Applied Soft Computing 49 (2016) 1034–1050

Contents lists available at ScienceDirect

Applied Soft Computing

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

An empirical framework for defect prediction using machine learning techniques with Android software

Ruchika Malhotra

Department of Software Engineering, Delhi Technological University, Bawana Road, Delhi, India

ARTICLE INFO

Article history:
Received 30 November 2015
Received in revised form 20 March 2016
Accepted 26 April 2016
Available online 4 May 2016

Keywords:
Object-oriented metrics
Machine-learning
Software defect proneness
Statistical tests
Inter-release validation

ABSTRACT

Context: Software defect prediction is important for identification of defect-prone parts of a software. Defect prediction models can be developed using software metrics in combination with defect data for predicting defective classes. Various studies have been conducted to find the relationship between software metrics and defect proneness, but there are few studies that statistically determine the effectiveness of the results.

Objective: The main objectives of the study are (i) comparison of the machine-learning techniques using data sets obtained from popular open source software (ii) use of appropriate performance measures for measuring the performance of defect prediction models (iii) use of statistical tests for effective comparison of machine-learning techniques and (iv) validation of models over different releases of data sets.

Method: In this study we use object-oriented metrics for predicting defective classes using 18 machine-learning techniques. The proposed framework has been applied to seven application packages of well known, widely used Android operating system viz. Contact, MMS, Bluetooth, Email, Calendar, Gallery2 and Telephony. The results are validated using 10-fold and inter-release validation methods. The reliability and significance of the results are evaluated using statistical test and post-hoc analysis.

Results: The results show that the area under the curve measure for Naïve Bayes, LogitBoost and Multilayer Perceptron is above 0.7 in most of the cases. The results also depict that the difference between the ML techniques is statistically significant. However, it is also proved that the Support Vector Machines based techniques such as Support Vector Machines and voted perceptron do not possess the predictive capability for predicting defects.

Conclusion: The results confirm the predictive capability of various ML techniques for developing defect prediction models. The results also confirm the superiority of one ML technique over the other ML techniques. Thus, the software engineers can use the results obtained from this study in the early phases of the software development for identifying defect-prone classes of given software.

1. Introduction

In software engineering, early detection of defective portions of the software can help the software developers and engineers in proper allocation of limited resources in testing and maintenance phases of the software development. The cost of correcting the defects increases exponentially if the defects are encountered later in the software development. The software defect prediction models can be used in the early phases of software development life cycle. Further, their use reduces the testing and maintenance time, cost and effort of the project and thus improves the quality of the software [1].

In order to increase the level of automation while developing software, models are effective and are gaining importance. Moreover, defect prediction models can be developed by using software metrics in conjunction with defect data obtained from historical repositories. The models can be trained using the historical releases of the same software and validated either on the same release or the subsequent releases of the software.

There are several machine-learning (ML) techniques proposed in the literature including neural networks, Support Vector Machines, ensemble learners and decision trees. But, it is difficult to establish the superiority of one ML technique over the other techniques using multiple data sets. Hence, more and more stud-

E-mail address: ruchikamalhotra2004@yahoo.com

http://dx.doi.org/10.1016/j.asoc.2016.04.032
1568-4946/

ies should be performed in order to draw well-formed, widely acceptable and generalized conclusions based on the experimental evidence gathered from the obtained results [1]. The results from the empirical studies will help to improve, refute and validate the results obtained from the past studies.

An effective empirical framework for the development of the prediction models should focus on the following issues: (i) use of appropriate and large number data sets (ii) performance of the predicted models assessed using appropriate performance measures (iii) reliability of the results using statistical tests (iv) validating the predicted models on data different from which they are trained. Lessman et al. [3] and Malhotra [4,5] observed that, the size of the study, performance measures used to assess the predicted model performance and the statistical tests to confirm the reliability of the results are three very important factors that need to be considered while conducting an empirical study. Also, the data sets available in software engineering research are scarce. There are only few studies that use statistical tests to analyze the suitability and validity of the results in literature. The evaluation of effectiveness of the performance of the predicted models is very crucial for the assessment of practical application of any defect prediction models. The reliability of empirical experiments can only be confirmed using the statistical tests [6]. Certain studies have pointed out that the statistical significance of the obtained results is rarely examined (Menzies et al. [7], Myrtviet et al. [8]). Further, previous studies have validated the developed models using the same data, on which they were trained.

In this work, we develop defect prediction models using object-oriented (OO) metrics over multiple application packages of Android operating system, open source software. Specifically, we address the following research issues in this work: (i) low repeatability of empirical studies, (ii) less usage of statistical tests for comparing the effectiveness of different models, and (iii) non-assessment of results on different releases of the software. This study will present an empirical framework of defect prediction models using 18 ML techniques, which will yield unbiased, accurate and repeatable results. The outcome of this research is assessed over various releases of seven application packages of Android software available in the Google code repository—Contact, MMS, Bluetooth, Email, Calendar, Gallery2 and Telephony.

As there is less use of statistical tests in the literature for statistically determining the comparative difference between predictive performances of developed models. Hence, after the models are generated, we will apply statistical techniques (such as Friedman test) to statistically determine whether there is a statistical difference between the performances of different ML techniques. We will also perform post-hoc analysis (using Nemenyi test) to evaluate the pairwise comparison amongst the results of different techniques. The results are evaluated using area under the curve (AUC) obtained from Receiver Operating Characteristics (ROC) analysis. Thus, the following research questions are addressed in this work:

- RQ1: What is the overall predictive capability of various ML techniques on seven application packages of Android software using 10-fold validation?

In this question we validate the results of predicted models using 10-fold validation with the help of various performance measures. The overall capability of the 18 ML techniques is assessed based on the results obtained using seven application packages of Android software.

- RQ2: What is the performance of defect prediction models when inter-release validation is carried out?

The performance of defect prediction models is validated using inter-release validation in this question. The results are evaluated using the AUC measure obtained using ROC analysis.

- RQ3: Is the performance of defect prediction models validated using inter-release validation comparable to 10-fold validation and is it statistically different than 10-fold validation?

We compare and assess the performance of models validated using inter-release with models validated using 10-fold validation. We determine the statistical difference between the results of inter-release validation and 10-fold validation for defect prediction using Wilcoxon test.

- RQ4: Which are the best and worst ML techniques for defect prediction using OO metrics?

The best and worst ML techniques are determined using the results of both 10-fold and inter-release validation over the seven application packages of Android software. These results are based on AUC measure and derived using the statistical test, Friedman.

- RQ5: Which pairs of ML techniques are statistically different from each other for defect prediction?

In this research question, we determine the pairs of ML techniques that are statistically different than each other. The results are based on post-hoc analysis using Nemenyi test.

The initial results carried out using one application package of Android software following the proposed approach are reported in Malhotra et al. [9]. Now, we present a major extension of the preliminary results presented in our previous study by evaluating the ML techniques over six additional application packages of Android software. The results of the previous study were not generalizable as they were only based on one application package of Android software. Also, we carry out post-hoc analysis using Nemenyi test to determine the effectiveness of the results. We also determine the statistical significance of inter-release validation and compare the ML techniques on the basis of inter-release validation. There is no study to the best of the authors' knowledge that extensively compares and assesses the performance of ML techniques to analyze the relationship between OO metrics and defect prediction using statistical tests. Hence, the main contributions of this paper are summarized below:

(1) An extensive comparison of 18 popular ML techniques in the context of defect prediction.
(2) The use of data collected from seven application packages over multiple releases of widely used Android software.
(3) Statistical analysis of the obtained results for comparison of ML techniques.
(4) An inter-release validation of models developed in order to obtain unbiased and generalized results.

The rest of the paper is organized as follows: Section 2 summarizes the related work and Section 3 describes the empirical research framework followed in this paper. Section 4 presents the research methodology and Section 5 provides the answers to the research questions. The threats to validity in the current research are summarized in Section 6 and the conclusions of the work are presented in Section 7.

Table 1
Summary of studies using ML techniques using OO metrics.

Study Refs.	Data type	Performance measures	Statistical test	Validation type
Briand 2002 [24]	Proprietary	Completeness, correctness	–	10-fold
Gyimothy 2005 [25]	Open source	Accuracy, correctness, completeness	–	10-fold, inter-release
Zhou and Leung 2006 [27]	Proprietary	Accuracy, precision, completeness	–	10-fold
Kanmani 2007 [29]	Academic	Correctness, completeness	–	Random training and testing
Pai 2007 [30]	Proprietary	Specificity, sensitivity, accuracy	–	10-fold
Catal 2007 [31]	Proprietary	Recall, accuracy, precision, gmean, f-measure	–	10-fold
Singh 2009 [34]	Proprietary	Sensitivity, specificity, completeness, Accuracy, AUC	–	10-fold
Malhotra 2010 [28]	Open source	Sensitivity, specificity, completeness, accuracy, AUC	–	10-fold
Zhou 2010 [35]	Open source	AUC	Statistical test based on Mann–Whitney statistic [46]	10-fold, inter-release
Arisholm 2010 [32]	Proprietary	AUC, accuracy, precision, recall	Wilcoxon	Hold-out
Carvalho 2010 [33]	Proprietary	AUC, accuracy, precision, recall, f-measure	Wilcoxon	10-fold
Martino 2011[36]	Open source	Accuracy, recall, precision, f-measure	–	Hold-out, 10-fold
Azar 2011 [38]	Open source	Accuracy	Wilcoxon	10-fold
Malhotra 2011 [39]	Open source	Sensitivity, specificity, AUC, Accuracy	–	10-fold
Malhotra 2012 [40]	Open source	Sensitivity, specificity, AUC, Accuracy	–	10-fold
Okutan2012 [37]	Open source	AUC	*t*-test	10-fold
Yang 2012 [41]	Open source	Recall, precision, f-measure	*t*-test	5-fold cross validation
Li 2012 [42]	Open source	f-measure, balance	Mann Whitney *U*-test	Random sampling validation
Canfora 2013 [43]	Proprietary	Precision, Recall	Wilcoxon	10-fold, inter-project
Harman 2014 [44]	Open source	G-mean, recall	Wilcoxon with Holm–Bonferroni correction	Inter-release

2. Related work

There are number of studies in literature that analyze relationship between OO metrics and fault proneness using a statistical technique, logistic regression (Basili et al. [10]; Binkley and Schach [11]; Briand et al. [12]; Briandet al. [13]; Cartwright and Shepperd [14]; Chidamber et al. [15]; Harrisonet al. [16]; El Emam et al. [17]; Li and Henry [18]; Yu et al. [19]; Aggarwal et al. [20]; Li and Shatnawi [21]; Olague et al. [22]; Aggarwal et al. [23]).

There are various studies that analyze the relationship between OO metrics and defect proneness that evaluate ML techniques. Briand et al. [24] validated Multivariate Adaptive Regression Splines (MARS) technique for the construction of defect proneness models and concluded that models developed using MARS technique outperformed logistic regression (LR) technique. Gyimothy et al. [25] used Mozilla data set to validate the results to evaluate the relationship of Chidamber & Kemerer (CK) metrics [26] with defect prediction using LR and ML techniques (decision tree and neural network). The results indicated number of children (NOC) as an insignificant metric for predicting defects and showed low accuracy of models developed using ML techniques. Zhou and Leung [27] validated the results using NASA data set KC1 (written in C++ language) with respect to two severity levels of defects—high and low. They employed statistical technique (LR) and three ML techniques (Naïve Bayes, Random Forest and NNge) to evaluate the effectiveness of CK metrics for defect prediction with regard to the high and low defect severity levels. The results showed that the defect prediction models developed at high severity level of defects had less prediction accuracy. The values of performance measures for defect prediction models were low with regard to ML techniques. Singh et al. [1] validated the results on the same NASA data set KC1 with three severity levels of defects—high, medium and low. They used logistic regression, decision tree and artificial neural network techniques and used ROC analysis to evaluate and assess the performance of developed defect prediction models. The results of this study depicted better performance as compared to the results obtained by Malhotra and Singh [28], particularly with regard to the models developed at high severity level of defects. The performance of the models developed using the chosen ML techniques were also high. In another study by Kanmani et al. [29], the Support Vector Machines (SVM) technique was evaluated using the public domain NASA KC1 data set at various severity levels of defects.

