An Empirical Comparison of Machine Learning Techniques for Software Defect Prediction

Ruchika Malhotra
Dept. of Computer and Information Science
Indiana University-Purdue University
Indianapolis, IN USA
ruchmalh@cs.iupui.edu

Rajeev Raje
Dept. of Computer and Information Science
Indiana University-Purdue University
Indianapolis, IN USA
rraje@cs.iupui.edu

ABSTRACT
Software systems are exposed to various types of defects. The timely identification of defective classes is essential in early phases of software development to reduce the cost of testing the software. This will guide the software practitioners and researchers for planning of the proper allocation of testing resources. Software metrics can be used in conjunction with defect data to develop models for predicting defective classes. There have been various machine learning techniques proposed in the literature for analyzing complex relationships and extracting useful information from problems in less time. However, more studies comparing these techniques are needed to provide evidence so that confidence is established on the performance of one technique over the other. In this paper we address four issues (i) comparison of the machine learning techniques over unpopular used data sets (ii) use of inappropriate performance measures for measuring the performance of defect prediction models (iii) less use of statistical tests and (iv) validation of models from the same data set from which they are trained. To resolve these issues, in this paper, we compare 18 machine learning techniques for investigating the effect of Object-Oriented metrics on defective classes. The results are validated on six releases of the 'MMS' application package of recent widely used mobile operating system – Android. The overall results of the study indicate the predictive capability of the machine learning techniques and an endorsement of one particular ML technique to predict defects.

Categories and Subject Descriptors
D.2 [**Software Engineering**]: D.2.8 Metrics, D.4.8 Performance

General Terms
Measurement, Performance, Reliability, Verification.

Keywords
Defect prediction, Object-oriented metrics, Machine Learning, Empirical Validation.

1. INTRODUCTION
As the complexity of the software system is increasing, it is difficult to produce a software without defects. One of the critical areas that can be considered while developing quality systems is minimization of defects. The cost of corrective maintenance increases exponentially if defects are detected in the later phases of software development [9]. Hence, software defect prediction is a very important activity that can result in reduction of testing efforts [28]. Detection of defects in early phases may result in delivering high quality and low cost software to the customers in a timely manner. Software metrics that capture various properties of the software can be used for developing models for predicting defective classes in a software. The software metrics collected from a similar project or past release can be used for developing defect prediction model. The developed defect prediction model can then be subsequently used for classifying classes of current projects as defective or not defective.

Various Machine Learning (ML) techniques have been proposed in the literature including neural networks, decision trees, support vector machines, and Bayesian learning. But, the performance of these techniques varies with different data sets and the superiority of one technique over others across many different studies is difficult to determine. Hence, it is essential to perform more studies in order to draw generalized conclusions to form widely acceptable and well-formed theories based on the experimental evidence obtained by the results [25]. The results from the empirical studies will help to improve, refute and validate the results obtained from the past studies.

As noted by Lessman et al. [15], the study size, the way in which the performance of the developed models is measured and statistical tests used are three very important factors for comparision of results across different studies. Also, the availability of data sets has always been a constraint in software engineering research. The use of unstable performance measures is another dimension to be considered. The statistical tests to compare the performance of techniques have been used only in few studies. The past studies have concluded that it is a common practice to derive results without computing statistical significance (Menzies et al. [20], Myrtviet et al. [21]). Further, in the past studies the models have been validated on the same data from which they were derived.

In order to address the above issues, in this work, we compare the performance of 18 ML techniques on six releases of 'MMS' application package of Android operating system. This enables the investigation whether one technique outperforms others and also provides insights on the selection of a particular ML technique. The 18 ML techniques are subset of 22 ML techniques

BICT 2014, December 01-03, Boston, United States
Copyright © 2015 ICST
DOI 10.4108/icst.bict.2014.257871

used by [15] to assess relationship between static code metrics (traditional, procedural and module based metrics [16]) and defect proneness. However, in this work we predict the defect prone classes using the Object-Oriented (OO) metrics design suite given by [3, 7, 8] instead of static code metrics. 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 Android 'MMS' data set?

RQ2: Which is the best ML technique for defect prediction using OO metrics?

RQ3: Which pairs of ML techniques are significantly different from each other for defect prediction?

RQ4: What is the performance of ML techniques when across-release validation is used for predicting post-release defects?

There have been few studies in the literature that evaluate the statistical significance of the ML models using static code attributes [9, 15, 20].

To the best of the authors' knowledge there is no study so far that extensively compares and analyzes the performance of the ML models for defect prediction using OO metrics. 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 six releases of widely used Android application package.

(3) Statistical comparison of obtained results.

