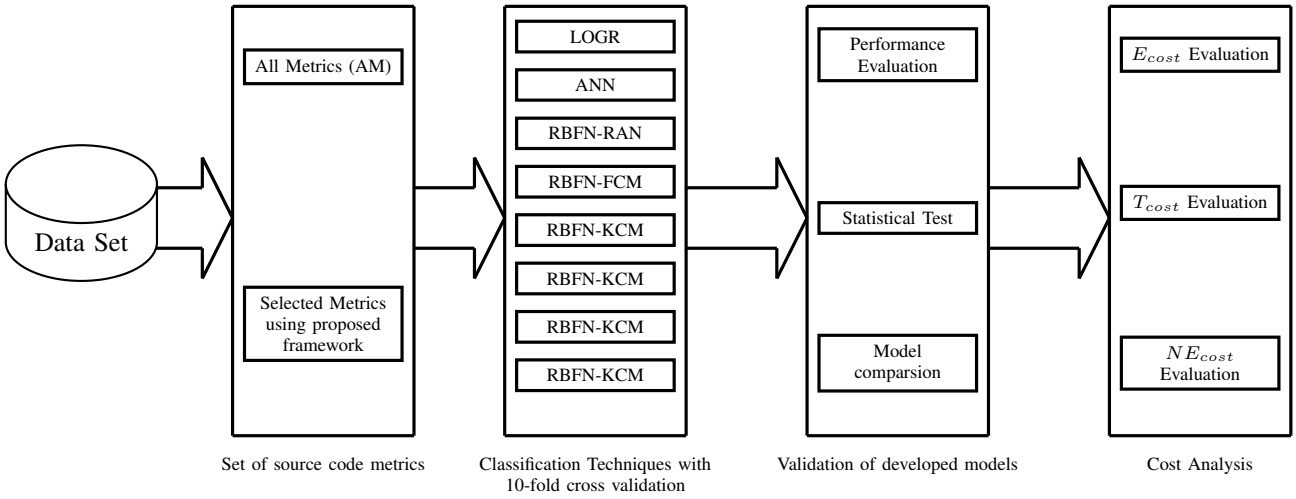


Fig. 2: Framework of Proposed work

$$G = \frac{\partial}{\partial W}(E_k) = \frac{\partial}{\partial W}(\frac{1}{2}(O'_k - O_k)^2) \tag{15}$$

In each iteration, weight vector W is updated using gradient vector G [3]. Weighted vector W is updated as:

$$W_{k+1} = -\alpha G_k = -\alpha \frac{\partial}{\partial W}(E_k) \tag{16}$$

where W_{k+1} is the updated weight vector, G_k is the gradient vector and α is the learning constant. We apply the cross-validation technique to compute the optimal value of α.

3) Radial Basis Function Neural Network (RBFN): RBFN network [15] is a popular alternative to the feed forward neural network, since it has simple network structure and fast training process [15]. In this analysis Hybrid RBFN model has been used for predicting software fault proneness. Basically RBFN model involves updating the centers and the weights of the model. In this paper, different approaches have been followed for updating centers and weight, details of which are mentioned below:

a. In the first approach, centers are generated randomly and weights are updated using Gradient Descent, is referred to as RBFN-RAN.
b. In the second instance, centers are identified using K-means clustering and weights are updated using Gradient Descent, and this approach is termed as RBFN-KMC.
c. In the third approach, centers are identified using Fuzzy C-mean clustering technique and weights are updated using Gradient Descent, and is referred to as RBFN-FCM.

D. Best Training Ensemble (BTE)

Best Training Ensemble (BTE) method takes the advantage of the fact that each classifier has a different performance across the used dataset partitions. Amongst these, we select the best model in the training dataset based on certain performance parameters.

[3]http://in.mathworks.com/help/nnet/ref/traingd.html

E. Majority Voting Ensemble (MVE) Method

In Majority Voting Ensemble (MVE) method, we have considered the output of each classifier on the test data, and the ensemble output (E_{out}) is the majority category classified by the base classifier.

F. Nonlinear Ensemble Decision Tree Forest (NDTF)

In nonlinear ensemble, we have considered the output corresponding to the training data of the base learner as the input to train the non-linear ensemble model. The trained non-linear ensemble model uses the output corresponding to the testing data of the base learner to make a final prediction on the test set. In this study, we have considered Decision tree forest (DTF) as a classifier for non-linear ensemble. The concept of DTF was proposed by Breiman in 2001. It is a collection of different decision trees where the result of each tree is combined to make a final decision.

