The Journal of Systems and Software 137 (2018) 686–712

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM)

Lov Kumar[a,*], Sai Krishna Sripada[b], Ashish Sureka[c], Santanu Ku. Rath[d]

[a] Department of Computer Science & Engineering, National Institute of Technology, Rourkela, India
[b] International Institute of Information Technology, Hyderabad, India
[c] ABB Corporate Research Center, Bangalore, India
[d] Department of Computer Science & Engineering, National Institute of Technology, Rourkela, India

ARTICLE INFO

Article history:
Received 15 April 2016
Revised 2 April 2017
Accepted 19 April 2017
Available online 20 April 2017

Keywords:
CK metrics
Cost analysis
Fault
Feature selection techniques
Least Squares Support Vector Machine (LSSVM)
Object-oriented software

ABSTRACT

Software developers and project teams spend considerable amount of time in identifying and fixing faults reported by testers and users. Predicting defects and identifying regions in the source code containing faults before it is discovered or invoked by users can be valuable in terms of saving maintenance resources, user satisfaction and preventing major system failures post deployment. Fault prediction can also improve the effectiveness of software quality assurance activities by guiding the test team to focus efforts on fault prone components. The work presented in this paper involves building an effective fault prediction tool by identifying and investigating the predictive power of several well-known and widely used software metrics for fault prediction. We apply ten different feature selection techniques to choose the best set of metrics from a set of twenty source code metrics. We build the fault prediction model using Least Squares Support Vector Machine (LSSVM) learning method associated with linear, polynomial and radial basis function kernel functions. We perform experiments on 30 Open Source Java projects. Experimental results reveals that our prediction model is best suitable for projects with faulty classes less than the threshold value depending on fault identification efficiency (low- 52.139%, median- 46.206%, and high- 32.080%).

1. Introduction

Modern software systems are built using Object Oriented (OO) paradigm for its effective design features such as enhancing code re-use and reducing fragility which enables faster product development. OO Paradigm also assists in optimal characterization (for example, abstraction, encapsulation, inheritance and polymorphism) of the software system compared to other paradigms. Along with these design strategies, industry follows certain quality assurance processes and practises like peer code reviews, deployment of bug tracking systems to ensure quality of software and reduce faults in a system. However, exhaustive testing and ensuring a complex large scale system to be fault-free is not practically possible. Software fault prediction has emerged as a novel research field to identify faulty classes among sub-systems at early stage of software development life cycle. Fault prediction helps the testing team in focusing their testing efforts and resources optimally. Fault prediction tools and statistical models are constructed based on the application of software metric suites (predictors or indicators) such as Abreu MOOD metrics suite (Abreu and Carapuça, 1994; Bieman and Kang, 1995; Briand et al., 2000; Halstead, 1977; Henderson-Sellers, 1996; Li and Henry, 1993; McCabe, 1976; Lorenz and Kidd, 1994), and CK metrics (Chidamber and Kemerer, 1994) suite. In the study presented in this paper, twenty object oriented metrics were considered while developing a statistical model to predict whether a class in an OO software sub-system is faulty or not.

Performance of the fault prediction tool depends on choosing the right set of object oriented source code metrics. The input features to the predictive model has a major impact on the performance of the fault prediction model. Feature selection is a process of selecting a suitable subset of object-oriented metrics from the list of available metrics. Feature selection methods are classified into two subclasses consisting of feature ranking and feature subset selection methods. In feature ranking methods, decisive factors are considered to rank each individual feature and higher ranked features applicable for a given project are chosen. In feature subset selection methods, subset of features are identified to collectively

* Corresponding author.
E-mail addresses: lovkumar505@gmail.com (L. Kumar), saikrishna.sripada@research.iiit.ac.in (S.K. Sripada), ashish.sureka@in.abb.com (A. Sureka), skrath@nitrkl.ac.in (S.Ku. Rath).

http://dx.doi.org/10.1016/j.jss.2017.04.016
0164-1212/

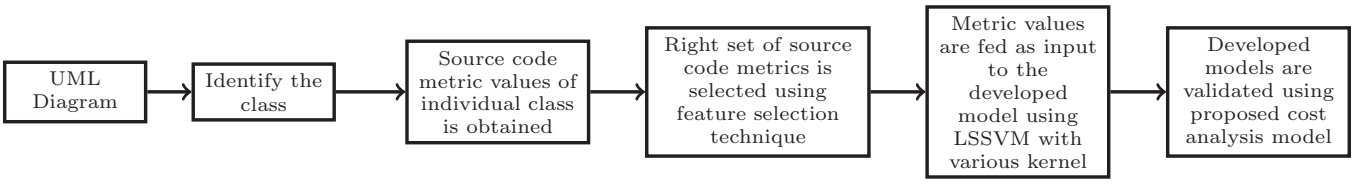


Fig. 1. Flow chart of the proposed work.

improve predictive capability. In our work, six different types of feature ranking and four different types of feature subset selection methods are applied for finding the right subset of source code metrics. These chosen subset of metrics help in reducing the value of mis-classification errors because it removes irrelevant features and retains features which have good discriminatory power (Kumar et al., 2016b; 2016c; Kumar and Rath, 2016; Kumar et al., 2016a; 2016d).

Our literature survey reveals that there are a several methods employed by researchers for software fault prediction. Some of the extensively employed approaches for fault prediction are regression, association rule mining, clustering, classification, neural network and decision tree learning algorithms. Development of accurate fault prediction model to predict classes which are prone to faults is still a challenging task and unsolved problem in software engineering discipline. In this study, Least Square Support Vector Machine (LSSVM) method with linear kernel, polynomial kernel, and radial basis function kernels are considered while developing a model to predict whether a class is faulty or not. Least Square Support Vector Machine(LSSVM) is a least square version of support vector machine(SVM) which is based on the theory of statistical learning. The application of LSSVM are many, such as analyzing data and recognizing patterns which are used for classification and regression analysis. In this study, LSSVM method used as a classification method to develop a model to predict faulty and non-faulty classes (Kumar et al., 2016b; 2016c; Kumar and Rath, 2016; Kumar et al., 2016a; 2016d).

Previous research (Erturk and Sezer, 2015; Goyal et al., 2014; Abaei et al., 2015) focuses on building and improving the performance of fault prediction models out of which only few studies (Jiang et al., 2008; Arisholm et al., 2010; Mende and Koschke, 2009; 2010) focus on effectiveness of fault prediction. In this study, we propose a cost evaluation framework (one of the main focus of our work and unique contribution) to perform cost based analysis for evaluating the effectiveness of fault prediction model. We considered 30 open source software projects based on object oriented paradigm to conduct this study. Each software projects differ in terms of percentage of faulty classes. The cost analysis framework depend on the misclassified faulty classes among these projects.

The list of steps followed by us in developing an effective fault prediction models are shown in Fig. 1. Initially the classes of the respective software are identified from the class diagram and next, the different metric values of a class are extracted using different publicly available tools. Further, right set of source code metrics is selected using feature selection technique. Next these selected set of source code metrics are considered as input to develop a model using LSSVM with different kernel. Finally, these developed models are validated using our proposed cost analysis models.

The unique and novel contributions of the study presented in this paper are as follows:

- Development of a fault prediction model involving object oriented source code metrics suitable for all object oriented software projects.
- A cost analysis framework to evaluate the fault prediction models.

The rest of the paper is organized as follows: Section 2 presents the existing literature in the field of fault prediction. Section 3 highlights on research background. Section 4 describes the feature selection methods and Section 5 describes the Least Square Support Vector Machines (LSSVM) with various kernels for fault prediction. We propose the cost analysis of the model in Section 6 and describe the performance parameters used for evaluating the models in Section 7. Sections 8 and 9 presents the experimental study and results. In Section 10, threats to validity have been discussed and Section 11 concludes the paper with our thoughts on scope for future work.

2. Related work

This section presents a review of literature on the use of software metrics and their application in fault prediction as well as work done on cost based evaluation framework for fault prediction techniques. The relationship between object-oriented metrics and the fault proneness derived from existing literature is shown in Table 1. In Table 1, the first column indicates the name of the author, and the year in which the work was carried out. The second column indicates the data-set used, and the third column represents the methods employed for fault prediction. Fourth column represents subset of object-oriented metrics considered while developing a model to predict faulty or non-faulty classes.

From Table 1, it is observed that, independent variables are the subset of the object-oriented metrics and the dependent variable is the fault proneness. This shows that the performance of fault prediction model depends on the source code metrics which have been considered as input to develop a model. In this work, a study on the influence of different subsets of object-oriented source code metrics (a set of all metrics and ten reduced sets of metrics identified using feature selection techniques) on quality attributes, and design of relevant models which help to predict these quality attributes. Our Observations suggests logistic regression, decision tree analysis, Naive Bayes classifier, neural network are commonly used by most authors. In this paper, Least Square Support Vector Machine (LSSVM) method with various kernel methods i.e., linear kernel, polynomial kernel, and radial basis function kernel are considered to design a model to predict faulty or non-faulty classes. In recent year, LSSVM models have seen an explosion of interest, and their applicability across a wide range of problem domains (Suykens et al., 2002). LSSVM models can be mainly used to solve problems related to prediction and classification.

Earlier, researchers presented cost based evaluation models for predicting the effectiveness of fault prediction. In this section, we present our literature survey on measuring cost effectiveness for fault prediction in Table 2. Table 2 emphasizes on the studies carried out by different authors to compare the prediction models, and the evaluation criterion is considered in choosing effective fault prediction techniques. In this study, cost-based evaluation framework is proposed to assess the effectiveness of developed fault prediction models by considering twenty object-oriented metrics as requisite input. We attempt to assess the influence of fault removal cost to know whether the performing fault

Table 1
Summary of literature on reliability prediction using static source code metrics.

Author	Data set	Method used	Metric used
Briand et al. (2000)	Hypothetical video rental business	Logistic regression, Principal component analysis	28 coupling, 10 cohesion, and 11 inheritance metrics
Cartwright and Shepperd (2000)	Large European telecommunication industry, which consists of 32 classes and 133KLOC.	Spearmans rank correlation	ATTRIB, DELS, DFCT, EVNT, LOC, NOC, READS, RWD. DIT, STATES, and WRITES
Emam et al. (2001)	Used two versions of Java application: Ver 0.5 and Ver 0.6 consisting of 69 and 42 classes respectively.	Logistic regression	CK, and Briand metrics
Gyimothy et al. (2005)	Source code of Mozilla with the use of Columbus framework.	Decision tree, Linear and Logistic regression, Neural network	CK metrics suite, LCOM, and LOC
Nagappan et al. (2005)	Open source eclipse plug-in.	Multiple linear regression, Principal component analysis	STREW-J metric suite (Number of Assertions, Number of Test cases)
Zhou and Leung (2006)	NASA consisting of 145 classes, 2107 methods and 40 KLOC.	Logistic regression, Naive Bayes, and Random forest	CK metrics suite, and SLOC
Olague et al. (2007)	Mozilla Rhino project.	Logistic regression	CK metrics, MOOD, and QMOOD metrics
Kanmani et al. (2007)	Library management system consists of 1185 classes.	Logistic regression, Neural network, Probabilistic neural network	10 cohesion, 18 inheritance, 29 coupling, and 7 size measures (total 64 metrics)
Pai and Dugan (2007)	Public domain dataset consists of 2107 methods, 145 classes, and 43 KLOC.	Bayesian linear regression, Bayesian poisson regression, Multiple regression, and Ordinary least square	CK metric suite, and SLOC
Tomaszewski et al. (2007)	Two telecommunication project developed by Ericsson, consisting of 800 classes (500KLOC) and 1000 classes(600KLOC) respectively.	Linear regression, and Expert estimation	CK metrics suite
Shatnawi and Li (2008)	Eclipse project: Bugzilla database and Change log.	Logistic regression	CK metrics suite, Lorenz and Kidd
Aggarwal et al. (2009)	Student projects at University School of Information Technology (USIT).	Statistical regression analysis, and Principal component analysis	Coupling, Cohesion, Inheritance, and Size metrics
Singh et al. (2009)	NASA consists of 145 classes, 2107 methods and 40K LOC.	Support vector machine	CK metrics suite, and SLOC
Cruz and Ochimizu (2009)	638 classes of Mylyn software	Logistic regression	RFC, CBO, and WMC
Burrows et al. (2010)	iBATIS, Health watcher, Mobile media.	Logistic regression	CAE, CBM, DIT, and coupling metrics
Singh et al. (2010)	NASA consists of 145 classes, 2107 methods, and 40K LOC.	Logistic regression, ANN, and Decision tree	CK metric suite, and SLOC
Zhou et al. (2010)	Three releases of Eclipse, consisting of 6751, 7909, 10,635 java classes and 796, 988, 1306 KLOC respectively.	Logistic regression	AMC, aVGloc, CCMax, LOC, NIM, NCM, NTM, NLM, SDMC, and WMC
Fokaefs et al. (2011)	NASA data set.	Decision tree analysis	WMC, NOC, LCOM, and CBO
Malhotra and Singh (2011)	Open source software.	ANN, Bagging, Random forest, Boosting, Naive Bayes, and Kstar	CK metrics suite, Ca, Ce, NPM, LCOM3, LOC, DAM, MOA, MFA, CAM, IC, CBM, Maxcc, and Avgcc
Mishra and Shukla (2012)	Eclipse and Equinox data sets.	SVM, Fuzzy, and Naive Bayes	Complexity metrics (FOUT, MLOC) and Abstract syntax tress based metrics
Malhotra and Jain (2012)	Apache POI.	Logistic regression, random forest, Adaboost, Bagging, Multilayer perceptron, SVM, and Genetic programming	CK metrics suite, Ca, Ce, NPM, LCOM3, LOC, DAM, MOA, MFA, CAM, IC, CBM, Maxcc and Avgcc
Kapila and Singh (2013)	Open Source Eclipse System.	Bayesian inference	CBO, and NOC
Goyal et al. (2014)	Eclipse, Mylyn, Equinox and PDE	K-Nearest Neighbor (KNN) regression	CK metrics and 11 OO (Object Oriented) metrics
Erturk and Sezer (2015)	promise software engineering repository data	ANN, SVM and ANFIS	McCabe metrics
Abaei et al. (2015)	NASA data sets	semi-supervised hybrid self-organizing map (HySOM)	McCabe and Halstead metrics

Table 2
Fault prediction effectiveness based on cost evaluation model (Kumar et al., 2016b; 2016c; Kumar and Rath, 2016; Kumar et al., 2016a; 2016d).

Author	Cost evaluation criteria
Jiang et al. (2008)	Introduced cost curve based on Receiver Operating Characteristic (ROC).
Mende and Koschke (2009)	Introduced a performance measure (P_{opt}) and compared prediction model with an optimal model. P_{opt} accounted module size to evaluate the performance of a fault-prediction technique.
Mende and Koschke (2010)	Proposed two strategies namely AD (effort-aware binary prediction) and DD (effort-aware prediction based on defect density) to include the notion of effort awareness into fault-prediction techniques.
Arisholm et al. (2010)	Proposed a cost performance measure - Cost Effectiveness (CE), a variation of lift charts where the x-axis contains the ratio of lines of code instead of modules.

Table 3
Software metrics.

OO metric	Description	Suggested by
Weighted methods per class (WMC)	Sum of the complexities of methods defined in class	Chidamber and Kemerer (1994)
Depth of inheritance tree (DIT)	Maximum height of the class hierarchy	Chidamber and Kemerer (1994)
Number of children (NOC)	Number of immediate descendants of the class	Chidamber and Kemerer (1994)
Coupling between object classes (CBO)	Number of classes coupled to a given class	Chidamber and Kemerer (1994)
Response for a Class (RFC)	Number of different methods that can be executed when an object of that class receives a message	Chidamber and Kemerer (1994)
Lack of cohesion in methods (LCOM)	Number of sets of methods in a class that are not related through the sharing of some of the class's fields	Chidamber and Kemerer (1994)
Afferent coupling (Ca)	Number of other classes use the specific class	Martin (1994)
Efferent coupling (Ce)	Number of classes used by the specific class	Martin (1994)
Number of public methods (NPM)	Number of methods in a class that are declared as public.	Bansiya and Davis (2002)
LCOM3	Lack of cohesion in methods Henderson-Sellers version	Henderson-Sellers (1996)
Lines of code (LOC)	Number of lines in the text of the source code	Halstead (1977)
Data access metric (DAM)	Ratio of the number of private (protected) attributes to the total number of attributes declared in the class	Bansiya and Davis (2002)
Measure of aggregation (MOA)	Number of data declarations (class fields) whose types are user defined classes	Bansiya and Davis (2002)
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.	Bansiya and Davis (2002)
Cohesion among methods of class (CAM)	Sum of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.	Bansiya and Davis (2002)
Inheritance coupling (IC)	Number of parent classes to which a given class is coupled.	Tang et al. (1999)
Coupling between methods (CBM)	Number of new/redefined methods to which all the inherited methods are coupled	Tang et al. (1999)
Average method complexity (AMC)	Average method size for each class.	Tang et al. (1999)
Maximum McCabe's cyclomatic complexity ($Max-CC$)	Maximum cyclomatic complexity of methods defined in a class	McCabe (1976)
Average McCabe's cyclomatic complexity ($Avg-CC$)	Average cyclomatic complexity of methods defined in a class	McCabe (1976)

prediction analysis is useful or not (Kumar et al., 2016b; 2016c; Kumar and Rath, 2016; Kumar et al., 2016a; 2016d).

3. Research background

The following sub-sections describes the dataset being used for fault prediction. Data was further normalized to obtain better accuracy and identify dependent & independent variables for fault prediction.

3.1. Software metrics

Number of software metrics have been proposed for predicting faults in software by Chidamber and Kemerer (1994), Basili et al. (1996) and Li and Henry (1993). In this study 20 object-oriented metrics of size, complexity, coupling, cohesion, and inheritance are considered to develop fault prediction models. Table 3 displays the basic definitions of various software metrics considered. These metrics include the popular metrics suite proposed by Chidamber and Kemerer (1994), Martin (1994), Tang et al. (1999); McCabe (1976), and Henderson-Sellers (1996).

3.2. Effectiveness of metrics

The goal of this study is to explore the relationship between Object-Oriented metrics and fault proneness of associated classes. In this paper, a fault in a class is considered as a dependent variable and software metrics are considered as independent variables. Fault is assumed to be a function of various metrics such as WMC, DIT, NOC, CBO, RFC, LCOM, Ca, Ce, NPM, LCOM3, LOC, DAM, MOA, MFA, CAM, IC, CBM, AMC, Max-CC, and Avg-CC.