Pai and Dugan [30] evaluated an approach that was based on neural networks and the results were based on the software system written in the Java programming language. The study illustrated that the performance of models developed using neural networks was high as compared to the models developed using the statistical technique. Catal et al. [31] assessed the Bayesian approach and the results were validated using the NASA KC1 data set. The results showed that the models developed using the Bayesian techniques were comparable to the existing techniques for defect prediction. In another study conducted by Arisholm et al. [32], Artificial Immune Recognition System was evaluated using NASA KC1 data set. Carvalho et al. [33] used Java Telecom system for comparing various variations of decision tree techniques. In addition to the decision tree technique (C4.5), they also evaluated neural networks, SVM and LR techniques and concluded that the model developed using the C4.5 technique showed high accuracy. The predictive ability of MultiObjective Particle Swarm Optimization technique and SVM technique (using Jedit software) were evaluated by Carvalho et al. [33] and Singh et al. [34], respectively. Zhou et al. [35] proposed the usefulness of odds ratio in predicting defective classes using CK and complexity measures. Martino et al. [36] developed model using SVM technique and the model was configured using Genetic algorithm for prediction of defective classes with regard to OO metrics. Okutan et al. [37] assessed the relationship between CK metrics and defect proneness using the Bayesian approaches. Azar and Vybihal [38] developed defect prediction models using Ant Colony Optimization (ACO) technique and concluded these models were better than decision tree (C4.5) and random guessing techniques. Studies by Malhotra and Singh [39] and Malhotra and Jain [40] developed defect prediction models on open-source data sets using 10-fold cross validation. However, Yang et al. [41] used five-fold cross validation technique for developing defect prediction models. Canfora et al. [43] developed defect prediction models using both 10-fold and inter-project results. Harman et al. [44] developed temporal defect prediction models using eight releases of open-source Hadoop software.

Table 1 provides the information about the use of data set type (Academic, proprietary or open-source), statistical test, performance measures and the type of validation in the literature. It

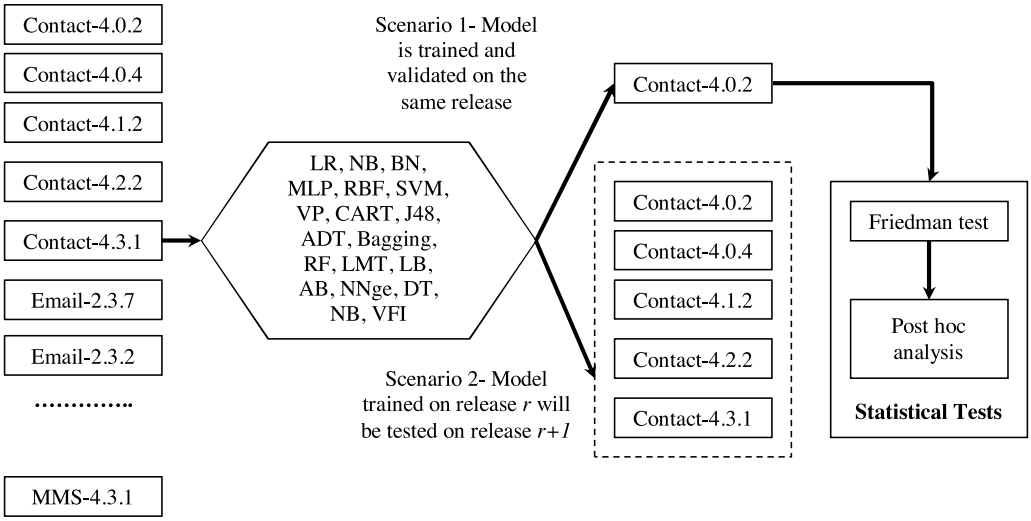

Fig. 1. Framework for defect prediction—an example using Contact.

can be seen from the table that out of 21 studies only 5 studies use inter-release validation. Though two studies in literature use *t*-test, it is not effective, as it requires a lot of pre-assumptions about the underlying data such as the data should follow normal distribution etc. Thus, the parametric nature of the test limits its applicability as it has very stringent assumptions. However, a commonly used test in literature is Wilcoxon test for ascertaining the differences amongst the performance of different techniques, but the use of Wilcoxon test without Bonferroni correction is not advisable, as it does not take into account family-wise error. The use of a *p*-value in multiple comparisons only signifies the probability of an error of a certain comparison but ignores the remaining comparisons which belong to the same family. It is important to adjust the *p*-value otherwise the probability of committing Type I error is quite high. Demšar recommends the use of Nemenyi test as a post-hoc test for comparing the pairwise differences amongst the performance of different techniques [45] as it reduces family-wise error. It is also advocated by Lessmann et al. [3].

Although, the above studies use ML techniques to evaluate the relationship between OO metrics and defect prediction, however there are no studies to the best of the author's knowledge that extensively compares the ML techniques to evaluate the relationship between OO metrics and defect prediction. In this study we not only compare and assess the effectiveness of the 18 ML techniques for defect prediction, we also use appropriate statistical tests to evaluate the statistical significant differences amongst the ML techniques. We also perform inter-release validation on various releases of seven application packages of Android software. Hence, we provide a completely generalizable framework and repeatable results in this study.

3. Empirical research framework

This section presents the empirical research framework followed in this work in order to conduct the extensive comparison and assessment of ML techniques. Fig. 1 presents the framework for development of defect prediction models for application package 'contact' of Android software. The figure depicts that various releases of application packages of Android software are analyzed using 18 ML techniques. It also shows that two types of validations (scenario 1 and 2) are carried out. In scenario 1, each version of contact is used and the predicted model is trained and validated on the same release of the software. In scenario 2, the model is trained on release *r* and validated on the subsequent release *r*+1. For example, model is trained using Contact-4.0.2 and validated on Contact-4.0.4. Similarly, model is trained using Contact-4.0.4 and validated on Contact-4.1.2. The procedure is repeated over all the releases of seven application packages of Android software.

The subsequent sub sections describe the procedure of data collection and independent and dependent variables used in this study.

3.1. Data collection procedure

Smartphones have revolutionized the modern day scenario as they can easily perform a number of tasks with increase in there computing and communication facilities [47]. Recent studies like Espada et al. [48], focus on improving user experience for smartphones. In this study, the empirical results are computed using various application packages of widely used Android operating system, which is popularly used in smartphones. The Android OS has six releases with three code names namely—Ginger Bread (2.3.2 and 2.3.7), Ice Cream Sandwich (4.0.2 and 4.0.4) and JellyBean (4.1.2, 4.2.2 and 4.3.1). The data has been collected using seven application packages of Android operating system namely—Contact, MMS, Bluetooth, Email, Calendar, Gallery2 and Telephony. The source code of various releases of these application packages has been obtained from Google's Git repository (https://android.googlesource.com). The characteristics of the seven application packages over various releases are given in Table 2.

Fig. 2 shows the procedure for data collection in this study. The open source repositories are used to extract the source code with OO metrics using Chidamber and Kemerer Java Metrics (CKJM) tool. The defects are collected from open sources bug repository, GIT. Defect Collection and Reporting System (DCRS) tool developed at the Delhi Technological University [49] in the Java language is used for collection of the defects. After following the procedure described below, finally the data is collected for each application package of Android operating system over various releases.

Table 2 presents the total classes, LOC, number of defects and percentage of defective classes in each release of the seven application packages of Android software. The steps stated below are followed to collect the defects from the open source software defect logs [9,49]:

(1) *Step 1: obtain change logs*: The DCRS obtains change logs from Git repository of two predetermined consecutive releases. The change may occur due to defect occurrence, improvement in

Table 2
Summary of various releases of Android software over application packages.

Data set name	Version	Total classes	Total LOC	Defective #	Defective %
Gallery2	4.0.2	305	18,853	24	7.86
	4.0.4	310	19,290	69	22.25
	4.1.2	330	20,446	170	51.51
	4.2.2	374	28,223	93	24.86
	4.3.1	647	50,168	130	20.09
Contact	2.3.7	85	7788	29	34.11
	4.0.2	325	22,722	19	5.84
	4.0.4	331	22,834	107	32.32
	4.1.2	357	24,644	35	9.80
	4.2.2	375	25,860	14	3.73
	4.3.1	210	14,807	98	46.66
Email	2.3.2	385	25,760	21	5.45
	2.3.7	394	26,839	41	10.41
	4.0.2	469	33,730	17	3.62
	4.0.4	624	43,147	274	43.91
	4.1.2	475	34,255	61	12.84
	4.2.2	472	34,025	7	1.48
	4.3.1	472	34,037	77	16.31
MMS	2.3.7	195	13,157	54	27.69
	4.0.2	201	13,538	11	5.47
	4.0.4	206	13,804	68	33.01
	4.1.2	223	14,759	42	18.83
	4.2.2	225	14,932	12	5.33
	4.3.1	224	14,915	23	10.27
Calendar	4.0.2	77	8042	22	28.57
	4.0.4	78	8216	45	57.69
	4.1.2	86	9285	12	13.95
	4.2.2	88	9465	41	46.59
Bluetooth	4.1.2	39	2517	15	38.46
	4.2.2	63	6246	10	15.87
	4.3.1	72	7550	13	18.05
Telephony	4.2.2	249	30,325	137	55.02
	4.3.1	224	28,331	154	68.75

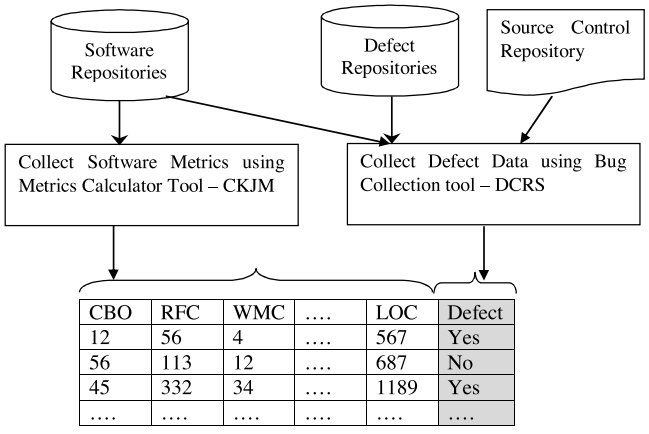

Fig. 2. The procedure for defect data collection.

quality of the software, or new functionality add-on. A single change record represents a change. A change record includes unique identifier, timestamp of committing, description of change (optional), and a list of modified lines of source code. The change logs of the available releases (Android 2.3.2, Android 2.3.7, Android 4.0.2, Android 4.0.4, Android 4.1.2, Android 4.1.2, Android 4.2.2 and Android 4.3.1) of seven application packages of Android software were obtained from the repository. The releases that did not have any change or were not available are not included in this analysis.

(2) *Step 2: process the changes related to defects*: The relevant information is extracted by preprocessing of changes related to fixed defects. The change log consists of information related to defects from the beginning (i.e., the first release of the software). Hence, preprocessing is required as in this study defects that are encountered from the immediate previous release to the subsequent release are only considered. For example, the defects between Contact 2.3.7 and 4.0.2 were taken into account; the defects between Contact 4.0.2 and 4.0.4 were taken into account and so on. The records of defects are retrieved by searching defect-Ids and if defect-Ids are not found to be present then keywords "issue"; "fault"; "bug" or "defect" are searched in the given description of the change.

(3) *Step 3: calculating OO metrics with respect to each Java file*: The open source CKJM tool (http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html) is used for computing the values of OO metrics (indicated in Section 3.2). The Java source code files are only considered in this step and all the other types of files are ignored. The OO metrics were calculated for each of the Java class with respect to each release of the specified package.

(4) *Step 4: associating or linking defects to the Java classes*: The collected defects in step 2 are finally linked to the Java classes in the source code. Hence, each class found in step 3 is related to defect count corresponding to that class. The classes in each release of the application package of the Android software are related with the defects fixed from the current release that is being considered to the immediate successive release (i.e., defects obtained from current release to next succeeding release).

3.2. Variables

The independent variables (used in this study) are the OO metrics. The metrics used in this project, are the ones, which are most commonly used by various researchers, to account for software

characteristics such as size, coupling, cohesion and inheritance. In addition to the metrics proposed by Chidamber and Kemerer [26] we also used the metrics proposed by Henderson-Sellers [50] and Bansiya and Davis [51]. The dependent variable of the study is defect proneness, which is defined as probability of occurrence of defect in a class [1]. The OO metrics used in this work are summarized below:

Chidamber & Kemerer metrics [26]:

- Coupling between objects (CBO): counts number of other classes to which a class is coupled with.
- Response for a class (RFC): counts number of external and internal classes.
- Lack of cohesion in methods (LCOM): measures dissimilarity of methods in a class.
- Number of children (NOC): counts number of descendants of a class.
- Depth of inheritance (DIT): measures number of ancestor classes.
- Weighted methods per class (WMC): counts number of methods in a class weighted by complexity.

Henderson-Sellers metric [50]:

- Lack of cohesion in methods (LCOM3): $\mathrm{LCOM3} = \frac{\frac{1}{N}\sum_{i=1}^{n}\mu(D_i) - m}{1-m}$.

Martin's metrics [52]:

- Afferent couplings (Ca): counts how many other classes use a given class.
- Efferent Couplings (Ce): counts the number of classes, a class is dependent upon.