(4) An across-release validation of results 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 research background. Section 4 presents the research methodology and Section 5 provides the results of the work. The conclusion and future work of the work are presented in Section 6.

2. RELATED WORK

There are many studies that related software metrics with defect proneness in the literature. Due to space constraint, we will limit the description in this section to studies that evaluate the ML techniques for predicting defects using OO metrics. Gyimothy et al. [11] validated Logistic Regression (LR) and ML techniques (decision tree and neural network) using Mozilla data set for evaluating relationship between Chidamber & Kemerer (CK) metrics [7] and defect prediction. The results indicated Number Of Children (NOC) to be insignificant metric for defect prediction and low precision values of developed ML models. Zhou and Leung [28] validated NASA data set KC1 (written in C++ language) with respect to two severity categories of defects. They employed LR and ML (Naïve Bayes, Random Forest and NNge) techniques to evaluate the usefulness of CK metrics for defect prediction with regard to defect severities. The results showed that the prediction capability of CK metrics was limited to prediction of high severity faults. The values of performance measures for models predicted using ML techniques were low. Singh et al. [25] validated the same NASA data set with three severity levels of defects – high, medium and low. They used decision tree and neural network techniques and employed the ROC analysis to evaluate the performance of defect prediction models. Their results showed were better as compared to Zhou and Leung particularly at the high severity level. The performance of the ML techniques was also high. In another study conducted by Malhotra et al. [17], the predicted capability of the Support Vector Machines (SVM) technique was analyzed using the NASA KC1 data set at various severity levels.

An approach using Neural Networks was evaluated by Kanmani et al. [14] by using the software system written in the Java language. The results showed that the two neural networks performed better than the statistical techniques. Pai and Dugan [23] evaluated Bayesian approach using the NASA KC1 data set. They concluded that the Bayesian techniques are comparable to the existing techniques for defect prediction. Another study by Catal et al. [5] analyzed Artificial Immune Recognition System using NASA KC1 data set. Arisholm et al. [1] compared various variants of decision tree techniques using the Java Telecom system. They applied neural networks, SVM and LR techniques and concluded that the decision tree technique (C4.5) performed very well. Carvalho et al. [4] and Singh et al. [24] confirmed the effectiveness of the MultiObjective Particle Swarm Optimization technique and SVM techniques (using Jedit software) for defect prediction, respectively. Zhou et al. [30] proposed the appropriate usefulness of Odds ratio in precutting defective classes using CK and complexity measures. Martino et al. [20] used a Genetic algorithm to configure the SVM for prediction of defective classes on the basis of OO metrics. Okutan et al. [22] used Bayesian networks to evaluate the relationship between CK metrics and defect proneness. Azar and Vybihal [2] validated Ant Colony Optimization (ACO) technique and concluded it to be better than decision tree (C4.5) and random guessing techniques.

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 compare the ML techniques to evaluate the relationship between OO metrics and defect prediction. In this study we compare and assess the effectiveness of the 18 ML techniques for defect prediction. We use statistical tests to obtain the statistical significant differences among the ML techniques and also perform across-release validation on various releases of Android data set.

3. RESEARCH BACKGROUND

This section presents the process of empirical data collection and the dependent and independent variables used in this study.

3.1 Empirical Data Collection

In this study, the techniques are obtained using the popular, open source Android Operating System (OS). The six releases of Android OS have been selected with three code names namely – Ginger Bread (GB), Ice Cream Sandwich (ICS) and JellyBean (JB). The source code of these releases has been obtained from Google's Git repository (https://android.googlesource.com). The source code of Android OS is available in various application packages. We have used the 'Multimedia Messaging Service (MMS)' application package to collect the data. The characteristics of the six releases of 'MMS' application package are given in Table 1. The defects are collected using Defect Collection and Reporting System (DCRS) developed in the Java programming language at the Delhi Technological University [18]. Table 1 presents the total classes, LOC, number of defects and percentage of defects in each release of the MMS software.

The defects from the open source software defect logs were collected by the following steps:

(1) *Obtaining change logs*: The DCRS processes the Git repository and obtains change logs of two predetermined consecutive releases. The change may be due to defect fixation, addition of new functionality, refactoring or other related enhancements. Each change constitutes a single change record. A change consists of various information including timestamp of committing, unique identifier, change description (optional) and list of changed lines of source code. The change logs of six releases of Android MMS software were obtained namely 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.