To analyze the effectiveness of the metrics used, they are categorized into three groups as follows:

a. **Analysis 1 (A1)**: The effectiveness of object oriented metrics are used for predicting fault proneness at the class level. The relationship between faults and metrics can be represented in the following manner:

$$Faults = f(WMC, DIT, NOC, CBO, RFC, LCOM, Ca, Ce, NPM, LCOM3, LOC, DAM, MOA, MFA, CAM, IC, CBM, AMC, Max-CC, Avg-CC)$$

b. **Analysis 2 (A2)**: In this study, reduced feature attributes using *feature ranking* methods are considered as input to develop a model for predicting fault proneness at class-level in object oriented software. In this study, six feature ranking methods are considered to rank the features (static source code metrics) based on their performance. According to Gao et al. (2009), it is appropriate to select top $\lceil log_2 n \rceil$ features when using feature ranking methods to find reduced feature set. In this paper, top $\lceil log_2 n \rceil$ metrics have been considered to predict whether the

class is faulty or not, where "n" shows the number of software metrics in the original data set. The relationship can be represented as follows:

$$Faults = f(\text{Reduced subset of metrics using feature ranking methods})$$

c. **Analysis 3 (A3)**: In this study, reduced feature attributes using *feature subset* selection methods are considered as input to develop a model for predicting fault proneness at the class level of object oriented software. Four feature subset selection methods are considered by us in our experiments to find reduced sub set of features. The relationship can be represented as follows:

$$Faults = f(\text{Reduced subset of metrics using feature subset selection methods})$$

3.3. Dataset description

Previous studies (Table1) consider small number of software projects to investigate the relationships between fault proneness and Object-Oriented metrics. Hence it is not possible to draw generic conclusions applicable to all software projects and systems. In this study, thirty open-source software projects are considered to strengthen and generalize our conclusions. We fetch the experimental dataset for our work from promise repository.[1] Table 4 shows project name, number of classes, number of faulty classes and % faulty classes. The data set also consists of static source code metrics in Table 3 calculated for each of the project chosen. The descriptive statistics and correlation analysis of source code object oriented metrics are available at our repository on GitHub.[2]

3.4. Research questions

An experimental study is carried out to find the effectiveness of fault prediction techniques using the cost based evaluation framework (Kumar et al., 2016b; 2016c; Kumar and Rath, 2016; Kumar et al., 2016a; 2016d). A cost evaluation framework has been proposed to perform cost based analysis for misclassified faults. This study also focuses on identifying the best possible subset of object-oriented source code metrics to predict whether a class is faulty or not. We address the following research questions:

RQ1: Which fault prediction model is most suitable among the different techniques ?
This question helps to investigate the performance of different types of fault-prediction techniques. In this study, different types of fault-prediction techniques have been considered for developing a model by considering object-oriented software metrics as input which are better to predict as to whether the class is a faulty one or not.

RQ2: For a given software product, whether performing fault prediction analysis is economically effective or not?
This question investigates the effectiveness of different fault prediction techniques. In this work, a cost evaluation framework has been proposed which performs cost based analysis for mis-classification of faults.

RQ3: Does a subset of all the object-oriented software metrics performs better than all metrics for the task of predicting if a class is faulty or not ?
In this research question, our aim to evaluate the metrics and examine their relationship with faulty and non-faulty classes. In this study, different types of feature reduction methods have been considered for finding subset of object-oriented software metrics which are better to predict whether the class is faulty or not.

RQ4: Which feature ranking methods work best for the task of predicting if a class is faulty or not?
In feature ranking method, performance of the classifier is influences by the nature and characteristics of the fault dataset. Different methods have been considered with different parameters to rank the features. In our work, two different performance parameters have been considered to compare different feature-ranking methods.

RQ5: Which feature subset selection method performs best for the task of predicting if a class is faulty or not?
Each feature subset selection method has a different mechanism to find the subset of features which are better to predict as to whether a class is faulty or not. In our work, different performance parameters have also been considered to compare different methods.

RQ6: How do the feature ranking methods compare with feature subset selection methods ?
In this study, pairwise *t*-test has been employed to determine whether feature ranking methods work better than feature subset selection methods or they both perform equally well.

RQ7: Do the feature selection methods effect the performance of the classification methods ?
It is observed that some of the feature selection methods work very well with specific classification method. Thus, in this study, different feature selection methods are evaluated using different classification methods to assess their performance. This question focuses on variation of performance of classification method over different classification method.

4. Feature selection methods

From Table 1, it is also observed that in all studies, independent variables are different subset of the OO metrics and the dependent variable is the fault proneness. This shows that the performance of fault prediction model depends on the software metrics which have been considered as input to design a model. Selection of right set of feature is an important data pre-processing task in different application of data mining and machine learning. Various applications of feature selection techniques in different domains have been reported (Forman, 2003; Furlanello et al., 2003; Doraisamy et al., 2008). In area of software fault prediction, some authors uses different feature selection techniques to select suitable set of software metrics. Aggarwal et al. (2009) conducted an empirical validation of OO metrics. Initially they compute, the interrelationships among selected metrics and then they find the individual and combined effect of selected metrics on fault proneness. They apply all these algorithm on three small software system. They conclude that OO metrics found in the predicted model appear to be well suited to develop practical quality benchmarks. Shatnawi and Li (2008) investigated whether the OO metrics could predict the class error probability in the post-release evolution of Eclipse or not. The limitations of the above discussed studies include that no studies were performed on large number of datasets. Consequently, it is not clear whether a particular conclusion could be generalized to other systems. Gondra (2008) investigated the machine learning application for software fault-proneness model development. In their work, they worked on two aspects. First, they conducted software metrics selection using sensitivity analysis and compared it with some other methods such as PCA and concluded

[1] http://openscience.us/repo/defect/.
[2] https://github.com/lov505/JSS-Dataset/.

that the proposed feature selection methods do not use the derived non-intuitive metrics. The second aspect consisted of performing a comparison between performance of model developed using SVM and ANN and they observed superior performance of SVMs over ANNs (Kumar et al., 2016b; 2016c; Kumar and Rath, 2016; Kumar et al., 2016a; 2016d).

In this work, ten different types of feature selection methods are applied on thirty different datasets to find the best subset of OO metrics which help to predict fault proneness with higher accuracy and also reduce the value of mis-classification errors. The following sub-sections highlight on different feature selection methods to find a small subset of object-oriented metrics out of total available object-oriented metrics that collectively have good predictive capability. Feature selection methods can be broadly classified into two subclasses:

- *Feature ranking methods*: In Feature ranking method, some decisive factors have been considered to rank each individual feature and then some of these features are selected which are suitable for a given project.
- *Feature subset selection method*: In feature subset selection, subset of features are searched which collectively have good predictive capability.

4.1. Feature ranking method

Feature ranking methods rank features independently without using any learning algorithm. In feature ranking methods, ranking of features are based on the score of the features. Based on our analysis of the literature, a good number of methods are available for computing the score of each feature. In this study, six feature ranking methods have been considered for computing the score of feature (Kumar et al., 2016b; 2016c; Kumar and Rath, 2016; Kumar et al., 2016a; 2016d). These feature-ranking methods are described below:

4.1.1. Chi Squared test

Chi Squared test is used to test the independence between two events (Plackett, 1983), where, ranking of features are based on the value of the Chi squared statistic with respect to the class. The high value of Chi squared indicates the rejection of the null hypothesis, and thus these features can be identified as good significance.

4.1.2. Gain ratio feature evaluation

In gain ratio feature evaluation method, ranking of features are based on the value of the gain ratio with respect to the class (Novakovic, 2010). The gain ratio of feature 'F' is defined as:

$$\text{Gain Ratio} = \frac{Gain(F)}{SplitInfo_F(N)} \tag{1}$$

Where $Gain(F)$ is defined as follow:

$$Gain(F) = I(N) - E(F) \tag{2}$$

where N represents the set containing n number of samples with m distinct classes. The expected information needed to classify a given sample is computed using following equation:

$$I(N) = -\sum_{i=1}^{m} P_i \log_2(p_i) \tag{3}$$

Where P_i is the probability that an arbitrary sample belongs to class C_i and is estimated by n_i/n.

Let n_{ij} shows the number of data samples of class C_i in subset N_j. The entropy, or expected information based on the partitioning into subsets by F, is given by

$$E(F) = -\sum_{i=1}^{M} I(N) \frac{n_{1i} + n_{2i} + \ldots + n_{mi}}{n} \tag{4}$$

The value of $SplitInfo_F(N)$ is computed using following equation:

$$SplitInfo_F(N) = -\sum_{i=1}^{t} \frac{|N_i|}{N} \log_2 \frac{|N_i|}{N} \tag{5}$$

The value of $SplitInfo_F(N)$ represents the information generated by partitioning the training data set N into t partitions corresponding to t outcomes of a test on the attribute F.

4.1.3. OneR feature evaluation

In oneR feature evaluation method, oneR classifier has been considered for ranking of features (Novakovic, 2010). OneR classifier uses classification rates to rank the features. It considers all numerically valued features as continuous ones and divides the range of values into several disjoint intervals using straightforward method. Features with higher classification rates are considered to be of more significance.

4.1.4. Information gain feature evaluation

In info gain feature evaluation, importance of features are based on the value of the information gain with respect to the class (Novakovic, 2010).

4.1.5. Logistic regression analysis

Logistic regression analysis is a statistical analysis method (Cruz and Ochimizu, 2009). In this analysis, Univariate Logistic Regression (ULR) analysis has been considered to check the level of significance of each of the object-oriented metric. In this study, two parameters of LR model have been considered to find the level of significance of each metric and also used to rank the features. These parameters are

1. *Value of regression coefficient*: The coefficient value of metrics shows the amount of correlation of each metrics with faults.
2. *P-value*: P-value i.e., significance level shows the significance of correlation.

4.1.6. Principal Component Analysis (PCA)

Attribute reduction using Principal Component Analysis (PCA) is achieved by transforming high dimension data space into lower dimension data space. The lower dimension data have the most significant features (Wang and Romagnoli, 2005). Since correlation between some of metrics are high, so PCA is used to transfer these metrics into new metrics which are not highly correlated. The new metrics are called principal component domain metrics. A small number of principal components (PCs) are observed to be sufficient enough to represent most of the significant patterns in the data. The detail steps of PCA is describe below. The detail steps of PCA is described in Fig. 2.

4.2. Feature subset selection method

Feature-subset selection methods are used to find suitable subset of features which collectively have good predictive capability. Feature-subset selection methods are based on the assumption that model has higher accuracy and reduced value of misclassification errors when combined with some other features as compared to when used by itself. Based on our analysis of the literature, a good number of methods are available for finding the subset of features. In this study, four feature subset selection methods have been considered for computing the score of feature. These feature subset selection methods are described below:

4.2.1. Correlation based feature selection

Correlation based feature selection (CFS) method selects a subset of features which are highly correlated with the class. In this study, Pearson's correlations (r: Coefficient of correlation) has been

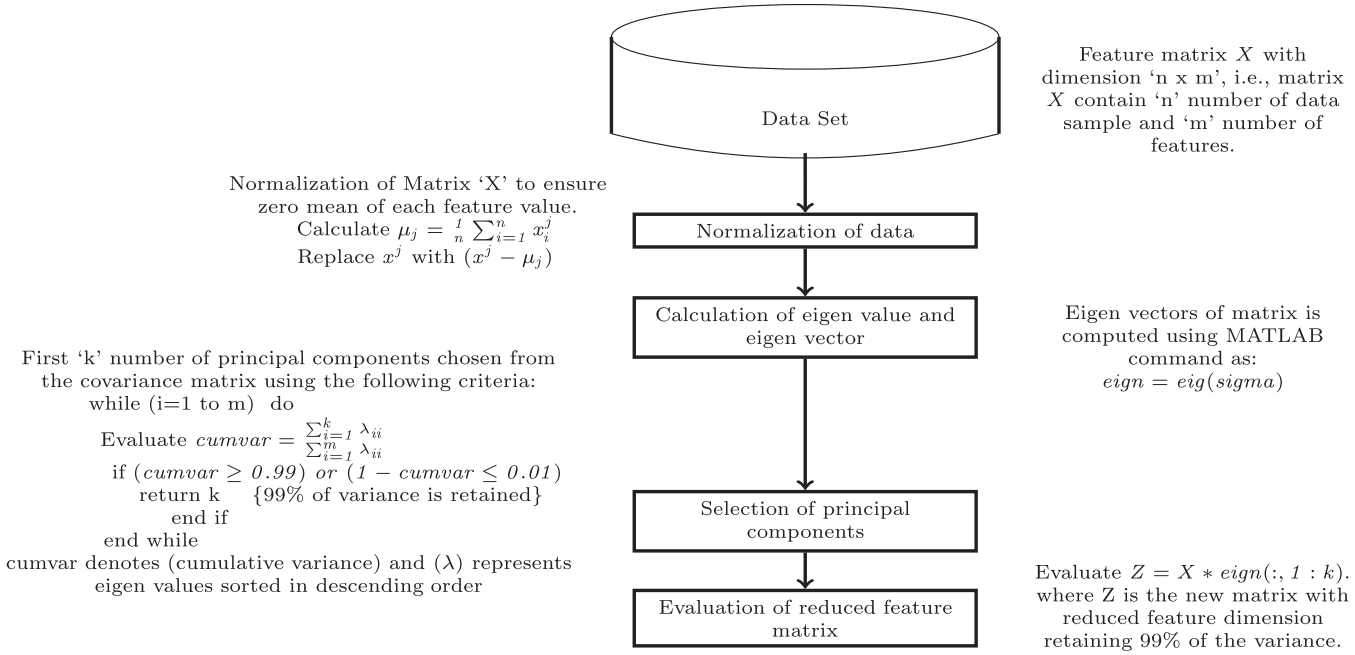


Fig. 2. Framework of PCA calculation.

considered for finding the dependency between metrics. If value of coefficient of correlation is higher between the pair of metrics, then it indicates a strong structural association between these metrics. This aspects suggests that, although metrics measure different features of class design, there is a significant statistical reason to believe that classes with low (or high) metric values also have low (or high) values of other highly correlated metrics.

4.2.2. Rough set analysis (RSA)

Rough set is a formal approximation of a conventional crisp set[3] (i.e., conventional set) in terms of a pair of sets which give the lower and the upper approximation of the original set (Pawlak, 1982). This formal approximation, represents the lower, and upper bound of the original set. The application of rough set analysis makes data pattern more visible by lowering the 'degree of precision'(Huang, 1992). In general, rough set analysis can be viewed as a formal framework for mining facts from imperfect data. In this analysis, RSA is used to find reduced feature set of object oriented metrics. Three different types of notations are used in RSA such as information system, approximations, and reduced attributes.

i. *Information system*: It is defined as I=(U,A), where U is a universe containing non-empty set of finite objects and A is the finite attribute sets. There exist a mapping F_a: $U \rightarrow V_a$ for each $a \in A$, where V_a is the set of value of attribute a. For each set of attribute $B \subset A$, there is an associated equivalence relation called B-indiscerbility (Ind(B)) relation. Ind(B) is defined as follow:

$$IND_A(B) = \{(x, y) \in U^2 | \forall a \in B, a(x) = a(y)\} \quad (6)$$

ii. *Approximation*: Let $S = (U, A)$, $B \subseteq A$ and $X \subseteq U$. B-upper ($\bar{B}X$) and B-lower ($< uline > B < /uline > X$) approximations of X are used to approximate X. The upper approximation contains all objects which possibly belong to the set and the lower approximation consists of all objects which surely belong to the set and the. The $\bar{B}X$ and $< uline > B < /uline > X$ are calculated using following equations:

$$\bar{B}X = \{x_i \in U | [x_i]_{Ind(B)} \cap X \neq 0\} \quad (7)$$

$$\underline{B}X = \{x_i \in U | [x_i]_{Ind(B)} \subset X\} \quad (8)$$

Where $[x_i]_{Ind(B)}$ is the equivalence class of x_i in relation $Ind(B)$

iii. *Reduced attributes:* An accuracy measure of the set X (Acc(X)) in $B \subseteq A$ is defined as:

$$\mu_B(X) = \frac{card(\underline{B}X)}{card(\bar{B}X)} \quad (9)$$

The cardinality of a set is the number of objects contained in the upper or lower approximation of the set X. All possible sets are selected whose accuracy is equal to the accuracy of universal set.

The steps followed to obtain reduced attributes set using RSA are shown in Fig. 3.

4.2.3. Consistency subset evaluation technique

Consistency subset evaluation technique evaluates the worth of a subset of attributes by the level of consistency in the class values when the training instances are projected onto the subset of attributes (Dash and Liu, 2003). The consistency rate is computed using inconsistency rate where two data points are considered inconsistent if they have the same feature values but different class labels. In this study, target variable i.e., fault has two different values (0 for non faulty classes and 1 for faulty classes). A set of feature (FS) have N number of instance, there are n number of patterns such that $N = P_1 + P_2 + \ldots + P_n$. Pattern P_i appears in totally M instances out of which M_0 number of instances are labeled *0* and M_1 number of instances are labeled *1* where $M = M_0 + M_1$. If M_0 is greater than M_1, then the inconsistency count for the pattern P_i is $INC_i = M - M_0$. The inconsistency rate is ratio of sum of all the inconsistent counts over all patterns and number of instances. The inconsistency rate (INCR) of feature set is calculated using following equation:

$$INCR = \frac{\sum_{i=1}^{n} INC_i}{N} \quad (10)$$

4.2.4. Filtered subset evaluation

Filtered subset evaluation is a method for running a random subset evaluator on dataset that has been passed through an ar-

[3] https://en.wikipedia.org/wiki/Rough_set.

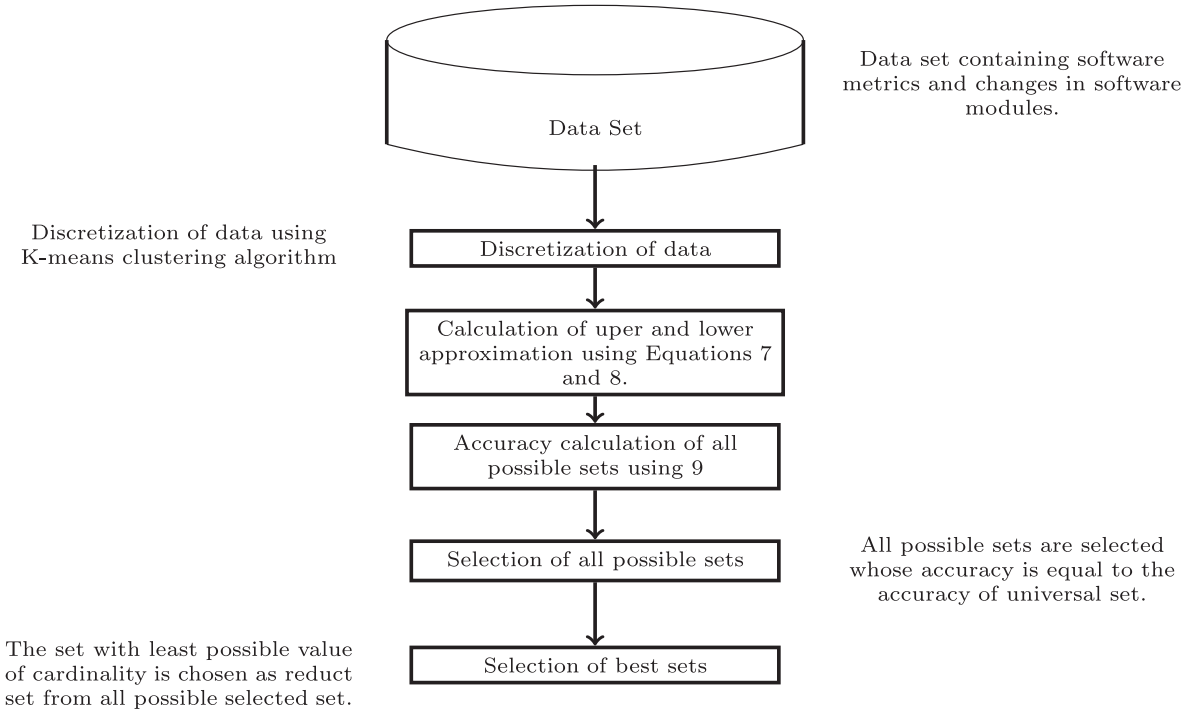


Fig. 3. Framework of rough set theory.

Fig. 4. Filter approach for feature selection.

bitrary filter (Kohavi and John, 1997). The filter approach does not depend on a any learning induction algorithm. The computational complexity of filter approach is fast and scalable. Fig. 4 shows the steps follow to find subset of features using filter approach.