G. Performance Parameter

In order to evaluate the fault prediction model, various performance parameters need to be analyzed which indicate the effectiveness of the developed fault prediction models. In this work, we have considered two different performance parameters: accuracy and F-Measure. These parameters are computed using values of various elements in a confusion matrix as shown in Table VI.

TABLE VI: Confusion matrix to classify a class as faulty and not-faulty

	Non Faulty	**Faulty**
Non Faulty	$N_{NF->NF}$	$N_{NF->F}$
Faulty	$N_{F->NF}$	$N_{F->F}$

$$Accuracy = \frac{N_{NF->NF} + N_{F->F}}{N_{classes}} \tag{17}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * N_{F->F}}{2 * N_{F->F} + N_{NF->F} + N_{F->NF}} \quad (18)$$

H. Validation method

The objective of this work is to apply our developed model to predict faulty classes for future releases and unseen similar natured projects. Hence, it is necessary to validate the developed fault prediction model on different data set from which they are trained. In this experiment, we have considered 10-fold cross validation to validate the proposed fault prediction model. Cross-validation is a statistical learning method, being used to evaluate and compare the models by partitioning the data into two portions. One portion of the divided set is used to train or learn the model and the rest of the data is used to validate the model, based on training.

I. Statistical tests

In order to statistically analyze the results, we have considered pairwise Wilcoxon signed rank test. We conducted this test to determine which of prediction methods and feature selection techniques work better or all have performed equally well. We have analyzed all results at 0.05 significance level.

J. Validation of Developed Fault Prediction Model

Finally these developed models are validated using proposed cost analysis framework.

VI. Experimental results

A. Source Code Metrics Validation

This section presents a detailed description of selection the right set of metrics for fault prediction. We started the statistical analysis with 20 source code object-oriented metrics. Figure 3 shows the selected set of metrics in each step for all 45 projects. For the purpose of simplicity, the graphs are represented using four different symbols as described below:

- Empty circle (○): source code metrics selected after t-test analysis; and
- Circle with star (⊛): source code metrics selected after t-test and MLR stepwise forward selection method.

From Figure 3, it is observed that wmc, cbo rfc, lcom, ca are commonly referred as relevant metrics to the fault of classes in most of the projects.

B. Performance Evaluation Parameters

We have considered two different set of metrics based on source code as input to design a model to estimate fault proneness of Java classes developed using 5 classification techniques i.e., LOGR, ANN, RBFN-RAN, RBFN-FCM, and RBFN-KMC and 3 ensemble methods i.e., BTE, MVE, and NDTF. Accuracy (%) and F-Measure are considered as performance parameters to measure the performance of the developed fault prediction models. Figure 4 and Figure 5 show the box-plot diagrams for each of the experimental results respectively

TABLE VII: t-test

(a) Training Methods

Accuracy								
Mean								
	LOGR	**ANN**	**RBFN-RAN**	**RBFN-FCM**	**RBFN-KCM**	**BTE**	**MVE**	**NDTF**
LOGR	0.00	-6.46	-8.78	-8.85	-8.78	-2.98	-9.18	-8.86
ANN	6.46	0.00	-2.32	-2.40	-2.32	3.48	-2.72	-2.41
RBFN-RAN	8.78	2.32	0.00	-0.08	0.00	5.80	-0.40	-0.09
RBFN-FCM	8.85	2.40	0.08	0.00	0.08	5.87	-0.33	-0.01
RBFN-KCM	8.78	2.32	0.00	-0.08	0.00	5.80	-0.40	-0.09
BTE	2.98	-3.48	-5.80	-5.87	-5.80	0.00	-6.20	-5.88
MVE	9.18	2.72	0.40	0.33	0.40	6.20	0.00	0.31
NDTF	8.86	2.41	0.09	0.01	0.09	5.88	-0.31	0.00
p-value								
	LOGR	**ANN**	**RBFN-RAN**	**RBFN-FCM**	**RBFN-KCM**	**BTE**	**MVE**	**NDTF**
LOGR	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ANN	0.00	1.00	0.00	0.00	0.00	0.24	0.00	0.00
RBFN-RAN	0.00	0.00	1.00	0.17	1.00	0.00	0.00	0.93
RBFN-FCM	0.00	0.00	0.17	1.00	0.17	0.00	0.26	0.12
RBFN-KCM	0.00	0.00	1.00	0.17	1.00	0.00	0.00	0.93
BTE	0.00	0.24	0.00	0.00	0.00	1.00	0.00	0.00
MVE	0.00	0.00	0.00	0.26	0.00	0.00	1.00	0.02
NDTF	0.00	0.00	0.93	0.12	0.93	0.00	0.02	1.00

