Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

An effective fault prediction model developed using an extreme learning machine with various kernel methods

Lov KUMAR‡, Anand TIRKEY, Santanu-Ku. RATH

Department of Computer Science and Engineering, National Institute of Technology Rourkela, Rourkela 769008, India

E-mail: lovkumar505@gmail.com; andy9c@gmail.com; skrath@nitrkl.ac.in

Received Aug. 27, 2016; Revision accepted Feb. 21, 2017; Crosschecked July 8, 2018

Abstract: System analysts often use software fault prediction models to identify fault-prone modules during the design phase of the software development life cycle. The models help predict faulty modules based on the software metrics that are input to the models. In this study, we consider 20 types of metrics to develop a model using an extreme learning machine associated with various kernel methods. We evaluate the effectiveness of the mode using a proposed framework based on the cost and efficiency in the testing phases. The evaluation process is carried out by considering case studies for 30 object-oriented software systems. Experimental results demonstrate that the application of a fault prediction model is suitable for projects with the percentage of faulty classes below a certain threshold, which depends on the efficiency of fault identification (low: 47.28%; median: 39.24%; high: 25.72%). We consider nine feature selection techniques to remove the irrelevant metrics and to select the best set of source code metrics for fault prediction.

Key words: CK metrics; Cost analysis; Extreme learning machine; Feature selection techniques; Object-oriented software

https://doi.org/10.1631/FITEE.1601501 **CLC number:** TP311

1 Introduction

One of the most active areas of research in software engineering is software quality assurance (SQA), where careful attention is given to every stage of the software development life cycle (SDLC) in production and delivery.

Software companies invest 50% of total software development costs in SQA activities, in which defect-detection techniques are often applied to detect defective modules (Wagner, 2006). Measuring the quality of software and understanding its relationship to the source code metrics help developers control and estimate reliability through an analysis of the source code. Many software metrics have been proposed by different researchers for measuring software source code. The most commonly used metric suites are from McCabe (1976), Halstead (1977), Li and Henry (1993), Lorenz and Kidd (1994), Chidamber and Kemerer (1994), and Briand et al. (2000). In this study, we consider 20 object-oriented (OO) metrics to develop a model using an extreme learning machine (ELM) with various kernel methods, i.e., linear kernel, polynomial kernel, sigmoid kernel, and radial basis function (RBF) kernel, to predict whether the class in an OO software system is faulty.

Since fault prediction models are constructed using software metrics, the performance of the models depends very much on the choice of the right kinds of software metrics. In this study, we consider nine feature selection methods to find a small subset of OO metrics out of the total available OO metrics to predict, with a higher accuracy and a reduced misclassification error, how prone to fault various classes are.

‡ Corresponding author

From a literature survey (Goyal et al., 2014; Abaei et al., 2015; Erturk and Sezer, 2015), it is observed that several researchers focused on improving the performance of fault prediction models; however, few studies (Jiang et al., 2008; Mende and Koschke, 2009, 2010; Arisholm et al., 2010) focused on the effectiveness of fault prediction. In the present work, we propose a framework for performing a cost analysis of the fault prediction models to evaluate their effectiveness in predicting faulty classes. We perform experiments on 30 open-source Java projects. Each software project differs in terms of the percentage of faulty classes.

The contributions made in this study are outlined below:

1. We select a suitable set of source code metrics for fault prediction in OO software.

2. We develop a fault prediction model using ELM with various kernel methods.

3. We propose a cost-analysis framework to determine the usefulness of the fault-prediction model.

2 Related work and research background

In this section, we present a summary of studies closely related to our research and various probable research questions that guide our development of the fault prediction model. We organize the closely related studies into four categories: prediction techniques, software metrics, cost analysis models, and case studies.

2.1 Prediction techniques

Different methods have been analyzed and the relationship between OO metrics and fault proneness is derived (Table 1).

From Table 1, it is observed that a statistical method, i.e., the logistic regression method, has been used by many researchers. In this study, we consider the ELM method associated with various kernel methods, such as linear, polynomial, sigmoid, and RBF, in designing the model to predict faulty or non-faulty classes. The ELM method is adopted for its qualitative performance in dataset applications (Huang et al., 2006).

2.2 Software metrics

In the literature, it is observed that various researchers considered different sets of source code metrics to develop prediction models (Table 2). In this study, we consider 20 OO metrics of varying size, complexity, coupling, cohesion, and inheritance to develop a fault prediction model. Table 3 gives the basic definitions of the software metrics. These metrics include the metric suites proposed by Chidamber and Kemerer (1991), Li and Henry (1993), Lorenz and Kidd (1994), and Briand et al. (2000). They have been adopted by many researchers in their work.

From Table 2, independent variables are considered different subsets of OO metrics and the dependent variable represents fault proneness by different researchers. This shows that the performance of a fault prediction model depends on the software metrics which have been considered the input in designing a model. Selecting the right sets of features is an important data preprocessing task in different data mining and machine learning systems. There have been various feature selection techniques applied in different domains (Forman, 2003; Furlanello et al., 2003; Doraisamy et al., 2008). In the area of software fault prediction, few researchers have considered different feature selection techniques when selecting a suitable set of software metrics. Gao et al. (2011) used seven filter-based feature-ranking techniques and three filter-based subset selection search algorithms for software quality modeling. They applied these algorithms on a large real-world software system. They observed that feature selection algorithms help speed up the overall process, and also improve the prediction performances. Aggarwal et al. (2009) conducted an empirical validation of OO metrics. Initially they computed the interrelationships among the selected metrics and then the individual and combined effects of the selected metrics on fault proneness. They applied these algorithms to three small software systems. They concluded that the OO metrics identified in the predicted model appear to be well suitable for developing quality benchmarks. Shatnawi and Li (2008) investigated whether the OO metrics could predict the class error probability in the post-release evolution of Eclipse. The limitations of the above studies are that the researchers considered a small amount of data to validate their

Table 1 Summary of the literature on reliability prediction

Literature	Method
Briand et al. (2000)	Logistic regression and principal component analysis
Cartwright and Shepperd (2000)	Spearman's rank correlation
El Emam et al. (2001)	Logistic regression
Gyimothy et al. (2005)	Decision tree, linear and logistic regression, and neural network
Nagappan et al. (2005)	Multiple linear regression and principal component analysis
Zhou and Leung (2006)	Logistic regression, Naive Bayes, and random forest
Olague et al. (2007)	Logistic regression
Kanmani et al. (2007)	Logistic regression, neural network, and probabilistic neural network
Pai and Dugan (2007)	Bayesian linear regression, Bayesian Poisson regression, multiple regression, and ordinary least squares
Tomaszewski et al. (2007)	Linear regression and expert estimation
Aggarwal et al. (2009)	Statistical regression analysis and principal component analysis
Camargo Cruz and Ochimizu (2009)	Logistic regression
Singh et al. (2010)	Logistic regression, ANN, and decision tree
Zhou et al. (2010)	Logistic regression
Fokaefs et al. (2011)	Decision tree analysis
Malhotra and Singh (2011)	ANN, boosting, bagging, Naive Bayes, random forest, and Kstar
Mishra and Shukla (2012)	SVM, fuzzy, and naive Bayes
Malhotra and Jain (2012)	Logistic regression, random forest, Adaboost, bagging, multilayer perceptron, SVM, and genetic programming
Kapila and Singh (2013)	Bayesian inference
Goyal et al. (2014)	K-nearest neighbor (KNN) regression
Jing et al. (2014b)	Collaborative representation classification
Jing et al. (2014a)	Cost-sensitive discriminative dictionary learning
Erturk and Sezer (2015)	ANN, SVM, and ANFIS
Abaei et al. (2015)	Semi-supervised hybrid self-organizing map (HySOM)
Jing et al. (2015)	Canonical correlation analysis (CCA), effective transfer learning method
Wang et al. (2016)	Multiple kernel ensemble learning
Jing et al. (2017)	Improved subclass discriminant analysis (ISDA), semi-supervised transfer component analysis (SSTCA), and SSTCA+ISDA

proposed feature selection techniques. Consequently, it is not clear whether a particular conclusion based on a set of selected source code metrics could be generalized to other software systems. In this study, we apply nine selection methods to 30 datasets to find the best subset of OO metrics, which helps predict fault proneness with higher accuracy and reduce the misclassification errors.

2.3 Cost analysis model

Based on our analysis, we observe that many researchers have used different parameters to evaluate the performance of fault prediction models (Li and Henry, 1993; Malhotra and Jain, 2012). Some of the widely used parameters are accuracy, mean absolute error, F-measure, precision, recall, and area under curve (AUC); however, these parameters do not clearly indicate parameters based on software development cost. It is further observed that few researchers presented cost-based evaluation models for predicting the effectiveness of fault prediction (Table 4). From Table 4, it may be observed that some researchers have focused only on a cost analysis of the fault prediction model developed for comparing the different models. Considering the importance of cost, we propose a cost-based evaluation framework to assess the usability of the fault prediction models developed; in other words, an attempt has been made to assess the influence of the cost of fault removal, to know whether performing fault prediction analysis is useful.

2.4 Case study

In the literature, it is observed that researchers have used different case studies to validate their proposed work (Table 5) and a limited number of software systems to investigate the relationships between fault proneness and OO metrics. Consequently, there is some doubt about whether a particular conclusion based on a small number of

Table 2 Summary of the literature on fault prediction using different metrics

Literature	Metrics (independent variables)
Briand et al. (2000)	28 coupling, 10 cohesion, and 11 inheritance metrics
Cartwright and Shepperd (2000)	ATTRIB, DELS, DFCT, EVNT, LOC, NOC, READS, RWD.DIT, STATES, and WRITES
El Emam et al. (2001)	CK and Briand metrics
Gyimothy et al. (2005)	CK metric suite, LCOM3, and LOC
Nagappan et al. (2005)	STREW-J metric suite
Zhou and Leung (2006)	CK metric suite and SLOC
Olague et al. (2007)	CK metrics, MOOD, and QMOOD metrics
Kanmani et al. (2007)	10 cohesion, 18 inheritance, 29 coupling, and 7 size measures
Pai and Dugan (2007)	CK metric suite and SLOC
Tomaszewski et al. (2007)	CK metric suite
Aggarwal et al. (2009)	Coupling, cohesion, inheritance, and size metrics
Camargo Cruz and Ochimizu (2009)	RFC, CBO, and WMC
Singh et al. (2010)	CK metric suite and SLOC
Zhou et al. (2010)	LOC, AMC, aVGloc, CCMax, NCM, NTM, NIM, SDMC, NLM, and WMC
Fokaefs et al. (2011)	WMC, NOC, LCOM, and CBO
Malhotra and Singh (2011)	CK metric suite, Ce, Ca, NPM, LOC, MOA, LCOM3, MFA, DAM, CAM, IC, CBM, and MVG
Mishra and Shukla (2012)	Complexity metrics (FOUT, MLOC) and abstract syntax tree based metrics
Malhotra and Jain (2012)	CK metric suite, Ce, Ca, NPM, LOC, MOA, LCOM3, MFA, DAM, CAM, IC, CBM, and MVG
Kapila and Singh (2013)	CBO and NOC
Goyal et al. (2014)	CK metrics and 11 OO metrics
Jing et al. (2014b)	5 LOC metrics, 3 McCabe metrics, and 12 Halstead metrics
Jing et al. (2014a)	5 LOC metrics, 3 McCabe metrics, and 12 Halstead metrics
Erturk and Sezer (2015)	McCabe metrics
Abaei et al. (2015)	5 LOC metrics, 3 McCabe metrics, and 12 Halstead metrics
Abaei et al. (2015)	Semi-supervised hybrid self-organizing map (HySOM)
Jing et al. (2015)	LOC metrics, McCabe metrics, Halstead metrics, and CK metrics
Wang et al. (2016)	5 LOC metrics, 3 McCabe metrics, and 12 Halstead metrics
Jing et al. (2017)	5 LOC metrics, 3 McCabe metrics, and 12 Halstead metrics

software systems can be generalized to other systems. We consider a case study of 30 software systems to come to a generalized conclusion. We have used dataset from the PROMISE Repository (http://openscience.us/repo/defect/). Table 6 shows the project name, the number of classes, the number of faulty classes, and the percentage of faulty classes.

2.5 Effectiveness of metrics

The objective of this experiment is to investigate the relationship between OO metrics and fault proneness of associated classes. We consider a fault in a class a dependent variable, and each of the software metrics is considered an independent variable. To analyze the effectiveness of the metrics used, the metrics are categorized into different groups as follows:

Analysis 1 (A1) The effectiveness of OO metrics is used to predict class-level fault proneness. The relationship between faults and metrics can be represented in the following manner:

$$\begin{aligned}\text{faults} = f(&\text{WMC, CBO, DIT, RFC, NOC,}\\ &\text{LCOM, Ce, Ca, NPM, LOC, LCOM3,}\\ &\text{DAM, MOA, MFA, IC, CAM, AMC,}\\ &\text{Max-CC, CBM, Avg-CC}).\end{aligned}$$

Analysis 2 (A2) Reduced feature attributes using feature selection methods are considered the input in developing the model for predicting class-level fault proneness in OO software. The model can be represented as

$$\begin{aligned}\text{faults} = \ & f(\text{reduced subset of metrics using}\\ & \text{feature selection methods}).\end{aligned}$$

2.6 Research questions

Based on our analysis of the research conducted by various researchers, the following are the research questions concerning the issue of fault prediction:

Table 3 Software metrics

OO metric	Description
Weighted methods per class (WMC)	Sum of the complexities of methods defined in a class
Depth of inheritance tree (DIT)	Maximum height of the class hierarchy
Number of children (NOC)	Number of immediate descendants of the class
Coupling between object classes (CBO)	Number of classes coupled to a given class
Response for a class (RFC)	Number of different methods that can be executed when an object of that class receives a message
Lack of cohesion in methods (LCOM)	Number of the sets of methods in a class that are not related through the sharing of some of the class's fields
Afferent coupling (Ca)	Number of other classes using the specific class
Efferent coupling (Ce)	Number of classes used by the specific class
Number of public methods (NPM)	Number of methods in a class that are declared to be public
LCOM3	Lack of cohesion in methods with Henderson-Sellers version
Lines of code (LOC)	Number of lines in the text of the source code
Data access metric (DAM)	Ratio of the number of private (protected) attributes to the total number of attributes declared in the class
Measure of aggregation (MOA)	Number of data declarations, whose types are user-defined classes
Measure of functional abstraction (MFA)	Ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class
Cohesion among methods (CAM)	Sum of the number of different types of method parameters in every method divided by multiplication of the number of different method parameter types in the whole class and number of methods
Inheritance coupling (IC)	Number of parent classes to which a given class is coupled
Coupling between methods (CBM)	Number of methods to which all the inherited methods are coupled
Average method complexity (AMC)	Average method size for each class
Maximum McCabe's cyclomatic complexity (Max-CC)	Maximum cyclomatic complexity of methods defined in a class
Average McCabe's cyclomatic complexity (Avg-CC)	Average cyclomatic complexity of methods defined in a class

Table 4 Fault prediction effectiveness based on the cost evaluation model

Literature	Cost evaluation criterion
Jiang et al. (2008)	Introducing a cost curve based on the receiver operating characteristic (ROC)
Mende and Koschke (2009)	Introducing a performance measure (P_{opt}) and comparing the prediction model with an optimal model. P_{opt} accounts for the module size to evaluate the performance of a fault-prediction technique
Mende and Koschke (2010)	Proposing effort-aware binary prediction and the effort-aware prediction based on defect density for evaluating the effort awareness in fault-prediction techniques

RQ1: Which fault prediction technique is the most suitable among the different techniques?

This question helps investigate the performance of different types of fault-prediction techniques. In this study, we consider different types of fault-prediction techniques in developing a model where OO software metrics are the input for predicting whether the class is faulty.

RQ2: For a given software product, is performing fault prediction analysis economically effective?

This question investigates the effectiveness of different fault prediction techniques. In this work, a cost evaluation framework is proposed which performs cost-based analysis for misclassification of faults.

RQ3: Is a subset of OO software metrics used better at predicting whether a class is faulty?

This step is aimed to evaluate the metrics for testing their relationship with faulty and non-faulty classes. In this study, we consider different types of feature reduction methods to find a subset of OO software metrics that is better at predicting whether the class is faulty.