(2) *Process the defect related changes*: This step involves preprocessing of changes related to fixed defects in order to extract useful information. The preprocessing is required as the change log consists of defect information from the beginning (i.e., the first release of the software). But, in our study defects that are incurred from the immediate previous release to the subsequent are only taken. For example, the defects between Android release 2.3.3 and 4.0.2 were taken into account; the defects between Android release 4.0.2 and 4.0.4 were taken into account and so on. The defect records are retrieved by searching defect-Ids and if defect-Ids are not present then keywords "issue", "bug" or "defect" are searched in the given description.

(3) *Computation of OO metrics on each Java file*: The values of OO metrics (indicated in Table 2) are calculated using open source CKJM tool (http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html). In this step, only Java source code files are considered and other type of files are ignored. The six releases of Android 'MMS', as indicated in Table 1, application 2.3.7, 4.0.2, 4.0.4, 4.1.2, 4.1.2, 4.2.2 and 4.3.1 contained 195, 201, 206, 223, 225, 224 Java classes, respectively. The OO metrics were collected for each of these classes.

(4) *Associating or linking defects to the corresponding classes*: The collected defects are then mapped to the classes in the source code. Finally, the total number of defects found in step 2 are related with each class found in step 3. The classes in each release of the software are associated with the defects fixed from the release under consideration to the immediate subsequent release (i.e., defects obtained from current release to next release).

Table 1. Android 'MMS' Data Set Details

Software	Code name	Total Classes	Total LOC	Defectiv e #	Defect ive %
Android 2.3.7	GB	195	13,157	54	27.69
Android 4.0.2	ICS	201	13,538	11	5.47
Android 4.0.4	ICS	206	13,804	68	33.01
Android 4.1.2	JB	223	14,759	42	18.83
Android 4.2.2	JB	225	14,932	12	5.33
Android 4.3.1	JB	224	14,915	23	10.27

This data set is used to empirically evaluate the approaches used in this study.

3.2 Variables

The independent variables (used in this study) are the OO metrics. In addition to the metrics proposed by Chidamber and Kemerer [7] we also used the metrics proposed by Henderson-Sellers [6] and Bansiya and Davis [3]. The dependent variable is the defect proneness defined as probability of occurrence of defect in a class [25, 29]. Table 2 presents the OO metrics used in this study for defect prediction.

4. RESEARCH METHODOLOGY

In order to answer the research questions described in Section 1, we conduct an empirical validation of various techniques on the six releases of the Android OS given in Table 1 using the following steps.

1) Preprocessing of collected data sets.
2) Selection of various ML Techniques for defect prediction.
3) Selection of performance measures and model validation techniques (refer section 4.3) for analyzing the performance of the models developed using Android data sets.
4) Selection of relevant OO metrics using Correlation Feature Sub selection (CFS) method (refer section 4.1).
5) Model development for defect prediction using ML techniques in step 2 (refer section 4.2).

Table 2. Description of Object-Oriented Metrics Used in the Study [3, 6, 7]

Abb.	Metric	Definition
WMC	Weighted Methods Per Class	Count of sum of complexities of number of methods in a class.
NOC	Number Of Children	Number of sub classes of a given class.
DIT	Depth of Inheritance Tree	Provides the maximum steps from the root to the leaf node.
LCOM	Lack of Cohesion Among Methods of a Class	Null pairs not having common attributes.
CBO	Coupling Between Objects	Number of classes to which a class is coupled.
RFC	Response For a Class	Number of external and internal methods in a class.
DAM	Data Access Metric	Ratio of the number of private (and/or protected) attributes to the total number of attributes of a class.
MOA	Measure Of Aggression	Percentage of data declarations (user defined) in a class.
MFA	Method of Functional Abstraction	Ratio of total number of inherited methods to the number of methods in a class.
CAM	Cohesion Among the Methods of a Class	Computes method similarity based on their signatures.
AMC	Average Method Complexity	Computed using McCabe's Cyclomatic Complexity method.
LCOM3	Lack Of Cohesion Among Methods of a Class	Revision of LCOM metric given by Henderson sellers
LOC	Line Of Code	Number of lines of source code of a given class.
NPM	Number of Public Methods	Number of public methods in a given class.
Ca	Afferent Couplings	Number of classes calling a given class.
Ce	Efferent Couplings	Number of other classes called by a class.
IC	Inheritance Coupling	Number of parent classes to which a class is coupled.
Defects	Defect Count	Binary variable indicating the presence or absence of the defects.

6) Model validation using two validations methods 10-cross validation and across-release validation (refer section 4.4).
7) Testing whether the difference between the performances of ML techniques is statistically significant using Friedman test and post-hoc analysis as suggested in [9] (refer section 4.5).

The techniques used and procedures followed in the above steps are explained in the following sub sections.