(b) All metrics and Selected Metrics

	Accuracy					**F-Measure**			
	Mean		**P-value**			**Mean**		**P-value**	
	AM	**SM**	**AM**	**SM**		**AM**	**SM**	**AM**	**SM**
AM	0.00	-3.15	1.00	0.00	AM	0.00	-0.02	1.00	0.00
SM	3.15	0.00	0.00	1.00	SM	0.02	0.00	0.00	1.00

enabling a visual comparison. The middle line of the boxes show the median of accuracy and F-Measure. We apply 10-fold cross validation for all the combinations and the accuracy and f-measure metric values are summarized in the box blots. From the box-plot diagram, it can be inferred that:

- In all cases, the selected set of source code metrics has a high median value. Based on the boxplots, SM produced the best result, i.e. the proposed software metrics validation method computes the best set of source code metrics for predicting faulty and non-faulty classes of object-oriented software as compared to all metrics.
- Among all classification, ensemble methods have outperformed as compared to individual models. Further, It is observed that MVE yields better results compared to other techniques.

C. Comparison of results

In this work, we have considered pairwise Wilcoxon signed rank test to determine which of the classification techniques and selected sets of source code metrics work better or weather they all perform equally well. The use of Wilcoxon test without Bonferroni correction is not advisable because it does not take into account family-wise errors. In this work, we have considered Wilcoxon test with Bonferroni correction for comparison analysis.

1) Classification Techniques: Five different classification techniques and three different ensemble methods have been considered to develop a model to predict whether the class is faulty or not. Two different sets of metrics, one containing all metrics and one selected set using the proposed metrics validation techniques, have been considered as the input to develop fault prediction models over 45 different projects with two different performance parameters, i.e. accuracy, and F-Measure. In other words, for each technique a total number

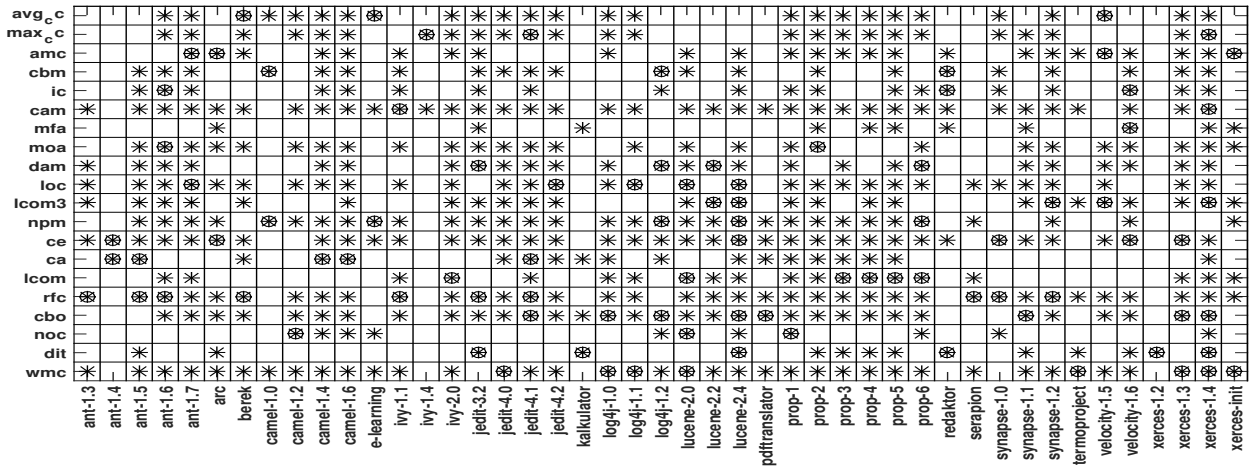
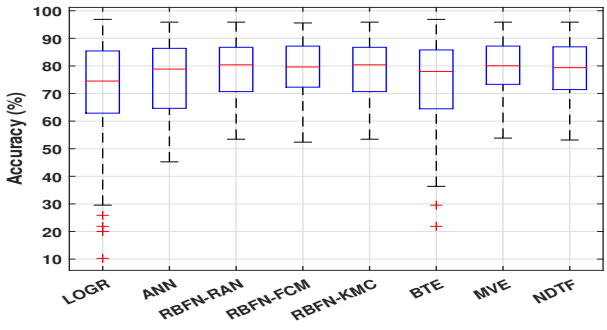
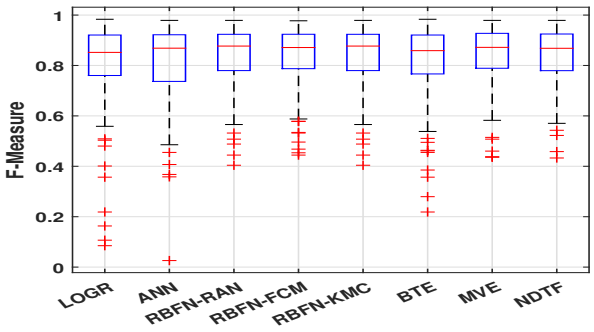