QMOOD metrics suite (Bansiya and Davis [51])

- Number of public methods (NPM): counts the number of public methods in a class.
- Data Access Metric (DAM): defined as the number of private methods divided by the total number of methods.
- Measure of Aggregation (MOA): counts the number of abstract data types in a class.
- Measure of Functional Abstraction (MFA): defined as the number of inherited methods divided by total number of methods accessible by its member functions.
- Cohesion Among Methods of Class (CAM): based upon the parameters in a method.

Tang et al. metrics [53]:

- Inheritance Coupling (IC): it is based upon inheritance-based coupling.
- Coupling Between Methods (CBM): counts the newly added functions with which inherited based methods are coupled.
- Average Method Complexity (AMC): counts average size of method in a class.

LOC: it counts the number of lines of source code.

4. Experimental setup

In order to answer the research questions formed in this work the following steps must be conducted in an empirical framework:

1. Selection of feature section technique as described in Section 4.1.
2. Selection of appropriate data analysis techniques.
3. Selection of performance measures that can deal with imbalanced data sets for evaluating the predictive capability of data analysis techniques selected at step 2 (refer Section 4.3).
4. Use of efficient validation methods to determine the true applicability of the models predicted.
5. Selection of appropriate statistical tests to determine the superiority of one ML technique over the other ML techniques.

After the above steps have been conducted, an empirical methodology is followed that includes (i) feature sub selection (ii) model training (iii) model validation (iv) model evaluation and (v) statistical evaluation of predicted models.

In the subsequent sections, we describe the feature selection method, ML techniques, performance measures, validation methods and statistical tests used to carry out the empirical study.

4.1. Feature selection method

In this work, 18 OO metrics have been used for defect prediction. It is essential to select relevant and important attributes out of these OO metrics so that only uncorrelated and relevant OO metrics are included in the construction of defect prediction models. In order to achieve this aim, we used Correlation based Feature Selection (CFS) method for feature selection. The CFS method is simple, fast and is widely used with ML techniques for selecting features [54]. In this method a subset of attributes is selected by searching the best combination of attributes [54]. The CFS method is a supervised technique that is based on the evaluation of correlation between independent variables and the dependent variable. The method follows the principle for selecting the independent variables that are highly correlated with the dependent variable and uncorrelated with each other [54]. Thus, in this case OO metrics that are highly correlated with defect prediction are selected and the OO metrics that are highly correlated with each other are avoided. Hence the CFS method removes irrelevant and redundant OO metrics.

4.2. ML techniques

We have carefully selected the ML techniques from each of the categories as shown in Table 3.

4.3. Performance measures

In order to evaluate the learning process, various performance measures need to be analyzed which indicate the effectiveness of the developed defect prediction models. These performance measures include specificity, sensitivity, and Area Under the Receiver Operating Characteristics (ROC) Curve amongst others. A high value for these performance measures denotes a good prediction model.

Sensitivity: It is defined as the ratio of correctly predicted defect-prone classes to the total number of classes.

Specificity: It is defined as the ratio of correctly predicted non defect-prone classes to the total number of classes.

The data having disproportionate ratio of defective and not defective classes is often known as imbalanced data. Given the imbalanced nature of the data sets, the ROC analysis is a commonly used performance measure [7,55]. The ROC analysis is one of the most popular techniques used to determine the accurateness of the prediction model. It determines how well the model has worked on the validation data [55]. It is a plot of sensitivity and '1-specificity' at various cutoff points. Here, sensitivity is plotted on *y*-axis and '1-specificity' is plotted on *x*-axis. In ROC analysis, the optimal cutoff

Table 3
Description of ML techniques used in the study.

Machine learning technique		Description
Statistical classifier		
LR	Logistic regression	LR uses statistical method to estimate the response. NB assumes interdependence of various attributes and uses Bayes theorem. BN uses conditional probability with K2 as a search technique.
NB	NaiveBayes	
BN	Bayesian networks	
Neural networks		
MLP	Multilayer perceptron	MLP uses two separate passes i.e. backward and forward which is formally known as the back propagation method for learning. A learning rate of 0.3 with 500 epochs was used. RBF uses Gaussian radial basis function which is normalized.
RBF	Radial basis function	
Support Vector Machines based		
SVM	Support Vector Machines	These techniques use the concept of a hyperplane to distinguish the data set into defective and not defective instances. In case of a nonlinear region, a kernel function is used appropriate mapping. Both SVM and VP techniques use a polynomial kernel function.
VP	Voted perceptron	
Decision tree methods		
CART	Classification & regression trees	These techniques involve successful construction of trees where attributes that are most efficient at classifying the defective classes are chosen at each node, based on the splitting criteria. C4.5 uses entropy and information gain, while CART uses Gini Index for splitting criteria.
J48	C4.5 based technique	
ADT	Alternating decision trees	
Ensemble learning		
Bag	Bagging	These techniques are meta learning in nature and use voting in order to effectively classify a class as defect or not defective. The CART technique is used by RF and the LR technique is used by LMT as base learners. Both LB and AB techniques use decision stump i.e. one level decision tree algorithm as boosting algorithms. LB is said to perform additive logistic regression.
RF	Random forest	
LMT	Logistic model trees	
LB	Logit boost	
AB	Ada boost	
Rule based learning		
NNge	Nearest neighbour with generalised exemplars	NNge is a nearest-neighbor technique that formulates if-then rules. DTNB is hybrid in nature as it incorporates the strengths of DT as well as NB techniques for all the attributes at each step.
DTNB	Decision table Naïve Bayes	
Miscellaneous		
VFI	Voting feature intervals	This technique establishes intervals specific to each attribute for classifying each class.

point is chosen that maximizes the values of both sensitivity and specificity measures.

AUC computed using ROC analysis is used to determine the accuracy of the predicted model. Carvalho et al. [33] advocated AUC to be the relevant criterion for dealing with unbalanced and noisy data as AUC is insensitive to the changes in distribution of class. He and Garcia [56] have recommended the use of AUC for dealing the issues of imbalanced data as with regard to class distributions, it provides a visual representation of the relative trade-offs between the advantages (represented by true positives) and costs (represented by false positives) of classification. Thus, in order to determine the performance of the ML techniques we used AUC measure derived using ROC analysis.

4.4. Validation methods

In any empirical study it is essential to obtain unbiased and generalizable results that can be applied to future releases and unseen similar-natured projects. Thus, it is important to validate the models predicted on different data set from which they are trained. In this study we used two validation methods—10-fold cross validation and inter-release validation. In 10-fold cross validation, the data is divided into 10 folds where each time nine parts are used for training purpose and one part is used for validation purpose. This procedure is repeated 10 times and the results from each fold are combined to produce the model validation results [57].

In inter-release validation, the model is trained using a specific release and validated on its immediate release. Thus, we used each release of application packages of Android software and constructed models and validated these models on the subsequent releases. For example, MMS-2.3.7 release was used to train the model using ML techniques and the developed model is henceforth validated on MMS-4.0.2 release.

4.5. Statistical tests

In order to statistically analyze the results, we use Friedman test, which is followed by post-hoc Nemenyi test. Both the tests are non-parametric in nature and are advocated by Demšar [45] and Lessmann et al. [3] to compare the effectiveness of various techniques. The tests used do not require the underlying data to follow normal distribution.

The Friedman [58] test allocates mean ranks to all the techniques based on their performance, which is evaluated using the AUC measure. It is used to ascertain whether the performance of different techniques differs significantly when compared with each other or not. The technique achieving the lowest rank is termed as the best performing technique. The Friedman test statistic has $n-1$ degrees of freedom and is based on chi-square distribution, where n is the total number of compared techniques. The Friedman statistic is calculated on the basis of the given formula, where k corresponds to the number of data sets, R_i is the rank allocated to the ith technique by the Friedman test.

$$\chi 2 = \frac{12}{nk(k+1)} \sum_{i=1}^{k} {R_i}^2 - 3n(k+1)$$

If Friedman test yields significant results, we perform post-hoc Nemenyi test to compare pairwise differences amongst the performance of various techniques, based on AUC performance measure. The Nemenyi test involves computation of a critical distance, which is then compared with the difference between mean ranks allocated to the two techniques using Friedman test. In case, the difference amongst mean ranks of the two compared ranks is greater than or equal to computed the critical distance, we conclude that the performance of the compared techniques differs significantly at the chosen significance level (α). The critical distance for the Nemenyi test is based on the following formula.

$$\text{Critical Distance} = q_\alpha \sqrt{\frac{n(n+1)}{6k}}$$

Here n corresponds to the number of compared techniques, k corresponds to the number of data sets on which the comparison is based and q_α is the studentized range statistic divided by $\sqrt{2}$. The study also evaluates whether there is any significant difference between the results obtained using 10-fold validation and inter-release val-

idation using Wilcoxon test. The test evaluates the significance by computing the difference amongst the results of the two validation techniques and then ranking them on the basis of their absolute values.

5. Experimental results

This section presents the results of comparison of ML techniques using OO metrics for defect prediction. The reliability of the results is assessed using the Friedman Test followed by post-hoc analysis using Nemenyi test as described in Section 4.5. The first step is data preprocessing that includes summarizing the descriptive statistics of the data sets and reducing the number of OO metrics by applying CFS method (refer Section 4.1). The effectiveness of predicted models is analyzed using 10-fold validation and inter-release validation.

5.1. Data preprocessing results

The number of OO metrics were reduced by applying the CFS method. Table 4 presents the relevant OO metrics selected after applying CFS method over all the releases of seven application packages of Android software. It can be seen that LOC, CAM, WMC, Ce and LCOM3 are commonly found relevant metrics related to defect proneness of classes in most of the releases of seven application packages of Android software.

Table 4
Metrics selected across Data sets using CFS Method.

Data set name	Release	OO metrics selected
Gallery2	4.0.2	Ce, LCOM3, LOC, MOA, CAM
	4.0.4	WMC, Ce, NPM, LOC, CAM
	4.1.2	WMC, CBO, RFC, Ce, LOC, AMC
	4.2.2	NPM, LOC, CAM
	4.3.1	Ce, NPM, LOC, MOA, CAM
Contact	2.3.7	DIT, NPM
	4.0.2	WMC, RFC, Ce, LOC, CAM
	4.0.4	Ce, LOC, DAM, CAM
	4.1.2	WMC, CBO, Ce, NPM, LOC, CAM
	4.2.2	LOC
	4.3.1	WMC, DIT, CBO, NPM, LCOM3, LOC, MOA, AMC
Email	2.3.2	NPM, LCOM3, LOC, DAM, CAM
	2.3.7	DIT, LOC, CAM
	4.0.2	CAM
	4.0.4	CAM
	4.1.2	DIT, LCOM3, DAM, CAM
	4.2.2	WMC
	4.3.1	CAM
MMS	2.3.7	Ce, LCOM3, LOC, DAM, MOA, CAM, AMC
	4.0.2	WMC, RFC, LCOM, LCOM3, DAM
	4.0.4	Ce, NPM, LCOM3, LOC, DAM, CAM
	4.1.2	Ce, CAM
	4.2.2	DAM
	4.3.1	Ce, LOC, DAM, MOA
Calendar	4.0.2	WMC, CAM
	4.0.4	WMC, DIT, RFC, LOC, MOA, CAM
	4.1.2	WMC, DIT, RFC, LOC, MOA, CAM
	4.2.2	WMC, RFC, Ce, CAM
Bluetooth	4.1.2	CBO, Ce
	4.2.2	LOC, CAM
	4.3.1	WMC, LOC, CAM
Telephony	4.2.2	WMC, CBO, LCOM3, LOC, MFA
	4.3.1	CBO, LCOM3, LOC, MFA,CAM

5.2. RQ1—what is the overall predictive capability of various ML techniques on seven application packages of Android software using 10-fold validation?

After applying CFS method, the models are predicted using 18 ML techniques and the predicted models are validated using 10-fold validation. We used ROC analysis to obtain the balance between sensitivity and specificity over multiple cutoff points. The value of sensitivity (sen.) and specificity (sp.) over appropriate cutoff point is shown in Tables 5–11 . The results are presented over various releases for each application package of Android data set viz. Contact, MMS, Bluetooth, Email, Calendar, Gallery2 and Telephony.

The performance of the defect prediction models is evaluated using AUC obtained using ROC analysis as shown in Table 12a and b. The AUC of most of the models predicted using the ML techniques is 0.7, which highlights the predictive capability of the ML techniques. The last column depicts the average AUC over all the releases of each application package of the Android software. The ML technique yielding the best AUC for a given release is depicted in bold. The results show that the model predicted using the NB, AB, RBF, Bag, ADT, MLP, LB and RF techniques have AUC greater than 0.7 corresponding to most of the releases of the application packages of Android software. Fig. 3 depicts the average AUC values of each ML technique on all the seven corresponding data sets.

From Fig. 3, it can be clearly seen that NB, LR, MLP, LB and Bagging techniques show the highest performance. The average AUC of NB

Table 5
Sensitivity and specificity values of Gallery2 data set.