5. Least Square Support Vector Machine (LSSVM) classifier

Least Square Support Vector Machines (LSSVM) are the set of supervised learning methods used for different application such as: classification, regression and outliers detection (Suykens et al., 2002). The basic form of LSSVM classifier, deals with two-class problems, in which data are separated by the optimal hyperplane defined by a number of support vectors. Support vectors are the subset of the training set which define the boundary values between two classes. In this work, Least Square Support Vector Machine (LSSVM) with various kernels have been considered as a classifier for developing a model to classify faulty and non faulty classes. The general form of LSSVM function is defined as:

$$y(x) = w^T \phi(x) + c \tag{11}$$

where y, and x are the output and input vector. $\phi(x)$ is nonlinear mapping function and it is used for mapping the input data into higher dimensional feature space. w, and c show the adjust weight vector and scalar threshold value respectively. The objective in LSSVM is to optimize the following equation:

$$Minimize \ \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{i=1}^{l} E_i^2$$

$$Subject \ to \ y(x) = w^T \phi(x) + c + E_i, i = 1, 2, 3, ..n$$

where E_i is the error value of input instance i and γ is the cost function. After solving the above optimization problem, fault prediction values are obtained using Eq. (12).

$$Y' = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \phi(x_i) * \phi(x) + b = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) * K(x_i, x) + b \tag{12}$$

where $K(x_i, x)$ is the kernel function, which enables the dot product to be performed in high-dimensional feature space using low-dimensional space data. The different Kernel functions considered in this study are mentioned below:

1. Linear function:

$$K(x_i, x_j) = x_i^T x_j \tag{13}$$

2. Polynomial function:

$$K(x_i, x_j) = (x_i^T * x_j + C)^d \tag{14}$$

3. Radial basis function:

$$K(x_i, x_j) = e^{(-\gamma \|x_i - x_j\|^2)}, \gamma > 0 \tag{15}$$

6. Cost analysis model

This section describes the construction of a cost evaluation model, which accounts for realistic cost 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, which are as follows:

i. Different phases of testing account for varying fault removal cost.
ii. It is not possible to detect 100% faults in testing phase.
iii. It is not practically possible to perform unit testing on all classes.

Normalized fault removal cost approach suggested by Wagner (2006) has been applied to formulate the proposed cost evaluation

model. Cost varies because of various projects being developed on different platforms following several organization standards. The normalized fault removal costs are summarized in Table 5.

The fault identification efficiencies for different testing phases are taken from the study of Jones (2010). The efficiencies of testing phases are summarized in Table 6. Huitt and Wilde (1992) have stated that more than fifty percent of classes are usually very small in size, hence performing unit testing on these classes may not be helpful.

The formulation of E_{cost}, T_{cost} and the NE_{cost} of the proposed cost based evaluation framework is discussed in the following subsections:

* Estimated fault removal cost (E_{cost}): The detailed analysis to compute Ecost of the framework is as follows:
  - Total number of faulty classes identified by the predictor are equal to the summation of $No._{F->F}$ (number of are classes correctly classified as faulty classes) and $No._{NF->F}$ (number of not-faulty classes incorrectly labeled as faulty classes) values. Hence, it is necessary to compute testing and verification cost at class level which implies that this cost is equal to the cost of unit testing (C_u). Total unit testing cost of software system is defined as:

$$Cost_{unit} = (No._{F->F} + No._{NF->F}) * C_u \tag{16}$$

  - As it is not possible to identify 100% fault in a specific testing phase, there is a possibility that some of the correctly predicted faulty classes remain undetected in unit testing. Further, there is a scope that these faulty classes and the faulty classes which were predicted as non-faulty classes ($No._{F->NF}$) are identified by the predictor in later phases of testing such as integration testing (C_i), system testing and field testing. The fault removal cost in integration testing may be computed as:

$$Cost_{integration} = \delta_i * C_i * (No._{F->NF} + (1-\delta_u) * No._{F->F}) \tag{17}$$

  - Remaining faulty classes which were not identified in integration testing will be further identified in system testing. The fault removal cost in case of system testing (C_s) is computed as:

$$Cost_{system} = \delta_s * C_s * ((1-\delta_i) * (No._{F->NF} + (1-\delta_u) * No._{F->F})) \tag{18}$$

  - Remaining faulty classes which were not identified in system testing will be further identified in field testing. The fault removal cost in case of field testing (C_f) is computed as:

$$Cost_{field} = (1-\delta_s) * C_f * ((1-\delta_i) * (No._{F->NF} + (1-\delta_u) * No._{F->F})) \tag{19}$$

  - The estimated overall fault removal cost can be determined as:

$$Ecost = C_{initial} + C_u * (No._{F->F} + No._{NF->F}) + \delta_i * C_i * (No._{F->NF} + (1-\delta_u) * No._{F->F}) + \delta_s * C_s * ((1-\delta_i) * (No._{F->NF} + (1-\delta_u) * No._{F->F})) + (1-\delta_s) C_f * ((1-\delta_i) * (No._{F->NF} + (1-\delta_u) * No._{F->F})) \tag{20}$$

* Estimated testing cost (T_{cost}): The detailed steps to compute T_{cost} of the framework are as follows:
  - In testing, if fault prediction analysis is not carried out, then the tester performs unit testing on all the classes. So the total unit testing may be computed as:

$$Cost_{unit} = M_p * C_u * No._{classes} \tag{21}$$

  - Further, there is a possibility that some of the faulty classes that remains undetected in unit testing can be identified in integration testing. The total system testing cost will be computed as:

$$Cost_{integration} = \delta_i * C_i * (1-\delta_u) * No._{faculty\ classes} \tag{22}$$

  - Further, there is a possibility that some of the faulty classes that remained undetected in integration testing may be identified in system testing. The total system testing cost will be computed as:

$$Cost_{system} = \delta_s * C_s * ((1-\delta_i) * (1-\delta_u) * No._{faculty\ classes}) \tag{23}$$

  - Remaining faulty classes which were not identified in system testing may be further identified in field testing. The fault removal cost in case of field testing will be computed as:

$$Cost_{field} = (1-\delta_s) * C_f * ((1-\delta_i) * (1-\delta_u) * No._{faculty\ classes}) \tag{24}$$

  - The estimated overall fault removal cost without the use of fault prediction will be determined by using following equation as:

$$Tcost = M_p * C_u * No._{classes} + \delta_i * C_i * (1-\delta_u) * No._{faculty\ classes} + \delta_s * C_s * ((1-\delta_i) * (1-\delta_u) * No._{faculty\ classes}) + (1-\delta_s) * C_f * ((1-\delta_i) * (1-\delta_u) No._{faculty\ classes}) \tag{25}$$

* Normalized fault removal cost (NE_{cost}): Normalized fault removal cost and its interpretation can be modeled as:

$$NEcost = \frac{Ecost}{Tcost} = \begin{cases} < 1, & Fault\ Prediction\ is\ useful \\ => 1, & Perform\ Testing \end{cases} \tag{26}$$

where, E_{cost} represents for Estimated fault removal cost of the software when fault prediction is performed. T_{cost} is the Estimated fault removal cost of the software without using fault prediction approach. NE_{cost} represents the Normalized Estimated fault removal cost of the software when fault prediction is utilized.

Notations used in this cost evaluation analysis are:

i. $C_{initial}$: Initial setup cost for fault-prediction technique applied, C_u, C_i, C_s, and C_f are the Normalized fault removal cost in unit, integration, system, and field testing respectively.
ii. M_p: percentage of classes unit tested.
iii. $No._{F->F}$ (number of classes correctly classified as faulty classes), $No._{NF->F}$ (number of not-faulty classes incorrectly labeled as faulty classes), $No._{NF->NF}$ (number of not-faulty classes correctly classified as not-faulty classes), $No._{F->NF}$ (number of faulty classes incorrectly classified as not-faulty classes), $No._{classes}$ (total number of classes), $No._{faulty\ classes}$ (total number of faulty classes).
iv. δ_u , δ_i , and δ_s are the Fault identification efficiency of unit, integration, and system testing respectively.

7. Performance evaluation parameters

The following sub-sections give the basic definitions of the performance parameters used for fault prediction. All these parameters are computed using Confusion matrix. Confusion matrix contains actual and predicted classifications information done by fault

Table 4
Use project.

	Project	No. of class	No. of faulty class	Faulty (%)
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

Table 5
Removal costs of test techniques (in staff hour per defects).

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

prediction technique. Table 7 shows the confusion matrix for fault prediction model. In this study, two performance parameters such as accuracy and F-Measure are used for measuring the performance of fault prediction models.

$$Accuracy = \frac{No._{NF->NF} + No._{F->F}}{No._{NF->NF} + No._{NF->F} + No._{F->NF} + No._{F->F}} \quad (27)$$

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * No._{F->F}}{2 * No._{F->F} + No._{NF->F} + No._{F->NF}} \quad (28)$$

8. Experimental setup

In this section, the experimental setup done to find the effectiveness of fault prediction model using the cost based evaluation framework is presented. LSSVM with three different kernel methods i.e., linear kernel, Polynomial kernel, and radial basis function kernel are employed to develop a model that predicts whether a class is faulty or not. These techniques are applied on thirty object oriented software projects chosen from PROMISE and NASA data repository as shown in Table 4. All these projects have varying percentage of faulty classes which are sufficient to perform our analysis. The NE_{cost} of the proposed cost based analysis framework has been used to evaluate the effectiveness of fault prediction techniques. The values of the metrics calculated in these datasets describe the complexity of the projects chosen.

Table 6
Fault identification efficiencies of different test phase.

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

Table 7
Confusion matrix to classify a class as faulty and not-faulty.

	Non faulty	Faulty
Non faulty	$No._{NF->NF}$	$No._{NF->F}$
Faulty	$No._{F->NF}$	$No._{F->F}$

In this experiment, the values tabulated in Table 6 are used in design of cost evaluation model. δ_u, δ_i, and δ_s show the fault identification efficiency values of unit testing, integration testing, and system testing, respectively. The values of δ_u, δ_i, and δ_s have been collected from the survey report "Software Quality in 2010" of Jones (2010). M_p (0.5) shows the fraction of classes unit tested, obtained from the paper of Huitt and Wilde (1992). The objective is to provide the bench marks to approximate the overall fault removal cost. Fig. 5 shows the work flow of the proposed cost based evaluation framework.

The following steps are followed while selecting a subset of metrics to develop the fault prediction model which predicts whether a class is faulty or not. Feature selection methods (both feature ranking and feature subset selection) are applied on 30 data sets. Therefore, a total of 990 ((10 feature selection method + 1 considering all features) × 30 datasets(subsets of metrics specific to data sets identified after performing feature selection) × 3 prediction techniques) distinct prediction models have been developed in the study.

1. In this paper, six feature ranking methods and four feature subset selection methods are applied on 30 object oriented software projects to select right set of source code metrics for fault prediction.
2. The subsets of object-oriented metrics obtained from above two steps are passed as input to LSSVM with three different kernel methods i.e., linear kernel, polynomial kernel, and radial basis function kernel while developing a model. In this paper, 20 fold cross-validation is used for comparing the models. Cross-validation (Kohavi, 1995) is a statistical learning method which is used to evaluate and compare the models by partitioning the data into two portions. One portion of the divided subset is used to train or learn the model and the rest of data is used to validate the model, based on training. K-fold cross-validation (Kohavi, 1995) is the basic form of cross validation technique. In K-fold cross-validation technique the data are first partitioned into K equal (or nearly equal) sized portions or folds. For each of the model, K-1 folds are used for training and the remaining one fold is used for testing purpose. The significance of K-fold-cross-validation lies in it's ability to use the data set for both training and testing. In this paper, 20-fold cross-validation is used for comparing the models, i.e., data sets are divided into 20 parts. The performance of all developed fault prediction models are compared using two different performance evaluation parameters such as Accuracy, and F-measure.
3. The models developed using above two steps are passed to proposed cost analysis framework to check whether the developed fault prediction model is effective for this project or not.

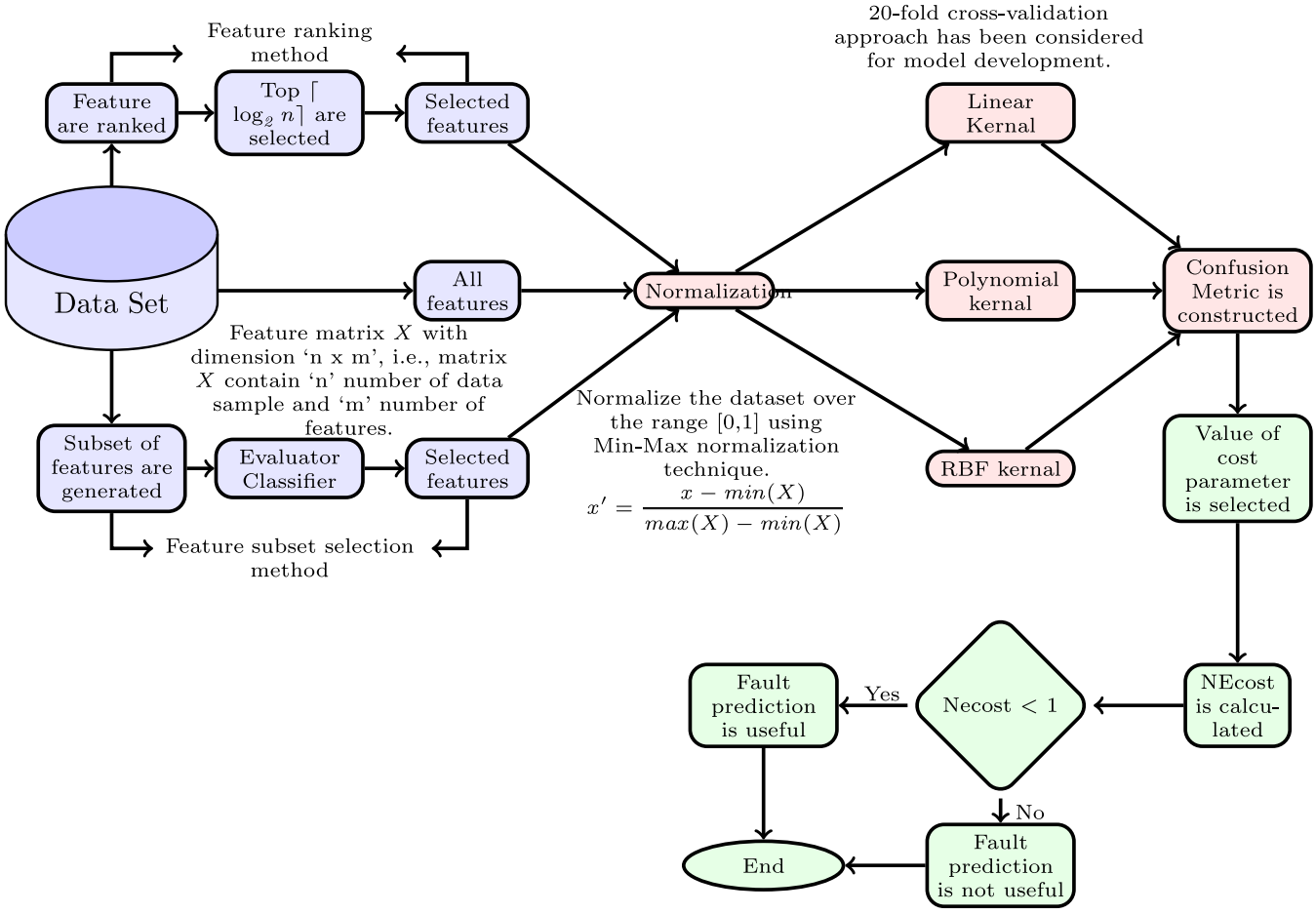


Fig. 5. Framework of proposed work.

Table 8
Used naming conventions for different techniques.

Abbreviation	Corresponding name
AM	All Metrics
FS1	Correlation based Feature Selection
FR1	Chi Squared test
FS2	Classifier Subset Evaluation
FR2	Gain Ratio Feature Evaluation
FS3	Filtered Subset Evaluation
FR3	OneR Feature Evaluation
FS4	Rough Set Analysis (RSA)
FR4	Information Gain Feature Evaluation
FR6	Principal Component Analysis (PCA)
FR5	Logistic regression analysis
DS	Data set

9. Results

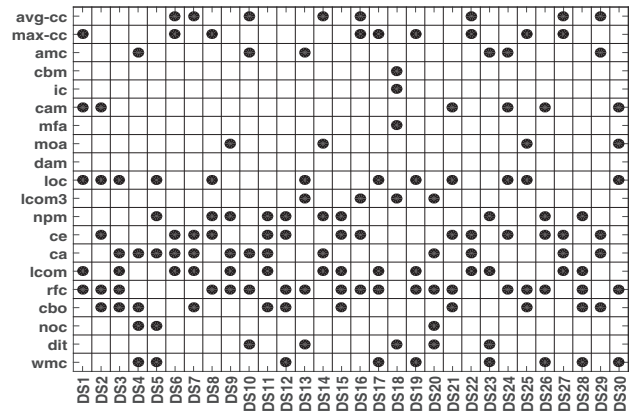
In this section, the relationship between Object-Oriented metrics and fault proneness at the class level is presented. Software metrics are considered as input data and the present the ratio of faulty and non-faulty classes in a project. Accuracy, and F-measure are considered as performance evaluation parameter to compare performance of fault proneness estimation model developed by using LSSVM with three different kernel methods. To represent the results we use the following abbreviations as shown in Table 8. Table 8 shows the mapping of these abbreviations to their actual names.

9.1. Feature ranking methods

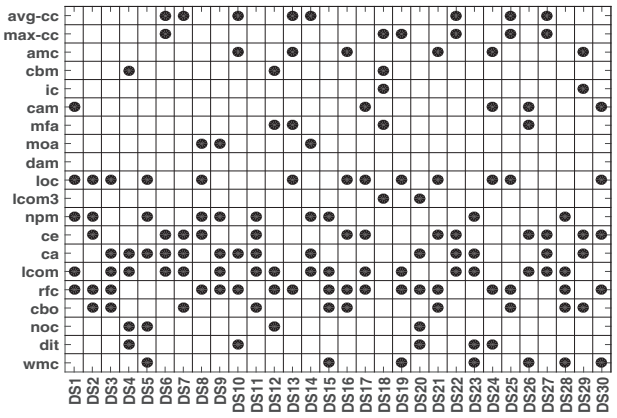
In this study, six feature ranking methods: Chi squared test, gain ratio feature evaluation, information gain, oneR feature evaluation, information gain feature evaluation, logistic regression analysis, and principal component analysis are applied on different object oriented softwares. Each method uses different performance parameters to rank the features. Further top $\lceil log_2 n \rceil$ metrics out of "n" number of metrics have been considered to develop a model for predicting fault proneness. For first four feature ranking methods(Chi-squared test, Gain-ratio feature evaluation, OneR feature evaluation), top $\lceil log_2 n \rceil$ are selected as a subsets of features, where n is the number of object-oriented metrics in the original dataset (in this study n=20). But in case of univariate logistic regression (ULR) analysis, only those metrics are selected which have positive value of regression coefficient, and value of p-value is less than 0.05 and in case PCA, only those metrics are selected which have Eigenvalue being more than 1. Selected set of object-oriented metrics after feature ranking methods are shown in Fig. 6.