RQ4: Which feature-ranking methods work best in predicting whether a class is faulty?

In the feature-ranking method, performance is based on the nature of the fault dataset. Different methods have been considered with different param-

Table 5 Summary of case studies in the literature

Literature	Dataset
Briand et al. (2000)	Hypothetical video rental business
Cartwright and Shepperd (2000)	Large European telecommunication industry consisting of 32 classes and 133 KLOC
El Emam et al. (2001)	Two versions of Java applications
Gyimothy et al. (2005)	Mozilla source code with the Columbus framework
Nagappan et al. (2005)	Open-source Eclipse plug-in
Zhou and Leung (2006)	NASA comprising 145 classes, 2107 methods, and 40 KLOC
Olague et al. (2007)	Mozilla Rhino project
Kanmani et al. (2007)	Library management system consisting of 1185 classes
Pai and Dugan (2007)	Public domain dataset comprising 145 classes
Tomaszewski et al. (2007)	Two telecommunication projects developed by Ericsson, comprising 800 classes (500 KLOC) and 1000 classes (600 KLOC), respectively
Aggarwal et al. (2009)	Student projects at University School of Information Technology (USIT)
Camargo Cruz and Ochimizu (2009)	638 classes of Mylyn software
Singh et al. (2010)	NASA comprising 145 classes, 2107 methods, and 40 KLOC
Zhou et al. (2010)	Three releases of Eclipse, consisting of 6751, 7909, and 10 635 Java classes and 796, 988, and 1306 KLOC, respectively
Fokaefs et al. (2011)	NASA datasets
Malhotra and Singh (2011)	Open-source software
Mishra and Shukla (2012)	Eclipse and Equinox datasets
Malhotra and Jain (2012)	Apache POI
Kapila and Singh (2013)	Open-source Eclipse system
Goyal et al. (2014)	Eclipse, Mylyn, Equinox, and PDE
Jing et al. (2014b)	Ten datasets from NASA
Jing et al. (2014a)	CM1, JM1, KC1, KC3, MC2, MW1, PC1, PC3, PC4, and PC5
Erturk and Sezer (2015)	PROMISE software engineering repository data
Abaei et al. (2015)	NASA datasets
Jing et al. (2015)	CM1, MW1, PC1, AR3, AR4, AR5, Apache, OpenIntents Safe, Zxing, EQ, JDT, LC, ML, and PDE
Wang et al. (2016)	CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, and PC5
Jing et al. (2017)	EQ, JDT, LC, ML, PDE, CM1, JM1, KC1, KC3, MC2, MW1, PC1, PC3, PC4, PC5, Apache, Safe, ZXing, AR3, AR4, and AR5

eters to rank the features.

RQ5: Which feature subset selection method works best in predicting whether a class is faulty?

Each feature subset selection method uses a different way to find the subset of features that are better at predicting whether a class is faulty.

RQ6: How do the feature-ranking methods compare with feature subset selection methods?

In this study, we employ the pairwise t-test to determine whether feature-ranking methods work better than feature subset selection methods or if they perform equally well.

RQ7: Do the feature selection methods affect the performance of the classification methods?

It is observed that, some of the feature selection methods quite possibly work very well with specific classification methods. Thus, in this study, different feature selection methods are evaluated using different classification methods to assess their performances. This question focuses on variation in the performance of the classification methods over different sets of software metrics.

3 Feature selection methods

The performance of a prediction model is highly influenced by the reliability of the dataset, based on software metrics and fault information. Hence, the selection of the right set of software metrics becomes an important step in the reliability prediction process. The following subsections highlight the different feature selection methods in finding a small subset of OO metrics out of the total available OO metrics that collectively have good predictive capabilities. Feature selection methods can be broadly classified into two subclasses: feature-ranking methods and feature subset selection methods.

Table 6 Project used in our case study

Project ID	Project name	Number of classes	Number of faulty classes	Percentage (%)
DS1	Ant	745	166	22.28
DS2	Arc	234	27	11.53
DS3	Berek	43	16	37.20
DS4	Camel	965	188	19.48
DS5	Elearning	64	5	7.81
DS6	Forrest	32	2	6.25
DS7	Intercafe	27	4	14.81
DS8	Ivy	352	40	11.36
DS9	Jedit	492	11	2.23
DS10	Kalkulator	27	6	22.22
DS11	Log4j	205	189	92.19
DS12	Lucene	340	203	59.70
DS13	Nieruchomosci	27	10	37.03
DS14	Pbeans	51	10	19.60
DS15	Pdftranslator	33	15	45.45
DS16	Poi	442	281	63.57
DS17	Prop	660	66	10
DS18	Redaktor	176	27	15.34
DS19	Serapion	45	9	20
DS20	Skarbonka	45	9	20
DS21	Synapse	256	86	33.59
DS22	Systemdata	65	9	13.84
DS23	Szybkafucha	25	14	56
DS24	Termoproject	42	13	30.95
DS25	Tomcat	858	77	8.97
DS26	Velocity	229	78	34.06
DS27	Workflow	39	20	51.28
DS28	Xalan	909	898	98.78
DS29	Xerces	588	437	74.31
DS30	Zuzel	29	13	44.82

3.1 Feature-ranking methods

Feature-ranking methods rank features independently without using any learning algorithm. In the method, ranking of features is based on the score of the features. In this work, we consider five feature-ranking methods for computing the score of feature. A small note on each of the feature-ranking methods is given below:

1. Chi-squared test

Chi-squared test is used to test the independence between two events (Plackett, 1983), where the ranking of features is based on the value of the Chi-squared statistics with respect to the class.

2. Gain ratio feature evaluation

In the gain ratio feature evaluation method, the ranking of features is based on the value of the gain ratio with respect to the class (Novakovic, 2010).

3. OneR feature evaluation

In the OneR feature evaluation method, the classifier is considered for ranking the features (Novakovic, 2010). The OneR classifier uses classification rates to rank the features. The features with higher classification rates are considered to be more significant.

4. Logistic regression analysis

Logistic regression (LR) analysis is a statistical analysis method (Chidamber and Kemerer, 1991). In this analysis, univariate logistic regression (ULR) analysis is used to check the level of significance of each metric. In this study, two parameters, i.e., regression coefficient value and p-value of the LR model, are used to find the level of significance for each metric.

5. Principal component analysis

Attribute reduction is achieved using principal component analysis (PCA) by transforming a high dimensional data space into a lower dimensional data space. The lower dimensional data has the most significant features (Wang and Romagnoli, 2005). The transferred metrics are called principal component domain metrics.

3.2 Feature subset selection method

The feature-subset selection method is used to find a right set of features that are relevant in terms of high predictive power. The concept of feature-subset selection method is based on the assumption that a model has better performance when filtering irrelevant features that have little or no impact on the classifier accuracy. In this study, we consider four feature subset selection methods to remove irrelevant features and select the right sets of features for fault prediction. The feature subset selection methods are described below.

1. Correlation-based feature selection

The correlation-based feature selection (CFS) method is based on the hypothesis that the right set of features is one that contains features having a high predictive correlation with the class, and low correlation with each other. We compute the correlation between all source code metrics using the Pearson correlation coefficient (r). This aspect suggests that, although the source code metrics measure different properties of the class design, there is a significant statistical reason to believe that classes with high (or low) source code metric values also have high (or low) values for other highly correlated source code metrics.

2. Classifier subset evaluation

The classifier subset evaluation method uses a classifier method to estimate the 'merit' of the possible subsets of features in the project (Dash and Liu, 2003). The 'merit' is considered the minimum classification error. It uses a search method which finds a small subset of features assessed using the evaluation method. In this study, we consider the naive Bayes classifier for classification purpose.

3. Filtered subset evaluation

Filtered subset evaluation is a method for running a random subset evaluator on a dataset that has passed through an arbitrary filter (Kohavi and John, 1997). The filter approach does not depend on a learning induction algorithm. So, the computation of the filter approach is observed to be fast and scalable.

4. Rough set analysis

Rough set theory is a formal approximation of a conventional (CRISP) set (Pawlak, 1982). This formal approximation represents the lower and upper bounds of the original set. The application of rough set analysis (RSA) makes data patterns more visible by lowering the 'degree of precision' (Slowinski, 1992). In general, RSA can be viewed as a formal framework for mining facts from imperfect data. In this analysis, RSA is used to find a reduced feature set of OO metrics.

4 Extreme learning machine classifier

In this work, we consider ELM with various kernels as a classifier for developing a model to classify faulty and non-faulty classes. An ELM represents an algorithm in which hidden neurons are randomly generated and these hidden neurons need not be changed, which helps reduce the training time and improve the learning speed (Huang et al., 2006). Fig. 1 shows the basic architecture of an ELM, which contains three layers, i.e., the input layer, hidden layer, and output layer, similar to those of a single hidden layer feed-forward neural network.

The output function O_n of ELM can be represented as

$$O_n(\boldsymbol{I}) = \sum_{i=1}^{n} \beta_i F(a_i, b_i, \boldsymbol{I}) = \boldsymbol{\beta}^{\mathrm{T}} h(\boldsymbol{I}), \qquad (1)$$

where n is the number of hidden layer neurons and $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_n]^{\mathrm{T}}$ is the weight vector with the i^{th} item representing the connection between the i^{th} hidden neuron and the output neuron. $F(a_i, b_i, \boldsymbol{I})$ is the output/activation function of the i^{th} hidden neuron and $h(\boldsymbol{x}) = [F(a_1, b_1, \boldsymbol{I}), F(a_2, b_2, \boldsymbol{I}), \ldots, F(a_n, b_n, \boldsymbol{I})]$ is hidden layer mapping.

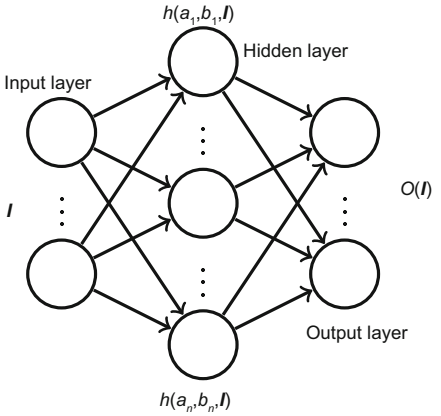

Fig. 1 Structure of an extreme learning machine

Let $\boldsymbol{T}_{\mathrm{s}}$ be the training sample such that $\boldsymbol{T}_{\mathrm{s}} = (I_i, O_i)$, where $i = 1, 2, \ldots, n$ with n being the number of hidden neurons and $F(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{I})$ the hidden neuron activation function. The output function of ELM can be formulated using the following equation:

$$O_n(\boldsymbol{I}) = \sum_{i=1}^{n} \beta_i F(a_i, b_i, \boldsymbol{I}) = O_j, \qquad (2)$$

where $j = 1, 2, \ldots, N$. Eq. (2) can be expressed as

$$\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{O}, \qquad (3)$$

where

$$\boldsymbol{H} = \begin{bmatrix} h(\boldsymbol{I}_1) \\ \vdots \\ h(\boldsymbol{I}_N) \end{bmatrix} = \begin{bmatrix} F(a_1, b_1, I_1) & \ldots & F(a_n, b_n, I_1) \\ \vdots & & \vdots \\ F(a_1, b_1, I_n) & \ldots & F(a_n, b_n, I_n) \end{bmatrix}. \qquad (4)$$

In Eq. (4), $\boldsymbol{H}$ is the hidden layer output matrix, with the i^{th} column of $\boldsymbol{H}$ being the output of the i^{th} hidden neuron with respect to inputs $\boldsymbol{I}_1, \boldsymbol{I}_2, \ldots, \boldsymbol{I}_N$. Output weight vector $\boldsymbol{\beta}$ can be calculated as

$$\boldsymbol{\beta} = \boldsymbol{H}^{\dagger}\boldsymbol{O}, \qquad (5)$$

where $\boldsymbol{H}^{\dagger}$ stands for the Moore-Penrose generalized inverse of the hidden layer output matrix $\boldsymbol{H}$. As discussed, until now the basic ELM resultant solution is not stable. To make it stable as well as general, the regularization factor is set to C. As per ridge regression theory guidelines, a positive value $1/C$ may

be added to the diagonal of $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ or $\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}}$ of the Moore-Penrose generalized inverse $\boldsymbol{H}$, which helps in making ELM stable. After finding the value of C, the output weight vector $\boldsymbol{\beta}$ can be calculated using the following equation:

$$\boldsymbol{\beta} = \boldsymbol{H}^{\mathrm{T}}\left(\frac{1}{C} + \boldsymbol{H}\boldsymbol{H}^{\mathrm{T}}\right)^{-1}\boldsymbol{O}. \tag{6}$$

The output function of ELM corresponding to output weight vector $\boldsymbol{\beta}$ is represented as

$$O(\boldsymbol{I}) = h(\boldsymbol{I})\boldsymbol{\beta} = h(\boldsymbol{I})\boldsymbol{H}^{\mathrm{T}}\left(\frac{1}{C} + \boldsymbol{H}\boldsymbol{H}^{\mathrm{T}}\right)^{-1}\boldsymbol{O}. \tag{7}$$

When hidden layer mapping is known, we can use a basic ELM; otherwise, a kernel-based ELM is used. Kernel matrix $\boldsymbol{\Omega}$ for ELM can be defined as follows:

$$\Omega_{ij} = h(\boldsymbol{I}_i)h(\boldsymbol{I}_j) = k(\boldsymbol{I}_i, \boldsymbol{I}_j). \tag{8}$$

The output function $O_n(\boldsymbol{I})$ for a kernel-based ELM is expressed as

$$O(\boldsymbol{I}) = \begin{bmatrix} k(\boldsymbol{I}, \boldsymbol{I}_1) \\ \vdots \\ k(\boldsymbol{I}, \boldsymbol{I}_N) \end{bmatrix}\left(\frac{1}{C} + \boldsymbol{\Omega}_{\mathrm{ELM}}\right)^{-1}\boldsymbol{O}. \tag{9}$$

In this study, we consider four types of kernels to develop a classifier for classifying faulty and non-faulty classes. These kernels are as follows:

1. Linear kernel: $k(\boldsymbol{I}_i, \boldsymbol{I}_j) = \boldsymbol{I}_i^{\mathrm{T}}\boldsymbol{I}_j$.

2. Polynomial kernel: $k(\boldsymbol{I}_i, \boldsymbol{I}_j) = (\boldsymbol{I}_i^{\mathrm{T}}\boldsymbol{I}_j + b)^{\gamma}, \gamma > 0$.

3. Sigmoid kernel: $k(\boldsymbol{I}_i, \boldsymbol{I}_j) = \tanh(\gamma\boldsymbol{I}_i^{\mathrm{T}}\boldsymbol{I}_j + \gamma)$.

4. RBF kernel: $k(\boldsymbol{I}_i, \boldsymbol{I}_j) = \exp(-\gamma||\boldsymbol{I}_i - \boldsymbol{I}_j||^2), \gamma > 0$.

5 Cost analysis model

This section describes the construction of a cost evaluation model, which accounts for the realistic costs required to remove a fault and compute the estimated fault removal cost for a specific fault prediction technique based on the concept proposed by Wagner (2006). Certain constraints are assumed in designing this cost evaluation model:

1. Different phases of testing account for varying fault removal costs.

2. It is not possible to fully detect all faults within one testing phase.

3. It is not possible in practice to perform unit testing on all classes.

The normalized fault removal cost approach suggested by Wagner (2006) is applied to formulate the proposed cost evaluation model. The cost varies because various projects are developed on different platforms following several organizational standards. The normalized fault removal costs are summarized in Table 7. Fault identification efficiencies for different testing phases are taken from Jones (2010). The testing phase efficiencies are summarized in Table 8. Huitt and Wilde (1992) stated that more than 50% of the classes are usually very small; hence, performing unit testing on these classes may not be very helpful.

Table 7 Removal costs for test techniques (in staff hours per defect)

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 8 Fault identification efficiencies for different 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 for E_{cost}, T_{cost}, and $\mathrm{NE}_{\mathrm{cost}}$ of the proposed cost-based evaluation framework are discussed, considering various notations as mentioned below:

1. C_{i}: initial setup cost of the fault-prediction technique used. C_{u}, C_{i}, C_{s}, and C_{f} are the normalized fault removal costs in unit, integration, system, and field testing, respectively. M_{p} is the percentage of classes unit tested.