Fig. 3: Selected Set of Metrics

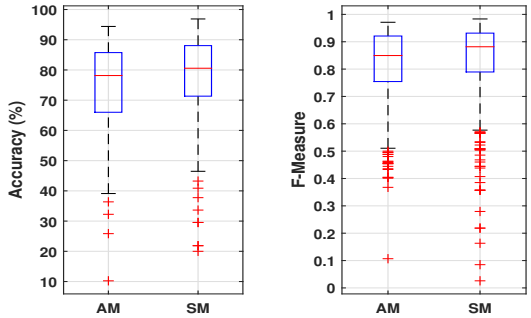


(a) Accuracy



(b) F-Measure

Fig. 4: Classification Techniques



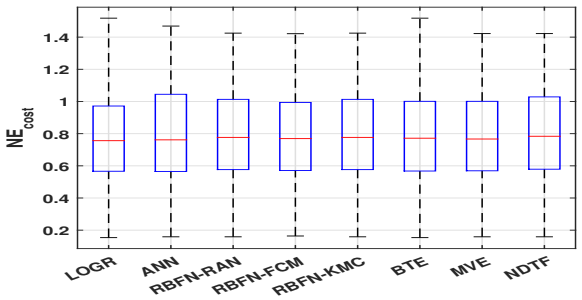
(a) Accuracy (b) F-Measure

Fig. 5: Selected set of source code metrics

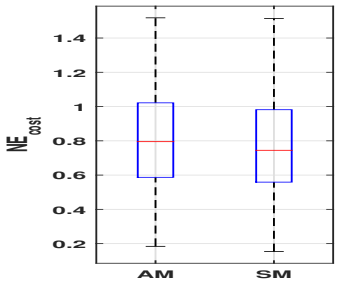
of two sets (one for each performance) is used, each with 90 data points ((1 feature selection method + 1 considering all features) * 45 datasets)). The results of the pair-wise comparisons of different training algorithms are shown in VIIa.

Table VIIa contains two parts; the first part of shows the mean difference values and the second part shows the p-value between different pairs. The Bonferroni correction sets the significance cutoff at $\frac{\alpha}{n}$. In this study, eight different techniques have been considered for analysis, i.e. total number of twenty eight (28) different pairs are possible ($^{8technique}C_2 = 8*7/2 = 28$) and all results are analyzed at a 0.05 significance level. Hence, we can only reject a null hypothesis if the p-value is less then $\frac{0.05}{28} = 0.0018$. The null hypothesis of the Wilcoxon test is that there is no significant difference between the two techniques. From Table VIIa, it is evident that in most of the cases there is a significant difference between these approaches due to the fact that the p-value is smaller than 0.0018, out of 28 pairs of training methods, 19 are found to have significant results. From Table VIIa, it is also observed that the ensemble methods have outperformed when compared to individual models. Further, It is observed that MVE yields better results compared to other techniques.