Dataset	Gallery2-4.0.2		Gallery2-4.0.4		Gallery2-4.1.2		Gallery2-4.2.2		Gallery2-4.3.1	
Tech.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.
LR	70.8	72.1	71.0	70.5	68.2	67.5	60.2	57.2	61.5	64.2
NB	75.0	74.0	68.1	67.2	60.5	60.0	53.7	57.2	59.2	59.7
BN	66.7	68.7	72.5	72.6	70.6	70.6	54.8	50.2	66.2	63.4
MLP	70.8	69.8	69.6	70.1	66.5	68.7	59.1	61.2	60.7	62.2
RBF	75.0	75.0	71.0	71.0	65.8	69.3	55.9	59.7	62.3	59.3
SVM	100.0	0.0	1.4	99.6	59.4	63.7	0.0	100.0	0.0	100.0
VP	100.0	0.0	2.9	99.2	81.7	40.6	0.0	100.0	0.0	100.0
CART	50.0	39.5	63.8	55.2	69.4	70.6	51.6	54.1	0.0	100.0
J48	37.5	57.3	59.4	61.4	67.6	70.0	55.9	45.1	64.6	41.3
ADT	54.2	53.7	68.1	69.3	70.5	71.2	52.6	52.3	63.0	65.3
Bag	62.5	61.9	72.5	72.6	74.1	71.2	58.0	59.4	63.8	63.0
RF	70.8	69.8	72.5	72.2	71.7	75.6	60.2	61.9	72.3	59.3
LMT	66.7	65.8	73.9	73.9	68.2	68.7	61.2	59.4	64.6	62.2
LB	66.7	68.0	71.0	70.5	75.8	72.5	56.9	56.2	64.6	64.4
AB	62.5	63.7	68.	69.7	71.1	71.8	59.1	52.6	63.0	60.3
Nnge	16.7	95.7	46.4	90.0	66.4	63.1	35.4	79.7	36.9	87.2
DTNB	37.5	57.7	73.9	73.0	69.4	70.6	53.8	58.7	74.6	31.9
VFI	58.3	64.1	73.9	32.0	55.3	63.7	76.3	25.9	67.6	53.7

Table 6
Sensitivity and specificity values of Contact data set using 10-fold validation.

Dataset	Contact-2.3.7		Contact-4.0.2		Contact-4.0.4		Contact-4.1.2		Contact-4.2.2		Contact-4.3.1	
Tech.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.
LR	48.2	47.2	73.6	76.0	70.1	72.3	71.4	70.2	64.3	62.3	70.4	67.9
NB	51.7	50.9	73.6	71.4	72.9	69.6	74.3	73.0	57.1	60.7	68.4	66.1
BN	48.2	45.4	73.6	70.1	69.2	72.3	65.7	66.1	35.7	45.2	70.4	67.9
MLP	58.6	56.3	68.4	71.8	69.2	70.1	65.7	68.3	64.3	62.0	72.4	70.5
RBF	62.1	61.8	68.4	70.8	71.0	72.8	71.4	68.0	64.3	64.0	66.3	67.0
SVM	100.0	0.0	0.0	100.0	20.6	97.8	0.0	100.0	0.0	100.0	51.0	85.7
VP	6.8	89.1	0.0	100.0	45.8	62.5	0.0	100.0	0.0	100.0	96.9	24.1
CART	37.9	60.0	89.4	17.7	59.8	63.8	28.6	68.3	42.9	39.9	66.3	69.6
J48	37.9	60.0	89.4	15.7	59.8	66.1	34.3	59.6	42.9	39.9	71.4	71.4
ADT	65.5	60.0	73.6	76.3	69.2	70.1	57.1	57.1	64.3	62.6	70.4	70.5
Bag	58.6	63.6	68.4	66.8	70.1	71.9	65.7	64.3	50.0	66.5	74.5	70.5
RF	62.1	63.6	78.9	79.0	65.4	71.9	60.0	64.0	28.6	87.3	66.3	74.1
LMT	48.2	50.9	63.1	62.9	69.2	66.5	68.6	72.4	64.3	62.9	70.4	67.9
LB	65.5	61.8	73.6	74.4	70.1	73.2	68.6	68.0	57.1	59.6	68.4	69.6
AB	55.1	63.6	73.6	76.7	67.3	66.5	68.6	66.8	57.1	57.6	64.3	66.1
Nnge	48.2	65.4	36.8	95.7	57.9	77.2	8.6	94.4	0.0	94.5	67.3	69.6
DTNB	37.9	60.0	52.6	69.1	70.1	73.2	60.0	53.7	42.9	39.9	68.4	70.5
VFI	51.7	60.0	68.4	60.3	68.2	67.0	77.1	62.1	57.1	57.9	60.2	60.7

Table 7
Sensitivity and specificity values of MMS data set.

Dataset	MMS-2.3.7		MMS-4.0.2		MMS-4.0.4		MMS-4.1.2		MMS-4.2.2		MMS-4.3.1	
Tech.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.
LR	68.5	64.5	63.6	58.9	75.0	79.0	69.0	72.4	50.0	54.0	60.9	63.7
NB	74.1	70.9	63.6	64.2	77.9	76.8	69.0	68.0	58.3	51.6	73.9	70.6
BN	68.5	66.0	81.8	10.0	76.5	73.2	61.9	70.2	25.0	69.0	60.9	40.3
MLP	66.7	63.8	63.6	66.8	73.5	74.6	69.0	70.7	58.3	53.5	65.2	63.7
RBF	63.0	64.5	63.6	63.7	73.5	70.3	66.7	69.1	66.7	68.1	60.9	59.7
SVM	5.6	99.3	0.0	100	66.2	86.2	0.0	100.0	0.0	100.0	0.0	100.0
VP	59.3	53.2	27.3	90.5	92.6	39.1	14.3	97.8	0.0	100.0	0.0	100.0
CART	72.2	73.0	75.0	14.3	70.6	66.7	69.0	66.9	66.7	19.7	60.9	29.9
J48	59.3	61.7	45.5	49.5	72.1	69.6	57.1	66.3	66.7	19.7	47.8	45.8
ADT	70.4	68.8	63.6	67.4	76.5	78.3	64.3	64.6	50.0	54.0	60.9	64.7
Bag	72.2	69.5	72.7	61.1	73.5	73.9	66.7	66.3	66.7	59.2	65.2	66.2
RF	70.4	66.7	45.5	78.9	70.6	74.6	61.9	59.7	58.3	59.2	73.9	67.7
LMT	74.1	70.9	72.7	51.1	76.5	78.3	66.7	67.4	50.0	54.0	69.6	71.1
LB	72.2	71.6	72.7	75.8	75.0	74.6	59.5	68.0	58.3	59.6	60.9	62.2
AB	68.5	69.5	63.6	63.7	67.6	71.0	64.3	63.0	58.3	62.0	60.9	60.7
Nnge	51.9	85.8	9.1	97.4	66.2	83.3	42.9	89.5	41.7	87.8	8.7	93.5
DTNB	66.7	70.2	45.5	50.0	75.0	77.5	61.9	66.9	66.7	19.7	65.2	56.2
VFI	63.0	68.1	72.7	65.8	64.7	65.2	54.8	62.4	66.7	69.5	65.2	66.7

Table 8
Sensitivity and specificity values of Email data set using 10-fold validation.

Dataset	Email-2.3.2		Email-2.3.7		Email-4.0.2		Email-4.0.4		Email-4.1.2		Email-4.2.2		Email-4.3.1	
Tech.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.
LR	76.2	79.9	68.3	67.4	88.2	85.0	54.0	57.4	70.5	69.1	57.1	66.9	59.1	62.6
NB	81.0	81.0	73.2	72.0	82.4	85.6	56.6	54.3	68.9	66.4	57.1	61.9	58.3	60.9
BN	81.0	83.2	68.3	70.5	82.4	84.7	55.5	44.9	67.2	66.7	0.0	69.7	55.1	54.8
MLP	76.2	78.3	70.7	71.1	88.2	84.5	52.2	55.1	67.2	71.0	71.4	68.0	58.3	61.7
RBF	71.4	75.0	70.7	69.1	82.4	86.1	54.0	55.7	68.9	69.1	57.1	55.5	58.3	61.4
SVM	0.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	100.0
VP	0.0	100.0	0.0	100.0	0.0	100.0	4.7	94.6	0.0	100.0	0.0	100.0	0.0	100.0
CART	85.7	9.9	29.3	69.4	17.6	69.7	49.3	56.9	83.6	16.7	0.0	77.8	56.7	44.6
J48	66.7	64.3	29.3	69.4	17.6	69.7	49.6	53.4	50.8	55.3	0.0	69.7	48.8	49.3
ADT	76.2	79.7	68.3	71.4	76.5	78.1	51.8	52.6	70.5	69.3	57.1	60.2	57.5	50.7
Bag	76.2	78.0	70.7	69.1	82.4	86.1	48.5	52.3	72.1	71.3	71.4	55.5	55.9	57.7
RF	71.4	79.9	65.9	64.9	29.4	87.6	49.6	46.3	62.3	76.1	28.6	82.2	52.0	55.7
LMT	76.2	77.2	56.1	57.8	88.2	82.1	55.8	52.9	65.6	66.4	71.4	69.5	56.7	59.4
LB	71.4	74.7	70.7	72.2	82.4	84.7	48.9	49.4	73.8	70.5	57.1	64.9	55.9	54.5
AB	81.0	78.8	68.3	71.1	82.4	84.7	48.2	53.1	65.6	68.4	57.1	45.4	58.3	55.1
NNge	9.5	97.3	12.2	89.8	0.0	96.2	41.6	56.9	31.1	91.1	28.6	97.8	30.7	78.0
DTNB	38.1	59.3	68.3	30.0	17.6	69.7	49.6	53.4	68.9	35.0	0.0	69.7	38.6	59.4
VFI	66.7	69.2	68.3	55.8	70.6	55.1	47.1	53.4	72.1	64.7	42.9	54.0	50.4	47.5

Table 9
Sensitivity and specificity values of Calendar data set using 10-fold validation.

Dataset	Calendar-4.0.2		Calendar-4.0.4		Calendar-4.1.2		Calendar-4.2.2	
Tech.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.
LR	63.6	67.3	75.6	78.8	66.7	73.0	68.3	66
NB	63.6	67.3	73.3	75.8	58.3	64.9	63.4	59.6
BN	59.1	58.2	64.4	63.6	33.3	60.8	53.7	51.1
MLP	63.6	67.3	75.6	72.7	58.3	62.2	68.3	66
RBF	72.7	70.9	64.4	66.7	75.0	78.4	63.4	61.7
SVM	4.5	100	88.9	36.4	0.0	100	34.1	87.2
VP	50.0	52.7	0.0	100.0	0.0	98.6	92.7	8.5
CART	54.5	52.7	73.3	66.7	50.0	47.3	58.5	59.6
J48	63.6	60	55.6	57.6	50.0	47.3	61	57.4
ADT	59.1	60	68.9	66.7	66.7	64.9	51.2	53.2
Bag	68.2	69.1	73.3	75.8	66.7	67.6	63.4	61.7
RF	63.6	61.8	71.1	72.7	66.7	63.5	53.7	61.7
LMT	63.6	65.5	73.3	75.8	75.0	78.4	63.4	59.6
LB	54.5	58.2	71.1	72.7	66.7	66.2	58.5	61.7
AB	54.5	60	66.7	66.7	66.7	70.3	68.3	66
NNge	54.5	81.8	73.3	60.6	33.3	95.9	58.5	59.6
DTNB	59.1	52.7	57.8	72.7	50.0	45.9	58.5	53.2
VFI	40.9	50.9	66.7	66.7	66.7	59.5	63.4	59.6

Table 10
Sensitivity and specificity values of Bluetooth data set using 10-fold validation.

Dataset	Bluetooth-4.1.2		Bluetooth-4.2.2		Bluetooth-4.3.1	
Tech.	Sen.	Sp.	Sen.	Sp.	Sen.	Sp.
LR	46.7	45.8	70.0	73.6	76.9	79.7
NB	53.3	54.2	70.0	71.7	76.9	79.7
BN	33.3	41.7	70.0	67.9	69.2	72.9
MLP	40.0	45.8	70.0	66.0	76.9	79.7
RBF	60.0	58.3	70.0	67.9	69.2	72.9
SVM	26.7	100.0	0.0	100.0	23.1	100
VP	66.7	58.3	10.0	90.6	46.2	55.9
CART	66.7	54.2	30.0	66.0	46.2	45.8
J48	46.7	41.7	30.0	64.2	53.8	54.2
ADT	53.3	50.0	60.0	62.3	61.5	64.4
Bag	46.7	54.2	70.0	64.2	69.2	69.5
RF	53.3	54.2	70.0	73.6	61.5	64.4
LMT	40.0	37.5	70.0	64.2	76.9	79.7
LB	53.3	50.0	70.0	64.2	69.2	72.9
AB	33.3	29.2	70.0	67.9	69.2	64.4
NNge	33.3	45.8	30.0	86.8	23.1	93.2
DTNB	53.3	54.2	20.0	77.4	69.2	74.6
VFI	46.7	50.0	80.0	71.7	61.5	62.7

Table 11
Sensitivity and specificity values of Telephony data set using 10-fold validation.