2. δ_{u}, δ_{i}, and δ_{s} are the fault identification efficiencies of unit, integration, and system testing, respectively.

3. FC and TC are the number of faulty modules and the total number of modules in the software projects, respectively. TN, FN, FP, and TP are the true negative, false negative, false positive, and true positive, respectively.

Estimated fault removal cost (E_{cost}): The detailed analyses of computation of the estimated fault removal cost of the software when fault prediction is

performed (E_{cost}) are given as follows:

1. The total number of faulty classes identified by the predictor is equal to the summation of true positive (TP) and false positive (FP) values. Hence, it is necessary to compute the testing and verification costs at the class level, which indicates that the cost value is equal to the cost of unit testing (C_{u}). The total cost of unit testing of the software system is defined as

$$\text{Cost}_{\text{u}} = (\text{TP}+\text{FP})C_{\text{u}}. \tag{10}$$

2. As it is not possible to identify all faults 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 likelihood that these faulty classes and the faulty classes predicted as non-faulty classes (count of FN) are identified by the predictor in the later testing phases, such as in the integration (C_{i}), system, and field testing. The fault removal cost in integration, system, and field testing may be computed as

$$\text{Cost}_{\text{i}} = C_{\text{i}}\delta_{\text{i}}(\text{FN}+\text{TP}(1-\delta_{\text{u}})), \tag{11}$$

$$\text{Cost}_{\text{s}} = \delta_{\text{s}}C_{\text{s}}(1-\delta_{\text{i}})(\text{TP}(1-\delta_{\text{u}})+\text{FN}), \tag{12}$$

$$\text{Cost}_{\text{f}} = (1-\delta_{\text{s}})C_{\text{f}}(1-\delta_{\text{i}})(\text{TP}(1-\delta_{\text{u}})+\text{FN}). \tag{13}$$

3. The estimated overall fault removal cost can be determined as

$$\begin{aligned} E_{\text{cost}} =& C_{\text{i}} + C_{\text{u}}(\text{FP}+\text{TP}) + \delta_{\text{i}}C_{\text{i}}(\text{TP}(1-\delta_{\text{u}})+\text{FN}) \\ &+ \delta_{\text{s}}C_{\text{s}}(1-\delta_{\text{i}})(\text{TP}(1-\delta_{\text{u}})+\text{FN}) \\ &+ (1-\delta_{\text{s}})C_{\text{f}}(1-\delta_{\text{i}})(\text{TP}(1-\delta_{\text{u}})+\text{FN}). \end{aligned} \tag{14}$$

Estimated testing cost (T_{cost}): The detailed analyses of computation of the estimated fault removal cost of the software without using a fault prediction approach (T_{cost}) are given as follows:

1. In testing, if fault prediction analysis is not carried out, then the tester often performs unit testing on all the classes. Therefore, the total unit testing may be computed as

$$\text{Cost}_{\text{u}} = M_{\text{p}}C_{\text{u}}\text{TC}. \tag{15}$$

2. Furthermore, there is a possibility that some of the faulty classes that remain undetected in unit testing may be identified in integration, system, and field testing. The total integration, system, and field testing cost may be computed as

$$\text{Cost}_{\text{i}} = \delta_{\text{i}}C_{\text{i}}(1-\delta_{\text{u}})\text{FC}, \tag{16}$$

$$\text{Cost}_{\text{s}} = \delta_{\text{s}}C_{\text{s}}(1-\delta_{\text{i}})(1-\delta_{\text{u}})\text{FC}, \tag{17}$$

$$\text{Cost}_{\text{f}} = (1-\delta_{\text{s}})C_{\text{f}}(1-\delta_{\text{i}})(1-\delta_{\text{u}})\text{FC}. \tag{18}$$

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

$$\begin{aligned} T_{\text{cost}} =& M_{\text{p}}C_{\text{u}}\text{TC} + \delta_{\text{i}}C_{\text{i}}(1-\delta_{\text{u}})\text{FC} \\ &+ \delta_{\text{s}}C_{\text{s}}((1-\delta_{\text{i}})(1-\delta_{\text{u}})\text{FC} \\ &+ (1-\delta_{\text{s}})C_{\text{f}}(1-\delta_{\text{i}})(1-\delta_{\text{u}})\text{FC}. \end{aligned} \tag{19}$$

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

$$\text{If the value of } \text{NE}_{\text{cost}} \begin{cases} < 1, \text{then application of fault prediction is useful,} \\ \geq 1, \text{then application of testing methodologies may be helpful.} \end{cases} \tag{20}$$

6 Experimental setup and results

In this section, an experimental study is presented to find the effectiveness of fault prediction techniques using the cost-based evaluation framework. In this work, extreme learning with four kernel methods is considered in the development of a model to predict whether the class is faulty. All techniques are tested using 30 software systems chosen from the PROMISE data repository (Table 6). These projects have varying percentages of faulty classes which are sufficient for performing an analysis.

6.1 Experimental setup

In this experiment, the values tabulated in Tables 7 and 8 are used in the design of the cost evaluation model. The objective is to provide benchmarks to approximate the overall fault removal cost. Fig. 2 shows the flowchart for the proposed cost-based evaluation framework. The steps mentioned below are followed while developing an effective fault prediction model. Each of the feature selection methods (both feature ranking and feature subset selection) is applied to 30 datasets. Therefore, a total of 1200 ((9 feature selection methods + 1 considering all features) × 30 datasets × 4 prediction techniques) distinct prediction models are developed in the study.

1. In this study, five feature-ranking methods and four feature subset selection methods are applied to find a suitable set of OO metrics on 30 OO software systems.

2. The subsets of OO metrics obtained from the above step are evaluated using an extreme learning method with four kernel functions. In this study, cross validation is used to compare the models. Cross validation is a statistical learning method used to evaluate and compare the models by partitioning the data into two portions. In the K-fold cross-validation technique, the data is first partitioned into K equally (or nearly equally) sized portions or folds. For each model, $K-1$ folds are used for training and the remaining one fold is used for testing (Kohavi, 1995). The significance of K-fold cross validation lies in its ability to use the dataset for both training and testing. In this work, 10-fold cross validation is used to compare the models; i.e., datasets are divided into 10 parts. Before designing the models, an analysis of the outliers is performed to eliminate the extreme values, which may unduly relegate the models. The effectiveness of outliers is examined using the following equation:

$$e_i = \begin{cases} \text{effective outliers}, & \text{if } |y_{ji} - \hat{y_j}| > 3\sigma, \\ \text{non-effective outliers}, & \text{if } |y_{ji} - \hat{y_j}| \leq 3\sigma. \end{cases} \tag{21}$$

3. The performances of all prediction models are compared using two performance evaluation parameters, accuracy and F-measure. We also validate the selected metrics to estimate the overall prediction accuracy.

4. The models developed using the above two steps are passed to the proposed cost analysis framework to determine whether the developed fault prediction model is effective for the project.

6.2 Analysis of results

In this section, the relationship between OO metrics and class-level fault proneness is determined. Software metrics are considered the input data for the model and its output is the faulty and non-faulty classes. Accuracy and F-measure are considered performance evaluation parameters to compare the performances of the fault proneness models developed using extreme learning with four kernel methods.

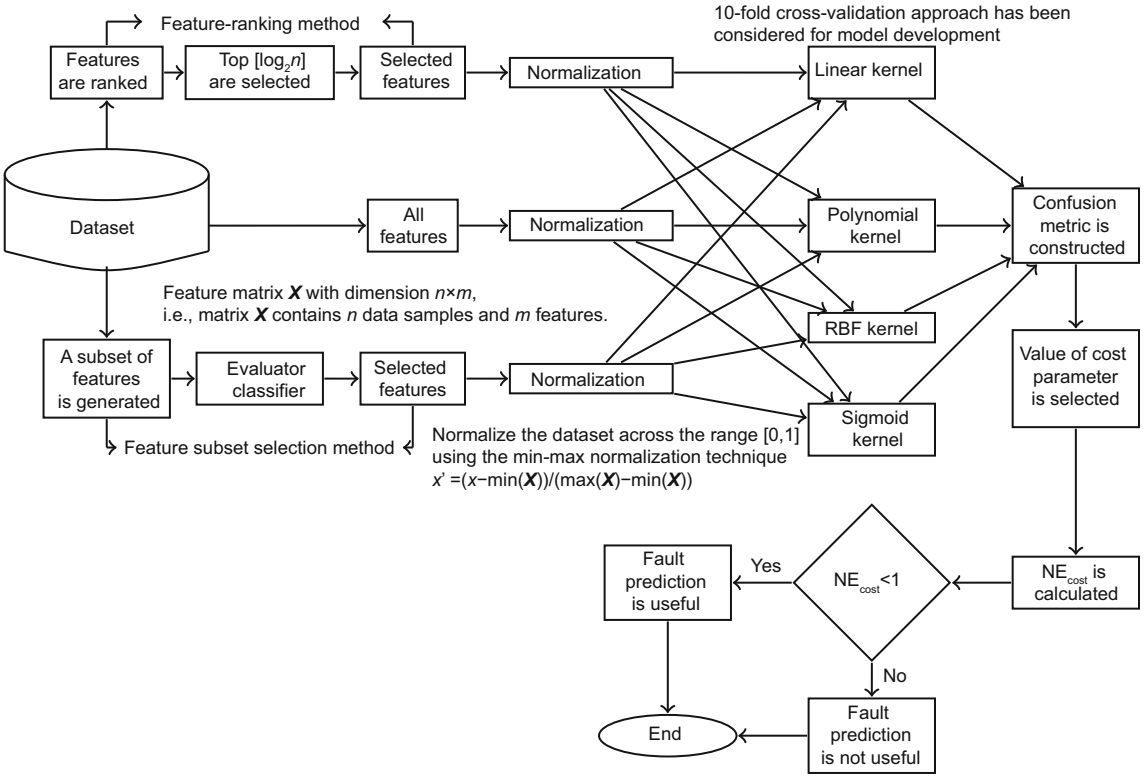

Fig. 2 Framework of the proposed cost-based evaluation framework (RBF: radial basis function)

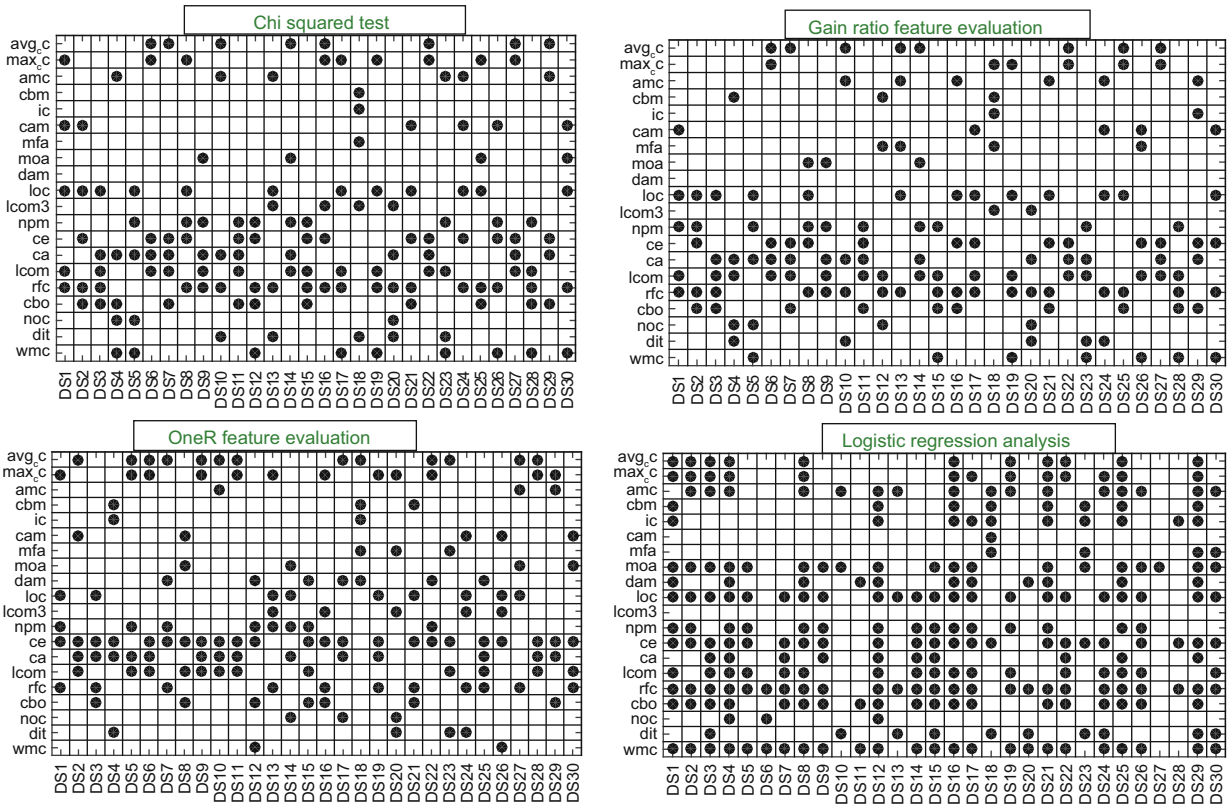

Fig. 3 Selected features using feature-ranking methods

6.2.1 Feature-ranking methods

Five feature-ranking methods, Chi-squared test, gain ratio feature evaluation, OneR feature evaluation, logistic regression (ULR) analysis, and PCA, are applied to 30 OO software systems. Each method uses different performance parameters to rank the features. The top $\lceil \log_2 n \rceil$ metrics out of n metrics are considered in developing a model to predict whether the class is faulty (Goyal et al., 2014) (in this study $n = 20$). However, in case of ULR analysis, only the metrics that have positive regression coefficients are selected, and the p-value is less than 0.05. The set of OO metrics selected using feature-ranking methods is shown in Fig. 3. The vertical and horizontal axes show the names of the selected source code metrics and the project IDs, respectively. From Fig. 3, of all source code metrics, only RFC, LCOM, LOC, CAM, and MAX-CC source code metrics are selected using the Chi-squared test for the first dataset, i.e., ant data. In case of PCA, only the principal components (PC) that have eigenvalues larger than 1 are selected. The PCA and varimax rotation concept are applied to all the OO metrics. The rotated component metrics for DS1 are tabulated in Table 9.

Table 9 shows the relationship between the original OO metrics and the domain metrics. The results for the other datasets are similar in nature.

6.2.2 Feature subset selection methods

Four feature subset selection methods are applied to 30 OO software systems to select a subset of metrics. Subsequently, the selected subsets of metrics are considered the input for developing a model to predict whether the class is faulty. The set of OO metrics selected using feature subset selection methods is shown in Fig. 4. The vertical and horizontal axes indicate the names of the selected source code metrics and project IDs, respectively.