9.2. Feature subset selection methods

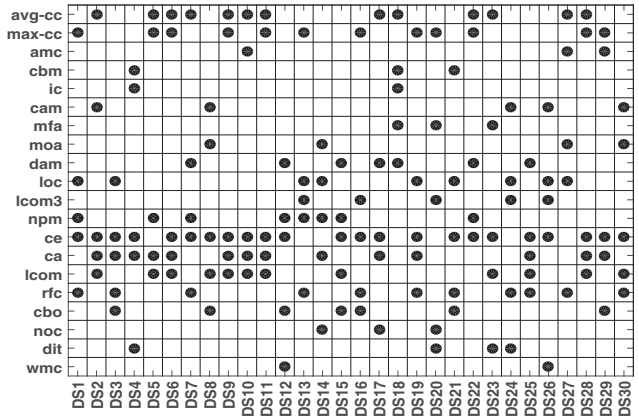
In this study, four different types of feature subset selection methods are applied over 30 object oriented software projects data sets one-by-one. Feature-subset selection methods are based on the assumption that model has higher accuracy and reduced value of misclassified errors, when combined with some other features as compared to when used by itself. Subsequently, these selected subset of metrics have been considered as input for developing a



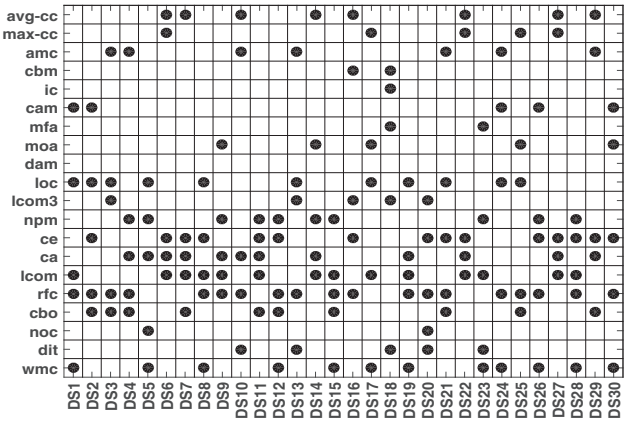
(a) Chi Squared test



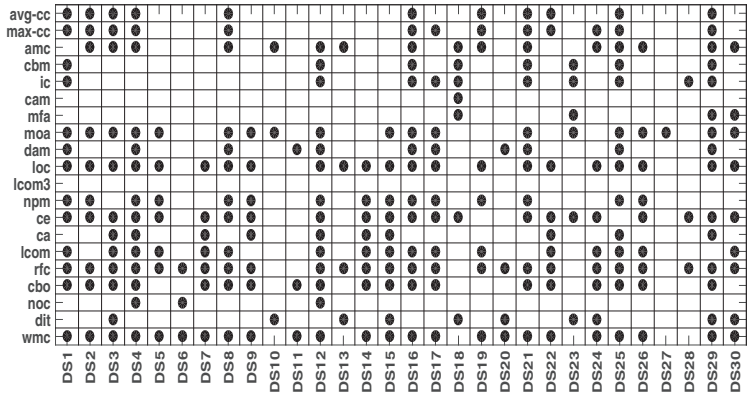
(b) Gain Ratio Feature Evaluation



(c) OneR Feature Evaluation



(d) Information Gain Feature Evaluation



(e) Logistic Regression Analysis

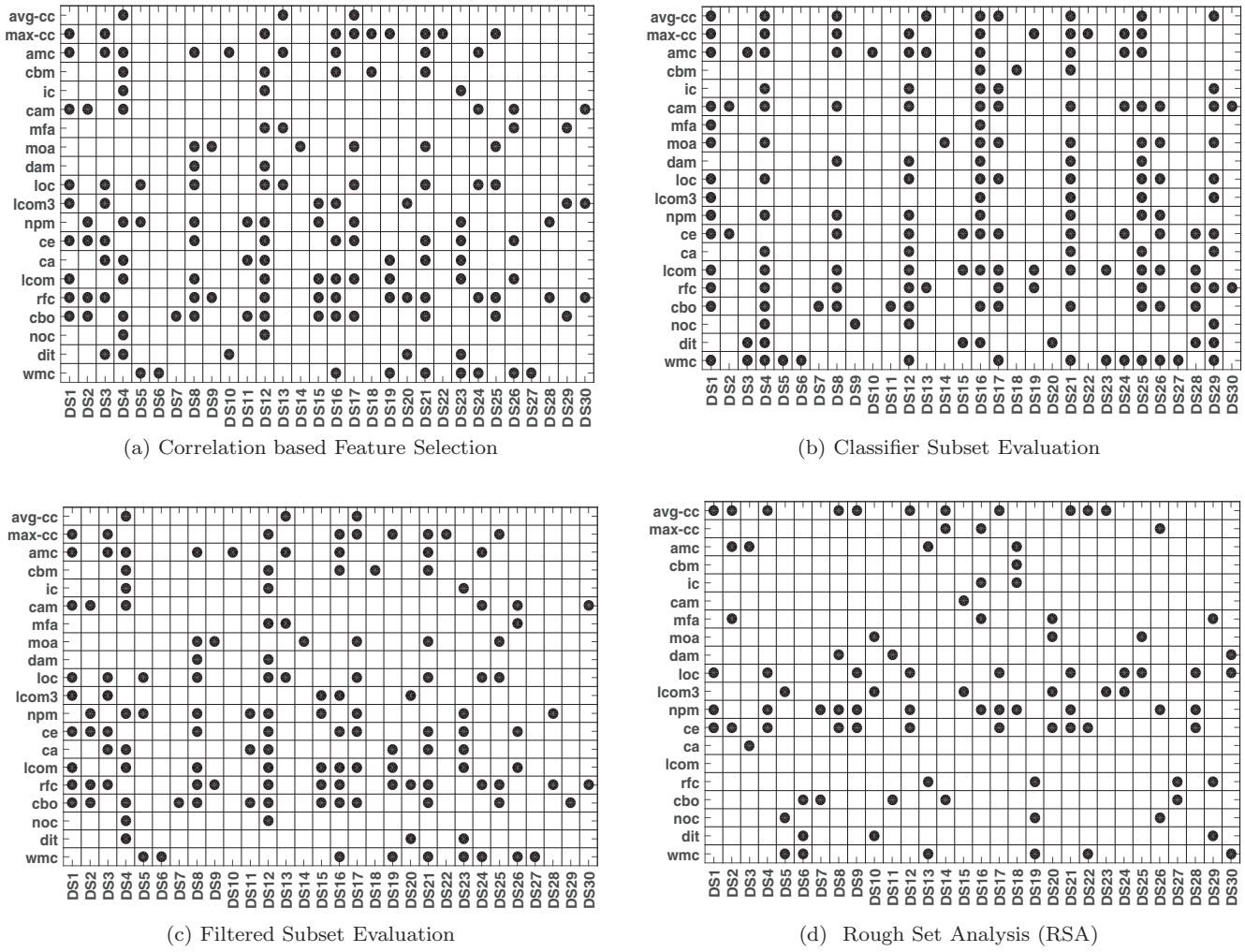
Fig. 6. Selected source code metrics using feature ranking methods.

model to predict whether the class is faulty or not. Selected set of object-oriented metrics after feature subset selection methods are shown in Fig. 7.

9.3. Least Square Support Vector Machine (LSSVM) classifier

In this paper, least square support vector machine (LSSVM) classifier with three different kernel methods have been considered to develop a model to predict whether a class is faulty or not. Eleven subsets of metrics (10 resulting from feature selection methods + 1 considering all metrics) are considered as input to develop a model for predicting fault proneness at the class level of object oriented software project. Hardware used to carry out this work: Core i5 processor with 4GB RAM and storage capacity of 250GB hard disk. Prediction models are developed using the licenced *MATLAB* environment at NIT-Rourkela.[4] The performance of each prediction

[4] http://www.nitrkl.ac.in.



(a) Correlation based Feature Selection

(b) Classifier Subset Evaluation

(c) Filtered Subset Evaluation

(d) Rough Set Analysis (RSA)

Fig. 7. Selected source code metrics using feature subset selection method.

model is evaluated in terms of two performance parameters i.e., accuracy and F-Measure. Table 9a–c, show the obtained performance values for different software projects using LSSVM with linear, polynomial and radial basic function. From Table 9a–c, it can be inferred that:

- In most cases, model developed by considering selected set of metrics using feature selection techniques as input obtained better performance i.e., high vales of accuracy and F-Measure for predicting reliability as compared to a model developed using all metrics.
- In case of LSSVM with linear kernel and polynomial kernel, fault prediction model develop by considering selected set of metrics using FS4 i.e., rough set analysis obtained better results when compared to other feature selection techniques.
- In case of LSSVM with RBF kernel, fault prediction model develop by considering selected set of metrics using FR6 i.e., PCA obtained better results when compared to other feature selection techniques.

In this study LSSVM with three kernel methods and two performance parameters are considered to predict whether the class is faulty or not. Six box-plot diagrams Fig. 8a–c are displayed (one for each combination). These diagrams show the box-plot diagrams for accuracy and F-Measure for each of the cases. Each of these figures contain eleven box-plots: one for all metrics, six for feature ranking methods and four for feature subset selection methods. The X-axis of the box-plot diagrams shows the name of the used feature selection technique. The model which has high median value and few number of outliers is the best model to predict as to whether the class is faulty or not. From the box-plot diagram, it can be inferred that:

- In case of *LSSVM with linear and polynomial kernel*, FS4 i.e., rough set analysis have high median value as well as few outliers. Based on these boxplots Fig. 8a–c, model developed using selected set of metrics using rough set analysis for predicting faulty and non-faulty class of object oriented software yield better result as compared to others.
- In case of *LSVM with RBF kernel* i.e. FR6, "PCA" has high median value and fewer outliers. Based on the boxplots in Fig. 8a–c model developed using select set of metrics using PCA for predicting faulty and non-faulty class of object oriented software yield better result as compared to others.
- Among all *feature ranking* techniques, FR6 have high median value as well as fewer outliers. Based on the boxplots in Fig. 8a–c, FR6 produced the best result i.e., feature ranking using PCA computes best set of object-oriented for predicting faulty and non-faulty class of object oriented software yield better result as compared to others.
- Among all *feature subset selection* techniques, FS4 yield better results compared to others i.e., feature subset selection using rough set analysis method computes best set of object-oriented

Table 9
Performance matrix.

(a) Linear kernel

	Accuracy											F-Measure										
	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
DS1	82.01	83.09	82.15	82.82	82.15	**83.22**	82.55	**83.22**	82.55	**83.22**	**83.22**	0.89	**0.9**	0.89	**0.9**	0.89	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**
DS2	88.46	88.46	88.46	88.46	88.03	88.46	88.46	88.89	88.46	88.46	**90.17**	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	**0.95**
DS3	83.72	86.05	86.05	93.02	88.37	**95.35**	93.02	93.02	83.72	93.02	**95.35**	0.89	0.9	0.9	0.95	0.91	**0.96**	0.95	0.95	0.89	0.95	**0.96**
DS4	80.83	81.24	81.24	80.93	81.04	81.66	81.76	81.24	**81.97**	81.35	**81.97**	0.89	0.89	0.89	0.89	0.89	**0.9**	**0.9**	0.89	**0.9**	**0.9**	**0.9**
DS5	93.75	92.19	92.19	92.19	93.75	93.75	95.31	93.75	92.19	93.75	**96.88**	0.97	0.96	0.96	0.96	0.97	0.97	**0.98**	0.97	0.96	0.97	**0.98**
DS6	**93.75**	**93.75**	**93.75**	**93.75**	**93.75**	**93.75**	**93.75**	**93.75**	**93.75**	**93.75**	**93.75**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
DS7	92.59	88.89	92.59	**96.3**	92.59	92.59	88.89	88.89	88.89	88.89	92.59	0.96	0.94	0.96	**0.98**	0.96	0.96	0.94	0.94	0.94	0.94	0.96
DS8	89.2	89.49	89.77	89.77	89.77	89.2	89.49	89.49	89.2	**90.06**	**90.06**	0.94	0.94	0.94	**0.95**	**0.95**	0.94	0.94	0.94	0.94	**0.95**	**0.95**
DS9	**97.76**	**97.76**	**97.76**	**97.76**	**97.76**	**97.76**	**97.76**	**97.76**	**97.76**	**97.76**	**97.76**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DS10	88.89	**92.59**	**92.59**	**92.59**	**92.59**	**92.59**	**92.59**	**92.59**	**92.59**	**92.59**	**92.59**	0.93	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
DS11	**92.2**	**92.2**	**92.2**	**92.2**	**92.2**	**92.2**	**92.2**	**92.2**	**92.2**	**92.2**	**92.2**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
DS12	66.47	70.59	65	69.71	70.88	70.29	70.29	69.71	71.18	70	**71.47**	0.44	0.57	0.51	0.53	0.59	0.64	0.62	0.62	0.64	0.63	**0.65**
DS13	74.07	88.89	85.19	88.89	88.89	88.89	88.89	85.19	77.78	85.19	**96.3**	0.83	0.92	0.89	0.91	0.92	0.92	0.92	0.89	0.85	0.89	**0.97**
DS14	**84.31**	82.35	82.35	**84.31**	82.35	**84.31**	**84.31**	78.43	80.39	80.39	**84.31**	**0.91**	0.9	0.9	**0.91**	0.9	**0.91**	**0.91**	0.88	0.89	0.89	**0.91**
DS15	75.76	84.85	84.85	**87.88**	84.85	84.85	**87.88**	**87.88**	81.82	**87.88**	84.85	0.76	0.88	0.88	0.89	0.88	0.87	0.88	**0.9**	0.85	**0.9**	0.87
DS16	76.47	71.49	63.57	71.27	75.57	77.38	**77.6**	76.24	76.92	75.57	77.38	0.66	0.48	NaN	0.46	0.61	0.67	**0.68**	0.63	0.65	0.61	0.64
DS17	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
DS18	90.34	90.34	90.34	90.34	90.34	90.34	**90.91**	90.34	90.34	90.34	**90.91**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
DS19	86.67	88.89	88.89	86.67	88.89	91.11	91.11	88.89	88.89	91.11	**93.33**	0.92	0.94	0.94	0.92	0.94	0.95	0.95	0.94	0.94	0.95	**0.96**
DS20	**80**	**80**	**80**	**80**	**80**	**80**	**80**	**80**	**80**	**80**	**80**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**	**0.89**
DS21	74.61	74.61	73.83	74.61	74.22	76.17	73.44	76.56	**76.95**	**76.95**	**76.95**	0.83	0.83	0.82	0.83	0.83	0.83	0.83	**0.84**	**0.84**	**0.84**	**0.84**
DS22	89.23	92.31	92.31	90.77	92.31	**93.85**	92.31	90.77	90.77	90.77	**93.85**	0.94	0.96	0.96	0.95	0.96	**0.97**	0.96	0.95	0.95	0.95	**0.97**
DS23	56	76	80	72	84	76	84	80	80	80	**92**	NaN	0.75	0.76	0.72	0.82	0.79	0.8	0.76	0.71	0.76	**0.9**
DS24	80.95	85.71	83.33	83.33	85.71	83.33	**88.1**	85.71	85.71	85.71	85.71	0.88	0.91	0.89	0.89	0.91	0.89	**0.92**	0.91	0.91	0.91	0.91
DS25	91.26	91.14	91.84	**92.31**	91.61	92.07	92.19	91.26	91.96	91.26	92.07	0.95	0.95	**0.96**	**0.96**	**0.96**	**0.96**	**0.96**	0.95	**0.96**	0.95	**0.96**
DS26	71.18	72.05	74.67	72.93	70.74	71.18	75.55	74.67	73.36	74.67	**75.98**	0.82	0.81	0.83	0.82	0.81	0.82	**0.84**	0.83	0.82	0.83	**0.84**
DS27	**76.92**	69.23	69.23	64.1	69.23	64.1	**76.92**	56.41	56.41	56.41	69.23	**0.77**	0.68	0.68	0.67	0.68	0.7	**0.77**	0.6	0.6	0.6	0.7
DS28	**98.79**	**98.79**	**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	NaN
DS29	74.32	74.32	74.32	74.32	74.32	80.44	76.53	**89.8**	82.48	74.32	83.16	NaN	NaN	NaN	NaN	NaN	0.47	0.19	**0.77**	0.58	NaN	0.61
DS30	79.31	82.76	89.66	82.76	82.76	**93.1**	89.66	79.31	86.21	86.21	82.76	0.84	0.86	0.91	0.86	0.86	**0.94**	0.91	0.84	0.88	0.88	0.86

(b) Polynomial kernel

	Accuracy											F-Measure										
	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
DS1	**88.99**	86.85	84.43	84.03	83.89	85.5	83.22	83.89	**88.99**	82.95	83.36	**0.93**	0.92	0.91	0.9	0.9	0.91	0.9	0.9	**0.93**	0.9	0.9
DS2	**94.87**	88.46	88.46	93.59	93.16	94.02	88.46	88.46	93.59	88.46	93.59	**0.97**	0.94	0.94	**0.97**	0.96	**0.97**	0.94	0.94	**0.97**	0.94	**0.97**
DS3	**100**	93.02	90.7	93.02	**100**	95.35	**100**	**100**	**100**	**100**	95.35	**1**	0.95	0.93	0.95	**1**	0.96	**1**	**1**	**1**	**1**	0.96
DS4	80.93	81.45	81.24	81.66	82.69	82.9	**85.7**	85.08	81.14	82.69	81.87	0.89	0.9	0.89	0.9	0.9	0.9	**0.92**	0.91	0.9	0.9	0.9
DS5	98.44	96.88	92.19	96.88	92.19	96.88	96.88	96.88	96.88	96.88	**100**	0.99	0.98	0.96	0.98	0.96	0.98	0.98	0.98	0.98	0.98	**1**
DS6	93.75	93.75	93.75	93.75	93.75	93.75	**100**	93.75	93.75	93.75	**100**	0.97	0.97	0.97	0.97	0.97	0.97	**1**	0.97	0.97	0.97	**1**
DS7	92.59	92.59	**100**	96.3	92.59	96.3	**100**	96.3	96.3	96.3	**100**	0.96	0.96	**1**	0.98	0.96	0.98	**1**	0.98	0.98	0.98	**1**
DS8	90.63	91.76	91.48	91.19	89.49	90.91	**94.6**	91.19	90.06	91.19	91.19	0.95	0.96	0.95	0.95	0.94	0.95	**0.97**	0.95	0.95	0.95	0.95
DS9	97.97	98.37	98.58	98.37	98.37	98.37	**99.19**	98.37	97.76	98.37	**99.19**	0.99	0.99	0.99	0.99	0.99	0.99	**1**	0.99	0.99	0.99	**1**
DS10	92.59	92.59	92.59	92.59	92.59	92.59	**96.3**	88.89	88.89	92.59	92.59	0.95	0.95	0.95	0.95	0.95	0.95	**0.98**	0.93	0.93	0.95	0.95
DS11	92.2	94.63	92.2	**100**	92.2	92.2	**100**	93.17	92.2	92.2	**100**	NaN	0.48	NaN	**1**	NaN	NaN	**1**	0.3	NaN	NaN	**1**
DS12	68.53	99.71	80.29	70	99.71	93.24	**100**	99.12	96.76	98.24	89.71	0.56	**1**	0.76	0.6	**1**	0.92	**1**	0.99	0.96	0.98	0.87
DS13	92.59	**100**	**100**	88.89	96.3	92.59	96.3	96.3	81.48	96.3	92.59	0.94	**1**	**1**	0.91	0.97	0.94	0.97	0.97	0.86	0.97	0.94

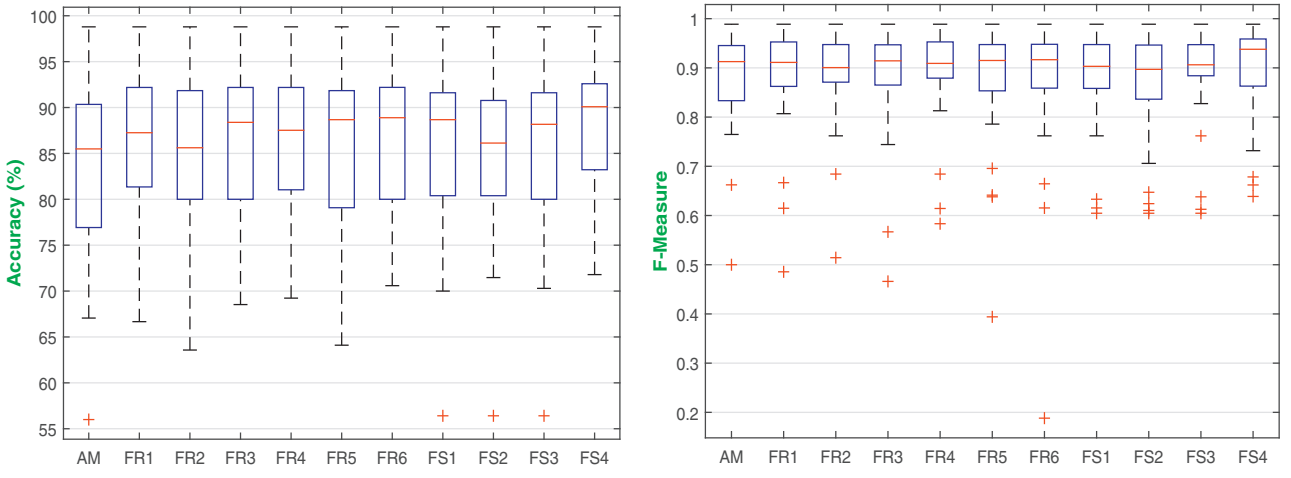
(*continued on next page*)