4.1 Data Preprocessing

In this study, we use 17 OO metrics for defect prediction. The uncorrelated and best attributes are selected out of a set of OO metrics using correlation based feature selection (CFS) technique [12]. This technique is simple, fast and widely used method in for sub selecting attributes using the ML techniques. The CFS technique searches all the combinations of attributes to find the best combination of the independent variables [16]. The CFS is a heuristic technique that evaluates the correlation between the independent variables and the dependent variable. The CFS technique is based on the principle that good attributes are highly correlated with the dependent variables and less correlated amongst themselves [12]. An attribute is selected if the correlation with the dependent variable is higher than the highest correlation amongst the attributes. The aim is to select individual variables that are correlated with the dependent variable and uncorrelated with other independent variables. Thus, the CFS technique handles both redundant and irrelevant attributes.

4.2 Machine Learning Techniques

Table 3 presents the summary of 18 ML techniques used in this study. We have carefully selected most of the important techniques that represent different domains given in table 3. We applied Weka 3.6 software to obtain the results.

4.3 Performance Measures

A variety of performance measures have been used in the literature to examine the strength of the developed models using different ML techniques. The data having disproportionate ratio of defective and not defective classes is often known as unbalanced data. Given the imbalanced nature of the datasets, the ROC analysis is a commonly used performance measure [4, 15]. The ROC curve depicts the percentage of correctly predicted defective classes (sensitivity) on the y-axis versus the one minus the percentage of correctly predicted non-defective classes (1-specificity) on the x-axis at various cut-off points [13]. Hence, in ROC analysis the sensitivity and 1-specificity of the developed model is calculated at each cut-off point. To compare various ML techniques the ROC curves are drawn for each ML technique.

The AUC computed using ROC analysis lies between 0 and 1 and higher the value of the AUC better is the predictive capability of the developed model. The advantage of using AUC for evaluating performance of the models is that it can deal with noisy and unbalanced data as it is insensitive to changes in class distribution.

4.4 Validation Methods

To obtain a realistic insight about the prediction of the model, it is necessary to apply the model to a different data set other than from which it is derived. We will employ two validation methods – 10-fold cross validation and across-release cross validation. In the 10-fold cross validation method the data set is divided into 10 parts where nine are used for training and one part is used for validation [26]. The procedure is repeated ten times and hence the results from all the folds are combined to produce the model result. In the across-release cross validation method, each release *r* of the Android data set is used as the training set to build the model and the model developed is then validated on the subsequent releases *r+1* of the Android data set. Hence, the model trained on the OO metrics of release *r* will be used for predicting defective classes of subsequent release *r+1*. As we are evaluating the six releases of the Android data set, we obtain across-release validation results for five releases.

4.5 Statistical Testing

To evaluate whether statistical difference exist between various ML techniques or not, we will use Friedman [10], a non parametric test, that can be used to rank a set of techniques over multiple data sets. The Friedman test is based on two hypotheses: Null Hypothesis (H_o): There is no statistical difference between the performance of the compared techniques. Alternative Hypothesis (H_i): There exists statistical difference between the performance of the compared techniques. The Friedman measure is defined as follows:

$$\chi_r^2 = \frac{12}{nk(k+1)} \sum R^2 - 3n(k+1)$$

Where R is the individual average rank (1, 2…., k), n is the number of data sets and k is number of compared techniques. The value of Friedman measure χ_r^2 is distributed over k-1 degree of freedom. If the value of Friedman measure is in the critical region (obtained from chi-squared table with specific level of significance i.e 0.01 or 0.05 and k-1 degree of freedom), then the Null hypothesis is rejected and it is concluded that there is difference among performance of compared techniques, otherwise Null hypothesis is accepted.

If the Null hypothesis is rejected, after applying the Friedman test, we will perform posthoc analysis using Wilcoxon signed-ranks test [27]. Wilcoxon signed-ranks test is a non-parametric test that performs pairwise comparisons of the difference in performance of the techniques. Friedman and Wilcoxon test is taken to be 0.05. We applied SPSS software to obtain the results.