6.2.3 Extreme learning methods

The extreme learning machine with four kernel methods has been applied to develop a model for predicting whether the class is faulty. Ten different subsets of metrics (9 feature selection methods + 1

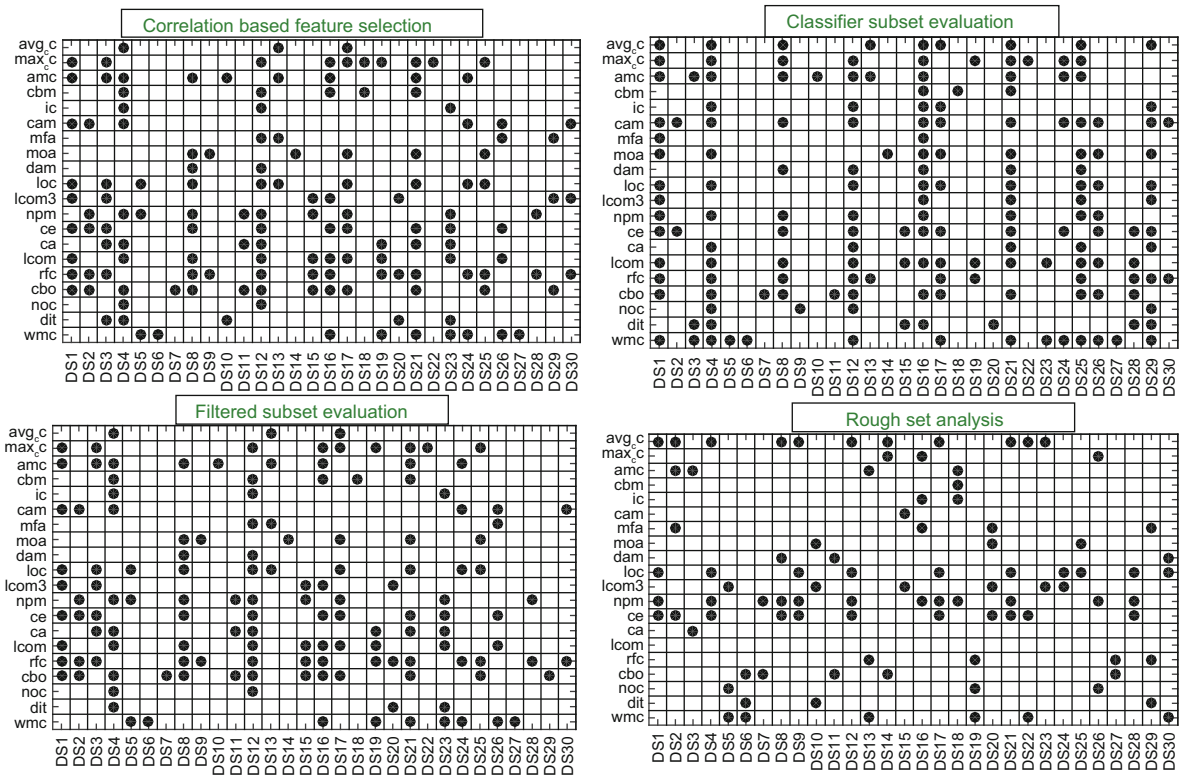

Fig. 4 Selected features using feature subset selection methods

Table 9 Principal component analysis for DS1

Metric	PC1	PC2	PC3	PC4	PC5	PC6
wmc	0.931	0.052	0.156	0.144	0.131	–0.021
dit	0.067	0.895	–0.046	0.02	–0.084	–0.036
noc	–0.053	–0.022	0.544	0.15	0.057	0.018
cbo	0.335	–0.002	0.903	–0.042	–0.007	–0.017
rfc	0.895	0.106	0.062	0.12	0.26	0.049
lcom	0.813	–0.038	0.276	–0.124	–0.066	–0.057
ca	0.193	–0.048	0.942	–0.069	–0.039	–0.019
ce	0.783	0.195	–0.033	0.11	0.142	0.005
npm	0.885	0.062	0.194	0.142	0.027	–0.07
lcom3	–0.096	–0.065	–0.059	–0.922	–0.03	0.028
loc	0.767	–0.002	0.027	0.06	0.372	0.3
dam	0.227	0.088	0.029	0.906	0.036	–0.032
moa	0.674	0.018	–0.003	0.146	0.095	–0.019
mfa	–0.088	0.83	–0.084	–0.04	–0.118	–0.03
cam	–0.548	–0.078	–0.038	–0.482	–0.182	–0.154
ic	0.168	0.84	0.003	0.111	0.11	0.021
cbm	0.124	0.773	0.045	0.11	0.154	0.028
amc	0.025	–0.017	–0.008	–0.023	0.024	0.983
$\max_c$c	0.385	0.002	0.003	0.045	0.842	0.03
avg_cc	0.148	0.032	0.028	0.063	0.916	0.005
Eigenvalues	5.507	2.873	2.161	2.074	1.903	1.098
Percentage of variance (%)	27.535	14.364	10.803	10.368	9.514	5.492

considering all features) are considered the input to develop a model for predicting class-level fault proneness in OO software. The hardware used to carry out our experiments is: Core i5 processor with 4 GB RAM and 250 GB hard disk. Prediction models are developed using a licensed MATLAB environment at NIT-Rourkela. Table 10 shows a list of abbreviations and their full names.

Table 10 Naming conventions used for different techniques

Abbreviation	Corresponding name
AM	All metrics
FR1	Chi-squared test
FR2	Gain ratio feature evaluation
FR3	OneR feature evaluation
FR4	Logistic regression analysis
FR5	Principal component analysis
FS1	Correlation based feature selection
FS2	Classifier subset evaluation
FS3	Filtered subset evaluation
FS4	Rough set analysis
C1	ELM (linear kernel)
C2	ELM (polynomial kernel)
C3	ELM (RBF kernel)
C4	ELM (sigmoid kernel)

The performance of each prediction model is evaluated in terms of accuracy and F-measure. Table 11 shows the performances obtained for different software products using the extreme learning method. From Table 11, it can be inferred that the model developed with a set of selected source code metrics from the feature selection techniques obtained better performance, i.e., high values of accuracy and F-measure, in predicting reliability compared with others.

Figs. 5 and 6 show the box-plot diagrams for accuracy and F-measure for each of the cases. The figures contain 10 types of box plots, one for all metrics, five for feature-ranking methods, and four for feature subset selection methods. Since in this study four fault prediction techniques and two performance parameters are considered for predicting whether the class is faulty, eight box-plot diagrams are displayed (one for each combination). The x-axis of the box-plot diagrams shows the name of the feature selection technique used. Box-plot diagrams help observe the performances of all feature selection methods based on a single diagram. The line in the middle of each box represents the median value. The model that has the highest median value and the smallest number of outliers is the best model for predicting whether the class is faulty. From the box-plot diagram, the following may be inferred:

1. From Figs. 5 and 6, the model developed with a set of selected source code metrics using feature selection techniques as the input has high median values for performance parameters and few outliers for each classification technique. This shows that the model developed using a set of selected source code metrics produces the best result compared to that developed using all metrics.

2. From Fig. 7, ELM with the polynomial kernel has high median values for performance parameters. This shows that ELM with the polynomial kernel yields better results compared to other kernels.

3. Regarding the feature-ranking techniques, FR3 has high median values for performance parameters (Fig. 8). This shows that FR3, i.e., the OneR feature evaluation technique, computes the best set of OO source code metrics in predicting faulty and non-faulty classes.

4. Regarding the feature subset selection techniques, FS3 has high median values for performance parameters (Fig. 8). This shows that feature subset selection using a filtered subset evaluation method computes the best set of OO metrics in predicting faulty and non-faulty classes.

6.2.4 Comparison of results

In this section, a pairwise t-test is employed to determine which one out of the different feature selection techniques and classifiers works better or if they all perform equally well.

Feature selection techniques: In this study 10 sets of metrics, i.e., one containing all metrics and nine sets of selected metrics using nine different feature selection techniques (five feature-ranking and four feature subset selection techniques), are the input to develop a model across 30 OO software systems. The extreme learning machine with four types of kernels is considered to develop a prediction model considering two performance parameters, i.e., accuracy and F-measure. So, for each feature selection technique, two sets (one for each performance measure) are used, each with 120 data points (4 classifiers $\times$ 30 datasets). The results of t-test analysis for performance parameters are summarized in Table 12. Table 12 contains two parts, the first part showing the mean difference values of performance parameters and the second part showing the p-value between different pairs. From this table, it is evident that, for most of the cases there is a significant difference between these approaches, as the p-value is lower than 0.05. However, by judging the mean difference, FS3, i.e., the selected source metrics using a filtered subset evaluation method, yields better results compared to other approaches.

Feature ranking and feature subset selection methods: In this study, a pairwise t-test is employed to determine whether feature-ranking methods work better than feature subset selection methods or if they perform equally well. Two sample pairs are considered between the average performances of the feature-ranking method and feature subset selection method. The results of t-test analysis for average performances of the feature ranking and feature subset selection techniques are summarized in Table 13. Since in this study four prediction techniques are applied to 30 OO software datasets with two performance parameters, for each feature selection technique, two sets (one for each performance measure) are used, which are feature ranking with 600 data points (5 feature-ranking techniques $\times$ 4 classifiers

× 30 datasets) and feature subset selection with 480 points (4 feature subset selection techniques × 4 classifiers × 30 datasets). From Table 13, it is evident that there is no significant difference between these two approaches, as the p-value is greater than 0.05. However, when judging the mean difference, the feature subset selection method yields better results than the feature-ranking method.

Classification methods: In this study, an extreme learning machine with four kernels is considered in the development of a model to predict whether the class is faulty. This study uses 10 subsets of metrics of 30 OO software datasets considering two performance parameters, i.e., accuracy and F-measure. So, for each prediction technique, two sets (one for each performance) are used, each with

Table 11 Performance matrix of the linear kernel (a), polynomial kernel (b), radial basis function kernel (c), and sigmoid kernel (d)

(a)

	Accuracy (%)										F-measure									
	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4
DS1	80.81	82.68	81.74	82.42	78.52	80.27	82.82	82.01	82.82	78.39	0.88	0.9	0.89	0.89	0.87	0.89	0.9	0.89	0.9	0.87
DS2	83.33	87.61	88.46	87.61	85.47	88.46	85.9	87.61	85.47	85.04	0.91	0.93	0.94	0.93	0.92	0.94	0.92	0.93	0.92	0.92
DS3	69.77	37.21	37.21	86.05	76.74	81.4	76.74	41.86	86.05	39.53	0.76	NaN	NaN	0.89	0.82	0.87	0.83	0.14	0.89	0.07
DS4	79.38	80.31	76.68	79.59	79.07	80.73	81.14	80.93	81.14	79.59	0.88	0.89	0.87	0.89	0.88	0.89	0.89	0.89	0.89	0.89
DS5	76.56	90.63	90.63	89.06	82.81	92.19	92.19	92.19	92.19	92.19	0.87	0.95	0.95	0.94	0.91	0.96	0.96	0.96	0.96	0.96
DS6	62.5	93.75	93.75	93.75	93.75	93.75	93.75	93.75	93.75	93.75	0.77	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
DS7	66.67	81.48	85.19	85.19	81.48	85.19	85.19	85.19	85.19	85.19	0.8	0.9	0.92	0.92	0.9	0.92	0.92	0.92	0.92	0.92
DS8	86.36	87.22	88.07	88.07	88.64	88.92	86.08	88.07	86.93	85.8	0.93	0.93	0.94	0.94	0.94	0.94	0.92	0.94	0.93	0.92
DS9	95.73	97.76	97.76	97.76	97.56	97.76	97.76	97.76	97.76	97.36	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
DS10	62.96	81.48	77.78	66.67	55.56	77.78	51.85	48.15	51.85	74.07	0.75	0.89	0.88	0.79	0.7	0.88	0.68	0.59	0.65	0.84
DS11	87.32	91.71	91.71	91.71	91.71	92.2	91.71	90.24	91.71	90.24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.17	NaN	0.17
DS12	65.59	59.71	58.82	59.71	60.59	62.94	60.88	65.88	62.35	60	0.53	NaN	0.04	NaN	0.04	0.28	0.23	0.5	0.27	0.01
DS13	62.96	70.37	55.56	66.67	44.44	74.07	37.04	51.85	59.26	70.37	0.77	0.73	0.54	0.71	0.21	0.83	0.26	0.48	0.59	0.73
DS14	60.78	76.47	76.47	78.43	66.67	82.35	78.43	80.39	80.39	74.51	0.74	0.87	0.87	0.88	0.8	0.9	0.88	0.88	0.89	0.85
DS15	75.76	45.45	45.45	45.45	48.48	66.67	63.64	51.52	60.61	66.67	0.79	NaN	NaN	NaN	0.11	0.72	0.65	0.2	0.61	0.74
DS16	70.81	68.78	63.57	73.98	63.57	63.12	75.57	73.98	75.11	71.72	0.54	0.34	NaN	0.58	NaN	0.01	0.6	0.59	0.6	0.43
DS17	88.48	89.55	90	89.55	89.39	89.55	89.7	90	89.7	89.55	0.94	0.94	0.95	0.94	0.94	0.94	0.95	0.95	0.95	0.94
DS18	84.66	87.5	87.5	88.07	89.2	84.09	88.07	77.84	79.55	88.64	0.91	0.93	0.93	0.93	0.94	0.91	0.93	0.87	0.88	0.94
DS19	57.78	80	80	80	82.22	82.22	80	82.22	82.22	75.56	0.72	0.89	0.89	0.89	0.9	0.9	0.89	0.9	0.9	0.86
DS20	57.78	80	80	80	66.67	77.78	80	80	80	82.22	0.73	0.89	0.89	0.89	0.8	0.88	0.89	0.89	0.89	0.9
DS21	69.53	74.61	69.14	67.19	72.66	72.66	70.7	73.83	69.92	73.83	0.79	0.82	0.78	0.77	0.81	0.82	0.79	0.82	0.79	0.82
DS22	70.77	84.62	84.62	87.69	86.15	86.15	86.15	86.15	86.15	86.15	0.83	0.92	0.92	0.93	0.92	0.93	0.93	0.93	0.93	0.93
DS23	44	76	76	76	60	64	68	44	76	44	0.61	0.75	0.79	0.79	0.44	0.57	0.64	0.61	0.7	0.22
DS24	52.38	78.57	78.57	78.57	71.43	71.43	78.57	80.95	80.95	57.14	0.64	0.86	0.86	0.86	0.8	0.83	0.86	0.87	0.87	0.7
DS25	89.86	89.74	88.58	89.86	89.39	90.79	89.86	90.91	89.28	91.03	0.95	0.95	0.94	0.95	0.94	0.95	0.95	0.95	0.94	0.95
DS26	66.38	69.87	69.87	69.43	60.7	69.43	69.87	68.56	70.31	45.85	0.77	0.8	0.81	0.8	0.69	0.81	0.81	0.79	0.81	0.5
DS27	64.1	66.67	58.97	58.97	66.67	61.54	53.85	53.85	53.85	69.23	0.67	0.58	0.47	0.47	0.67	0.62	0.1	0.1	0.1	0.74
DS28	97.69	98.79	98.79	98.35	98.35	98.79	98.02	98.79	98.02	98.57	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DS29	81.29	75	90.31	75	79.42	74.66	86.73	79.42	93.37	73.13	0.61	0.05	0.77	0.05	0.39	0.05	0.69	0.37	0.86	0.05
DS30	55.17	79.31	86.21	86.21	55.17	72.41	86.21	89.66	89.66	65.52	0.71	0.82	0.88	0.88	0.43	0.79	0.87	0.9	0.9	0.67

(b)