2) All metrics and Selected Metrics: In this work, two different sets of metrics have been considered as the input to develop a model over 45 different object-oriented softwares. Eight different classification methods have been considered to develop a prediction model considering two different performance parameters, i.e. accuracy, and F-Measure. Consequently, for each set of metrics a total number of two sets



(a) Classfication Techniques



(b) Source code metrics

Fig. 6: NE_{cost}

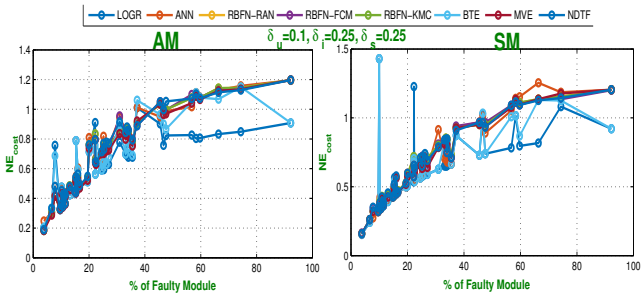


Fig. 7: NE_{cost} for $\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$

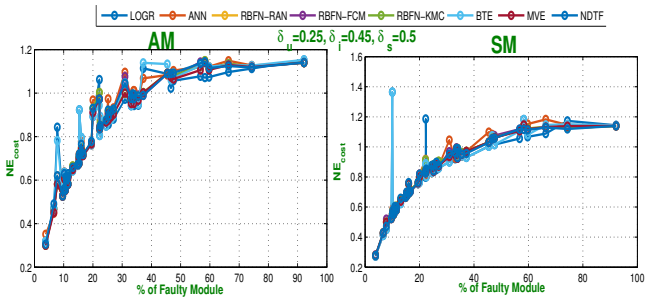


Fig. 8: NE_{cost} for $\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$

(one for each performance measure) is used, each with 360 data points (8 classification techniques * 45 datasets). The results of Wilcoxon signed rank test analysis for performance parameters are summarized in Table VIIb. From Table VIIb, it may be observed that there is a significant difference between these approaches due to the fact that the p-value is smaller than

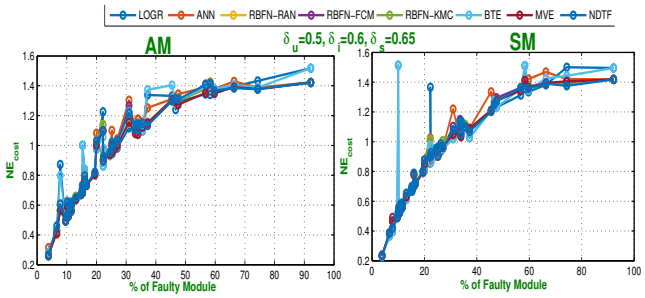


Fig. 9: NE_{cost} for $\delta_u = 0.5, \delta_i = 0.6, \delta_s = 0.65$

TABLE VIII: Threshold Value

(a) Training Methods

	Const.	Coeff.	Threshold
$\boldsymbol{\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25}$			
LOGR	-3.62	0.09	38.14
ANN	-13.16	0.27	48.75
RBFN-RAN	-23.98	0.48	49.75
RBFN-FCM	-23.98	0.48	49.75
RBFN-KCM	-23.98	0.48	49.75
BTE	-4.71	0.08	61.48
MVE	-23.98	0.48	49.75
NDTF	-24.57	0.51	47.94
$\boldsymbol{\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5}$			
LOGR	-6.67	0.17	39.64
ANN	-11.98	0.33	36.18
RBFN-RAN	-11.93	0.30	39.48
RBFN-FCM	-16.92	0.42	39.91
RBFN-KCM	-11.93	0.30	39.48
BTE	-8.47	0.21	39.55
MVE	-17.10	0.45	38.40
NDTF	-16.92	0.42	39.91
$\boldsymbol{\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65}$			
LOGR	-7.40	0.27	27.50
ANN	-10.88	0.42	25.78
RBFN-RAN	-11.56	0.46	25.17
RBFN-FCM	-18.80	0.68	27.68
RBFN-KCM	-11.56	0.46	25.17
BTE	-7.98	0.28	28.00
MVE	-12.13	0.44	27.47
NDTF	-12.26	0.47	26.23

(b) All Metrics and Selected Metrics

	Const.	Coeff.	Threshold
$\boldsymbol{\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25}$			
AM	-7.66	0.14	53.41
SM	-7.25	0.13	54.82
$\boldsymbol{\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5}$			
AM	-10.68	0.28	37.50
SM	-11.23	0.27	41.04
$\boldsymbol{\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65}$			
AM	-9.29	0.37	25.23
SM	-12.47	0.44	28.10

0.05. Yet, by judging the value of the mean difference, it is observed that the selected set of source code metrics using the proposed methods yields better results compared to all source code metrics.