Table 9 (*continued*)

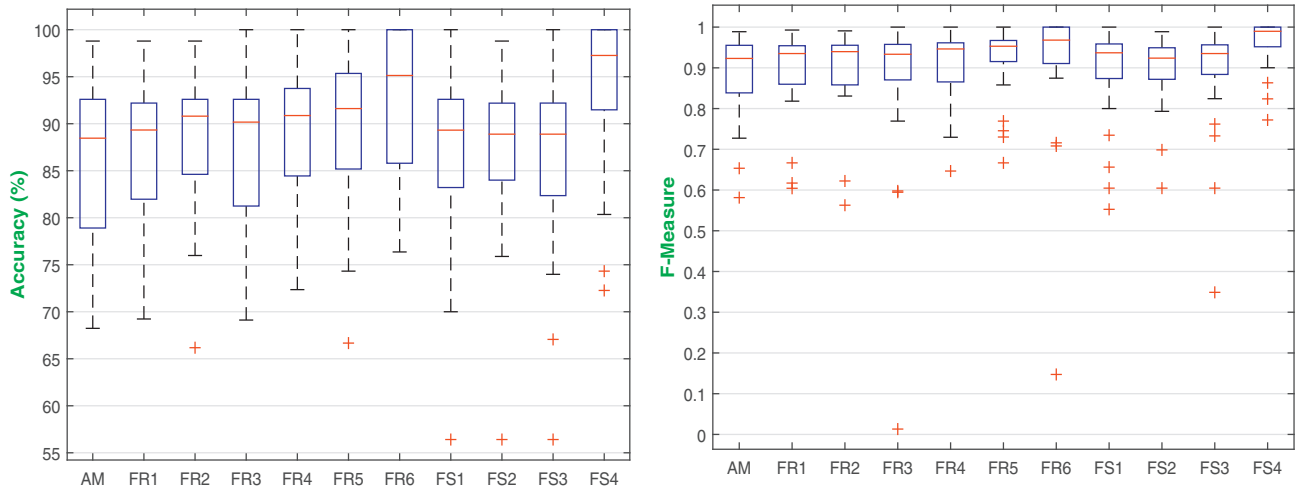
DS14	90.2	82.35	96.08	94.12	80.39	84.31	**98.04**	88.24	88.24	88.24	84.31	0.94	0.9	0.98	0.96	0.89	0.91	**0.99**	0.93	0.93	0.93	0.91
DS15	90.91	84.85	87.88	96.97	84.85	**100**	90.91	**100**	**100**	**100**	**100**	0.92	0.87	0.9	0.97	0.88	**1**	0.91	**1**	**1**	**1**	**1**
DS16	90.05	83.71	80.77	88.46	95.93	97.29	97.96	94.12	97.74	91.63	**98.87**	0.87	0.78	0.72	0.85	0.95	0.96	0.97	0.92	0.97	0.89	**0.98**
DS17	90	90	95.15	91.97	90	91.67	91.36	92.58	**96.06**	91.36	90	0.95	0.95	0.97	0.96	0.95	0.96	0.95	0.96	**0.98**	0.95	0.95
DS18	90.91	90.34	90.34	90.34	90.34	90.34	**91.48**	90.91	90.34	90.34	**91.48**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
DS19	**93.33**	88.89	88.89	86.67	88.89	91.11	91.11	88.89	88.89	88.89	91.11	**0.96**	0.94	0.94	0.92	0.94	0.95	0.95	0.94	0.94	0.94	0.95
DS20	**100**	95.56	91.11	84.44	**100**	88.89	80	80	84.44	80	**100**	**1**	0.97	0.95	0.91	**1**	0.94	0.89	0.89	0.91	0.89	**1**
DS21	90.63	75.39	75	**95.7**	74.61	88.28	90.23	79.69	87.11	89.45	89.45	0.93	0.83	0.83	**0.97**	0.83	0.92	0.93	0.86	0.91	0.92	0.92
DS22	**95.38**	92.31	92.31	93.85	92.31	92.31	92.31	90.77	90.77	90.77	92.31	**0.97**	0.96	0.96	**0.97**	0.96	0.96	0.96	0.95	0.95	0.95	0.96
DS23	**92**	80	88	76	84	56	84	88	**92**	84	80	**0.9**	0.76	0.86	0.77	0.8	NaN	0.8	0.87	**0.9**	0.8	0.8
DS24	85.71	83.33	80.95	80.95	85.71	83.33	85.71	**100**	83.33	85.71	83.33	0.91	0.89	0.88	0.88	0.91	0.89	0.91	**1**	0.89	0.91	0.89
DS25	91.49	93.59	92.77	91.61	**94.29**	92.07	92.31	94.06	92.07	92.89	92.42	0.96	**0.97**	0.96	0.96	**0.97**	0.96	0.96	**0.97**	0.96	0.96	0.96
DS26	71.18	75.55	74.24	79.04	75.55	77.29	77.29	74.24	77.29	75.55	**79.48**	0.82	0.83	0.83	0.85	0.83	0.84	0.84	0.82	0.84	0.83	**0.86**
DS27	79.49	97.44	64.1	97.44	66.67	66.67	84.62	56.41	56.41	56.41	**100**	0.79	0.97	0.63	0.97	0.67	0.67	0.85	0.6	0.6	0.6	**1**
DS28	99.56	99.67	99.67	99.34	99.78	99.56	99.67	99.56	99.45	99.78	**99.89**	0.78	0.84	0.84	0.63	0.9	0.78	0.84	0.8	0.71	0.9	**0.95**
DS29	75.17	93.71	97.96	92.86	93.2	98.3	97.45	94.22	**99.15**	93.37	98.64	0.14	0.87	0.96	0.85	0.86	0.97	0.95	0.88	**0.98**	0.86	0.97
DS30	82.76	**96.55**	82.76	**96.55**	82.76	89.66	**96.55**	89.66	86.21	86.21	82.76	0.85	**0.97**	0.86	**0.97**	0.86	0.91	**0.97**	0.91	0.88	0.88	0.86

(c) RBF Kkernel

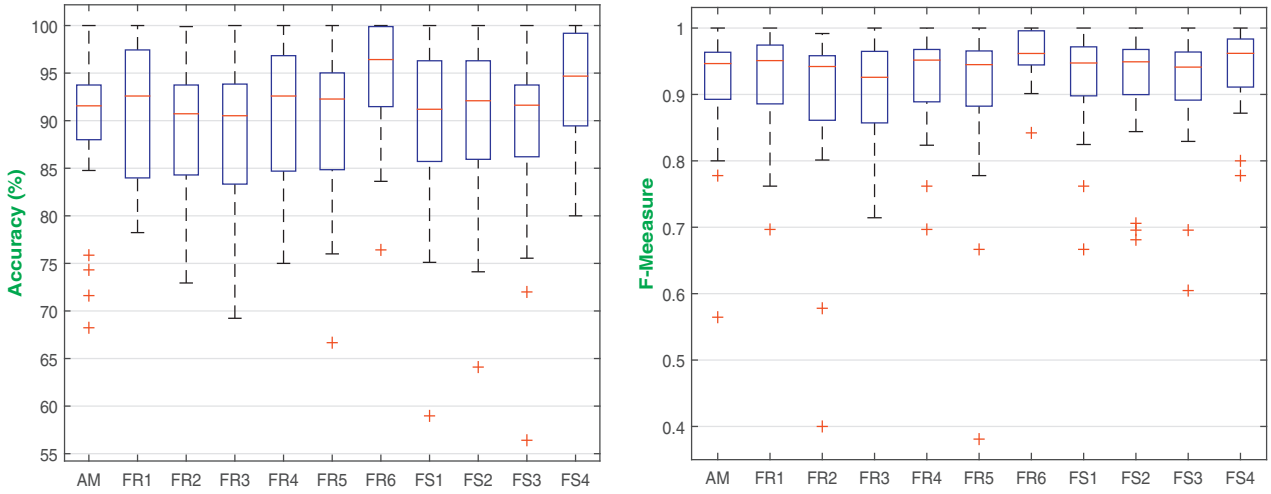
	Accuracy											F-Measure										
	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
DS1	83.09	83.36	83.89	83.22	82.95	87.25	85.1	83.09	81.61	83.89	**88.32**	0.9	0.9	0.9	0.9	0.9	0.92	0.91	0.9	0.89	0.9	**0.93**
DS2	88.03	88.46	89.74	88.03	88.46	**93.16**	91.88	89.74	86.75	89.32	**93.16**	0.94	0.94	0.94	0.94	0.94	**0.96**	**0.96**	0.94	0.93	0.94	**0.96**
DS3	93.02	90.7	90.7	90.7	95.35	95.35	**100**	97.67	95.35	95.35	**100**	0.95	0.93	0.93	0.93	0.96	0.96	**1**	0.98	0.96	0.96	**1**
DS4	81.35	81.97	81.66	81.24	81.55	**85.18**	82.38	83.21	83.94	82.28	82.28	0.9	0.9	0.9	0.9	0.9	**0.92**	0.9	0.9	0.91	0.9	0.9
DS5	92.19	92.19	93.75	96.88	93.75	96.88	96.88	92.19	92.19	92.19	**100**	0.96	0.96	0.97	0.98	0.97	0.98	0.98	0.96	0.96	0.96	**1**
DS6	96.88	93.75	93.75	93.75	93.75	93.75	**100**	93.75	93.75	93.75	**100**	0.98	0.97	0.97	0.97	0.97	0.97	**1**	0.97	0.97	0.97	**1**
DS7	92.59	96.3	92.59	92.59	96.3	96.3	**100**	88.89	88.89	88.89	**100**	0.96	0.98	0.96	0.96	0.98	0.98	**1**	0.94	0.94	0.94	**1**
DS8	91.19	89.77	91.19	91.19	90.63	92.9	90.63	93.18	89.77	92.9	**94.89**	0.95	0.95	0.95	0.95	0.95	0.96	0.95	0.96	0.95	0.96	**0.97**
DS9	97.76	**98.58**	98.17	97.97	98.17	**98.58**	97.76	97.97	97.76	97.97	97.97	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
DS10	92.59	92.59	92.59	88.89	92.59	92.59	92.59	88.89	88.89	88.89	**96.3**	0.95	0.95	0.95	0.93	0.95	0.95	0.95	0.93	0.93	0.93	**0.98**
DS11	**92.2**	**92.2**	**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	NaN
DS12	68.24	69.71	66.18	69.12	72.35	79.12	77.35	70	75.88	67.06	**95**	0.58	0.62	0.56	0.59	0.65	0.75	0.72	0.55	0.7	0.35	**0.94**
DS13	88.89	88.89	92.59	**100**	**100**	88.89	**100**	92.59	85.19	92.59	**100**	0.91	0.92	0.94	**1**	**1**	0.92	**1**	0.94	0.89	0.94	**1**
DS14	86.27	92.16	92.16	94.12	92.16	94.12	98.04	88.24	82.35	82.35	**100**	0.92	0.95	0.95	0.96	0.95	0.96	0.99	0.93	0.9	0.9	**1**
DS15	81.82	81.82	84.85	90.91	96.97	**100**	**100**	**100**	90.91	**100**	**100**	0.83	0.85	0.88	0.92	0.97	**1**	**1**	**1**	0.92	**1**	**1**
DS16	76.24	75.11	76.92	75.34	80.54	80.77	79.86	80.54	**84.62**	80.54	83.03	0.65	0.6	0.62	0.6	0.73	0.73	0.71	0.73	**0.8**	0.73	0.77
DS17	90	90.3	90.45	90	90.3	89.85	90	90	90.45	90.91	**91.06**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
DS18	90.34	90.34	90.91	90.34	90.34	90.34	85.8	90.91	90.34	90.34	**91.48**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	0.92	**0.95**	**0.95**	**0.95**	**0.95**
DS19	86.67	88.89	88.89	88.89	91.11	95.56	**100**	88.89	88.89	88.89	**100**	0.92	0.94	0.94	0.94	0.95	0.97	**1**	0.94	0.93	0.94	**1**
DS20	93.33	84.44	86.67	84.44	84.44	84.44	**100**	84.44	84.44	84.44	**100**	0.96	0.91	0.92	0.91	0.91	0.91	**1**	0.91	0.91	0.91	**1**
DS21	78.91	75.78	76.17	77.73	76.17	**88.28**	84.38	85.94	87.11	83.2	72.27	0.85	0.83	0.84	0.85	0.84	**0.92**	0.89	0.9	0.91	0.88	0.82
DS22	95.38	93.85	93.85	93.85	92.31	93.85	**100**	90.77	90.77	90.77	**100**	0.97	0.97	0.97	0.97	0.96	0.97	**1**	0.95	0.95	0.95	**1**
DS23	76	84	84	76	80	76	**100**	84	84	80	**100**	0.73	0.82	0.83	0.77	0.76	0.77	**1**	0.8	0.8	0.76	**1**
DS24	85.71	92.86	88.1	90.48	90.48	90.48	88.1	95.24	97.62	88.1	**100**	0.91	0.95	0.92	0.93	0.94	0.94	0.92	0.97	0.98	0.92	**1**
DS25	91.72	91.38	91.72	91.84	92.42	91.03	93.71	92.19	92.77	92.07	**94.87**	0.96	0.95	0.96	0.96	0.96	0.95	**0.97**	0.96	0.96	0.96	**0.97**
DS26	72.49	74.67	75.98	75.11	75.55	79.48	**84.72**	76.86	78.17	74.67	80.35	0.82	0.82	0.83	0.83	0.83	0.86	**0.89**	0.83	0.85	0.82	0.86
DS27	74.36	69.23	84.62	76.92	87.18	66.67	**100**	56.41	56.41	56.41	**100**	0.76	0.67	0.83	0.79	0.85	0.67	**1**	0.6	0.6	0.6	**1**
DS28	**98.79**	**98.79**	**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	NaN
DS29	74.32	74.32	**92.01**	74.49	91.33	74.32	76.36	81.29	89.63	73.98	74.32	NaN	NaN	**0.84**	0.01	0.81	NaN	0.15	0.66	0.79	NaN	NaN
DS30	75.86	86.21	**96.55**	89.66	86.21	**96.55**	**96.55**	82.76	82.76	82.76	**96.55**	0.8	0.89	**0.97**	0.91	0.88	**0.97**	**0.97**	0.85	0.85	0.86	**0.97**



(a) Linear Kernel



(b) Polynomial Kernel



(c) RBF Kernel

Fig. 8. Box-plot diagrams of accuracy and F-Measure.

Fig. 9. *t*-test analysis (p-value).

Table 10
Mean difference between performance of different feature selection techniques.

Accuracy											
	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
AM	0	−1.3	−0.76	−0.66	−1.87	−1.87	−4.72	−1.26	−1.07	−0.57	−5.2
FR1	1.3	0	0.54	0.64	−0.58	−0.58	−3.42	0.04	0.23	0.72	−3.9
FR2	0.76	−0.54	0	0.1	−1.11	−1.11	−3.95	−0.5	−0.31	0.19	−4.44
FR3	0.66	−0.64	−0.1	0	−1.22	−1.22	−4.06	−0.6	−0.41	0.08	−4.54
FR4	1.87	0.58	1.11	1.22	0	0	−2.84	0.61	0.8	1.3	−3.33
FR5	1.87	0.58	1.11	1.22	0	0	−2.84	0.62	0.8	1.3	−3.33
FR6	4.72	3.42	3.95	4.06	2.84	2.84	0	3.46	3.64	4.14	−0.48
FS1	1.26	−0.04	0.5	0.6	−0.61	−0.62	−3.46	0	0.19	0.68	−3.94
FS2	1.07	−0.23	0.31	0.41	−0.8	−0.8	−3.64	−0.19	0	0.5	−4.13
FS3	0.57	−0.72	−0.19	−0.08	−1.3	−1.3	−4.14	−0.68	−0.5	0	−4.62
FS4	**5.2**	**3.9**	**4.44**	**4.54**	**3.33**	**3.33**	**0.48**	**3.94**	**4.13**	**4.62**	**0**

source code metrics for predicting faulty and non-faulty classes with in object oriented software.

9.4. Comparison of results

Pairwise *t*-test has been employed to determine which out of the feature selection techniques and classifier work better or all have performed equally well.

1. *Feature Selection Techniques:* In this study eleven different set of metrics have been considered as input to develop a model over thirty different object oriented softwares. LSSVM with three different types of kernel methods have been considered to develop a prediction model considering two different performance parameters i.e., Accuracy, and F-Measure. For each feature selection method two sets (one for each performance measure) are used, each with 90 data points (3 classifier × 30 dataset). Here, ttest between different feature selection techniques are performed and compared the corresponding P-value value as to measure statistical significance. Fig. 9 shows the result of *t*-test analysis. For the sake of simplicity, the P-values are represented using two different symbols such as ●: P-value > 0.05 (no significance difference) and ●: P-value <= 0.05 (significance difference). From Fig. 9 it is evident that, most of the cell have green circle (●), this shows that there isn't significant difference between applied feature selection techniques. But, by judging the value of mean difference shown in Table 10, FS4 i.e., selected set of metrics using rough set analysis yield better results compared to other approaches.
Apart from comparison between feature selection based on two performance parameter i.e., Accuracy and F-Measure, this paper also make the comparison based on cost-benefit analysis. Cost-benefit analysis for each feature selection techniques is calculated using Eq. (29).

$$Cost - Benefit = (Based_{cost} + Benefit_{cost})/2 \tag{29}$$

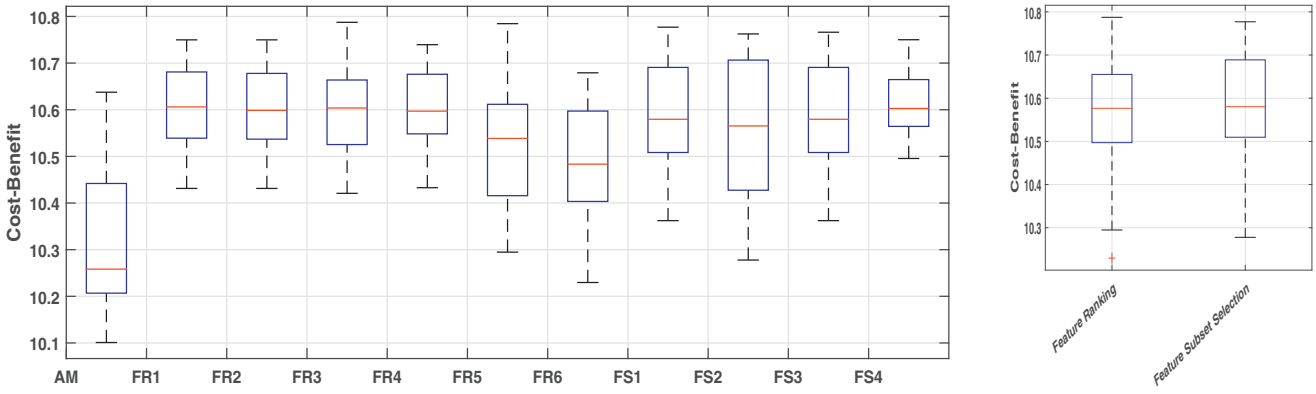
where $Based_{cost}$ is based on the correlation between selected set of source code metrics and fault in th class. $Based_{cost}$ is calculated using following equation:

$$Based_{cost} = Accuracy(SM) * \rho_{SM,fault} \tag{30}$$

where Accuracy(SM) is the classification accuracy of develop fault prediction model using selected set of source code metrics, $\rho_{SM,fault}$ is a multiple correlation coefficient between selected set of source code metrics and fault. The developed model producing high accuracy and having a high multiple correlation coefficient will also be obtained a high $Based_{cost}$
We define that *NAM* is a number of considered source code metrics and *NSM* is the number of selected source code metrics using feature selection techniques, then $Benefit_{cost}$ is defined as follow:

$$Benefit_{cost} = NAM - NSM/NAM \tag{31}$$

The feature selection techniques having high value of $Cost - Benefit$ is the best feature selection techniques as suggested by Chaikla and Qi (1999). The $Cost - Benefit$ of different feature selection techniques are shown in Fig. 10a. Based on Fig. 10a, we infer that FS4 i.e., selected set of metrics using rough set analysis having high median $Cost - Benefit$ value as compare to other techniques.
2. *Classification methods:* In this paper, LSSVM method with three different kernel methods have been considered to develop a



(a) Feature Selection Techniques

(b) Feature Ranking and Feature Subset Selection Techniques

Fig. 10. Cost-benefit value.