	Accuracy (%)										F-measure									
	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4
DS1	81.34	82.28	82.01	82.28	82.15	82.82	82.55	82.42	82.55	81.61	0.89	0.89	0.89	0.89	0.89	0.9	0.9	0.89	0.9	0.89
DS2	83.33	87.18	87.61	87.18	88.03	85.04	87.18	87.61	88.03	87.18	0.91	0.93	0.93	0.93	0.94	0.92	0.93	0.93	0.94	0.93
DS3	65.12	88.37	88.37	81.4	90.7	86.05	88.37	81.4	90.7	83.72	0.73	0.92	0.92	0.86	0.93	0.89	0.91	0.87	0.93	0.88
DS4	79.69	81.14	81.24	80.52	81.24	81.04	80.83	81.35	80.93	80.31	0.88	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89
DS5	73.44	92.19	92.19	92.19	92.19	90.63	90.63	90.63	92.19	90.63	0.84	0.96	0.96	0.96	0.96	0.95	0.95	0.95	0.96	0.95
DS6	59.38	93.75	90.63	93.75	93.75	87.5	93.75	93.75	93.75	93.75	0.75	0.97	0.95	0.97	0.97	0.93	0.97	0.97	0.97	0.97
DS7	66.67	88.89	88.89	85.19	88.89	77.78	88.89	88.89	88.89	88.89	0.78	0.94	0.94	0.92	0.94	0.88	0.94	0.94	0.94	0.94
DS8	86.36	88.64	89.2	89.2	88.92	88.92	89.49	88.92	89.2	89.2	0.93	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94
DS9	95.12	97.36	97.15	97.36	97.56	97.36	97.76	97.76	97.76	97.56	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
DS10	70.37	81.48	92.59	92.59	92.59	85.19	85.19	88.89	88.89	88.89	0.81	0.88	0.95	0.95	0.95	0.91	0.91	0.93	0.93	0.93
DS11	86.83	92.2	92.2	92.2	92.2	92.2	92.2	92.2	92.2	92.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DS12	67.35	69.71	64.12	69.41	66.47	67.94	67.94	67.35	69.12	68.24	0.6	0.6	0.46	0.59	0.59	0.59	0.61	0.59	0.62	0.54
DS13	62.96	88.89	77.78	85.19	81.48	77.78	81.48	85.19	74.07	70.37	0.77	0.92	0.84	0.89	0.86	0.84	0.86	0.89	0.82	0.78
DS14	58.82	78.43	80.39	78.43	80.39	72.55	78.43	78.43	80.39	82.35	0.73	0.87	0.89	0.87	0.89	0.84	0.88	0.88	0.89	0.9
DS15	51.52	84.85	81.82	78.79	78.79	66.67	78.79	81.82	81.82	75.76	0.6	0.88	0.85	0.82	0.82	0.73	0.83	0.84	0.85	0.76
DS16	73.3	72.17	66.06	76.47	77.6	76.24	76.24	74.89	76.7	73.98	0.63	0.49	0.15	0.63	0.68	0.65	0.64	0.64	0.66	0.63
DS17	87.58	89.55	89.55	89.7	89.39	89.7	89.85	89.7	89.55	89.7	0.93	0.94	0.94	0.95	0.94	0.95	0.95	0.95	0.94	0.95
DS18	84.66	90.34	90.34	90.34	90.34	90.34	90.34	90.34	90.34	90.34	0.91	0.95	0.95	0.95	0.95	0.94	0.95	0.95	0.95	0.95
DS19	62.22	84.44	88.89	86.67	86.67	80	86.67	88.89	88.89	86.67	0.75	0.91	0.94	0.92	0.92	0.88	0.92	0.94	0.94	0.92
DS20	57.78	80	80	80	71.11	73.33	80	80	73.33	77.78	0.73	0.89	0.89	0.89	0.83	0.85	0.89	0.89	0.85	0.88
DS21	74.22	73.44	74.22	74.22	73.05	73.83	73.44	73.05	74.22	74.22	0.82	0.82	0.83	0.83	0.82	0.82	0.82	0.81	0.82	0.83
DS22	70.77	92.31	92.31	92.31	92.31	89.23	90.77	90.77	90.77	86.15	0.83	0.96	0.96	0.96	0.96	0.94	0.95	0.95	0.95	0.92
DS23	44	68	80	72	72	68	64	76	72	48	0.61	0.67	0.76	0.72	0.74	0.64	0.61	0.63	0.7	0.13
DS24	47.62	78.57	73.81	78.57	76.19	73.81	76.19	78.57	78.57	80.95	0.59	0.85	0.82	0.85	0.84	0.83	0.84	0.86	0.85	0.88
DS25	89.86	91.14	91.38	91.03	91.14	91.38	91.03	91.49	91.26	91.26	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
DS26	66.38	70.74	72.93	68.12	72.05	74.67	73.8	70.31	73.8	70.31	0.76	0.8	0.81	0.79	0.82	0.83	0.82	0.8	0.82	0.81
DS27	51.28	56.41	56.41	53.85	64.1	69.23	56.41	56.41	56.41	66.67	0.49	0.54	0.51	0.53	0.7	0.68	0.6	0.6	0.6	0.7
DS28	97.69	98.79	98.79	98.79	98.79	98.79	98.79	98.79	98.79	98.79	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DS29	82.99	74.32	74.32	74.32	86.56	75.68	81.8	83.16	74.32	74.32	0.68	NaN	NaN	NaN	0.73	0.14	0.66	0.68	NaN	NaN
DS30	55.17	72.41	79.31	75.86	72.41	72.41	82.76	82.76	86.21	72.41	0.71	0.76	0.82	0.77	0.76	0.78	0.85	0.86	0.88	0.78

To be continued

Table 11

(c)

	Accuracy (%)										F-measure									
	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4
DS1	80.13	82.82	81.74	81.74	81.48	79.33	81.48	81.48	81.48	81.34	0.88	0.9	0.89	0.89	0.89	0.88	0.89	0.89	0.89	0.89
DS2	82.91	88.46	87.18	88.46	87.61	88.46	88.03	88.46	88.03	88.03	0.91	0.94	0.93	0.94	0.93	0.94	0.94	0.94	0.94	0.94
DS3	62.79	74.42	74.42	74.42	83.72	65.12	79.07	79.07	76.74	74.42	0.71	0.83	0.83	0.83	0.89	0.78	0.86	0.86	0.84	0.83
DS4	79.9	80.83	80.83	80.62	80.73	80.62	80.93	80.73	80.93	80.62	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89
DS5	76.56	92.19	92.19	92.19	92.19	92.19	92.19	92.19	92.19	92.19	0.87	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96
DS6	62.5	93.75	93.75	93.75	93.75	93.75	93.75	93.75	93.75	93.75	0.77	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
DS7	66.67	85.19	85.19	85.19	88.89	85.19	85.19	85.19	85.19	85.19	0.8	0.92	0.92	0.92	0.94	0.92	0.92	0.92	0.92	0.92
DS8	86.65	88.92	89.49	89.2	89.2	88.64	89.2	89.2	89.49	88.35	0.93	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94
DS9	95.73	97.76	97.76	97.76	97.56	97.76	97.76	97.76	97.76	97.76	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
DS10	59.26	77.78	77.78	77.78	77.78	77.78	77.78	77.78	77.78	77.78	0.74	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88
DS11	87.32	92.2	92.2	92.2	92.2	92.2	92.2	92.2	92.2	92.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DS12	67.35	62.35	59.71	61.76	68.53	59.41	68.24	66.76	68.82	60.29	0.57	0.15	NaN	0.12	0.58	0.03	0.59	0.56	0.6	0.03
DS13	62.96	81.48	77.78	81.48	81.48	62.96	74.07	74.07	77.78	70.37	0.77	0.87	0.85	0.87	0.87	0.77	0.83	0.83	0.85	0.81
DS14	60.78	82.35	82.35	82.35	84.31	80.39	80.39	80.39	80.39	80.39	0.75	0.9	0.9	0.9	0.91	0.89	0.89	0.89	0.89	0.89
DS15	75.76	69.7	75.76	75.76	78.79	66.67	75.76	78.79	72.73	66.67	0.8	0.78	0.82	0.82	0.83	0.76	0.82	0.83	0.8	0.72
DS16	72.17	70.81	63.57	70.81	73.98	63.57	71.27	74.89	70.59	69.68	0.56	0.46	NaN	0.46	0.6	NaN	0.47	0.6	0.45	0.4
DS17	88.33	90	89.85	89.85	89.85	90	89.85	89.85	90	89.85	0.94	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
DS18	84.66	88.07	86.93	87.5	87.5	84.66	85.8	85.8	84.09	86.36	0.91	0.93	0.93	0.93	0.93	0.92	0.92	0.92	0.91	0.93
DS19	60	82.22	82.22	82.22	84.44	80	82.22	80	80	80	0.74	0.9	0.9	0.9	0.91	0.89	0.9	0.89	0.89	0.89
DS20	57.78	80	80	80	77.78	80	80	80	80	80	0.73	0.89	0.89	0.89	0.88	0.89	0.89	0.89	0.89	0.89
DS21	70.31	72.66	73.05	73.05	73.44	69.53	73.44	72.27	72.27	71.09	0.8	0.82	0.82	0.82	0.82	0.81	0.82	0.81	0.81	0.81
DS22	70.77	86.15	86.15	86.15	86.15	86.15	86.15	86.15	86.15	86.15	0.83	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93
DS23	44	80	72	72	76	60	72	60	76	48	0.61	0.76	0.63	0.72	0.79	0.29	0.67	0.17	0.73	0.13
DS24	54.76	80.95	80.95	78.57	80.95	69.05	78.57	80.95	80.95	69.05	0.67	0.88	0.88	0.87	0.88	0.82	0.87	0.88	0.88	0.82
DS25	89.86	91.26	91.38	91.14	91.26	91.03	91.49	91.26	90.91	91.03	0.95	0.95	0.95	0.95	0.95	0.95	0.96	0.95	0.95	0.95
DS26	69.87	68.12	69	68.56	68.56	67.69	70.31	69	69.87	69.87	0.8	0.8	0.8	0.8	0.8	0.8	0.81	0.8	0.8	0.81
DS27	58.97	53.85	56.41	51.28	69.23	64.1	53.85	56.41	51.28	58.97	0.62	0.53	0.54	0.49	0.7	0.65	0.36	0.41	0.39	0.43
DS28	97.69	98.79	98.79	98.79	98.79	98.79	98.79	98.79	98.79	98.79	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DS29	80.44	74.32	74.32	74.32	87.41	74.32	86.9	78.06	74.32	74.32	0.61	NaN	NaN	NaN	0.7	NaN	0.69	0.46	NaN	NaN
DS30	55.17	75.86	82.76	79.31	75.86	82.76	79.31	82.76	79.31	75.86	0.71	0.8	0.86	0.82	0.8	0.86	0.83	0.86	0.83	0.81

(d)

	Accuracy (%)										F-measure									
	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4
DS1	80.27	81.74	81.74	81.48	73.96	78.26	82.55	81.74	81.88	75.3	0.88	0.89	0.89	0.89	0.84	0.88	0.9	0.89	0.89	0.86
DS2	84.19	88.46	88.46	88.46	87.18	88.46	88.46	88.46	88.03	85.04	0.91	0.94	0.94	0.94	0.93	0.94	0.94	0.94	0.94	0.92
DS3	51.16	37.21	37.21	86.05	58.14	62.79	90.7	37.21	90.7	37.21	0.6	NaN	NaN	0.89	0.5	0.77	0.92	NaN	0.92	NaN
DS4	79.59	80.41	76.79	80.41	79.69	80.52	80.52	80.41	80.62	80	0.89	0.89	0.87	0.89	0.89	0.89	0.89	0.89	0.89	0.89
DS5	76.56	92.19	92.19	92.19	92.19	92.19	92.19	92.19	92.19	92.19	0.87	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96
DS6	62.5	93.75	93.75	93.75	93.75	93.75	93.75	93.75	93.75	93.75	0.77	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
DS7	66.67	85.19	85.19	85.19	77.78	85.19	85.19	85.19	85.19	85.19	0.8	0.92	0.92	0.92	0.88	0.92	0.92	0.92	0.92	0.92
DS8	85.51	88.64	88.64	88.64	87.22	88.64	82.1	88.35	82.1	86.93	0.92	0.94	0.94	0.94	0.93	0.94	0.9	0.94	0.9	0.93
DS9	95.73	97.76	97.76	97.76	97.76	97.76	97.76	97.76	97.76	97.76	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
DS10	59.26	74.07	77.78	77.78	51.85	77.78	59.26	55.56	51.85	77.78	0.74	0.85	0.88	0.88	0.68	0.88	0.74	0.68	0.65	0.87
DS11	87.32	91.71	91.71	91.71	91.71	92.2	91.71	90.24	91.71	90.24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.17	NaN	0.17
DS12	66.76	59.71	60	59.71	59.71	59.71	62.94	66.76	62.94	59.71	0.51	NaN	0.03	NaN	NaN	NaN	0.18	0.49	0.17	NaN
DS13	62.96	74.07	37.04	74.07	37.04	62.96	40.74	51.85	48.15	66.67	0.77	0.77	0.19	0.77	NaN	0.77	0.27	0.38	0.36	0.71
DS14	60.78	80.39	80.39	76.47	60.78	80.39	80.39	80.39	80.39	74.51	0.76	0.89	0.89	0.87	0.75	0.89	0.89	0.89	0.89	0.85
DS15	48.48	45.45	45.45	45.45	45.45	63.64	63.64	51.52	63.64	66.67	0.56	NaN	NaN	NaN	NaN	0.7	0.63	0.2	0.65	0.77
DS16	71.72	63.57	63.57	67.65	63.57	63.57	70.59	73.3	70.36	69.91	0.56	NaN	NaN	0.28	NaN	NaN	0.43	0.56	0.43	0.34
DS17	88.48	90	90	90	83.64	89.85	89.85	90	89.85	89.85	0.94	0.95	0.95	0.95	0.91	0.95	0.95	0.95	0.95	0.95
DS18	78.98	84.66	84.66	84.66	84.66	84.66	84.66	80.11	84.66	84.66	0.88	0.92	0.92	0.92	0.92	0.92	0.92	0.89	0.92	0.92
DS19	57.78	80	80	80	80	80	80	80	80	80	0.73	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89	0.89
DS20	57.78	80	80	80	75.56	80	82.22	80	82.22	80	0.73	0.89	0.89	0.89	0.86	0.89	0.9	0.89	0.9	0.89
DS21	68.75	73.83	66.02	66.41	61.33	66.41	67.58	74.22	67.19	71.88	0.78	0.82	0.76	0.77	0.73	0.8	0.77	0.83	0.77	0.82
DS22	70.77	84.62	86.15	84.62	83.08	86.15	86.15	86.15	86.15	86.15	0.83	0.92	0.93	0.92	0.91	0.93	0.93	0.93	0.93	0.93
DS23	44	76	76	76	68	52	76	44	76	56	0.61	0.75	0.79	0.79	0.5	NaN	0.7	0.61	0.7	NaN
DS24	50	80.95	78.57	80.95	52.38	69.05	80.95	80.95	80.95	54.76	0.64	0.88	0.86	0.88	0.57	0.82	0.88	0.88	0.88	0.69
DS25	89.86	91.03	90.56	91.03	88.23	91.03	91.03	91.03	91.03	91.03	0.95	0.95	0.95	0.95	0.94	0.95	0.95	0.95	0.95	0.95
DS26	69.43	69.87	69.43	69.87	47.6	65.94	69.43	70.31	69.43	38.43	0.8	0.81	0.81	0.81	0.43	0.79	0.81	0.81	0.81	0.3
DS27	56.41	61.54	61.54	64.1	66.67	48.72	53.85	53.85	53.85	56.41	0.56	0.35	0.35	0.42	0.67	0.29	0.1	0.1	0.1	0.6
DS28	97.69	98.79	98.79	98.35	98.35	98.79	98.02	98.79	98.02	98.57	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DS29	73.64	75	90.31	75	75.85	74.32	74.32	74.66	93.37	73.13	0.15	0.05	0.77	0.05	0.11	NaN	NaN	0.04	0.86	0.05
DS30	55.17	86.21	86.21	86.21	41.38	72.41	93.1	93.1	86.21	68.97	0.71	0.88	0.87	0.88	NaN	0.79	0.94	0.94	0.88	0.69

300 data points ((9 feature selection methods + 1 considering all features) × 30 datasets)). The results of t-test analyses are summarized in Table 14. The results for other performance parameters are similar.

From Table 14, it is evident that in most of the cases there is a significant difference between these approaches, as the p-value is lower than 0.05. However, by judging the mean difference, C2, i.e., extreme learning with the polynomial kernel function, yields better results compared to other classifiers.

6.2.5 Interaction of classification methods with feature selection methods

This section focuses on the effect of classification methods across the feature selection method performances. In this study, extreme learning with four kernels is investigated. From Table 11 and Figs. 5 and 6, it can be inferred that for each classification method, using different feature selection methods produces better results.