Dataset	Telephony-4.2.2		Telephony-4.3.1	
Tech.	Sen.	Sp.	Sen.	Sp.
LR	68.6	68.8	78.6	78.6
NB	70.1	67.9	75.3	75.7
BN	68.6	67.0	75.3	74.3
MLP	67.2	67.9	77.9	78.6
RBF	62.0	63.4	72.1	72.9
SVM	89.8	40.4	90.9	61.4
VP	0.0	100.0	0.0	100.0
CART	68.6	69.6	72.1	65.7
J48	68.6	68.8	77.3	75.7
ADT	70.1	67.0	74.0	78.6
Bag	65.7	65.2	77.3	77.1
RF	59.9	59.8	79.2	78.6
LMT	67.9	67.9	76.0	75.7
LB	73.0	72.3	76.0	75.7
AB	71.5	68.8	70.8	68.6
NNge	56.9	58.0	85.7	60.0
DTNB	68.6	68.8	72.1	72.9
VFI	69.3	62.5	79.2	74.3

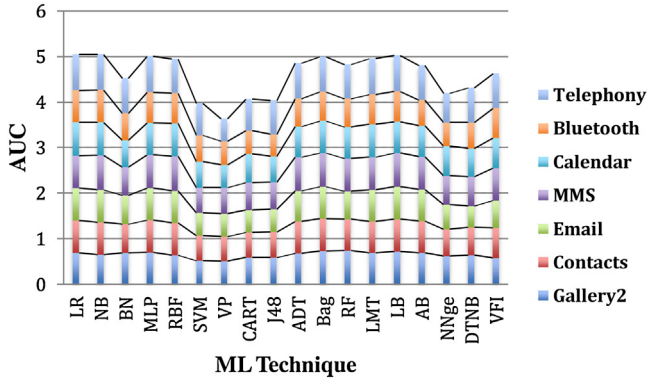

Fig. 3. Average AUC values of each ML technique on corresponding data sets using 10-fold validation.

technique with respect to Gallery2, Contact, Email, MMS, Calendar, Bluetooth and Telephony data sets is 0.65, 0.71, 0.71, 0.76, 0.73, 0.71 and 0.78, respectively. The average AUC of MLP technique is 0.70, 0.71, 0.70, 0.73, 0.70, 0.68 and 0.79 with respect to Gallery2, Contact, Email, MMS, Calendar, Bluetooth and Telephony data sets. Similarly, LB technique shows good values of average AUC over all the Android data sets (between 0.69 and 0.79).

Fig. 3 also depicts that defect prediction models constructed using VP technique over seven application packages of Android data sets showed the lowest performance. The results show that the average AUC of VP technique is around 0.5 for all the data sets that proves the poor or no predictive capability of VP technique. SVM and decision tree based techniques (CART and J48) follows VP technique.

5.3. RQ2—what is the performance of defect prediction models when inter-release validation is carried out?

The performance of defect prediction models was also evaluated using inter-release validation (refer Section 4.5). In order to do so, a model developed using a specific release was validated on its immediate subsequent release. The procedure for inter-release validation was performed for all the data sets. Table 15a and b present the results of inter-release validation in terms of AUC performance measure. As mentioned earlier that this measure has been recommended in literature for dealing with imbalanced data.

The tables show that the average AUC values for Gallery2 data set are in the range 0.50–0.65, with the MLP technique showing the best AUC value. Similarly, the Contact data set show average inter-release AUC values as 0.53–0.72, with NB, MLP, LMT and LR techniques showing the best AUC values for the data set. The inter-release AUC values for Email and MMS data sets are in the range 0.50–0.73 and 0.58–0.78, respectively. The NB and MLP techniques show the best inter-release results on Email data set and the Bagging technique on the MMS data set. An average inter-release AUC of 0.55–0.77 was obtained on Calendar data set with ADT and LMT showing the best results. The inter-release AUC results for Bluetooth and Telephony data sets were in the range 0.55–0.79 and 0.59–0.90 for Bluetooth and Telephony data sets respectively. The LMT technique was the best performing technique for Bluetooth data set and both the LMT and Bag techniques gave an AUC of 0.90 for telephony data set. It is observed from the tables that the models developed using NB, MLP, RBF, ADT, LMT, Bag, LB and AB techniques showed an AUC of 0.7 in most of the data sets. This reinforces the effective capability of ML techniques for developing defect prediction models.

Fig. 4 depicts the cumulative average AUC over all the ML techniques for all application packages of Android software. The figure

Table 12
AUC values of data sets using 10-fold validation.

(a)

Dataset	Gallery2						Contact							Email							
Tech.	4.0.2	4.0.4	4.1.2	4.2.2	4.3.1	Avg	2.3.7	4.0.2	4.0.4	4.1.2	4.2.2	4.3.1	Avg	2.3.2	2.3.7	4.0.2	4.0.4	4.1.2	4.2.2	4.3.1	Avg
LR	0.76	0.74	0.67	0.62	0.67	0.69	0.48	0.77	0.76	0.76	0.71	0.75	**0.71**	0.84	0.73	0.84	0.57	0.74	0.68	0.60	**0.71**
NB	0.78	0.71	0.56	0.58	0.64	0.65	0.56	0.77	0.74	0.76	0.66	0.75	**0.71**	0.83	0.79	0.84	0.57	0.72	0.65	0.60	**0.71**
BN	0.70	0.77	0.77	0.55	0.68	0.69	0.46	0.72	0.76	0.68	0.40	0.75	0.63	0.83	0.74	0.77	0.50	0.66	0.35	0.58	0.63
MLP	0.75	0.76	0.73	0.61	0.65	**0.70**	0.65	0.77	0.74	0.69	0.68	0.75	**0.71**	0.81	0.72	0.82	0.55	0.72	0.65	0.63	**0.70**
RBF	0.50	0.76	0.71	0.61	0.64	0.64	0.67	0.78	0.74	0.71	0.64	0.73	**0.71**	0.78	0.73	0.85	0.56	0.71	0.65	0.62	**0.70**
SVM	0.50	0.50	0.61	0.50	0.50	0.52	0.50	0.50	0.59	0.50	0.50	0.68	0.55	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
VP	0.45	0.51	0.61	0.50	0.50	0.51	0.48	0.64	0.53	0.50	0.50	0.61	0.54	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50
CART	0.49	0.66	0.75	0.53	0.50	0.59	0.50	0.54	0.65	0.50	0.41	0.71	0.55	0.48	0.49	0.44	0.54	0.51	0.43	0.52	0.49
J48	0.49	0.67	0.72	0.54	0.55	0.59	0.49	0.54	0.66	0.48	0.41	0.77	0.56	0.66	0.49	0.44	0.53	0.56	0.35	0.49	0.50
ADT	0.67	0.72	0.78	0.58	0.66	0.68	0.62	0.77	0.73	0.63	0.67	0.77	**0.70**	0.81	0.75	0.78	0.53	0.73	0.52	0.58	0.67
Bag	0.71	0.79	0.80	0.65	0.68	**0.73**	0.61	0.76	0.75	0.68	0.64	0.79	**0.71**	0.81	0.72	0.86	0.50	0.73	0.66	0.60	**0.70**
RF	0.77	0.80	0.80	0.64	0.71	**0.74**	0.64	0.79	0.71	0.66	0.56	0.80	0.69	0.78	0.63	0.57	0.44	0.74	0.58	0.55	0.61
LMT	0.66	0.76	0.74	0.63	0.67	0.69	0.46	0.72	0.74	0.75	0.71	0.73	0.69	0.81	0.66	0.84	0.57	0.66	0.73	0.58	0.69
LB	0.75	0.78	0.79	0.59	0.68	**0.72**	0.66	0.77	0.76	0.73	0.59	0.77	**0.71**	0.84	0.76	0.83	0.52	0.74	0.65	0.61	**0.71**
AB	0.69	0.75	0.78	0.59	0.65	0.69	0.66	0.79	0.74	0.70	0.62	0.71	**0.70**	0.82	0.72	0.81	0.54	0.66	0.65	0.61	0.69
NNge	0.56	0.68	0.65	0.57	0.62	0.62	0.56	0.66	0.68	0.51	0.47	0.68	0.59	0.53	0.51	0.48	0.49	0.61	0.63	0.54	0.54
DTNB	0.49	0.78	0.78	0.61	0.56	0.64	0.49	0.67	0.76	0.60	0.41	0.75	0.61	0.48	0.49	0.44	0.51	0.54	0.35	0.49	0.47
VFI	0.58	0.51	0.65	0.50	0.64	0.58	0.61	0.64	0.72	0.67	0.62	0.70	0.66	0.71	0.62	0.65	0.51	0.71	0.51	0.49	0.60

(b)

Dataset	MMS							Calendar					Bluetooth				Telephony		
Tech.	2.3.7	4.0.2	4.0.4	4.1.2	4.2.2	4.3.1	Avg	4.0.2	4.0.4	4.1.2	4.2.2	Avg	4.1.2	4.2.2	4.3.1	Avg	4.2.2	4.3.1	Avg
LR	0.76	0.66	0.85	0.73	0.56	0.68	**0.71**	0.66	0.80	0.82	0.69	**0.74**	0.49	0.79	0.81	**0.70**	0.73	0.85	**0.79**
NB	0.79	0.73	0.84	0.76	0.62	0.80	**0.76**	0.70	0.80	0.71	0.69	**0.73**	0.53	0.74	0.85	**0.71**	0.73	0.83	**0.78**
BN	0.76	0.46	0.84	0.73	0.43	0.52	0.62	0.58	0.70	0.51	0.56	0.59	0.37	0.73	0.71	0.60	0.70	0.83	**0.77**
MLP	0.75	0.71	0.85	0.71	0.61	0.76	**0.73**	0.65	0.80	0.69	0.65	**0.70**	0.46	0.78	0.80	0.68	0.72	0.85	**0.79**
RBF	0.74	0.76	0.80	0.74	0.76	0.74	**0.76**	0.71	0.73	0.75	0.69	**0.72**	0.52	0.70	0.80	0.67	0.69	0.81	**0.75**
SVM	0.52	0.50	0.76	0.50	0.50	0.50	0.55	0.52	0.63	0.50	0.61	0.57	0.63	0.50	0.62	0.58	0.65	0.76	**0.71**
VP	0.58	0.59	0.67	0.56	0.50	0.50	0.57	0.51	0.50	0.49	0.51	0.50	0.56	0.50	0.50	0.52	0.50	0.50	0.50
CART	0.77	0.45	0.75	0.74	0.43	0.45	0.60	0.56	0.73	0.58	0.63	0.63	0.57	0.48	0.49	0.51	0.65	0.75	**0.70**
J48	0.68	0.48	0.78	0.67	0.43	0.52	0.59	0.59	0.61	0.55	0.54	0.57	0.47	0.46	0.50	0.48	0.69	0.80	**0.75**
ADT	0.79	0.72	0.83	0.72	0.62	0.74	**0.74**	0.58	0.74	0.71	0.64	0.67	0.46	0.69	0.72	0.62	0.73	0.83	**0.78**
Bag	0.80	0.68	0.84	0.74	0.68	0.72	**0.74**	0.69	0.80	0.69	0.66	**0.71**	0.51	0.71	0.71	0.64	0.71	0.83	**0.77**
RF	0.76	0.65	0.82	0.67	0.70	0.73	**0.72**	0.64	0.78	0.72	0.61	0.69	0.49	0.68	0.71	0.63	0.65	0.85	**0.75**
LMT	0.78	0.66	0.83	0.75	0.56	0.73	**0.72**	0.65	0.81	0.79	0.63	**0.72**	0.43	0.76	0.80	0.66	0.73	0.84	**0.79**
LB	0.79	0.75	0.82	0.71	0.70	0.65	**0.74**	0.56	0.78	0.73	0.67	0.69	0.48	0.76	0.76	0.67	0.75	0.83	**0.79**
AB	0.77	0.70	0.81	0.69	0.70	0.65	**0.72**	0.57	0.71	0.75	0.69	0.68	0.31	0.72	0.68	0.57	0.72	0.80	**0.76**
NNge	0.69	0.53	0.75	0.66	0.65	0.51	0.63	0.68	0.67	0.65	0.59	0.65	0.40	0.58	0.58	0.52	0.57	0.72	0.65
DTNB	0.73	0.46	0.81	0.71	0.43	0.68	0.64	0.62	0.72	0.54	0.58	0.62	0.50	0.49	0.73	0.57	0.72	0.81	**0.77**
VFI	0.74	0.72	0.70	0.62	0.76	0.74	**0.71**	0.48	0.76	0.71	0.67	0.66	0.50	0.79	0.68	0.66	0.70	0.81	**0.76**

shows that similar to the results shown in Fig. 3, NB, LR, MLP, LB and Bagging techniques show the highest performance over all the data sets.