Table 11
Mean difference between performance of different classification methods.

Accuracy			
	Linear	Polynomial	RBF
Linear	0	−2.53	−5.87
Polynomial	2.53	0	−3.34
RBF	**5.87**	**3.34**	**0**

Table 12
T-test analysis between feature ranking and feature subset selection methods.

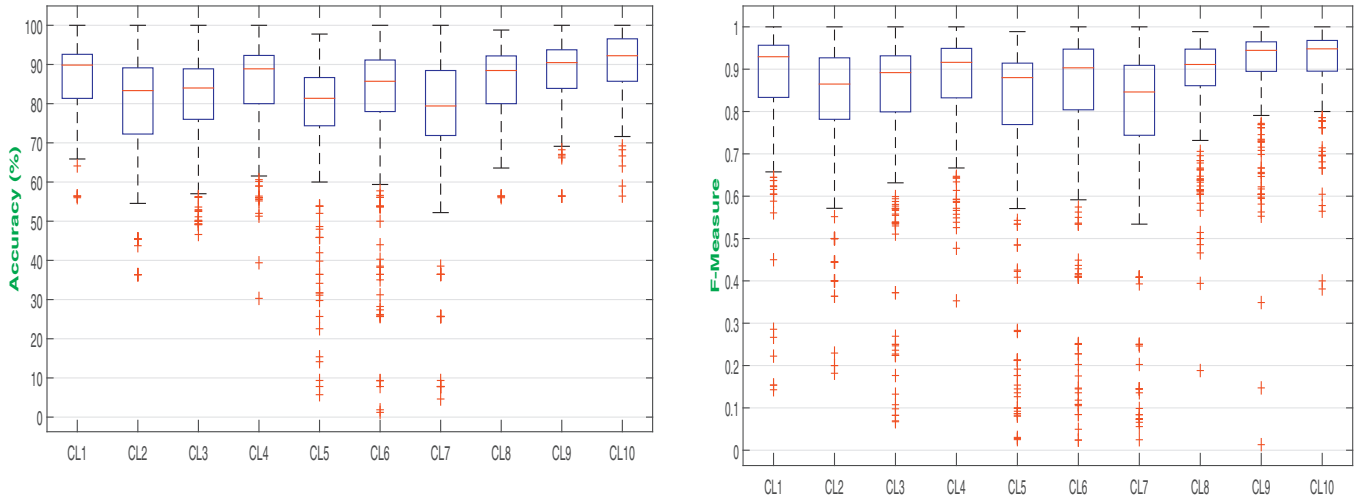
Accuracy		
Mean(FR -FS)	P-value P-value	t-value
−0.1808	0.755	−0.3112
F-Measure		
−0.0088	0.598	−0.5298

model to predict whether the class is faulty or not. This study uses eleven different subset of metrics (10 feature selection method + 1 considering all features) of thirty object oriented software data set with two different performance parameters i.e., accuracy, F-Measure are considered, so for each prediction technique a total number of two sets (one for each performance) are used, each with 330 data point ((10 feature selection method + 1 considering all features) × 30 datasets)). Fig. 9 shows the result of *t*-test analysis. From Fig.9, it is evident that, there is a significant difference between these approaches due to the fact that p-value is smaller than 0.05 (●). But upon judging the value of mean difference as shown in Table 11, LSSVM with radial basis function kernel function yields better results compared to other classifiers.

3. *Feature ranking and feature subset selection methods:* In this study, pairwise *t*-test has been employed to determine whether feature ranking methods work better than feature subset selection methods. Two sample pairs of performance measure are considered between the average performance of feature ranking method and the average performance of feature subset selection method. The results of *t*-test analysis for average performance of feature ranking and feature subset selection technique are summarized in Table 12. Since, in this study three different types of prediction techniques are applied over thirty object oriented software data set with two performance parameters, so for each feature selection techniques a total number of two sets (one for each performance measure) are used, feature ranking with 540 data points (6 feature ranking technique × 3 classifier × 30 datasets) and feature subset selection with 360 points (4 feature subset selection technique × 3 classifier × 30 datasets). From Table 12, it is evident that, there isn't a significant difference between two approaches, due to the fact that p-value being greater than 0.05. But, by judging the value of mean difference feature subset selection method yield better result compared to feature ranking method. It signifies that an average feature-subset selection method performs better as compared to average feature ranking method. From *Cost – Benefit* analysis as shown in Fig. 10b, we infer that both feature ranking and feature subset selection have almost same *Cost – Benefit* value. This show that the average cost and benefit of model developed using selected set of software metrics bu feature ranking and feature subset selection techniques having nearly same.

Apart from the comparative analysis done to find the best method to develop a model which can predict faults accurately, this paper also makes the comparison with other most frequently used classification methods in literature such as logistic regression (CL1), decision tree analysis (CL2), Naive Bayes classifier (CL3), neural network (CL4), SVM with linear kernel (CL5), SVM with polynomial kernel (CL6), SVM with RBF kernel (CL7). In this study eleven different subset of metrics (10 feature selection method + 1 considering all features) of thirty object oriented software dataset with two different performance parameters i.e., accuracy, F-Measure are considered, so for each classifier a total number of two sets (one for each performance) are used, each with 330 data point ((10 feature selection method + 1 considering all features) × 30 datasets)). Fig. 11 shows the box-plot diagrams for accuracy and F-Measure for most frequently used classifier and LSSVM (LSSVM with liner kernel (CL8), LSSVM with polynomial kernel (CL9), and LSSVM with RBF kernel (CL10)). From Fig. 11, LSSVM have high median value as well as few number of outliers. Based on these boxplots, LSSVM produced the best result i.e., model develop using LSSVM for predicting faulty and non-faulty classes within object oriented software yield better result as compared to other models.

We also employed pairwise *t*-test to determine which classifier produces better results. The results of *t*-test analysis for different classifier are in Fig. 12. From Fig. 12 it is evident that, among most of the cases there is a significant difference between these approaches due to the fact that p-value is smaller than 0.05 (●).



(a) Accuracy (b) F-Measure

Fig. 11. Performance box-plot diagrams of different classifier.

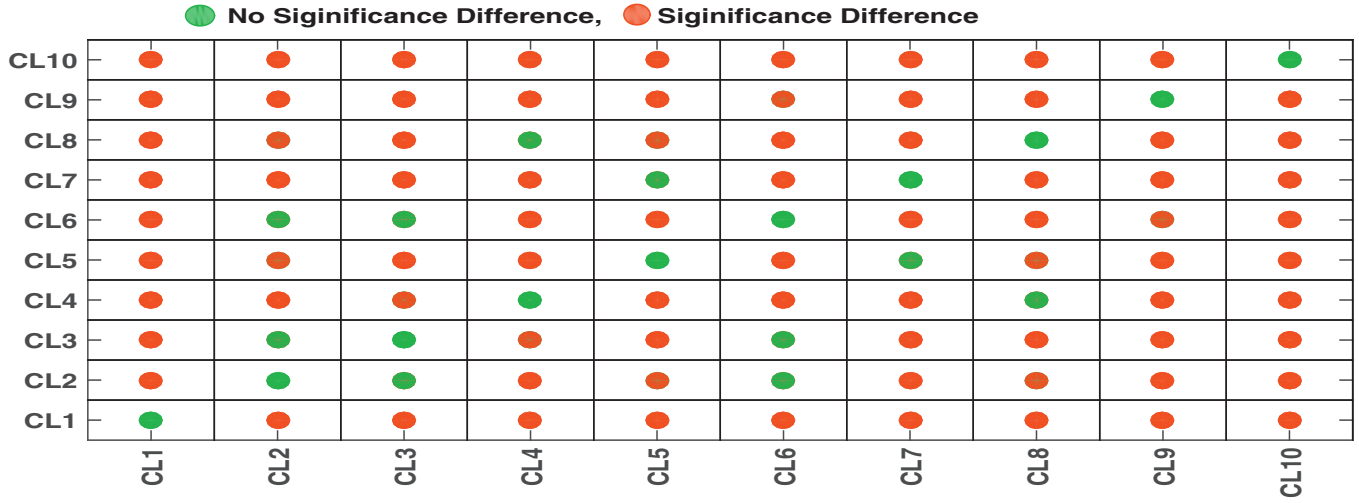


Fig. 12. *t*-test analysis (p-value).

Table 13
Mean difference between performance of different classifier.

Accuracy										
	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9	CL10
CL1	0	6.94	6.42	2.21	9.9	5.88	10.28	2.01	−1.18	−2.99
CL2	−6.94	0	−0.52	−4.73	2.96	−1.06	3.34	−4.92	−8.12	−9.93
CL3	−6.42	0.52	0	−4.21	3.47	−0.54	3.86	−4.41	−7.6	−9.41
CL4	−2.21	4.73	4.21	0	7.69	3.67	8.07	−0.19	−3.38	−5.2
CL5	−9.9	−2.96	−3.47	−7.69	0	−4.02	0.38	−7.88	−11.07	−12.88
CL6	−5.88	1.06	0.54	−3.67	4.02	0	4.4	−3.86	−7.05	−8.87
CL7	−10.28	−3.34	−3.86	−8.07	−0.38	−4.4	0	−8.27	−11.46	−13.27
CL8	−2.01	4.92	4.41	0.19	7.88	3.86	8.27	0	−3.19	−5
CL9	1.18	8.12	7.6	3.38	11.07	7.05	11.46	3.19	0	−1.81
CL10	**2.99**	**9.93**	**9.41**	**5.2**	**12.88**	**8.87**	**13.27**	**5**	**1.81**	**0**

But by observing the value of mean difference in Table 13 it can be observed that LSSVM yields better result compared to other classifiers.

9.5. Interaction of classification method with the feature selection method

This section focuses on the effect of classification methods over the performance of feature selection methods. In this study, LSSVM with three different types of kernel methods i.e., linear kernel, polynomial kernel, and radial basis kernel function have been implemented for investigation. From Fig. 8a–c, it can be inferred that for each classification method required different feature selection methods to produce better results.

Table 14
Testing cost (T_{cost}).

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$	$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$	$\delta_u = 0.5, \delta_i = 0.6, \delta_s = 0.65$
DS1	1079.78	2322.85	3913.59
DS2	260.25	518.84	975.02
DS3	82.47	187.88	290.79
DS4	1313.84	2782.28	4796.06
DS5	63.69	121.92	242.56 D
DS6	30.28	56.77	116.22
DS7	32.81	67.28	121.45
DS8	389.55	775.33	1460.48
DS9	403.53	707.21	1587.23
DS10	39.08	84.05	141.67
DS11	746.97	1840.66	2526.17
DS12	892.17	2126.77	3072.74
DS13	51.64	117.58	182.12
DS14	69.64	147.58	254.12
DS15	71.83	167	250.68
DS16	1213.49	2908.16	4167.47
DS17	702.16	1378.29	2647.39
DS18	216.75	446.34	801.02
DS19	62	131.7	226.01
S20	62	131.7	226.01
DS21	461.93	1040.95	1637.63
DS22	77	156.7	286.01
DS23	62.69	148.61	216.57
DS24	72.3	161.48	257.46
DS25	885.18	1718	3352.62
DS26	416.57	940.13	1475.74
DS27	92.03	216.41	319.24
DS28	3500.35	8664.3	11807.58
DS29	1812.63	4398.43	6182.94
DS30	62.55	145.23	218.46

9.6. Cost analysis

In this experiment, the values tabulated in Tables 5 and 6 have been used in design of cost evaluation model. Eq. (20), and (25) are used to calculate the estimated fault removal cost (E_{cost}) and estimated testing cost (T_{cost}) respectively. δ_u, δ_i, and δ_s in equations show the fault identification efficiency of unit, integration and system testing, respectively. The values of δ_u, δ_i, and δ_s have been collected from the survey report "Software Quality in 2010" of Jones (2010). Table 14 shows the value of estimated testing cost (T_{cost}) for different value of δ_u, δ_i, and δ_s of different data set. For the sake of simplicity, each dataset is numbered from DS1-DS30.

Tables A1–A3 show the estimated fault removal cost (E_{cost}) for δ_u=0.25, δ_i=0.45 and δ_s=0.5. The results for other value of δ_u, δ_i and δ_s are similar in nature.

Figs. 13–15 depict the normalized fault removal cost (NE_{cost}) of fault prediction techniques for different values of δ_u, δ_i, and δ_s. Figure these figures, it is observed that as the percentage value of faulty classes increases, the fault-prediction technique tends to have a higher value of NE_{cost} i.e., fault prediction can be useful for the projects with percentage of faulty classes having less than certain threshold.

9.7. WEYUKER'S properties analysis

Weyuker defined a list of for metrics which has been used by several researchers to evaluate their proposed metrics (Weyuker, 1988). In this section, we present our analysis of Weyuker's properties on our proposed cost analysis measurement and present our examination of the number of Weyuker's properties satisfied. Analysis of Weyuker's properties for software metrics is an area which has attracted several researcher's attention. The objective of analyzing Weyuker's properties is to validate our proposed cost analysis framework.

i. **Property 1:** $(\exists P)(\exists Q)(E_{cost}(P) \neq E_{cost}(Q))$:
The property is an essential requirement and states that there are projects P and Q such that the estimated fault removal cost of project P is not equal to the estimated fault removal cost of project Q. This property requires that the cost analysis framework should not produce the same value for every project. If all projects have same estimated fault removal cost value then there is no meaning for the estimated fault removal cost measurement making differentiation between all possible projects. From Tables A1–A3, we demonstrates that there exists two projects which have different estimated fault removal cost i.e., E_{cost}. Hence, the E_{cost} satisfies Property 1.

ii. **Property 2:** Let c be a non-negative number, and then there are only finite number of projects of estimated fault removal cost (E_{cost}) c
This property states that there are only a finite number of projects of the same estimated fault removal cost (E_{cost}). Since the universe of discourse deals with a finite set of projects, and each project has a finite number of classes, this property will be satisfied by E_{cost}. Here, c is assumed to be the largest possible number, and should be represented as an upper-bound on the number of classes of the projects.

iii. **Property 3:** There are distinct programs P and Q such that $E_{cost}(P) = E_{cost}(Q)$
This property states that there are multiple project of the same E_{cost}. From Tables A1–A3, we demonstrates that there exists two projects such as DS19 or DS20 with LSSVM-Polynomial kernel and selected set of source code metrics using FR6 have same estimated fault removal cost i.e., E_{cost}. Hence E_{cost} satisfy Property 3.

iv. **Property 4:** $(\exists P)(\exists Q)(P \equiv Q \ \& \ E_{cost}(P) \neq E_{cost}(Q))$:
This property ensures that at-least two different project with equal functionality yield distinct estimated fault removal cost i.e., E_{cost} values. This property is also satisfied by E_{cost}. This is due to the fact that even if two projects perform the same function, the details of design matters in determining the E_{cost} value. The choice of number of classes in projects is independent of the functionality of the project. So, it may be possible that the two projects with similar functionality have different number of classes. E_{cost} also depends on the number of classes. So if projects have different numbers of classes then their E_{cost} will be also different. Hence, the E_{cost} satisfies Property 4.

v. **Property 5:** $(\forall P)(\forall Q)(E_{cost}(P) \leq E_{cost}(P; Q) \ \& \ E_{cost}(Q) \leq E_{cost}(P; Q))$:
This property pertains to the monotonicity characteristic and ensures that the E_{cost} value only increases or grows and never decreases when composing a project body by other project. We demonstrate the satisfy-ability of Property 5 using concatenation. If we concatenate two projects, the number of classes is always greater or equal to number of classes of largest project. Hence, the E_{cost} satisfies Property 5.

vi. **Property 6a:** $(\exists P)(\exists Q)(\exists R)(E_{cost}(P) = E_{cost}(Q) \ \& \ E_{cost}(P; R) \neq E_{cost}(Q; R))$:
Property 6b: $(\exists P)(\exists Q)(\exists R)(E_{cost}(P) = E_{cost}(Q) \ \& \ E_{cost}(R; P) \neq E_{cost}(R; Q))$:
This property pertains to the interaction characteristics between project bodies. Let us say we have two projects (B and C) with equal number of classes. Let us say we have another project (A) which interacts with the two projects (B and C) having equal number of classes. The project A may

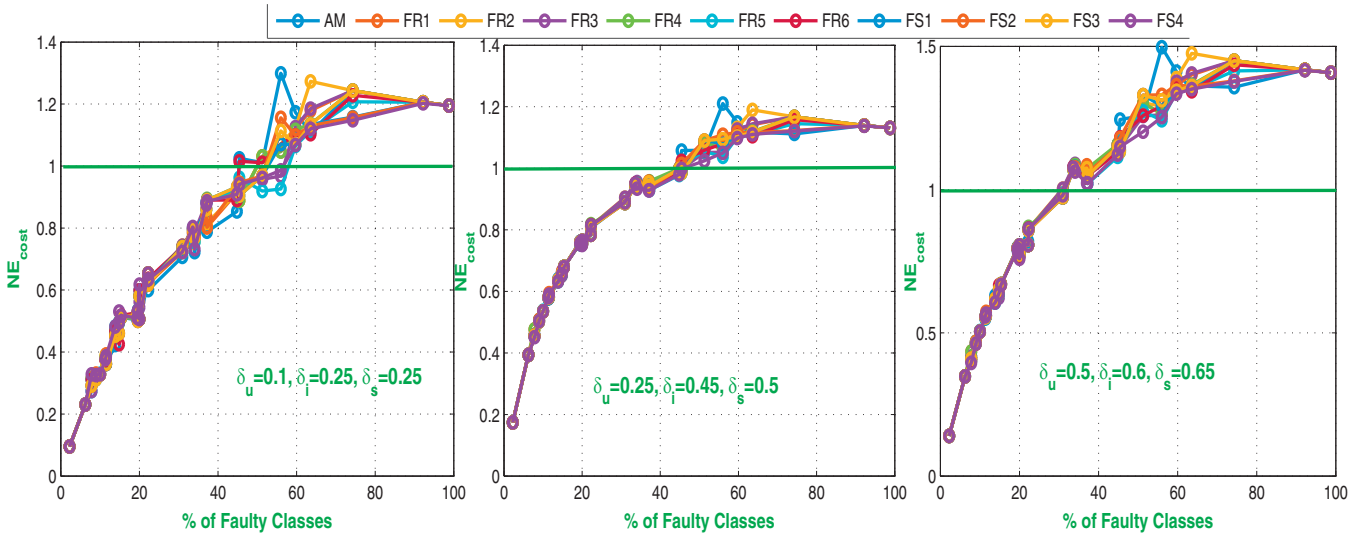


Fig. 13. NE_{cost}: Linear kernel.