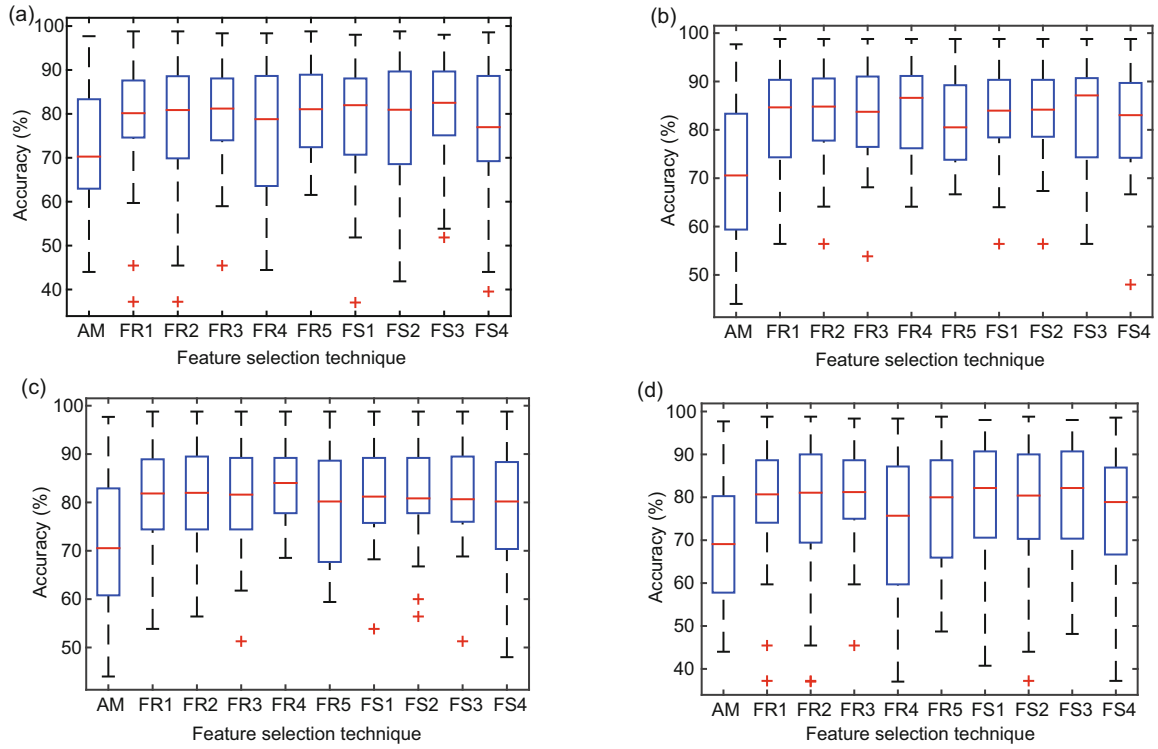

Fig. 5 Accuracy of the linear kernel (a), polynomial kernel (b), radial basis function kernel (c), and sigmoid kernel (d)

Table 12 *t*-test: feature selection techniques

(a) Accuracy (%)

	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4
AM	0.00	−9.16	−8.77	−9.80	−7.29	−8.14	−9.55	−8.45	−9.97	−6.86
FR1	9.16	0.00	0.40	−0.63	1.88	1.02	−0.39	0.71	−0.80	2.31
FR2	8.77	−0.40	0.00	−1.03	1.48	0.62	−0.79	0.32	−1.20	1.91
FR3	9.80	0.63	1.03	0.00	2.51	1.65	0.25	1.35	−0.17	2.94
FR4	7.29	−1.88	−1.48	−2.51	0.00	−0.86	−2.27	−1.16	−2.68	0.43
FR5	8.14	−1.02	−0.62	−1.65	0.86	0.00	−1.41	−0.30	−1.82	1.29
FS1	9.55	0.39	0.79	−0.25	2.27	1.41	0.00	1.10	−0.41	2.70
FS2	8.45	−0.71	−0.32	−1.35	1.16	0.30	−1.10	0.00	−1.52	1.59
FS3	9.97	0.80	1.20	0.17	2.68	1.82	0.41	1.52	0.00	3.11
FS4	6.86	−2.31	−1.91	−2.94	−0.43	−1.29	−2.70	−1.59	−3.11	0.00

(b) p-value

	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4
AM	–	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FR1	0.00	–	0.40	0.32	0.04	0.15	0.64	0.28	0.34	0.00
FR2	0.00	0.40	–	0.16	0.08	0.42	0.30	0.61	0.12	0.02
FR3	0.00	0.32	0.16	–	0.00	0.01	0.66	0.09	0.75	0.00
FR4	0.00	0.04	0.08	0.00	–	0.29	0.00	0.19	0.00	0.62
FR5	0.00	0.15	0.42	0.01	0.29	–	0.05	0.69	0.01	0.06
FS1	0.00	0.64	0.30	0.66	0.00	0.05	–	0.12	0.25	0.01
FS2	0.00	0.28	0.61	0.09	0.19	0.69	0.12	–	0.05	0.04
FS3	0.00	0.34	0.12	0.75	0.00	0.01	0.25	0.05	–	0.00
FS4	0.00	0.00	0.02	0.00	0.62	0.06	0.01	0.04	0.00	–

Table 13 *t*-test: feature ranking techniques versus feature subset selection techniques (average values)

Parameter	Mean (FR-FS)	p-value
Accuracy (%)	−0.0743	0.9231
F-measure	−0.0165	0.3964

6.3 Cost analysis

In this experiment, the values tabulated in Tables 7 and 8 are used in the design of the cost evaluation model. Eqs. (14) and (19) are used to calculate the estimated fault removal cost (E_{cost}) and estimated testing cost (T_{cost}), respectively. δ_{u}, δ_{i},

Table 14 *t*-test: classification methods

(a) Accuracy (%)

	Mean difference				p-value			
	C1	C2	C3	C4	C1	C2	C3	C4
C1	0.00	−4.01	−2.53	0.78	–	0.00	0.00	0.00
C2	4.01	0.00	1.49	4.79	0.00	–	0.00	0.00
C3	2.53	−1.49	0.00	3.31	0.00	0.00	–	0.00
C4	−0.78	−4.79	−3.31	0.00	0.00	0.00	0.00	–

(b) F-measure

	Mean difference				p-value			
	C1	C2	C3	C4	C1	C2	C3	C4
C1	0.00	−0.07	−0.04	0.03	–	0.00	0.00	0.00
C2	0.07	0.00	0.02	0.09	0.00	–	0.00	0.00
C3	0.04	−0.02	0.00	0.07	0.00	0.00	–	0.00
C4	−0.03	−0.09	−0.07	0.00	0.00	0.00	0.00	–

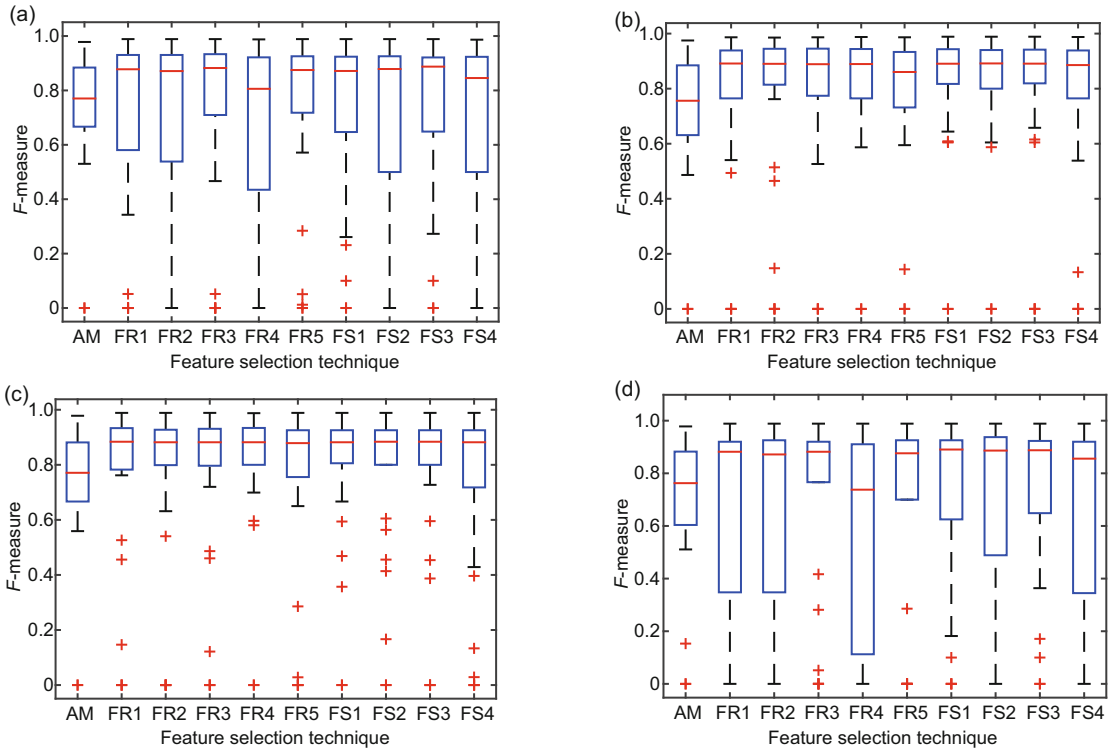

Fig. 6 ***F*-measure of the linear kernel (a), polynomial kernel (b), radial basis function kernel (c), and sigmoid kernel (d)**

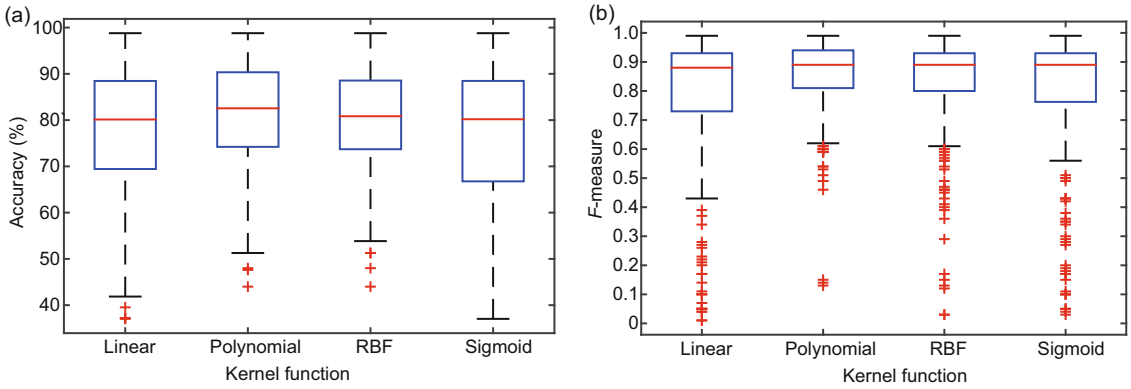

Fig. 7 Box plots displaying performance evaluation results for different kernel functions: (a) accuracy; (b) *F*-measure

and δ_s in the equations show the fault identification efficiencies of unit, integration, and system testing, respectively. The values of δ_u, δ_i, and δ_s are collected from the survey report "Software Quality in 2010" by Jones (2010). Table 15 shows the value of estimated testing cost (T_{cost}) for different datasets. Table 16 shows the estimated fault removal cost for δ_u=0.25, δ_i=0.45, and δ_s=0.5. The results for the other values of δ_u, δ_i, and δ_s are similar in nature.

Figs. 9a–9d 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 of faulty classes increases, the fault-prediction technique tends to have a higher value for NE_{cost}; i.e., fault prediction can be useful for the projects with the percentage of faulty classes below a certain threshold.

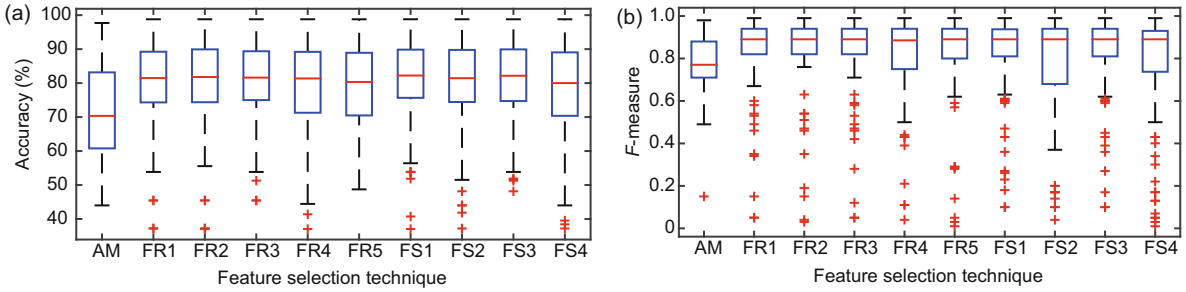

Fig. 8 Box plots displaying performance evaluation results for different sets of source code metrics: (a) accuracy; (b) *F*-measure

6.4 Threshold value

After finding the best model, an attempt is made to establish a relationship between $\mathrm{NE_{cost}}$ and the percentage of faulty classes (FP). In this study, a logistic regression method is considered to develop a model to calculate the probability of usefulness for the fault prediction techniques (P_{fault}). In the logistic regression method, a dependent variable can take only two values. So, the dependent variable of $\mathrm{NE_{cost}}$ is divided into two groups, one containing software for which fault prediction will be useful ($\mathrm{NE_{cost}} < 1$) and the other containing software for which it is not useful ($\mathrm{NE_{cost}} > 1$). Table 17 shows the constant, coefficient, and threshold value in terms of the percentage of faulty classes for different values of δ_{u}, δ_{i}, and δ_{s}. Logistic regression analysis is applied with a threshold of 0.5. It implies that fault prediction is useful if $P_{\mathrm{fault}} < 0.5$; otherwise, it is not useful. From Table 17, it is observed that the fault prediction model is useful for the projects with the percentage of faulty classes below a certain threshold depending on the fault identification efficiency (high: 25.72%; median: 39.24%; low: 47.28%).

6.5 Experimental finding

In this section, the overall findings for the experiments are presented. The investigation was carried out for 30 projects using the extreme learning method with four types of kernel functions, i.e., linear, polynomial, sigmoid, and RBF. The performances of the respective techniques were evaluated using the proposed cost-based evaluation framework and associated parameters such as δ_{u}, δ_{i}, and δ_{s}. Figs. 9a–9d depict the normalized fault removal cost ($\mathrm{NE_{cost}}$) of fault prediction techniques for different values of δ_{u}, δ_{i}, and δ_{s}. Table 17 shows the threshold values in terms of the percentage of faulty classes for different values of δ_{u}, δ_{i}, and δ_{s}. From these figures and tables, the following can be inferred:

1. In cases of the minimum fault identification efficiency for unit, integration, and system testing, i.e., $\delta_{\mathrm{u}} = 0.1$, $\delta_{\mathrm{i}} = 0.25$, and $\delta_{\mathrm{s}} = 0.25$, fault prediction may be useful when there are less than 47.28% faulty classes in the software.

2. Similarly, in cases of the median fault identification efficiency for unit, integration, and system testing, i.e., $\delta_{\mathrm{u}} = 0.25$, $\delta_{\mathrm{i}} = 0.45$, and $\delta_{\mathrm{s}} = 0.5$, fault prediction may be useful when there are less than 39.24% faulty classes in the software.

3. In cases of the high fault identification efficiency for unit, integration, and system testing, i.e., $\delta_{\mathrm{u}} = 0.5$, $\delta_{\mathrm{i}} = 0.6$, and $\delta_{\mathrm{s}} = 0.65$, fault prediction may be useful when there are less than 25.72% faulty classes in the software.