5.4. RQ3: is the performance of defect prediction models validated using inter-release validation comparable to 10-fold validation and is it statistically different than 10-fold validation?

We further compared the results of the defect prediction models developed using all the 18 ML techniques using 10-fold validation and inter-release validation. The comparative results using average AUC values are shown in Fig. 5. It can be observed from the figure that in majority of the cases the results of the inter-release models were found comparable or even better than 10-fold validation results. However, in certain cases such as in the case when the model developed by Contact 4.0.2 was validated on Contact 4.0.4, or another case when the model developed by Contact 4.2.2 was validated on Contact 4.3.1, there was decrease in the average AUC values. The average AUC values of 10-fold validation models were better than inter-release models in such cases. The reason for such a trend could be the ineffectiveness of the training models due to low percentage of defective classes. The percentage of defective classes in Contact 4.0.2 and Contact 4.2.2 are very less i.e. 5.84% and 3.73%,

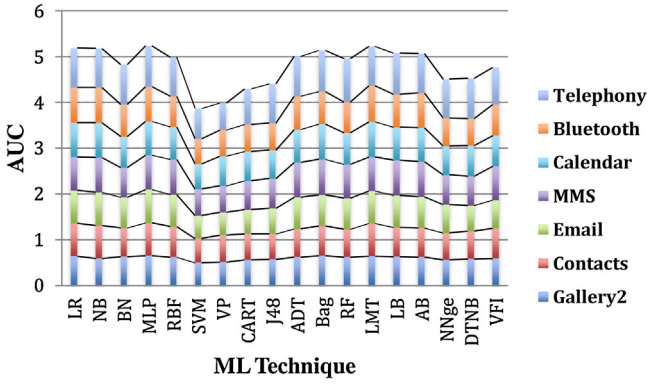

Fig. 4. Average AUC values of each ML technique on corresponding data sets using inter-release validation.

respectively as compared to the versions on which these models are validated i.e. Contact 4.0.4 has 32.32% of defective classes and Contact 4.3.1 has 46.66% defective classes. Due to low number of defective classes in the data sets, which are used for training, the models developed may not be able to learn effectively the values of OO metrics that are efficient in predicting defective classes leading to decrease in average AUC values.

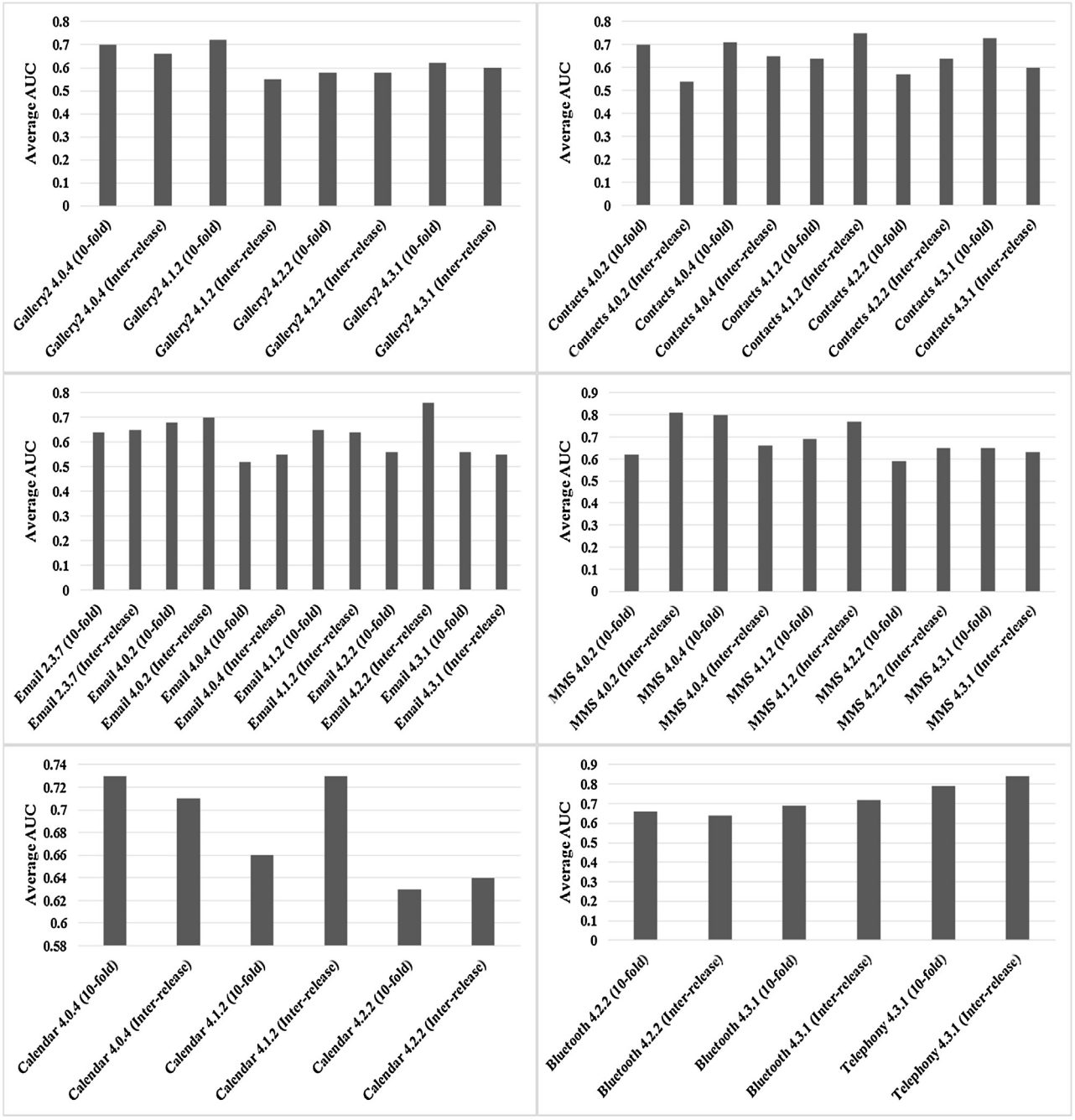

Fig. 5. Comparison of 10-fold and Inter-release validation models using average AUC values.

Moreover, we also validated whether there is any statistical difference amongst the results obtained using 10-fold and inter-release validation on all the data sets using Wilcoxon test. The average results of all the data sets on all the releases of a specific data set, were analyzed to compare the AUC obtained using 10-fold and inter-release validation. The test did not yield significant results, which means that the performance of the developed models using both 10-fold and inter-release validation is comparable to each other.

5.5. RQ4—which are the best and worst ML techniques for defect prediction using OO metrics?

The difference in performance amongst the ML techniques is assessed using Friedman test. It provides a combined ranking of the 18 ML techniques over multiple releases of seven application packages of Android data sets. This will allow us to determine whether the performance difference among the ML techniques is random or not and establish the superiority of one ML technique over the other ML techniques. The results are analyzed at the 0.5 significance level. We first obtain the average performance of each technique on each of the data set by computing the average AUC obtained on all the releases of a specific data set both using 10-fold as well as using inter-release validation. Then, each of the eighteen techniques are ranked by the Friedman test. Since, eighteen techniques are compared, the degrees of freedom is 17. The computed Friedman test statistic is 92.354. We used AUC for determining the difference in performance of the ML techniques. AUC is reported to be an appropriate measure for dealing with unbalanced and noisy data [56] as

Table 13
Friedman mean ranks of ML techniques based on AUC.

ML technique	Mean rank	ML technique	Mean rank
MLP	3.93	RF	8.50
LR, NB	4.68	BN	10.25
LB	4.71	VFI	10.64
Bag	4.82	DTNB	12.71
LMT	5.50	NNge	13.96
RBF	6.89	J48	14.79
ADT	7.71	CART	15.61
AB	7.96	SVM, VP	16.82

it is unaffected to the changes in distribution of outcome class (for example percentage of defective or non-defective classes).

The null hypothesis of the Friedman test is that there is no significant difference between the AUC of ML techniques. The results are found to be significant at 0.05 level. Hence, we reject the null hypothesis and accept the alternative hypothesis that there is a significant difference among the performance of the ML techniques. The results show that MLP is the best technique followed by NB and LR for defect prediction models. VP, SVM and CART techniques received the lowest ranks amongst the ML techniques. Hence, the results confirm the reliability of the difference amongst the ML techniques in the defect prediction models (Table 13).

The MLP technique achieved the best rank amongst all other ML techniques for developing defect prediction models using 10-fold as well as inter-release validation. MLP technique is effective and powerful as it allows insertion of a number of non-linear layers between the input and output layers. As advocated by Gondra [59], these intermediate layers allow complex interactions between the inputs and outputs that are capable of fitting complex functions. This also allows adaptiveness of the network to train new functions. The 10-fold validation results on Contact and Telephony data sets showed MLP as the best technique. Similarly, the inter-release validation results on Gallery2, Contact and Email depicted MLP as the best technique. It can be seen that in majority of the data sets except Telephony data set, the percentage of defective classes ranged from very low i.e. 1% to about 50%. Thus, MLP showed effective results with data sets having imbalanced distribution of defective classes (1%) to balanced distribution (50%). Moreover, MLP use back-propagation as a learning mechanism that allows two passes through the networks. During the forward pass of the network, the synaptic weights of the network are adjusted according to the training data, and the actual response of the network is produced. The backward pass propagates an error signal, which is the difference between the desired response and the actual response of the network. During this pass, the weights of the network are readjusted to decrease the difference between the actual and desired response. Thus, this training process helps in developing efficient models.

The average AUC of the NB model are between 0.65–0.78 for all the application packages of Android software. The results in this study confirm the previous findings that the NB technique is effective in defect prediction and may be used by researchers and practitioners in future applications. The NB technique is based on the assumption that the attributes are independent and unrelated. One of the reasons that the NB technique showed the best performance is that the features were reduced using the CFS method before applying the model prediction techniques in this work. The CFS method removes the features that are correlated with each other and retains the features that are correlated with the dependent variable. Hence, OO metrics selected by the CFS method for each data set are less correlated with each other and more correlated with the defect variable. This could be the reason that the NB technique showed the best results on data sets, which have few features, selected. For example, the 10-fold validation results on the Email 4.0.4 data set with just one feature selected (AUC—0.57), the MMS 4.1.2 data set with two features (AUC—0.76), the Email 2.3.7 data set (AUC—0.79) and the Bluetooth 4.3.1 data set (AUC—0.85) with three features each selected show the best AUC values using the NB technique. The NB technique is easy to understand and interpret (linear model can be obtained as a sum of logs) and is also computationally efficient [11,30]. However, the NB technique shows low AUC values in Email Android data set 4.0.4. This may be due to the reason the NB technique was not able to make accurate predictions of defects on the basis of only one OO metric (CAM). Thus, data sets with few features should use NB technique for developing defect prediction models.

Also, ensemble learners like RF, Bag, LB, AB and LMT show very good results on a number of data sets. For example, RF shows best results on Email 4.1.2, Contact 4.3.1, Gallery2 4.3.1 with AUC values of 0.74, 0.80 and 0.71 respectively. LB shows best results on Contact 4.0.4 with an AUC of 0.79 and Email 2.3.2 with an AUC of 0.84. Bag also exhibits best results on Email 4.0.2 data set with an AUC of 0.86 and so do LMT and AB on a number of data sets. Ensemble learners develop effective prediction models as they use a number of learning techniques together to obtain better predictive performance than constituent learning techniques. Thus, these techniques can also be used on various data sets for developing defect prediction models.

Table 14
Nemenyi test results.