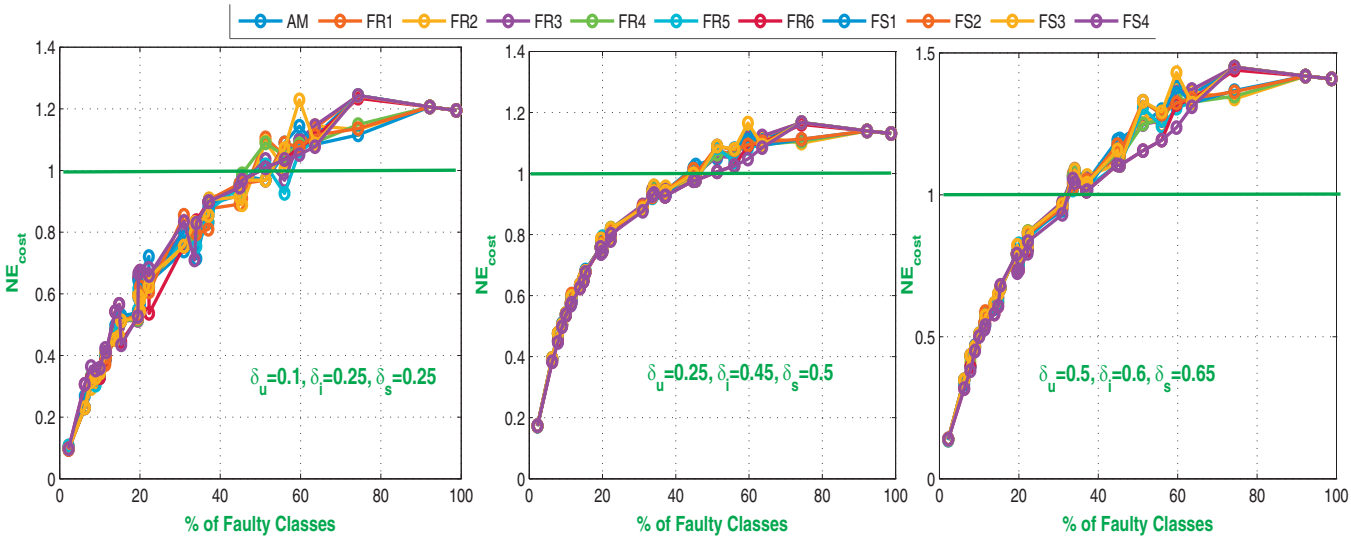


Fig. 14. NE_{cost}: Polynomial kernel.

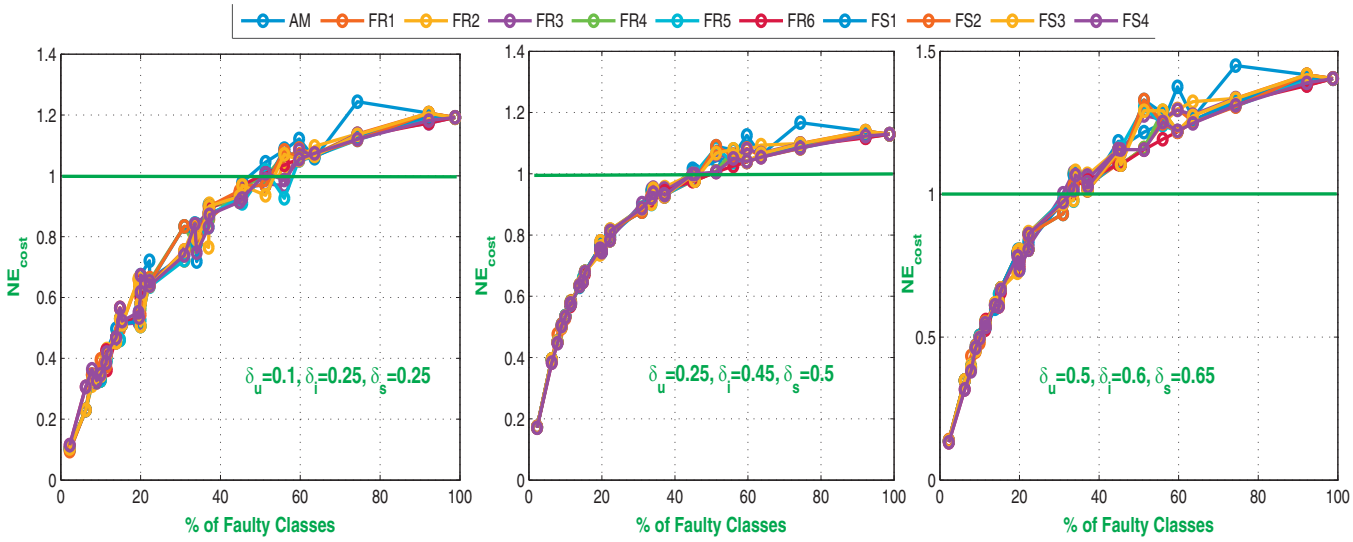


Fig. 15. NE_{cost}: RBF kernel.

Table 15
Confusion matrix.

(a) Project P

	Non faulty	Faulty
Non faulty	5	3
Faulty	2	1

(b) Project Q

	Non faulty	Faulty
Non faulty	4	2
Faulty	2	2

(c) Project P;Q

	Non faulty	Faulty
Non faulty	0	9
Faulty	7	0

interact differently with projects B and C. So final projects have different number of classes. So if projects have different numbers of classes then their E_{cost} will be also different. Hence, the E_{cost} satisfies Property 6.

vii. **Property 7:** There are project bodies P and Q such that Q is formed by permuting the order of the statements of P, and $(E_{cost}(P) \neq E_{cost}(Q))$
This property states that permutation is significant. It means that permutation of elements within the item being measured can change the metric values. The purpose is to ensure that metric values change due to the permutation of classes in project. E_{cost} does not satisfy this property because E_{cost} depends on the number of classes in project not the structure of the projects. Hence, the E_{cost} does not satisfies Property 7.

viii. **Property 8:** If P is a renaming of Q, then $(E_{cost}(P) = E_{cost}(Q))$ E_{cost} value of the project depends on the number of classes in the projects. The renaming of various elements of the classes do not change the number of classes in the project. Hence, the E_{cost} satisfies Property 8.

ix. **Property 9:** $(\exists P)(\exists Q)(E_{cost}(P) + E_{cost}(Q) < E_{cost}(P; Q))$:
This property ensures that the E_{cost} should result in a possible increase in E_{cost} value if an interaction of two projects occurs. The property requires the existence of two projects whose concatenation has a E_{cost} value which is more than the sum of its individual E_{cost} value. Table 15 show the confusion matrix of three project: P, Q, and concatenation of P and Q (P;Q). The estimated fault removal cost (E_{cost}) value of project P and Q are 38.23 and 49.12 respectively (Computed using Eq. (20)). The summation value of E_{cost} for P and Q projects is 87.35 which is less than the E_{cost} vale P;Q i.e., 113.24. Hence, the E_{cost} satisfies Property 9.

Table 16 the Weyuker's property analysis for proposed cost analysis framework. We present the detailed analysis of Weyuker's property in this paper only for the E_{cost} and not for others due to the limited space in the paper. Table 16 reveals that the proposed cost analysis framework satisfies 8 out of the 9 properties. Previous research shows (Misra and Akman, 2008) that satisfying all the 9 properties is not a necessary condition for a metric to be adopted. Satisfying various Weyuker's properties is a validation step which indicates the strength or quality of the metric but satisfying all the properties is not mandatory. We observe that in the original work of Weyuker's 1988, the author did not find a single measurement which is satisfied by all 9 properties.

Table 16
Weyuker properties and cost analysis framework metrics. The symbol (denoting yes or no) in each cell shows if the property is satisfied or not.

Properties	E_{cost}	T_{cost}	NE_{cost}
Property 1	✓	✓	✓
Property 2	✓	✓	✓
Property 3	✓	✓	✓
Property 4	✓	✓	✓
Property 5	✓	✓	✓
Property 6	✓	✓	✓
Property 7	×	×	×
Property 8	✓	✓	✓
Property 9	✓	✓	✓

9.8. Threshold value

After finding the best model, an attempt is made to establish a relationship between the NE_{cost} and % of faulty classes (FP). In this study, logistic regression has been considered to develop a model to calculate the probability of usefulness of fault prediction techniques (P_{fault}). In logistic regression, dependent variable can take only two values. So the dependent variable of a NE_{cost} is divided into two groups, one group containing software for which fault prediction will be useful ($NE_{cost} < 1$) or other not useful ($NE_{cost} >= 1$). Table 17 shows the constant, coefficient, and threshold value in term of % of faulty classes for different values of δ_u and δ_s. Logistic regression analysis is applied with a threshold value 0.5. It implies that the fault prediction is useful if ($P_{fault} < 0.5$) otherwise not useful. From Table 17, it is observed that, fault prediction useful for the projects with percentage of faulty classes less than a certain threshold.

9.9. Experiment finding

In this section, the overall findings of the experimental works are presented. The experimental work was carried out for thirty projects using LSSVM with three different types of kernel methods i.e., linear kernel, polynomial kernel, and radial basis kernel function. The performance of the respective technique was evaluated using the proposed cost based evaluation framework and the associated parameters such that δ_u, δ_i and δ_s. Figs. 13–15 depict the normalized fault removal cost (NE_{cost}) of fault prediction techniques for different values of δ_u, δ_i, and δ_s. Table 17 shows the threshold value in term of % of faulty classes for different values of δ_u, δ_i, and δ_s. From Figure and Table, it can be inferred that:

- In case of minimum fault identification efficiency of unit, integration and system testing i.e., δ_u= 0.1, δ_i=0.25 and δ_s= 0.25, fault prediction may be useful when the software has less than 52.139% of faulty classes.
- Similarly, in case of median fault identification efficiency of unit, integration and system testing i.e., δ_u= 0.25, δ_i=0.45, and δ_s= 0.5, fault prediction may be useful when the software has less than 46.206% of faulty classes.
- Also, in case of high fault identification efficiency of unit, integration and system testing i.e., δ_u= 0.5, δ_i=0.60, and δ_s= 0.65, fault prediction may be useful when the software has less than 32.080% of faulty classes.

Based on this study, this paper helps to answer the following research questions.

RQ1: In this study, LSSVM with three different types of kernel methods i.e., linear kernel, polynomial kernel, and radial basis kernel function have been considered to develop

Table 17
Threshold value.

(a) Feature selection techniques

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65$		
	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold
AM	−24.075	0.492	48.943	−2094.492	46.739	44.813	−815.627	25.268	32.279
FR1	−424.823	8.298	51.199	−2123.277	46.697	45.469	−815.627	25.268	32.279
FR2	−424.823	8.298	51.199	−613.682	12.662	48.466	−432.385	13.947	31.002
FR3	−25.241	0.447	56.412	−613.682	12.662	48.466	−414.827	13.424	30.901
FR4	−424.823	8.298	51.199	−613.682	12.662	48.466	−815.627	25.268	32.279
FR5	−18.139	0.330	54.966	−2222.684	48.914	45.440	−2064.905	61.488	33.582
FR6	−2123.277	46.697	45.469	−2123.277	46.697	45.469	−815.627	25.268	32.279
FS1	−798.369	14.881	53.648	−41.656	0.888	46.902	−815.627	25.268	32.279
FS2	−798.369	14.881	53.648	−43.508	0.948	45.874	−815.627	25.268	32.279
FS3	−798.369	14.881	53.648	−613.682	12.662	48.466	−2064.905	61.488	33.582
FS4	−19.441	0.363	53.500	−613.682	12.662	48.466	−815.627	25.268	32.279

(b) Classification methods

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65$		
	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold
Linear	−19.905	0.384	51.881	−2132.814	46.910	45.466	−442.648	14.252	31.058
Polynomial	−28.175	0.548	51.436	−44.405	0.957	46.418	−815.627	25.268	32.279
RBF	−28.541	0.537	53.125	−69.144	1.489	46.431	−54.946	1.696	32.390

(c) Threshold value (All)

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65$		
	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold
Overall	−24.061	0.461	52.139	−68.089	1.474	46.206	−66.544	2.074	32.080

a model to predict as to whether the class is faulty or not. From Table 9a–c, it can be inferred that model developed using LSSVM with radial basis function kernel function by considering selected set of metrics using rough set analysis as input yields better result as compared to others.

RQ2: To answer RQ2, Figs. 13–15 were analyzed. Here, it is found that, fault prediction process is useful for the projects with percentage of faulty classes less than a certain threshold value.

RQ3: In this study, six different types of feature ranking methods and four different types of feature subset selection methods are considered to find the reduced subset of object-oriented metrics. By using these methods, best possible subsets of the object-oriented metrics are determined which can be used to develop a model to predict as to whether the class is faulty or not. From Table 9a to c, in most of cases there exists a reduced subset of object-oriented metrics that is better for developing a prediction model as compared to considering all seventeen metrics.

RQ4: In this study, six different types of feature ranking methods are considered to find the reduced subset of object-oriented metrics. From *t*-test analysis, it is evident that feature selection using PCA method produces the best results as compared to others.

RQ5: In this study, four different types of feature subset selection methods have been considered to find the reduced subset of object-oriented metrics. From *t*-test analysis, it is observed that feature selection using rough set analysis produces the results which are convincingly quite better as compared to others.

RQ6: In this study, pairwise *t*-test has been employed to determine whether feature ranking methods work better than feature subset selection methods or they both perform equally well. From *t*-test results it is observed that, there is a significant difference between feature ranking and feature subset selection method. However, the mean difference value shows that feature ranking method outperforms the feature subset selection methods.

RQ7: From Section 9.5, it is observed that the performance of the feature selection methods is varied with the different classification methods used. It shows that selection of classification method to develop a prediction model to predict whether the class is faulty or not is affected by the feature selection methods.

10. Threat to validity

For the sake of completeness, some of the existing threats to validity of the proposed work have been considered. The proposed work may suffer from following threats:

i. *Internal Validity:* We identify two possible threats to the internal validity of our study. The first threat is the consistency of project fault data. We collect fault data of project from promise repository[5]. Any fault information not mention in the sources were not considered in the study. We do not make claims about the accuracy of data, but we believe that the data are collected consistently. The second threat is that cost parameter may be incorrectly collected. We take cost parameter (C_u, C_i, C_s, C_f) from Wagner (2006) and fault identification efficiencies (δ_u, δ_i, δ_s) from Jones (2010), and M_p from Huitt and Wilde (1992). However, all these values may vary with company benchmarks.

ii. *Construct Validity:* In this study, developed models for fault prediction only predicts that whether a class is faulty or not, but does not indicate on possible number of bugs in the class. Nonetheless, we did realize that this is a potential threat to the construct validity of the dependent variable (fault proneness) and needs to be eliminated in the future work.

iii. *External Validity:* In this work, all case studies are designed in Java language. However the models designed in this study

[5] http://openscience.us/repo/defect/.

are likely to be valid for other object-oriented programing languages. Further research can be extended to design a model for other programing paradigms too. Number of psychological factors also affect the reliability of software. But in this study, factors such as different level of expertise for developers, standards in which software is developed, types of developers involved, history of development of the system and other stockholders of the system are not considered.

11. Conclusion

This study emphasized on designing a cost based evaluation framework for determining efficiency of the developed fault prediction model which is created using object oriented source code metrics. In this study, twenty object-oriented source code metrics are used to develop a model using Least Square Support Vector Machines (LSSVM) with three different kernel methods i.e., linear kernel, polynomial kernel, and RBF kernel. The implementation process was carried out by taking help of thirty projects following object oriented paradigm. The experiments conducted and results generated are MATLAB environment.

Our observations are the following:

- 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 value of misclassified errors.
- From the experimental finding, we observed that even after removing 70% (Average) of the available number of source code metrics, the developed fault prediction models were not adversely affected; in fact, in most of the cases the results were better. This is as important concept for software engineers, since the models considers a small number of source code metrics from data collection, management, modeling, and analysis points of view.
- The MOA, CBM, CAM, AMC, LOC, NOC source code metrics were found to be significant predictors for fault prediction using feature selection methods.
- Based on *t*-test analysis, it is observed that there is no any significant difference between applied feature selection techniques. It is also observed that the models developed using different classification techniques are significantly different.
- Based on mean difference, it is observed that the model developed using selected set of source code metrics as input yields better performance compared to all metrics. It is also observed that the model developed using LSSVM with radial basis kernel function yields better results as compared to other kernels.
- Based on Cost-Benefit analysis, we infer that the selected set of metrics using rough set analysis having high median $Cost - Benefit$ value as compare to other techniques.
- We found that the selection of classification method to develop a fault prediction model is affected by the feature selection methods.
- Based on cost analysis framework, it is observed that as the percentage value of faulty classes increases, the fault-prediction technique tends to have a higher value of NE_{cost}. Cost analysis results reveal that our developed fault prediction model is best suitable for projects with faulty classes less than the threshold value depending on fault identification efficiency (low- 52.139%, median- 46.206%, and high- 32.080%)

In this study, developed models for fault prediction only predicts that whether a class is faulty or not. Future work can be extended to indicate on possible number of bugs in the class. Further, this work can be replicated over other open source projects which use soft computing models to achieve higher accuracy of fault prediction.

Appendix A

Table A1
Linear kernel: estimated fault removal cost (E_{cost}) for $\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$.

	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
DS1	1872.51	1879.27	1887.66	1883.69	1900.89	1891.92	1894.13	1885.30	1883.69	1881.19	1891.92
DS2	301.79	301.79	301.79	301.79	301.79	301.79	301.79	301.79	308.41	301.79	305.91
DS3	176.49	176.19	175.90	177.22	180.01	174.42	177.22	177.22	176.19	177.22	174.42
DS4	2102.69	2114.75	2112.54	2104.60	2104.31	2112.54	2117.69	2119.16	2113.57	2112.25	2106.22
DS5	55.89	58.09	55.59	55.89	58.09	55.30	55.30	55.59	55.59	55.59	55.00
DS6	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36	22.36
DS7	44.12	44.42	44.12	43.83	44.12	44.12	44.71	44.42	44.42	44.42	44.12
DS8	450.92	448.42	452.54	448.13	450.33	450.92	448.42	447.54	450.92	450.04	450.92
DS9	122.95	122.95	122.95	122.95	122.95	122.95	122.95	122.95	122.95	122.95	122.95
DS10	66.18	65.89	65.89	65.89	65.89	65.89	65.89	65.89	65.89	65.89	65.89
DS11	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2094.71
DS12	2444.72	2381.78	2407.52	2395.17	2371.19	2342.67	2363.84	2344.88	2375.32	2347.38	2336.20
DS13	111.92	109.71	112.21	111.92	109.71	109.71	109.71	112.21	112.51	112.21	109.13
DS14	111.19	111.48	111.48	111.19	111.48	111.19	111.48	111.78	111.78	111.78	111.19
DS15	176.63	164.72	164.72	166.63	164.72	169.13	171.04	164.42	169.42	164.42	166.63
DS16	3220.82	3319.93	3460.66	3329.34	3255.37	3224.64	3210.82	3250.08	3245.67	3250.08	3228.46
DS17	737.72	737.72	737.72	737.72	737.72	740.22	737.72	737.72	737.72	737.72	737.72
DS18	303.26	303.26	303.26	303.26	303.26	303.26	302.97	303.26	303.26	303.26	302.97
DS19	99.13	99.42	99.42	99.71	99.42	99.42	99.13	99.42	99.42	99.42	98.83
DS20	100.60	100.60	100.60	100.01	100.60	100.60	100.60	100.60	100.60	100.60	100.60
DS21	981.55	980.23	980.82	988.76	984.35	986.70	989.78	984.49	985.37	984.49	989.20
DS22	100.30	99.42	99.42	99.71	99.42	99.13	99.42	99.71	99.71	99.71	99.13
DS23	179.86	157.95	160.45	156.34	157.95	154.13	160.45	160.45	164.86	162.66	156.04
DS24	143.84	143.25	146.04	146.04	143.25	143.54	143.25	143.25	143.54	143.25	143.54
DS25	870.52	870.22	863.02	864.34	870.22	864.34	862.14	861.70	871.84	861.70	864.34
DS26	884.64	894.19	892.43	898.01	892.57	886.99	885.22	892.43	890.51	885.51	877.87
DS27	229.13	234.43	234.43	227.52	234.43	228.40	229.13	235.90	235.90	235.90	221.63
DS28	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55
DS29	5133.43	5133.43	5133.43	5133.43	5133.43	5037.69	5094.31	4890.49	4937.26	5133.43	4929.91
DS30	143.25	145.16	145.16	145.16	145.16	142.07	142.66	144.86	144.86	144.86	142.36

Table A2
Polynomial kernel: estimated fault removal cost (E_{cost}) for $\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$.