This study helps answer the following research questions:

RQ1: Fig. 7 and Table 11 reveal the perfor-

Table 15 Testing cost (T_{cost})

$\delta_{\mathrm{u}}, \delta_{\mathrm{i}}, \delta_{\mathrm{s}}$ values	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8	DS9	DS10	DS11	DS12	DS13	DS14	DS15
0.1, 0.25, 0.25	1079.78	260.25	82.47	1313.84	63.69	30.28	32.81	389.55	403.53	39.08	746.97	892.17	51.64	69.64	71.83
0.25, 0.45, 0.5	2322.85	518.84	187.88	2782.28	121.92	56.77	67.28	775.33	707.21	84.05	1840.66	2126.77	117.58	147.58	167.00
0.5, 0.6, 0.65	3913.59	975.02	290.79	4796.06	242.56	116.22	121.45	1460.48	1587.23	141.67	2526.17	3072.74	182.12	254.12	250.68
$\delta_{\mathrm{u}}, \delta_{\mathrm{i}}, \delta_{\mathrm{s}}$ values	DS16	DS17	DS18	DS19	DS20	DS21	DS22	DS23	DS24	DS25	DS26	DS27	DS28	DS29	DS30
0.1, 0.25, 0.25	1213.49	702.16	216.75	62.00	62.00	461.93	77.00	62.69	72.30	885.18	416.57	92.03	3500.35	1812.63	62.55
0.25, 0.45, 0.5	2908.16	1378.29	446.34	131.70	131.70	1040.95	156.70	148.61	161.48	1718.00	940.13	216.41	8664.30	4398.43	145.23
0.5, 0.6, 0.65	4167.47	2647.39	801.02	226.01	226.01	1637.63	286.01	216.57	257.46	3352.62	1475.74	319.24	11807.58	6182.94	218.46

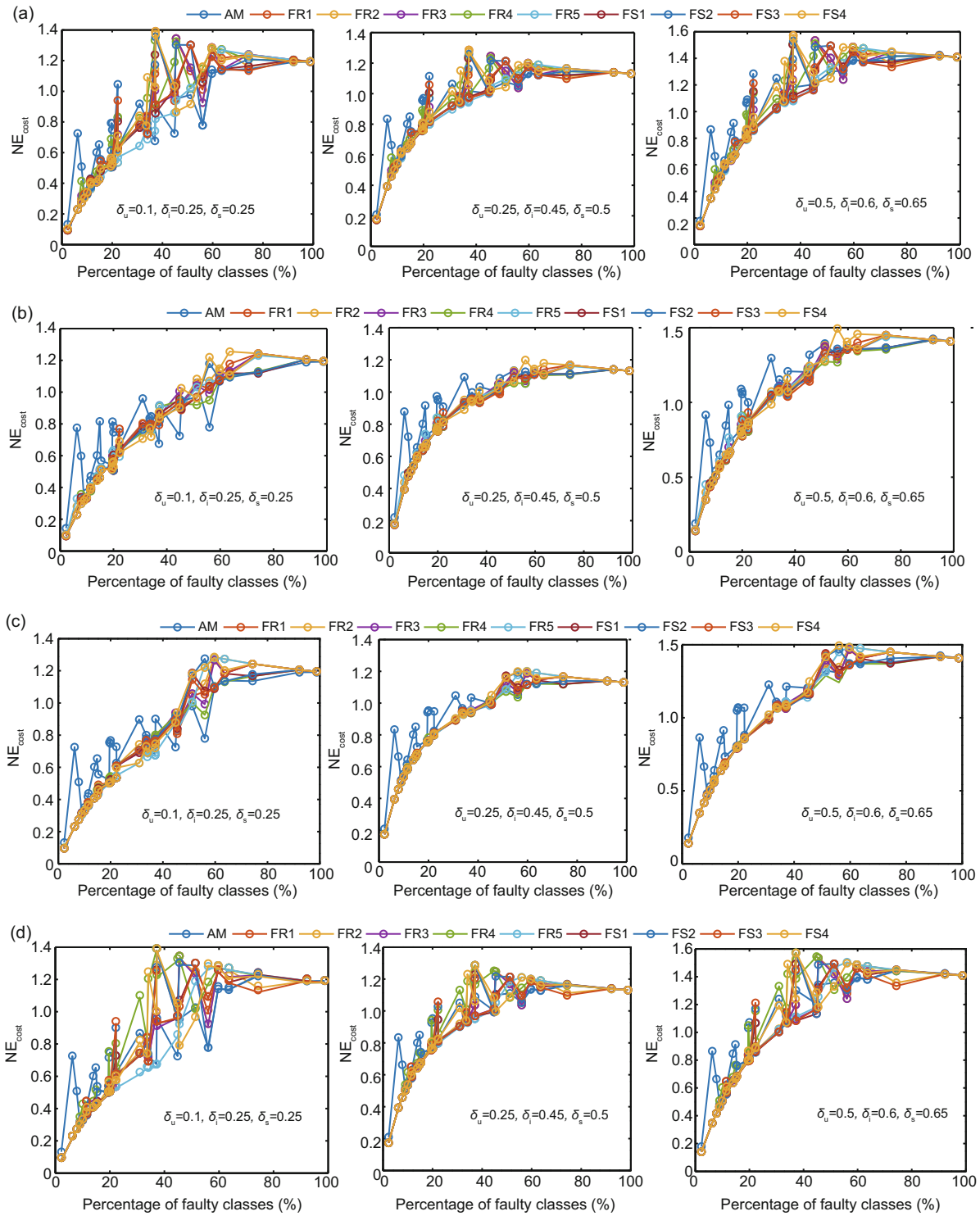

Fig. 9 NE_{cost} of the linear kernel (a), polynomial kernel (b), radial basis function kernel (c), and sigmoid kernel (d)

mances of the set of selected source code metrics and all of the metrics. The model developed using the extreme learning method with a polynomial kernel yields better results compared to others.

RQ2: The results shown in Figs. 9a–9d indicate that the fault prediction model developed is useful

for projects having a percentage of faulty classes below a certain threshold.

RQ3: In this work, nine feature selection techniques are considered to find the right set of source code metrics for fault prediction. From Fig. 8, it may be inferred that the model developed using a set of selected source code metrics produces better results than the model using all the 20 metrics.

RQ4: In this study, five feature-ranking methods are considered to find the reduced subset of OO metrics. From Fig. 8, it is observed that feature selection using FR3, i.e., the OneR feature evaluation method, produces the best results.

RQ5: In this study, four feature subset selection methods are considered to find the reduced subset of OO metrics. From Fig. 8, it is observed that feature selection using the filter subset evaluation method produces results that are convincingly quite better.

RQ6: In this study, a pairwise t-test is employed to determine whether feature-ranking methods work better than feature subset selection methods or if they perform equally well. From t-test results, there is no significant difference between feature ranking and feature subset selection methods. However, the

Table 16 Estimated fault removal cost (E_{cost}) for δ_{u}=0.25, δ_{i}=0.45, δ_{s}=0.5

	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FS1	FS2	FS3	FS4
	Linear kernel										Polynomial kernel									
DS1	1912.66	1908.54	1908.39	1904.71	1944.13	1860.90	1890.60	1894.57	1895.01	1913.55	1929.13	1896.19	1907.80	1902.80	1905.30	1888.39	1891.19	1898.10	1891.19	1893.25
DS2	331.79	306.79	301.79	306.79	319.29	304.00	319.00	306.79	321.50	321.79	340.62	313.70	311.20	313.70	306.50	321.79	311.50	309.00	308.70	313.70
DS3	191.19	241.63	241.63	182.51	185.90	178.69	183.69	236.63	182.51	239.13	193.99	175.60	175.60	180.90	179.72	182.51	177.81	178.69	175.31	178.40
DS4	2144.31	2119.60	2204.90	2134.90	2145.19	2102.99	2106.22	2113.43	2110.63	2126.08	2158.87	2115.04	2114.75	2116.81	2125.78	2126.37	2122.54	2123.28	2122.25	2119.60
DS5	80.89	58.39	58.39	60.89	70.89	55.89	55.89	55.89	55.89	55.89	88.09	58.09	58.09	58.09	60.30	60.59	60.59	58.39	58.09	58.39
DS6	47.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	49.86	22.36	24.86	22.36	22.36	27.36	22.36	22.36	22.36	22.36
DS7	57.21	47.21	44.71	44.71	47.21	44.71	44.71	44.71	44.71	44.71	61.62	44.42	44.42	46.92	44.42	49.71	44.42	44.42	44.42	44.42
DS8	475.92	461.81	452.10	452.10	455.92	446.81	480.63	456.51	473.13	472.10	484.75	460.33	455.33	457.54	460.04	460.04	455.04	457.83	457.54	453.13
DS9	147.95	122.95	122.95	122.95	125.45	122.95	122.95	122.95	122.95	127.95	155.45	127.95	130.45	127.95	125.45	127.95	122.95	122.95	122.95	125.45
DS10	81.48	68.98	67.07	76.77	84.27	67.07	84.57	93.68	88.98	73.98	76.48	73.39	65.89	65.89	65.89	66.48	68.68	66.18	66.18	68.39
DS11	2099.85	2097.21	2097.21	2097.21	2097.21	2096.91	2097.21	2093.68	2097.21	2093.68	2100.15	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91
DS12	2400.32	2551.77	2546.04	2551.77	2544.27	2493.40	2506.48	2417.67	2496.19	2549.27	2356.64	2374.14	2430.46	2374.43	2366.35	2367.08	2358.26	2369.88	2357.08	2404.28
DS13	111.78	124.42	134.42	124.71	146.33	110.89	144.71	136.92	131.92	124.42	111.78	109.71	112.80	112.21	112.51	112.80	112.51	110.01	113.10	117.80
DS14	141.19	116.78	116.78	114.28	131.48	111.48	116.48	118.39	113.98	119.28	143.69	118.69	116.19	118.69	116.19	123.98	114.28	114.28	113.98	113.69
DS15	172.22	208.25	208.25	208.25	205.75	175.31	182.22	203.25	184.72	170.90	181.19	164.72	167.22	169.72	169.72	173.10	167.51	169.42	167.22	176.63
DS16	3283.61	3374.49	3460.66	3270.67	3460.66	3459.04	3264.20	3264.05	3266.99	3344.19	3225.23	3317.13	3428.75	3245.37	3210.82	3228.02	3234.64	3229.79	3225.23	3235.38
DS17	762.72	749.63	737.72	747.42	749.92	745.22	747.13	737.72	744.92	749.63	779.92	745.22	745.22	742.72	747.72	742.72	740.22	744.92	745.22	744.92
DS18	323.85	304.73	304.73	304.44	303.85	304.29	302.23	336.20	333.12	304.14	326.05	303.26	303.26	303.26	303.26	305.47	303.26	303.26	303.26	303.26
DS19	127.80	100.60	100.60	100.60	100.30	100.30	100.60	102.51	100.30	105.60	125.01	102.21	99.42	101.92	101.92	107.21	101.92	99.42	99.42	99.71
DS20	125.60	100.60	100.60	100.60	115.60	103.10	102.80	100.60	102.80	100.30	125.60	100.60	100.60	100.60	110.60	108.10	100.60	100.60	108.10	103.10
DS21	1018.46	994.78	1020.96	1024.64	998.46	978.61	1017.58	990.96	1022.58	986.55	1008.31	991.26	986.26	986.26	995.96	993.17	995.67	1004.78	990.67	984.05
DS22	125.60	103.10	103.10	100.30	105.01	100.60	100.60	100.60	100.60	100.60	125.60	99.42	99.42	99.42	99.42	102.21	99.71	99.71	99.71	102.80
DS23	156.49	158.54	154.13	154.13	170.75	166.04	163.54	156.49	162.95	176.34	156.49	161.34	160.45	158.84	156.63	163.54	163.84	167.36	161.04	178.25
DS24	171.63	148.54	148.54	148.54	156.04	145.01	148.54	148.25	148.25	164.42	176.63	150.75	153.54	150.75	148.84	151.34	151.04	148.54	150.75	143.84
DS25	885.67	888.17	917.58	887.87	900.08	870.08	887.87	869.78	898.17	860.67	885.67	878.02	873.02	876.11	886.84	877.43	876.11	877.14	875.52	868.90
DS26	917.87	893.46	884.64	895.96	992.28	873.90	884.64	900.96	884.34	1081.69	937.72	901.69	900.22	905.66	889.78	883.60	895.22	901.99	899.63	882.14
DS27	232.81	243.55	248.84	248.84	234.72	237.52	262.66	262.66	262.66	225.61	245.31	242.52	244.72	242.81	228.40	234.43	235.90	235.90	235.90	230.31
DS28	9803.49	9800.55	9800.55	9801.72	9801.72	9800.55	9802.61	9800.55	9802.61	9801.14	9803.49	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55
DS29	4933.88	5123.43	4891.81	5123.43	5038.58	5124.01	4922.25	5045.19	4829.16	5126.66	4889.03	5133.43	5133.43	5133.43	4871.82	5104.60	4888.88	4890.94	5133.43	5133.43
DS30	145.31	147.66	147.07	147.07	169.57	146.04	149.28	146.78	146.78	157.66	145.31	150.45	147.66	152.36	150.45	148.25	147.36	145.16	144.86	148.25
	Radial basis function kernel										Sigmoid kernel									
DS1	1914.13	1868.54	1881.92	1881.92	1886.92	1851.93	1886.92	1886.92	1886.92	1876.19	1907.22	1899.57	1888.54	1893.54	1996.05	1854.29	1880.16	1890.75	1888.25	1915.90
DS2	334.29	304.00	309.29	301.79	306.79	301.79	304.29	301.79	304.29	304.29	326.79	301.79	301.79	301.79	309.29	301.79	301.79	301.79	304.29	321.79
DS3	194.28	177.37	177.37	179.57	176.19	178.55	176.78	176.78	177.07	177.37	204.57	241.63	241.63	182.51	219.13	178.84	181.92	241.63	181.92	241.63
DS4	2131.81	2102.69	2100.49	2103.28	2105.19	2101.08	2100.19	2109.60	2102.40	2103.28	2128.28	2103.87	2193.58	2103.87	2125.78	2101.37	2103.58	2106.08	2103.28	2113.87
DS5	80.89	55.89	55.89	55.89	55.89	55.89	55.89	55.89	55.89	55.89	80.89	55.89	55.89	55.89	55.89	55.89	55.89	55.89	55.89	55.89
DS6	47.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	47.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36
DS7	57.21	44.71	44.71	44.71	44.42	44.71	44.71	44.71	44.71	44.71	57.21	44.71	44.71	44.71	49.71	44.71	44.71	44.71	44.71	44.71
DS8	475.63	453.42	450.63	450.92	450.92	447.10	450.92	450.92	450.63	449.60	474.60	447.10	447.10	447.10	459.60	447.10	504.60	449.60	504.60	462.10
DS9	147.95	122.95	122.95	122.95	125.45	122.95	122.95	122.95	122.95	122.95	147.95	122.95	122.95	122.95	122.95	122.95	122.95	122.95	122.95	122.95
DS10	79.57	67.07	67.07	67.07	67.07	67.07	67.07	67.07	67.07	67.07	79.57	69.57	67.07	67.07	84.57	67.07	79.57	86.48	88.98	69.27
DS11	2099.85	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2099.85	2097.21	2097.21	2097.21	2097.21	2096.91	2097.21	2093.68	2097.21	2093.68
DS12	2378.70	2524.86	2551.77	2529.86	2379.73	2547.66	2368.99	2383.70	2370.61	2546.77	2414.58	2551.77	2547.07	2551.77	2551.77	2551.77	2517.66	2425.61	2519.86	2551.77
DS13	111.78	110.30	110.60	110.30	110.30	111.78	110.89	110.89	110.60	111.19	111.78	121.92	146.92	121.92	151.33	111.78	144.42	141.33	141.63	124.71
DS14	138.98	111.48	111.48	111.48	111.19	111.78	111.78	111.78	111.78	111.78	136.78	111.78	111.78	116.78	138.98	111.78	113.98	113.98	116.19	119.28
DS15	170.01	166.19	165.60	165.60	167.51	168.69	165.60	167.51	165.90	175.31	183.69	208.25	208.25	208.25	208.25	175.60	184.42	203.25	182.22	166.49
DS16	3277.43	3332.13	3460.66	3329.93	3259.64	3460.66	3327.13	3258.46	3332.43	3355.66	3278.02	3460.66	3460.66	3393.60	3460.66	3460.66	3343.46	3282.58	3341.55	3375.22
DS17	765.22	737.72	740.22	740.22	742.42	737.72	740.22	740.22	737.72	740.22	762.72	737.72	737.72	737.72	849.33	740.22	742.42	737.72	742.42	742.42
DS18	323.85	304.44	305.03	302.53	302.53	301.79	305.62	303.41	306.50	303.12	326.79	301.79	301.79	301.79	301.79	301.79	304.00	330.62	301.79	301.79
DS19	125.30	100.30	100.30	100.30	100.01	100.60	100.30	100.60	100.60	100.60	125.60	100.60	100.60	100.60	100.60	100.60	100.60	102.80	100.60	100.60
DS20	125.60	100.60	100.60	100.60	103.10	100.60	100.60	100.60	100.60	100.60	125.60	100.60	100.60	100.60	105.60	100.60	100.30	100.60	100.30	100.60
DS21	1006.85	980.82	980.52	980.52	984.64	967.73	984.64	992.14	987.73	977.58	1012.43	986.55	1025.52	1025.23	1051.11	965.68	1030.96	981.85	1033.46	979.20
DS22	125.60	100.60	100.60	100.60	100.60	100.60	100.60	100.60	100.60	100.60	125.60	103.10	100.60	103.10	105.60	100.60	100.60	100.60	100.60	100.60
DS23	156.49	160.45	165.45	158.84	154.13	175.16	163.25	177.36	160.75	178.25	156.49	158.54	154.13	154.13	170.16	180.16	162.95	156.49	162.95	179.86
DS24	169.13	143.84	146.04	144.13	143.84	145.31	144.13	143.84	143.84	145.31	169.72	146.04	148.54	146.04	182.66	145.31	146.04	146.04	146.04	164.72
DS25	885.67	866.70	861.99	869.20	864.49	860.67	863.90	864.49	869.78	865.08	885.67	860.67	872.87	860.67	925.08	860.67	860.67	860.67	860.67	860.67
DS26	895.66	888.02	891.84	892.14	883.31	870.67	890.96	889.64	891.25	873.61	887.14	882.43	876.11	884.64	1115.80	871.85	876.11	888.75	876.11	1157.27
DS27	235.61	242.81	242.52	245.31	232.22	235.02	253.84	251.34	251.93	251.05	240.31	255.16	255.16	252.66	234.72	256.63	262.66	262.66	262.66	235.90
DS28	9803.49	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9803.49	9800.55	9800.55	9801.72	9801.72	9800.55	9802.61	9800.55	9802.61	9801.14
DS29	4926.53	5133.43	5133.43	5133.43	4921.08	5133.43	4926.37	5007.85	5133.43	5133.43	5103.72	5123.43	4891.81	5123.43	5110.93	5133.43	5133.43	5126.22	4829.16	5126.66
DS30	145.31	147.95	145.16	147.66	147.95	142.95	145.45	145.16	145.45	145.75	145.31	147.07	149.28	147.07	181.78	146.04	144.28	144.28	147.07	157.36