	NB	Bag	RBF	ADT	LB	MLP	LR	LMT	AB	RF	VFI	BN	DTNB	NNge	CRT	J48	SVM	VP
NB		↑	↑	↑	↑	↓	=	↑	↑	↑	↑	↑	‡	‡	‡	‡	‡	‡
Bag			↑	↑	↓	↓	↓	↑	↑	↑	↑	↑	‡	‡	‡	‡	‡	‡
RBF				↑	↓	↓	↓	↓	↑	↑	↑	↑	↑	‡	‡	‡	‡	‡
ADT					↓	↓	↓	↑	↑	↑	↑	↑	↑	↑	‡	‡	‡	‡
LB						↓	↓	↓	↑	↑	↑	↑	‡	‡	‡	‡	‡	‡
MLP							↑	↑	↑	↑	↑	↑	‡	‡	‡	‡	‡	‡
LR								↑	↑	↑	↑	↑	‡	‡	‡	‡	‡	‡
LMT									↑	↑	↑	↑	‡	‡	‡	‡	‡	‡
AB										↑	↑	↑	↑	↑	‡	↑	‡	‡
RF											↑	↑	↑	↑	‡	↑	‡	‡
VFI												↓	↑	↑	↑	↑	↑	↑
BN													↑	↑	↑	↑	↑	↑
DTNB														↑	↑	↑	↑	↑
NNge															↑	↑	↑	↑
CART																↓	↑	↑
J48																	↑	↑
SVM																		=
VP																		

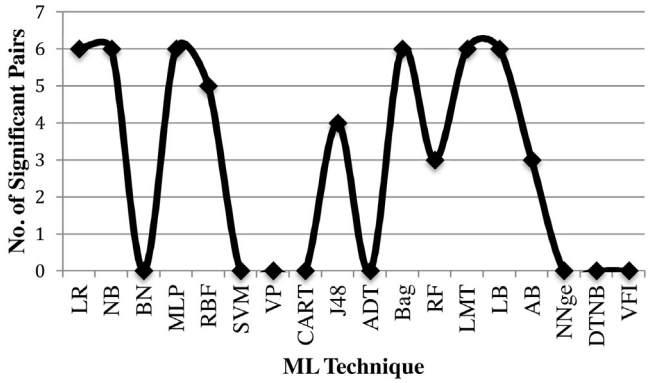

Fig. 6. Number of significant pairs corresponding to each ML technique.

Support Vector Machines like SVM and VP do not show good performance in terms of AUC for most of the data sets. This could be because these techniques are efficient in handling high dimensional data sets with redundant features. Since, we have applied CFS to the features before developing models using ML techniques, very few features (on an average 3–5) are left to develop the model. However, in certain data sets where after feature reduction, the number of features is 6–7, the SVM and VP techniques give better results. For example, in 10-fold validation results of Contact 4.3.1 data set with seven features, the SVM technique shows an AUC of 0.68 and the VP technique shows an AUC of 0.61, which is much higher than an AUC of 0.50 shown by SVM and VP in most other versions of Contact data set. Similarly, in MMS 4.0.4 data set, six features are used to develop models for 10-fold validation. The AUC of SVM was 0.76 and that of VP was 0.67, which is much higher than an AUC of 0.50–0.58, obtained by SVM and VP techniques in other MMS data sets. Thus, SVM and VP techniques are better suited to data sets with high dimensionality and redundant features.

5.6. RQ5—which pairs of ML techniques are statistically different from each other for defect prediction?

After obtaining significant results using the Friedman test, we performed post-hoc analysis using the Nemenyi test. The Nemenyi test was used to examine the statistical difference between the pairs of different ML techniques. The critical distance computed for Nemenyi test was 6.97 at $\alpha = 0.05$. The distance between the mean ranks allocated by the Friedman test to each of the techniques is computed for each pair of techniques and compared with this critical distance. If the computed difference is greater than or at least equal to the critical distance, the performance of pair of techniques is said to differ significantly at the significance level $\alpha = 0.05$.

The results of the pair-wise comparisons of the ML techniques are shown in Table 14. Out of 153 pairs of ML techniques, 51 were found to have significant results. The results shown in Table 14 depict that MLP, NB, LR, LB, Bagging and LMT techniques are significantly superior to six techniques namely DTNB, NNge, J48, CART, SVM and VP techniques. Similarly, RBF technique is significantly superior to NNge, J48, CART, SVM and VP techniques, whereas J48 technique is significantly superior to J48, CART, SVM and VP techniques. AB and RF techniques are significantly superior to CART, SVM and VP techniques. Hence, the results show that 51 pairs of ML techniques depict significant results. It also confirms that MLP, NB, LR, LB, Bagging and LMT are top six ML techniques with maximum number of significant pairs. Fig. 6 depicts the number of pairs corresponding to each ML technique, in which it performs significantly better. For example, LR performs significantly better than four other ML techniques when pairwise comparisons were performed using Nemenyi test.

6. Threats to validity

This section describes the various threats to validity of the study.

6.1. Conclusion validity

All possible threats that may bias the results and affect the relationship of dependent and independent variables pose a threat to conclusion validity [61]. Conclusion validity also concerns itself with statistical validity of the results. This study involves use of non-parametric tests i.e. Friedman and Nemenyi, which are carefully chosen as these tests do not require certain data pre-conditions to be fulfilled such as data normality, homogeneity of variances etc. The study also evaluates its results at a widely acceptable cut-off value of $\alpha = 0.05$. The results are also evaluated using post-hoc analysis, which reinforces confidence in the results of the study. Moreover, the use of 10-fold validation omits validation bias.

6.2. Internal validity

The threat to internal validity is present if there is "causal effect" of the OO metrics on defect proneness attribute. However, in order to determine this effect, it is necessary to perform experiments where values of other OO metrics are controlled in order determine the "causal effect" of one OO metric [13]. But such experiments are difficult to perform. Since, the objective of the study was not to determine the causal effect, this threat exists in the study.

6.3. Construct validity

Construct validity is used to ascertain whether the dependent and independent variables accurately represent the concepts they are supposed to measure. The dependent variable of the study is computed by the DCRS tool, which is efficient in extracting defect information from change logs. Thus, the dependent variable is accurately measured. Moreover, certain researchers have performed extensive experiments to validate the accurateness of certain OO metrics used in the study [12,62,63], which further reduces this threat for the independent variables of the study.

6.4. External validity

The extent to which the results of the study are generalizable aids the external validity of the study [64]. Since, the study uses seven application packages of widely used mobile operating system Android, the results of the study are widely acceptable and can be used in similar scenarios. Furthermore, the data sets used in the study are open source on nature, which aids the replicability of the study. However, future studies should evaluate the results on software which are not operating systems to increase the generalizability of the obtained results.

7. Conclusions

Development of a defect prediction model helps in ascertaining software quality attributes and focused use of constraint resources. They also guide researchers and practitioners to perform preventive actions in the early phases of software development and commit themselves for creation of better quality software. In this work, we not only conduct an extensive empirical experimentation on publically available application packages of Android software systems but also provide a repeatable and pragmatic approach to achieve such models. Specifically, the main contributions of the paper are: (i) create defect prediction models using various ML techniques on collected multiple data sets from the Google code repository for the Android platform, (ii) pre-process the data using

Table 15
AUC values of data sets using inter-release validation.

(a)

Dataset	Gallery2					Contact						Email						
Tech.	4.0.2 on 4.0.4	4.0.4 on 4.1.2	4.1.2 on 4.2.2	4.2.2 on 4.3.1	Avg	2.3.7 on 4.0.2	4.0.2 on 4.0.4	4.0.4 on 4.1.2	4.1.2 on 4.2.2	4.2.2 on 4.3.1	Avg	2.3.2 on 2.3.7	2.3.7 on 4.0.2	4.0.2 on 4.0.4	4.0.4 on 4.1.2	4.1.2 on 4.2.2	4.2.2 on 4.3.1	Avg
LR	0.77	0.55	0.54	0.68	0.64	0.61	0.72	0.77	0.77	0.73	**0.72**	0.73	0.85	0.58	0.72	0.78	0.63	**0.72**
NB	0.75	0.49	0.44	0.67	0.59	0.62	0.74	0.76	0.75	0.73	**0.72**	0.71	0.82	0.58	0.72	0.89	0.63	**0.73**
BN	0.76	0.55	0.61	0.61	0.63	0.50	0.70	0.77	0.67	0.50	0.63	0.73	0.81	0.57	0.64	0.77	0.5	0.67
MLP	0.77	0.65	0.51	0.68	0.65	0.62	0.75	0.76	0.76	0.73	**0.72**	0.74	0.86	0.58	0.72	0.83	0.63	**0.73**
RBF	0.76	0.53	0.57	0.63	0.62	0.50	0.74	0.74	0.75	0.64	0.67	0.73	0.76	0.58	0.73	0.78	0.61	**0.70**
SVM	0.50	0.50	0.49	0.50	0.50	0.50	0.50	0.66	0.50	0.50	0.53	0.50	0.50	0.50	0.50	0.50	0.50	0.50
VP	0.50	0.51	0.53	0.50	0.51	0.72	0.50	0.73	0.50	0.50	0.59	0.50	0.50	0.50	0.50	0.50	0.50	0.50
CART	0.50	0.58	0.59	0.57	0.56	0.61	0.50	0.72	0.50	0.50	0.57	0.50	0.50	0.50	0.66	0.50	0.50	0.53
J48	0.50	0.60	0.61	0.57	0.57	0.61	0.50	0.71	0.50	0.50	0.56	0.54	0.50	0.50	0.64	0.69	0.50	0.56
ADT	0.74	0.50	0.63	0.60	0.62	0.43	0.66	0.77	0.62	0.61	0.62	0.72	0.80	0.57	0.65	0.84	0.55	0.69
Bag	0.78	0.57	0.63	0.63	0.65	0.42	0.66	0.77	0.74	0.66	0.65	0.77	0.78	0.56	0.59	0.86	0.52	0.68
RF	0.68	0.58	0.61	0.60	0.62	0.44	0.66	0.79	0.61	0.53	0.61	0.74	0.76	0.53	0.63	0.93	0.50	0.68
LMT	0.72	0.55	0.63	0.65	0.64	0.61	0.75	0.76	0.74	0.73	**0.72**	0.62	0.85	0.58	0.73	0.83	0.63	**0.71**
LB	0.73	0.51	0.63	0.65	0.63	0.41	0.68	0.76	0.69	0.64	0.64	0.73	0.83	0.58	0.66	0.83	0.58	**0.70**
AB	0.73	0.55	0.64	0.57	0.62	0.41	0.67	0.73	0.73	0.66	0.64	0.71	0.77	0.58	0.67	0.77	0.58	0.68
NNge	0.54	0.56	0.57	0.56	0.56	0.57	0.56	0.73	0.57	0.50	0.59	0.60	0.56	0.51	0.66	0.88	0.50	0.62
DTNB	0.50	0.58	0.61	0.61	0.58	0.59	0.67	0.77	0.50	0.50	0.61	0.50	0.50	0.50	0.64	0.73	0.50	0.56
VFI	0.65	0.65	0.52	0.55	0.59	0.52	0.71	0.71	0.69	0.66	0.66	0.70	0.57	0.55	0.54	0.75	0.60	0.62

(b)

Dataset	MMS						Calendar				Bluetooth			Telephony	
Tech.	2.3.7 on 4.0.2	4.0.2 on 4.0.4	4.0.4 on 4.1.2	4.1.2 on 4.2.2	4.2.2 on 4.3.1	Avg	4.0.2 on 4.0.4	4.0.4 on 4.1.2	4.1.2 on 4.2.2	Avg	4.1.2 on 4.2.2	4.2.2 on 4.3.1	Avg	4.2.2 on 4.3.1	Avg
LR	0.81	0.80	0.80	0.66	0.58	**0.73**	0.81	0.79	0.66	**0.75**	0.72	0.82	**0.77**	0.86	**0.86**
NB	0.83	0.83	0.78	0.68	0.70	**0.76**	0.80	0.78	0.70	**0.76**	0.71	0.83	**0.77**	0.85	**0.85**
BN	0.85	0.50	0.78	0.63	0.50	0.65	0.72	0.78	0.55	0.68	0.60	0.79	**0.70**	0.87	**0.87**
MLP	0.85	0.82	0.81	0.68	0.58	**0.75**	0.81	0.76	0.65	**0.74**	0.65	0.87	**0.76**	0.89	**0.89**
RBF	0.82	0.75	0.78	0.72	0.80	**0.77**	0.76	0.74	0.63	**0.71**	0.48	0.86	0.67	0.85	**0.85**
SVM	0.68	0.50	0.71	0.50	0.50	0.58	0.51	0.63	0.51	0.55	0.59	0.50	0.55	0.66	0.66
VP	0.70	0.50	0.71	0.50	0.50	0.58	0.79	0.51	0.64	0.65	0.63	0.50	0.57	0.59	0.59
CART	0.80	0.50	0.71	0.63	0.50	0.63	0.66	0.71	0.55	0.64	0.68	0.50	0.59	0.78	**0.78**
J48	0.84	0.50	0.77	0.63	0.50	0.65	0.59	0.72	0.55	0.62	0.68	0.50	0.59	0.86	**0.86**
ADT	0.83	0.69	0.81	0.74	0.71	**0.76**	0.70	0.78	0.68	**0.72**	0.62	0.84	**0.73**	0.87	**0.87**
Bag	0.85	0.72	0.79	0.72	0.81	**0.78**	0.83	0.74	0.75	**0.77**	0.61	0.81	**0.71**	0.90	**0.90**
RF	0.88	0.61	0.82	0.68	0.71	**0.74**	0.65	0.74	0.68	0.69	0.60	0.74	0.67	0.95	**0.95**
LMT	0.81	0.79	0.80	0.74	0.58	**0.74**	0.80	0.79	0.73	**0.77**	0.73	0.85	**0.79**	0.86	**0.86**
LB	0.85	0.69	0.81	0.69	0.80	**0.77**	0.71	0.76	0.68	**0.72**	0.63	0.81	**0.72**	0.90	**0.90**
AB	0.82	0.78	0.78	0.66	0.80	**0.77**	0.72	0.79	0.70	**0.74**	0.68	0.84	**0.76**	0.86	**0.86**
NNge	0.78	0.54	0.71	0.65	0.56	0.65	0.67	0.60	0.62	0.63	0.57	0.64	0.61	0.85	**0.85**
DTNB	0.80	0.52	0.77	0.63	0.50	0.64	0.66	0.81	0.55	0.67	0.68	0.50	0.59	0.88	**0.88**
VFI	0.84	0.79	0.72	0.58	0.78	**0.74**	0.48	0.76	0.71	0.67	0.50	0.79	0.68	0.70	**0.81**

the attribute reduction techniques, (iii) empirically validate the constructed models on various releases of the application packages of Android data sets in order to obtained generalized results, (iv) perform statistical tests to evaluate the significance of the obtained results and, (v) analyze the outcomes to deduce meaningful and generalized conclusions. The results are based on AUC and the primary results of the paper are as follows:

1. The LOC, CAM, WMC, Ce and LCOM3 metrics were found to be significant predictors over the various releases of seven application packages of the Android software using the CFS method.
2. The work confirms the overall predictive ability of the various ML techniques including MLP, NB, AB, ADT, Bagging, LB and RF techniques (AUC greater than 0.7 in most cases) for defect prediction. The results are confirmed using both 10-fold and inter-release validation.
3. The results of inter-release validation depict comparable results with 10-fold validation. This conforms the generalizability of the results with respect to the defect prediction models developed in this work.
4. The results of the comparison of ML techniques evaluated using Friedman test show the superiority of the MLP, LB, LR and NB technique in defect prediction. The MLP technique is found to be the best followed by NB and LR techniques. On the other hand, support vector based techniques namely SVM and VP show the worst performance for predicting defective classes. They are followed by two decision tree based techniques CART and J48 among the lowest rankers.
5. The statistical test, Nemenyi, depict that there is significant difference between the performances of 51 pairs (out of 153) of the ML techniques, i.e. about 33.3%.

Thus, we conclude that the ML techniques have predictive capability and the empirical framework proposed in this paper is repeatable and the results can be generalized on future releases of Android software and similar natured software. Moreover, the study statistically examines the results using Friedman and Nemenyi test (RQ4 and RQ5), thus overcoming the weakness of low use of statistical tests in the previous literature studies.

Once a defect prediction model is trained it can be used for quality assessment and for predictions on future unseen data. The predictions of this work can be utilized for assessing software quality processes and procedures as we evaluate the software products that are a result of these processes. The defect prediction models are efficient and effective, can be used by researchers and practitioners to quickly identify defect-prone classes and will aid any software organization to reduce associated costs, time of development and allow an efficient allocation of available resources. In future, we propose to replicate the work done in this paper using metaheuristic techniques and hybridized algorithms.

Acknowledgement

The author is thankful to Ms. Megha Khanna, Research Scholar, Department of Software Engineering, Delhi Technological University, India for her research inputs and scholarly discussions.

References

[1] Y. Singh, A. Kaur, R. Malhotra, Empirical validation of object-oriented metrics for predicting fault proneness models, Softw. Qual. J. 18 (2010) 3–35.
[3] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (2008) 485–496.
[4] R. Malhotra, Empirical Research in Software Engineering—Methods, Techniques and Applications, CRC Press, USA, 2015.
[5] R. Malhotra, A systematic review of machine learning techniques for software fault prediction, Appl. Soft Comput. 27 (2015) 504–518.
[6] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, Proceedings of the 33rd International Conference on Software Engineering (2011) 1–10.
[7] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (2007) 2–13.
[8] I. Myrtveit, E. Stensrud, M. Shepperd, Reliability and validity in comparative studies of software prediction models, IEEE Trans. Softw. Eng. 31 (2005) 380–391.
[9] R. Malhotra, R. Raje, An empirical comparison of machine learning techniques for software defect prediction, Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies (2014) 320–327.
[10] V. Basili, L. Briand, W. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Trans. Softw. Eng. 22 (1996) 751–761.
[11] A. Binkley, S. Schach, Validation of the coupling dependency metric as a risk predictor, Proceedings of the International Conference on Software Engineering (1998) 452–455.
[12] L.C. Briand, J. Wust, J.W. Daly, D.V. Porter, Exploring the relationships between design measures and software quality in object-oriented systems, J. Syst. Softw. 51 (2000) 245–273.
[13] L. Briand, J. Wüst, H. Lounis, Replicated case studies for investigating quality factors in object-oriented designs, Empir. Softw. Eng.: Int. J. 6 (2001) 11–58.
[14] M. Cartwright, M. Shepperd, An empirical investigation of an object-oriented software system, IEEE Trans. Softw. Eng. 26 (1999) 786–796.
[15] S. Chidamber, D. Darcy, C. Kemerer, Managerial use of metrics for object-oriented software: an exploratory analysis, IEEE Trans. Softw. Eng. 24 (1998) 629–639.
[16] R. Harrison, S.J. Counsell, R.V. Nithi, An evaluation of MOOD set of object-oriented software metrics, IEEE Trans. Softw. Eng. 24 (1998) 491–496.
[17] K. El Emam, S. Benlarbi, N. Goel, S. Rai, The confounding effect of class size on the validity of object-oriented metrics, IEEE Trans. Softw. Eng. 27 (2001) 630–650.
[18] W. Li, S. Henry, Object-oriented metrics that predict maintainability, J. Syst. Softw. 23 (1993) 111–122.
[19] P. Yu, T. Systa, H. Muller, Predicting fault-proneness using OO metrics: an industrial case study, in: Proceedings of Sixth European Conference on Software Maintenance and Reengineering, Budapest, Hungary, 2002, pp. 99–107.
[20] K.K. Aggarwal, Y. Singh, A. Kaur, R. Malhotra, Investigating the effect of coupling metrics on fault proneness in object-oriented systems, Softw. Qual. Prof. 8 (2006) 4–16.
[21] W. Li, R. Shatnawi, An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution, J. Syst. Softw. 80 (2007) 1120–1128.
[22] H. Olague, L. Etzkorn, S. Gholston, S. Quattlebaum, Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes, IEEE Trans. Softw. Eng. 33 (2007) 402–419.
[23] K.K. Aggarwal, Y. Singh, A. Kaur, R. Malhotra, Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study, Softw. Process Improv. Pract. 14 (2009) 39–62.
[24] L.C. Briand, W. Melo, J. Wuest, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE Trans. Softw. Eng. 28 (July (7)) (2002).
[25] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, IEEE Trans. Softw. Eng. 31 (2005) 897–910.
[26] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Trans. Softw. Eng. 20 (1994) 476–493.
[27] Y. Zhou, H. Leung, Empirical analysis of object-oriented design metrics for predicting high and low severity faults, IEEE Trans. Softw. Eng. 32 (2006) 771–789.
[28] R. Malhotra, Y. Singh, A. Kaur, Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines, Int. J. Syst. Assur. Eng. Manag. 1 (2010) 269–281.
[29] S. Kanmani, V.R. Uthariaraj, V. Sankaranarayanan, P. Thambidurai, Object-oriented software fault prediction using neural networks, Inf. Softw. Technol. 49 (2007) 483–492.
[30] G.J. Pai, J.B. Dugan, Empirical analysis of software fault content and fault proneness using Bayesian methods, IEEE Trans. Softw. Eng. 33 (2007) 675–686.
[31] C. Catal, B. Diri, B. Ozumut, An artificial immune system approach for fault prediction in object-oriented software, 2nd Int. Conf. Dependability Comput. Syst. (2007) 1–8.
[32] E. Arisholm, L.C. Briand, E.B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, J. Syst. Softw. 83 (2010) 2–17.
[33] A.B. Carvalho, A. Pozo, S.R. Vegilio, A symbolic fault-prediction model based on multiobjective particle swarm optimization, J. Syst. Softw. 83 (2010) 868–882.
[34] Y. Singh, A. Kaur, R. Malhotra, Software fault pronennes prediction using support vector machines, Procedings World Congr. Eng. (2009) 240–245.
[35] Y. Zhou, B. Xu, H. Leung, On the ability of complexity metrics to predict fault-prone classes in object-oriented systems, J. Syst. Softw. 83 (2010) 660–674.

[36] S. Martino, F. ferrucci, C. Gravino, F. Sarro, A genetic algorithm to configure support vetcor machine for predicting fault prone components, Product Focus Software Process Improvement (2011) 247–261.
[37] A. Okutan, O.T. Yıldız, Software defect prediction using Bayesian networks, Empir. Softw. Eng. 19 (2012) 154–181.
[38] D. Azar, J. Vybihal, An ant colony optimization algorithm to improve software quality prediction models: case of class stability, Inf. Softw. Technol. 53 (2011) 388–393.
[39] R. Malhotra, Y. Singh, On the applicability of machine learning techniques for object-oriented software fault prediction, Softw. Eng. Int. J. 1 (2011) 24–37.
[40] R. Malhotra, A. Jain, Fault prediction using statistical and machine learning methods for improving software quality, J. Inf. Process. Syst. 8 (2012) 241–262.
[41] Y. Yang, Z. He, F. Shu, M. Li, Q. Wang, An investigation on the feasibility of cross-project defect prediction, Autom. Softw. Eng. 19 (2012) 167–199.
[42] M. Li, H. Zhang, R. Whu, Z. Zhou, Sample-based software defect prediction with active and semi-supervised learning, Autom. Softw. Eng. 19 (2012) 201–230.
[43] G. Canfora, A.D. Lucia, M.D. Penta, R. Oliveto, A. Panichella, S. Panichella, Multi-objective cross-project defect prediction, Proceedings of 6th International Conference on Software Testing, Verification and Validation, Luxembourg (2013) 252–261.
[44] M. Harman, S. Islam, Y. Jia, L.L. Minku, F. Sarro, K. Sirivisut, Less is more: temporal fault predictive performance over multiple hadoop releases, in: Proceedings of 6th International Symposium on Search Based Software Engineering, Fortaleza, 2014, pp. 240–246.
[45] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[46] E.R. Delong, D.M. Delong, D.L. Clarke-Pearson, Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach, Biometrics 44 (1998) 837–845.
[47] J.P. Espada, V.G. Diaz, R.G. Crespo, O.S. Martinez, B.C.P. G-Bustelo, J.M.C. Lovella, Using extended web technologies to dvelop bluetooth multi-platform mobile applications for interact with smart things, Inf. Fusion 21 (2015) 30–41.
[48] J.P. Espada, V.G. Diaz, R.G. Crespo, B.C.P. G-Bustelo, J.M.C. Lovella, An intelligent mobile web browser to adapt the mobile web as a function of the physical environment, IEEE Latin Am. Trans. 13 (2015) 503–509.
[49] R. Malhotra, K. Nagpal, P. Upmanyu, N. Pritam, Defect collection and reporting system for git based open source software, ICACCI (2014).
[50] B. Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.
[51] J. Bansiya, A. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Trans. Softw. Eng. 28 (2002) 4–17.
[52] R. Martin, OO design quality metrics—an analysis of dependencies, Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94 (1994).
[53] M.H. Tang, M.H. Kao, M.H. Chen, An empirical study on object-oriented metrics, Proceedings of Metrics (1999) 242–249.
[54] M. Hall, Correlation-based feature selection for discrete and numeric class machine learning, Proceedings of the 17th Int. Conference on Machine Learning (2007) 359–366.
[55] J. Hanley, B.J. McNeil, The meaning and use of the area under a receiver operating characteristic ROC curve, Radiology 143 (1982) 29–36.
[56] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (2009) 1263–1284.
[57] M. Stone, Cross-validatory choice and assessment of statistical predictions, J. R. Soc. Ser. B 36 (2) (1974) 111–114.
[58] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, Ann. Math. Stat. 11 (1940) 86–92.
[59] I. Gondra, Applying machine learning to software fault-proneness prediction, J. Syst. Softw. 81 (2008) 186–195.
[61] T.M. Khoshgoftaar, N. Seliya, N. Sundaresh, An empirical study of predicting faults with case-based reasoning, Softw. Qual. J. 14 (2006) 85–111.
[62] L. Briand, J. Daly, J. Wust, A unified framework for cohesion measurement in object-oriented systems, Empir. Softw. Eng. 3 (1998) 65–117.
[63] L. Briand, J. Daly, J. Wust, A unified framework for coupling measurement in object-oriented systems, IEEE Trans. Softw. Eng. 25 (1999) 91–121.
[64] R. Malhotra, Comparative analysis of statistical and machine learning methods for predicting faulty modules, Appl. Soft Comput. 21 (2014) 286–297.