	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
DS1	1896.63	1904.86	1906.04	1903.24	1907.65	1874.12	1855.47	1883.39	1882.22	1903.68	1861.91
DS2	308.70	310.62	305.32	313.12	310.62	298.55	301.64	305.32	314.00	310.03	298.55
DS3	177.22	177.51	179.72	177.51	176.92	174.72	174.13	174.42	176.92	176.92	174.13
DS4	2112.25	2108.28	2113.57	2108.13	2106.95	2111.21	2102.39	2108.86	2110.92	2107.69	2109.60
DS5	58.09	58.09	55.59	55.00	57.80	55.00	55.00	58.09	58.09	58.09	54.42
DS6	22.06	22.36	22.36	22.36	22.36	22.36	21.77	22.36	22.36	22.36	21.77
DS7	44.12	43.83	44.12	44.12	43.83	43.83	43.53	44.42	44.42	44.42	43.53
DS8	446.66	445.92	444.45	444.45	449.45	442.68	447.25	442.39	448.13	444.89	440.62
DS9	122.95	121.78	122.36	122.66	122.36	121.78	122.95	122.66	122.95	122.66	122.66
DS10	68.09	65.89	65.89	66.18	65.89	65.89	65.89	66.18	66.18	66.18	65.59
DS11	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91	2096.91
DS12	2377.82	2358.70	2382.08	2372.52	2349.43	2302.96	2320.17	2402.52	2326.05	2478.25	2227.51
DS13	111.92	109.71	109.42	108.83	108.83	109.71	108.83	109.42	110.01	109.42	108.83
DS14	113.10	110.01	110.01	111.92	110.01	109.71	109.13	115.01	115.89	115.89	108.83
DS15	171.63	167.22	164.72	166.34	165.75	163.25	163.25	163.25	166.34	163.25	163.25
DS16	3225.82	3260.37	3258.02	3266.70	3182.73	3184.64	3201.26	3176.11	3210.08	3178.32	3155.23
DS17	737.72	745.36	739.04	737.72	743.15	740.22	737.72	746.54	743.74	746.54	740.80
DS18	303.26	303.26	302.97	303.26	303.26	303.26	301.20	305.17	303.26	303.26	301.79
DS19	99.71	99.42	99.42	98.83	98.83	98.54	97.95	99.42	101.63	99.42	97.95
DS20	101.04	100.01	101.92	100.01	104.42	104.42	97.95	100.01	100.01	100.01	97.95
DS21	978.31	987.28	980.37	983.61	980.37	958.01	971.99	961.99	961.10	977.28	965.67
DS22	98.83	99.13	99.13	99.13	99.42	99.13	97.95	99.71	99.71	99.71	97.95
DS23	160.75	157.95	155.75	156.34	157.95	154.13	152.36	160.16	160.16	160.45	152.36
DS24	143.25	142.36	142.95	144.86	142.66	142.66	142.95	142.07	143.98	142.95	141.48
DS25	872.14	864.20	867.72	871.84	865.96	860.67	862.72	870.96	862.87	866.84	855.36
DS26	874.05	901.25	899.78	892.57	894.04	875.51	878.15	900.66	885.66	902.28	879.04
DS27	227.22	232.66	232.96	228.55	230.16	234.72	217.66	235.90	235.90	235.90	217.66
DS28	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55	9800.55
DS29	5133.43	5133.43	4835.93	5130.93	4865.78	5133.43	5105.93	4890.06	4892.11	5134.01	5133.43
DS30	147.95	142.66	141.78	144.57	144.86	141.78	141.78	147.36	147.36	145.16	141.78

Table A3
RBF kernel: estimated fault removal cost (E_{cost}) for $\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$.

	AM	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
DS1	1852.94	1886.47	1898.98	1832.64	1894.71	1894.42	1890.15	1896.77	1893.39	1899.42	1890.45
DS2	297.38	298.26	302.97	301.35	298.26	299.73	301.79	301.05	298.26	301.05	297.97
DS3	174.13	175.01	179.72	177.51	174.13	174.72	177.22	174.13	174.13	174.13	174.72
DS4	2107.98	2114.16	2116.07	2106.95	2116.66	2115.19	2116.51	2113.57	2053.68	2049.12	2101.06
DS5	54.71	58.09	55.30	55.00	55.00	55.00	54.71	55.00	54.71	55.00	54.42
DS6	22.36	22.36	22.36	22.36	22.36	22.36	21.77	22.36	22.36	22.36	21.77
DS7	43.83	44.12	43.53	43.83	44.42	44.42	43.53	43.83	43.83	43.83	43.53
DS8	446.66	446.36	444.45	444.75	450.63	446.95	440.33	444.16	448.42	446.95	447.25
DS9	122.66	121.78	122.07	122.07	121.78	122.36	120.89	122.07	122.95	122.07	120.89
DS10	68.09	65.89	65.89	65.89	65.89	65.89	65.89	65.89	65.89	65.89	65.89
DS11	2096.91	2096.91	2059.41	2057.21	2064.41	2079.71	2056.91	2087.21	2096.91	2096.91	2066.91
DS12	2393.26	2302.37	2292.96	2297.66	2211.77	2209.57	2209.27	2210.45	2209.27	2210.16	2212.36
DS13	111.63	108.83	110.60	111.92	109.42	109.71	109.13	109.13	109.13	109.13	109.71
DS14	113.69	111.19	111.48	111.19	109.71	109.13	109.13	115.01	115.01	115.01	111.19
DS15	168.84	163.25	163.25	166.34	163.25	167.22	163.25	163.25	163.25	163.25	166.92
DS16	3086.40	3073.60	3180.96	3104.49	3071.10	3072.57	3068.01	3076.25	3067.72	3098.31	3064.04
DS17	737.72	737.72	732.12	737.72	737.72	737.72	734.77	736.24	728.15	735.07	734.18
DS18	303.26	303.26	303.26	303.26	303.26	303.26	299.88	303.26	303.26	303.26	302.67
DS19	99.42	99.42	99.42	99.42	99.71	99.42	99.13	99.42	99.42	99.42	98.83
DS20	97.95	99.13	98.54	98.54	100.60	100.30	98.24	100.60	100.01	100.60	97.95
DS21	962.13	966.40	983.17	989.49	981.85	938.30	949.93	983.61	968.31	938.89	958.75
DS22	98.83	99.42	99.42	99.42	99.42	99.42	99.42	99.71	99.71	99.71	99.42
DS23	157.36	157.95	155.75	156.34	160.45	154.13	152.36	160.16	157.36	160.45	156.04
DS24	143.25	143.54	143.54	146.04	141.48	143.54	142.95	141.48	141.48	142.95	143.25
DS25	872.72	872.14	865.22	862.43	860.37	861.55	858.46	855.22	852.13	857.43	864.05
DS26	879.34	889.48	892.43	898.01	880.80	885.95	879.48	890.51	887.28	894.63	881.99
DS27	225.46	217.96	217.96	230.02	217.96	234.72	217.66	235.90	235.90	230.90	217.66
DS28	9778.05	9780.55	9783.05	9788.05	9778.05	9783.05	9780.55	9778.05	9785.55	9778.34	9783.05
DS29	5133.43	4833.58	4833.28	4836.08	4816.08	4764.60	4797.69	4800.78	4761.81	4829.16	4772.40
DS30	147.66	141.78	145.16	145.16	145.16	142.07	141.78	144.86	144.57	144.86	145.16

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.

Abreu, F.B., Carapuça, R., 1994. Object-oriented software engineering: Measuring and controlling the development process. In: Proceedings of the 4th international conference on software quality, Vol. 186, pp. 1–8.

Aggarwal, K., Singh, Y., Kaur, A., Malhotra, R., 2009. Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study. Software Process 14, 39–62.

Arisholm, E., Briand, L., Johannessen, E.B, 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Empirical Software Eng. 83, 2–17.

Bansiya, J., Davis, C.G., 2002. A hierarchical model for object-oriented design quality assessment. IEEE Trans. Software Eng. 28, 4–17.

Basili, V.R., Briand, L.C., Melo, W.L., 1996. How reuse influences productivity in object-oriented systems. Commun. ACM 39, 104–116.
Bieman, J.M., Kang, B.-K., 1995. Cohesion and reuse in an object-oriented system. In: ACM SIGSOFT Software Engineering Notes, Vol. 20, pp. 259–262. ACM.
Briand, L.C., Wüst, J., Daly, J.W., Porter, D.V., 2000. Exploring the relationships between design measures and software quality in object-oriented systems. J. Syst. Software 51, 245–273.
Burrows, R., Ferrari, F.C., Lemos, O.A., Garcia, A., Taiani, F., 2010. The impact of coupling on the fault-proneness of aspect-oriented programs: An empirical study. In: 2010 IEEE 21st International Symposium on Software Reliability Engineering, IEEE, pp. 329–338.
Jones, C., 2010. Software quality in 2010: a survey of the state of the art. Founder and Chief Scientist Emeritus.
Cartwright, M., Shepperd, M., 2000. An empirical investigation of an object-oriented software system. IEEE Trans. Software Eng. 26, 786–796.
Chaikla, N., Qi, Y., 1999. Genetic algorithms in feature selection. In: Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on, Vol. 5, pp. 538–540. IEEE.
Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Software Eng. 20, 476–493.
Cruz, A.E.C., Ochimizu, K., 2009. Towards logistic regression models for predicting fault-prone code across software projects. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, pp. 460–463.
Dash, M., Liu, H., 2003. Consistency-based search in feature selection. Artif. Intell. 151, 155–176.
Doraisamy, S., Golzari, S., Mohd, N., Sulaiman, M.N., Udzir, N.I., 2008. A study on feature selection and classification techniques for automatic genre classification of traditional malay music.. In: ISMIR, pp. 331–336.
Emam, K.E., Melo, W., Machado, J.C., 2001. The prediction of faulty classes using object-oriented design metrics. J. Syst. Software 56, 63–75.
Erturk, E., Sezer, E.A., 2015. A comparison of some soft computing methods for software fault prediction. Expert Syst. Appl. 42, 1872–1879.
Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A., 2011. An empirical study on web service evolution. In: Web Services (ICWS), 2011 IEEE International Conference on, IEEE, pp. 49–56.
Forman, G., 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3, 1289–1305.
Furlanello, C., Serafini, M., Merler, S., Jurman, G., 2003. Entropy-based gene ranking without selection bias for the predictive classification of microarray data. BMC Bioinf. 4, 1.
Gao, K., Khoshgoftaar, T.M., Napolitano, A., 2009. Exploring software quality classification with a wrapper-based feature ranking technique. In: 2009 21st IEEE International Conference on Tools with Artificial Intelligence, IEEE, pp. 67–74.
Gondra, I., 2008. Applying machine learning to software fault-proneness prediction. J. Syst. Software 81, 186–195.
Goyal, R., Chandra, P., Singh, Y., 2014. Suitability of knn regression in the development of interaction based software fault prediction models. IERI Procedia 6, 15–21.
Gyimothy, T., Ferenc, R., Siket, I., 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. Software Eng. 31, 897–910.
Halstead, M.H., 1977. Elements of Software Science, 7. Elsevier New York.
Henderson-Sellers, B., 1996. Software metrics.
Huang, S.-Y., 1992. Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory, 11. Springer Science & Business Media.
Huitt, R., Wilde, N., 1992. Maintenance support for object-oriented programs. IEEE Trans. Software Eng. 18, 1038–1044.
Jiang, Y., Cukic, B., Ma, Y., 2008. Techniques for evaluating fault prediction models. Empirical Software Eng. 13, 561–595.
Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V., Thambidurai, P., 2007. Object-oriented software fault prediction using neural networks. Inf. Software Technol. 49, 483–492.
Kapila, H., Singh, S., 2013. Analysis of ck metrics to predict software fault-proneness using bayesian inference. Int. J. Comput. Appl. 74.
Kohavi, R., et al., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai, Vol. 14, pp. 1137–1145.
Kohavi, R., John, G.H., 1997. Wrappers for feature subset selection. Artif. Intell. 97, 273–324.
Kumar, L., Jetley, R., Sureka, A., 2016a. Source code metrics for programmable logic controller (plc) ladder diagram (ld) visual programming language. In: Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics, ACM, pp. 15–21.
Kumar, L., Krishna, A., Rath, S.K., 2016b. The impact of feature selection on maintainability prediction of service-oriented applications. Serv. Oriented Comput. Appl. 1–25.
Kumar, L., Kumar, M., Rath, S.K., 2016c. Maintainability prediction of web service using support vector machine with various kernel methods. Int. J. Syst. Assur. Eng. Manage. 1–18.
Kumar, L., Rath, S.K., 2016. Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software. J. Syst. Software 121, 170–190.
Kumar, L., Rath, S.K., Sureka, A., 2016d. Predicting quality of service (qos) parameters using extreme learning machines with various kernel methods. In: 4th International Workshop on Quantitative Approaches to Software Quality, p. 11.
Li, W., Henry, S., 1993. Maintenance metrics for the object oriented paradigm. In: Software Metrics Symposium, 1993. Proceedings., First International, IEEE, pp. 52–60.
Lorenz, M., Kidd, J., 1994. Object-Oriented Software Metrics: A Practical Guide. Prentice-Hall, Inc.
Malhotra, R., Jain, A., 2012. Fault prediction using statistical and machine learning methods for improving software quality. J. Inf. Process. Syst. 8, 241–262.
Malhotra, R., Singh, Y., 2011. On the applicability of machine learning techniques for object oriented software fault prediction. Software Eng. 1, 24–37.
Martin, R., 1994. Oo design quality metrics. Analy. Dependencies 12, 151–170.
McCabe, T.J., 1976. A complexity measure. IEEE Trans. Software Eng. 308–320.
Mishra, B., Shukla, K., et al., 2012. Defect prediction for object oriented software using support vector based fuzzy classification model. Int. J. Comput. Appl. 60.
Misra, S., Akman, I., 2008. Applicability of weyukers properties on oo metrics: some misunderstandings. ComSIS 5, 17–24.
Nagappan, N., Williams, L., Vouk, M., Osborne, J., 2005. Early estimation of software quality using in-process testing metrics: a controlled case study. In: ACM SIGSOFT Software Engineering Notes, Vol. 30, pp. 1–7. ACM.
Novakovic, J., 2010. The Impact of Feature Selection on the Accuracy of naïve Bayes Classifier. In: 18th Telecommunications forum TELFOR, Vol. 2, pp. 1113–1116.
Olague, H.M., Etzkorn, L.H., Gholston, S., Quattlebaum, S., 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. Software Eng. 33, 402–419.
Pai, G.J., Dugan, J.B., 2007. Empirical analysis of software fault content and fault proneness using bayesian methods. IEEE Trans. Software Eng. 33, 675–686.
Pawlak, Z., 1982. Rough sets. Int. J. Comput. Inf. Sci. 11, 341–356.
Plackett, R.L., 1983. Karl pearson and the chi-squared test. Int. Stat. Rev./Revue Internationale de Statistique 59–72.
Wagner, S., 2006. A literature survey of the quality economics of defect-detection techniques. In: Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE), pp. 194–203.
Shatnawi, R., Li, W., 2008. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. J. Syst. Software 81, 1868–1882.
Singh, Y., Kaur, A., Malhotra, R., 2009. Software fault proneness prediction using support vector machines. In: Proceedings of the World Congress on Engineering, Vol. 1, pp. 1–3.
Singh, Y., Kaur, A., Malhotra, R., 2010. Empirical validation of object-oriented metrics for predicting fault proneness models. Software Qual. J. 18, 3–35.
Suykens, J.A., De Brabanter, J., Lukas, L., Vandewalle, J., 2002. Weighted least squares support vector machines: robustness and sparse approximation. Neurocomputing 48, 85–105.
Mende, T., Koschke, R., 2009. Revisiting the evaluation of defect prediction models. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering (PROMISE), pp. 1–10.
Mende, T., Koschke, R., 2010. Effort-aware defect prediction models. In: Proceedings of 14th IEEE European Conference on Software Maintenance and Re-engineering (CSMR), pp. 107–116.
Tang, M.-H., Kao, M.-H., Chen, M.-H., 1999. An empirical study on object-oriented metrics. In: Software Metrics Symposium, 1999. Proceedings. Sixth International, IEEE, pp. 242–249.
Tomaszewski, P., Håkansson, J., Grahn, H., Lundberg, L., 2007. Statistical models vs. expert estimation for fault prediction in modified code–an industrial case study. J. Syst. Software 80, 1227–1238.
Wang, D., Romagnoli, J., 2005. Robust multi-scale principal components analysis with applications to process monitoring. J. Process Control 15, 869–882.
Weyuker, E.J., 1988. Evaluating software complexity measures. IEEE Trans. Software Eng. 14, 1357–1365.
Zhou, Y., Leung, H., 2006. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Trans. Software Eng. 32, 771–789.
Zhou, Y., Xu, B., Leung, H., 2010. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. J. Syst. Software 83, 660–674.

Lov Kumar did his B.tech in Computer Science and Engineering from C.V. Raman College of Engineering, Bhubaneswar and M.tech in Computer Science and Engineering with specialization of Software Engineering at NIT, Rourkela. Currently he is pursing his Ph. D. in Computer Science and Engineering with specialization of Software Engineering at NIT, Rourkela. His areas of interest are in Quality estimation, web service, Object-Oriented programing and service oriented programing.

Sai Krishna Sripada is persuing master's degree in Computer Science and Engineering with specialization of Software Engineering from IIIT,Hyderabad, India in 2014. His areas of interest is mostly on software engineering, Gamification and software defined networking. He is currently an intern with ABB Corporate research centre India, working on dependency analysis.

Ashish Sureka is a Principal Researcher at ABB Corporate Research Center (India). He was a Faculty Member (at IIIT-Delhi) from July 2009 to October 2014 and a visiting researcher at Siemens Corporate Research from August 2014 to July 2015. His current research interests are in the area of Mining Software Repositories, Software Analytics, and Social Media Analytics. He graduated with an MS and PhD degree in Computer Science from North Carolina State University (NCSU) in May 2002 and May 2005 respectively. He has worked at IBM Research Labs in USA, Siemens Research Lab (India) and was a Senior Research Associate at the R&D Unit of Infosys Technologies Limited before joining IIIT-D as a Faculty Member in July 2009. He has received research grants from Department of Information Technology (DIT, Government of India), Confederation of Indian Industry (CII) and Department of Science and Technology (DST, Government of India). He has published several research papers in international conferences and journals, graduated several PhD and MTech students, organized workshops co-located with conferences, and received best paper awards. He was selected for ACM India Eminent Speaker Program. He holds seven granted US patents.

S K Rath is a Professor in the Department of Computer Science and Engineering, NIT Rourkela since 1988. His research interests are in Software Engineering, System Engineering, Bioinformatics, and Management. He has published a large number of papers in international journals and conferences in these areas. He is a Senior Member of the IEEE, USA and ACM, USA and Petri Net Society, Germany.