Table 17 Threshold values of feature selection techniques (a), classification methods (b), and the on-average case (c)

(a)

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.6, \delta_s = 0.65$		
	Constant	Coefficient	Threshold	Constant	Coefficient	Threshold	Constant	Coefficient	Threshold
AM	–20.57	0.37	55.68	–10.51	0.29	35.93	–10.73	0.55	19.53
FR1	–11.67	0.26	44.60	–13.28	0.33	40.02	–409.67	13.27	30.87
FR2	–8.26	0.18	45.28	–13.49	0.34	39.99	–257.77	9.44	27.30
FR3	–9.28	0.19	49.95	–18.69	0.46	40.42	–14.58	0.53	27.34
FR4	–7.02	0.16	44.87	–7.91	0.22	35.56	–10.99	0.43	25.80
FR5	–613.68	12.66	48.47	–2065.05	46.09	44.80	–257.77	9.44	27.30
FS1	–11.89	0.26	45.51	–10.63	0.26	41.19	–12.67	0.48	26.38
FS2	–6.86	0.15	46.89	–7.75	0.19	40.18	–12.67	0.48	26.38
FS3	–12.10	0.26	46.45	–8.48	0.21	40.78	–12.67	0.48	26.38
FS4	–7.21	0.16	44.78	–10.01	0.28	35.98	–409.67	13.27	30.87

(b)

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.6, \delta_s = 0.65$		
	Constant	Coefficient	Threshold	Constant	Coefficient	Threshold	Constant	Coefficient	Threshold
C1	–7.22	0.17	43.66	–9.36	0.26	35.77	–17.53	0.75	23.28
C2	–18.87	0.36	51.73	–18.63	0.45	41.40	–13.46	0.52	26.02
C3	–23.78	0.47	50.78	–16.71	0.38	43.73	–11.24	0.40	28.28
C4	–7.50	0.17	43.18	–8.45	0.23	36.17	–15.02	0.64	23.65

(c)

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.6, \delta_s = 0.65$		
	Constant	Coefficient	Threshold	Constant	Coefficient	Threshold	Constant	Coefficient	Threshold
FR	–7.65	0.16	46.49	–9.07	0.29	31.37	–10.29	0.6	17.13
FS	–4.5	0.12	38.64	–10.39	0.35	29.5	–10.57	0.61	17.22
Overall	–8.98	0.19	47.28	–10.11	0.26	39.24	–11.68	0.45	25.72

mean difference shows that feature subset selection methods outperform feature-ranking methods.

RQ7: From Table 11 and Figs. 5 and 6, we conclude that the performance of the extreme learning method varies with the different sets of source code metrics. This shows that the selection of a classification technique to develop a model for predicting fault-prone classes is affected by the selection of source code metrics.

7 Threat to validity

For thoroughness, some existing threats to the validity of the proposed work are considered. The proposed work may suffer from the following threats:

1. The results obtained are based on historical data for open-source software, which has specific characteristics as well as behavior and cannot be generalized.

2. All case studies are designed in Java language. The models designed here are likely to be valid for other OO programming languages. Further research can be extended to design a model for other programming paradigms.

3. Only 20 metrics are used to develop a model. Some of other metrics that are also considered for OO software systems can be further considered for fault prediction.

4. Models developed for fault prediction predict only whether a class is faulty. They do not indicate the possible number of bugs in the class.

5. A number of human aspects associated with the software development process also affect the reliability of software systems. In this study, however, these are not considered, including different levels of developer expertise, standards to which software is developed, types of developers involved, development history of the desired system, and other stakeholders associated with the development process.

8 Conclusions and future work

In this study, we have proposed a cost-based evaluation framework for determining the efficiency of a fault prediction model developed using source code metrics. Ten sets of OO source code metrics were used to develop the model using an extreme learning machine with different kernels. The implementation process was carried out using 30 OO software systems. The results are generated using MATLAB. Our observations are outlined below:

1. Our experiment results suggest that it is possible to identify a small subset of OO source code metrics. The fault prediction model developed using this identified set of OO source code metrics is able to predict faulty and non-faulty classes with higher accuracy and reduced misclassified errors.

2. Based on t-test analysis, it is observed that there exists a significant difference between the results obtained using all metrics and the sets of selected source code metrics. It is also observed that the models developed using different classification techniques are significantly different.

3. Based on the mean difference, it is observed that the model developed using a set of selected source code metrics as the input has better performance than the model developed using all metrics. It is also observed that the model developed using extreme learning with the polynomial kernel yields better results compared to other kernels.

4. The selection of a classification method for developing a fault prediction model is affected by feature selection methods.

5. Based on the cost analysis framework, it is observed that as the percentage of faulty classes increases, the fault prediction technique tends to have a higher $\mathrm{NE}_{\mathrm{cost}}$. Cost analysis results reveal that our fault prediction model is most suitable for projects with the percentage of faulty classes below the threshold value depending on the fault identification efficiency (high: 25.72%; median: 39.24%; low: 47.28%).

In the future, this work can be replicated to other open-source projects using deep learning methods in the hope of achieving higher accuracy in fault prediction.

Acknowledgements

The researchers are grateful to the FIST project, of DST, government of India for sponsoring the work on web engineering and cloud based computing. The researchers are thankful to the Computer Science & Engineering Department, NIT Rourkela, for providing all facilities and guidance.

References

Abaei G, Selamat A, Fujita H, 2015. An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. *Knowl-Based Syst*, 74:28-39. https://doi.org/10.1016/j.knosys.2014.10.017

Aggarwal KK, Singh Y, Kaur A, et al., 2009. Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study. *Softw Process Improv Pract*, 14(1):39-62. https://doi.org/10.1002/spip.389

Arisholm E, Briand LC, Johannessen EB, 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Emp Softw Eng*, 83(1):2-17. https://doi.org/10.1016/j.jss.2009.06.055

Briand LC, Wüst J, Daly JW, et al., 2000. Exploring the relationships between design measures and software quality in object-oriented systems. *J Syst Softw*, 51(3):245-273. https://doi.org/10.1016/S0164-1212(99)00102-8

Camargo Cruz AE, Ochimizu K, 2009. Towards logistic regression models for predicting fault-prone code across software projects. Proc 3rd Int Symp on Empirical Software Engineering and Measurement, p.460-463. https://doi.org/10.1109/ESEM.2009.5316002

Cartwright M, Shepperd M, 2000. An empirical investigation of an object-oriented software system. *IEEE Trans Softw Eng*, 26(8):786-796. https://doi.org/10.1109/32.879814

Chidamber SR, Kemerer CF, 1991. Towards a metrics suite for object-oriented design. Proc 6th ACM Conf on Object-Oriented Programming Systems, Languages, and Applications, p.197-211. https://doi.org/10.1145/118014.117970

Chidamber SR, Kemerer CF, 1994. A metrics suite for object-oriented design. *IEEE Trans Softw Eng*, 20(6):476-493. https://doi.org/10.1109/32.295895

Dash M, Liu H, 2003. Consistency-based search in feature selection. *Artif Intell*, 151(1-2):155-176. https://doi.org/10.1016/S0004-3702(03)00079-1

Doraisamy S, Golzari S, Mohd N, et al., 2008. A study on feature selection and classification techniques for automatic genre classification of traditional malay music. ISMIR, p.331-336.

El Emam K, Melo W, Machado JC, 2001. The prediction of faulty classes using object-oriented design metrics. *J Syst Softw*, 56(1):63-75. https://doi.org/10.1016/S0164-1212(00)00086-8

Erturk E, Sezer EA, 2015. A comparison of some soft computing methods for software fault prediction. *Exp Syst Appl*, 42(4):1872-1879. https://doi.org/10.1016/j.eswa.2014.10.025

Fokaefs M, Mikhaiel R, Tsantalis N, et al., 2011. An empirical study on web service evolution. IEEE Int Conf on Web Services, p.49-56. https://doi.org/10.1109/ICWS.2011.114

Forman G, 2003. An extensive empirical study of feature selection metrics for text classification. *J Mach Learn Res*, 3(2):1289-1305.

Furlanello C, Serafini M, Merler S, et al., 2003. Entropy-based gene ranking without selection bias for the predictive classification of microarray data. *BMC Bioinform*, 4(1):54. https://doi.org/10.1186/1471-2105-4-54

Gao K, Khoshgoftaar TM, Wang H, et al., 2011. Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Softw Pract Exp*, 41(5):579-606. https://doi.org/10.1002/spe.1043

Goyal R, Chandra P, Singh Y, 2014. Suitability of KNN regression in the development of interaction based software fault prediction models. *IERI Proc*, 6:15-21. https://doi.org/10.1016/j.ieri.2014.03.004

Gyimothy T, Ferenc R, Siket I, 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans Softw*, 31(10):897-910. https://doi.org/10.1109/TSE.2005.112

Halstead MH, 1977. Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc., New York, NY, USA.

Huang GB, Zhu QY, Siew CK, 2006. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489-501. https://doi.org/10.1016/j.neucom.2005.12.126

Huitt R, Wilde N, 1992. Maintenance support for object-oriented programs. *IEEE Trans Softw Eng*, 18(12):1038-1044. https://doi.org/10.1109/ICSM.1991.160324

Jiang Y, Cukic B, Ma Y, 2008. Techniques for evaluating fault prediction models. *Emp Softw Eng*, 13(5):561-595. https://doi.org/10.1007/s10664-008-9079-3

Jing XY, Ying S, Zhang ZW, et al., 2014a. Dictionary learning based software defect prediction. Proc 36th Int Conf on Software Engineering, p.414-423. https://doi.org/10.1145/2568225.2568320

Jing XY, Zhang ZW, Ying S, et al., 2014b. Software defect prediction based on collaborative representation classification. Companion Proc 36th Int Conf on Software Engineering, p.632-633. https://doi.org/10.1145/2591062.2591151

Jing XY, Wu F, Dong XW, et al., 2015. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. Proc 10th Joint Meeting on Foundations of Software Engineering, p.496-507. https://doi.org/10.1145/2786805.2786813

Jing XY, Wu F, Dong XW, et al., 2017. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Trans Softw Eng*, 43(4):321-339. https://doi.org/10.1109/TSE.2016.2597849

Jones C, 2010. Software quality in 2010: a survey of the state of the art. http://semat.org/documents/20181/27952/software_quality_survey_2010.pdf/7cf00a73-c290-47fe-a5ff-4449ba32f65b

Kanmani S, Uthariaraj VR, Sankaranarayanan V, et al., 2007. Object-oriented software fault prediction using neural networks. *Inform Softw Technol*, 49(5):483-492. https://doi.org/10.1016/j.infsof.2006.07.005

Kapila H, Singh S, 2013. Analysis of CK metrics to predict software fault-proneness using Bayesian inference. *Int J Comput Appl*, 74(2):1-4. https://doi.org/10.5120/12854-9152

Kohavi R, 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. Proc 14th Int Joint Conf on Artificial Intelligence, p.1137-1143.

Kohavi R, John GH, 1997. Wrappers for feature subset selection. *Artif Intell*, 97(1):273-324. https://doi.org/10.1016/S0004-3702(97)00043-X

Li W, Henry S, 1993. Maintenance metrics for the object-oriented paradigm. Proc 1st Int Software Metrics Symp, p.52-60. https://doi.org/10.1109/METRIC.1993.263801

Lorenz M, Kidd J, 1994. Object-Oriented Software Metrics. Prentice-Hall, Englewood Cliffs, NJ.

Malhotra R, Jain A, 2012. Fault prediction using statistical and machine learning methods for improving software quality. *J Inform Process Syst*, 8(2):241-262. https://doi.org/10.3745/JIPS.2012.8.2.241

Malhotra R, Singh Y, 2011. On the applicability of machine learning techniques for object-oriented software fault prediction. *Softw Eng Int J*, 1(1):24-37.

McCabe TJ, 1976. A complexity measure. *IEEE Trans Softw Eng*, 2(4):308-320. https://doi.org/10.1109/TSE.1976.233837

Mende T, Koschke R, 2009. Revisiting the evaluation of defect prediction models. Proc 5th Int Conf on Predictor Models in Software Engineering, p.1-10. https://doi.org/10.1145/1540438.1540448

Mende T, Koschke R, 2010. Effort-aware defect prediction models. 14th European Conf on Software Maintenance and Reengineering, p.107-116. https://doi.org/10.1109/CSMR.2010.18

Mishra B, Shukla KK, 2012. Defect prediction for object oriented software using support vector based fuzzy classification model. *Int J Comput Appl*, 60(15):8-16. https://doi.org/10.5120/9766-3114

Nagappan N, Williams L, Vouk M, et al., 2005. Early estimation of software quality using in-process testing metrics: a controlled case study. *ACM SIGSOFT Softw Eng Notes*, 30(4):1-7. https://doi.org/10.1145/1082983.1083304

Novakovic J, 2010. The impact of feature selection on the accuracy of Naive Bayes classifier. 18th Telecommunications Forum TELFOR, p.1113-1116.

Olague HM, Etzkorn LH, Gholston S, et al., 2007. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Trans Softw Eng*, 33(6):402-419. https://doi.org/10.1109/TSE.2007.1015

Pai GJ, Dugan JB, 2007. Empirical analysis of software fault content and fault proneness using Bayesian methods. *IEEE Trans Softw Eng*, 33(10):675-686. https://doi.org/10.1109/TSE.2007.70722

Pawlak Z, 1982. Rough sets. *Int J Comput Inform Sci*, 11(5):341-356.

Plackett RL, 1983. Karl Pearson and the Chi-squared test. *Int Statist Rev*, 51(1):59-72. https://doi.org/10.2307/1402731

Shatnawi R, Li W, 2008. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *J Syst Softw*, 81(11):1868-1882.

Singh Y, Kaur A, Malhotra R, 2010. Empirical validation of object-oriented metrics for predicting fault proneness models. *Softw Qual J*, 18(1):3-35. https://doi.org/10.1007/s11219-009-9079-6

Slowinski R, 1992. Intelligent decision support. In: Handbook of Applications and Advances of the Rough Sets Theory. Kluwer Academic Publishers, Dordrecht, p.396. https://doi.org/10.1016/0165-0114(93)90040-O

Tomaszewski P, Håkansson J, Grahn H, et al., 2007. Statistical models vs. expert estimation for fault prediction in modified code—an industrial case study. *J Syst Softw*, 80(8):1227-1238. https://doi.org/10.1016/j.jss.2006.12.548

Wagner S, 2006. A literature survey of the quality economics of defect-detection techniques. Proc ACM/IEEE Int Symp on Empirical Software Engineering, p.194-203. https://doi.org/10.1145/1159733.1159763

Wang D, Romagnoli JA, 2005. Robust multi-scale principal components analysis with applications to process monitoring. *J Process Contr*, 15(8):869-882. https://doi.org/10.1016/j.jprocont.2005.04.001

Wang T, Zhang Z, Jing X, et al., 2016. Multiple kernel ensemble learning for software defect prediction. *Autom Softw Eng*, 23(4):569-590. https://doi.org/10.1007/s10515-015-0179-1

Zhou Y, Leung H, 2006. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Trans Softw Eng*, 32(10):771-789. https://doi.org/10.1109/TSE.2006.102

Zhou Y, Xu B, Leung H, 2010. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. *J Syst Softw*, 83(4):660-674. https://doi.org/10.1016/j.jss.2009.11.704