D. Cost analysis

In this experiment, the normalized fault removal cost approach suggested by Wagner *et al.* [14] and the fault identi-

fication efficiencies for different testing phases suggested by Jones [4] have been used in designing of our cost evaluation model. Equations 5 and 10 are used to calculate the estimated fault removal cost (E_{cost}), and estimated testing cost (T_{cost}), respectively. Figure 6 shows the normalized fault cost of different techniques and set of metrics. From Figure 6a, it can be seen that the MVE has low median value of NE_{cost} as compare to other techniques. This shows that the fault prediction model developed using MVE method consume less fault removal cost as compare to other techniques. Similarly From Figure 6b, it is observed that model developed using selected set of metrics obtained less NE_{cost} as compare to all metrics.

Figures 7 to 9 depict the normalized fault removal cost (NE_{cost}) of fault prediction techniques for different values of δ_u, δ_i, and δ_s. From these figures, it is observed that as the percentage value of faulty classes increases, the fault-prediction technique tends to have a higher value of NE_{cost}, i.e. fault prediction can be useful for the projects with percentage of faulty classes having less than certain threshold.

E. Threshold Value

Once we identify the best performing model, We analyze the results to establish a relationship or extent of association between the NE_{cost} and the percentage of faulty classes (FP). In this study, we use logistic regression approach as our dependent variable is dichotomous for the purpose of developing a statisticalmodel to calculate the probability of usefulness of fault prediction techniques (P_{fault}). In logistic or logit regression, the dependent variable is binary and hence can take only two values. Therefore, we divide the dependent variable of a NE_{cost} into two groups: one group containing software for which fault prediction will be useful ($NE_{cost} < 1$) and another group for which the fault prediction will not be useful ($NE_{cost} \geq 1$). Table VIII displays the constant, coefficient, and threshold values in terms of the percentage of faulty classes for different values of δ_u and δ_s. We apply logit regression by setting a fixed threshold value of 0.5. The threshold value implies that the fault prediction is useful if $P_{fault} < 0.5$, otherwise the fault prediction is not useful. From Table VIII, we can observe that the fault prediction is useful for the software projects having percentages of faulty classes less than a certain threshold. For example, Table VIII reveals that the threshold value expressed in percentage for LOGR and ANN for the case of $\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$ is 38.14 and 48.75 respectively.

VII. Conclusions

We conclude that the selected set of source code metrics has a high median value in terms of accuracy for all the classifier combinations in comparison to all metrics. This shows that identifying a subset of source code metrics is important. Our findings reveal that ensemble method learning algorithm outperforms individual classifiers. We observe that the MVE approach performs the best. We also observe that fault prediction model developed using MVE method consume less fault removal cost as compare to other techniques. We conclude that our fault prediction method is also effective for software projects with a percentage of faulty classes lower than the threshold value (low - 54.82%, medium - 41.04%, high - 28.10%).

References

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[2] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of Object-Oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, October 1996.

[3] R. Battiti. First and second-order methods for learning between steepest descent and newton's method. *Neural Computation*, 4(2):141–166, 1992.

[4] J. C. Software quality in 2010: a survey of the state of the art. In *Founder and Chief Scientist Emeritus*, 2010.

[5] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[6] J.-C. Chen and S.-J. Huang. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*, 82(6):981–992, 2009.

[7] S. R. Chidamber and C. F. Kemerer. *Towards a metrics suite for Object-Oriented design*, volume 26. ACM.

[8] S. Doraisamy, S. Golzari, N. Mohd, M. N. Sulaiman, and N. I. Udzir. A study on feature selection and classification techniques for automatic genre classification of traditional malay music. In *ISMIR*, pages 331–336, 2008.

[9] G. Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.

[10] C. Furlanello, M. Serafini, S. Merler, and G. Jurman. Entropy-based gene ranking without selection bias for the predictive classification of microarray data. *BMC bioinformatics*, 4(1):1, 2003.

[11] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[12] R. Huitt and N. Wilde. Maintenance support for object-oriented programs. *IEEE Transactions on Software Engineering*, 18(12):1038–1044, 1992.

[13] W. Li and S. Henry. Maintenance metrics for the Object-Oriented paradigm. In *Proceedings of First International Software Metrics Symposium*, pages 52–60, 1993.

[14] W. S. A literature survey of the quality economics of defect-detection techniques. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)*, pages 194–203, 2006.

[15] C. Zhang, H. Wei, L. Xie, Y. Shen, and K. Zhang. Direct interval forecasting of wind speed using radial basis func-

tion neural networks in a multi-objective optimization framework. *Neurocomputing*, 2016.