5. EVALUATION RESULTS

This section presents the evaluation results of the various ML techniques for defect prediction using earlier indicated OO metrics. The results are validated using six releases of the 'MMS' application package of the Android OS. The results of the ML techniques are compared by first applying the Friedman test, followed by post-hoc Wilcoxon signed rank test, as described in

Table 3. Description of ML Techniques Used in the Study

Machine Learning Technique		Description
Statistical Classifier		
LR	Logistic Regression	LR is based on statistical and NB is based on Bayes theorem and
NB	NaiveBayes	interdependence of attributes. BN is also based on conditional

BN	Bayesian Networks	probabilities and uses K2 as search algorithm.
Neural Networks		MLP uses back propagation to learn instances and was trained using 0.3 learning rate with 500 epochs. RBF implements normalized Gaussian radial basis function.
MLP	Multilayer Perceptron	
RBF	Radial Basis Function	
Support Vector Machines based		They create hyperplane to separate the data set into defective and non defective classes. When the points are separated by nonlinear region, a kernel function is used for mapping the data into different dimensional space. The polynomial kernel function is used for VP and SVM.
SVM	Support Vector Machines	
VP	Voted Perceptron	
Decision Tree Methods		In these methods, while constructing trees, at each node the independent variable that best classifies the dependent variable is selected based on the specified splitting criterion. CRT uses Gini Index and C4.5 is based on the concept of entropy and used information gain.
CART	Classification & Regression Trees	
J48	C4.5 based technique	
ADT	Alternating decision trees	
Ensemble Learning		These are meta learning techniques that use voting procedure to obtain the defective or not defective prediction. RF uses CART and LMT uses LR as base learners. LB performs additive logistic regression. LB and AB use decisionstump (creates binary one-level decision tree algorithm) classification algorithm designed to be used for boosting algorithms.
Bag	Bagging	
RF	Random Forest	
LMT	Logistic Model Trees	
LB	Logit Boost	
AB	Ada Boost	
Rule Based Learning		NNge is like nearest-neighbor algorithm that generates if-then rules. DTNB is a hybrid classifier and it evaluates the advantages of both the DT and NB algorithms for each attribute at every step.
NNge	Nearest Neighbour with Generalised Exemplars	
DTNB	Decision Table Naïve Bayes	
Miscellaneous		
VFI	Voting Feature Intervals	It constructs intervals around each attribute for each class.

section 4.5 if the results in the Friedman test are significant. The predicted models are validated using 10-fold cross validation. Further the predictive capabilities of the ML techniques are evaluated using the across-release validation.

We generated models using all the independent variables selected using the CFS technique. The results obtained using the reduced set of variables were slightly better as compared to the results obtained using all the independent variables. Table 4 presents the relevant metrics found in each release of Android data set after applying the CFS technique. The results show that Ce, LOC, LCOM3, CAM, DAM were the most commonly selected OO metrics over the six releases of the Android data sets.

After this we empirically compared the ML techniques and the results were evaluated in terms of the AUC. The AUC has been advocated as a primary indicator of comparative performance of the predicted models [15]. The AUC measure can deal with noisy and unbalanced data and is insensitive to the changes in the class distributions [4]. Table 5 reports the 10-fold cross validation results of 18 ML techniques on six releases of Android OS. The ML technique yielding 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 Android data set.

Table 4. Relevant OO Metrics

Release	Relevant Features
Android 2.3.7	Ce, LCOM3, LOC, DAM, MOA, CAM, AMC
Android 4.0.2	WMC, RFC, LCOM, LCOM3, DAM
Android 4.0.4	Ce, NPM, LOC, LCOM3, DAM, CAM
Android 4.1.2	Ce, CAM
Android 4.2.2	DAM
Android 4.3.1	Ce, LOC, DAM, MOA

Table 5. 10-fold Cross Validation Results of 18 ML Techniques with respect to AUC

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

> ***RQ1:*** What is the overall predictive capability of various ML techniques on Android 'MMS' data set?
>
> ***A1:*** The AUC of most of the models predicted using the ML techniques is 0.7 which highlights the predictive capability of the ML techniques.

In order to confirm that the performance difference among the ML models is not random, we used Friedman test to evaluate the superiority of one ML technique over the other ML techniques. The Friedman test resulted in significant value of zero. The results were significant at the 0.05 level of significance over 17 degrees of freedom. Thus, the null hypothesis that all the ML techniques

have similar performance in terms of AUC is rejected. The results given in Table 6 show that NB technique is the best for predicting defect proneness of a class using OO metrics. The result supports the finding of Menzies et al. [20] that NB is the best technique for building defect prediction models. It can be also seen the models predicted using support vector machines based techniques, SVM and VP, performed worst.

> ***RQ2:*** Which is the best ML technique for defect prediction using OO metrics?
>
> ***A2:*** The outcome of the Friedman test indicates that the performance of the NB technique for defect prediction is the best. The performance of the Bagging and RBF techniques for defect prediction are the second best among the 18 compared ML techniques.

Table 6. Friedman Test Results

ML Tech.	Mean Rank	ML Tech.	Mean Rank
NB	3.58	RF	8.58
Bag	5.67	VFI	9.08
RBF	5.67	BN	10.83
ADT	5.92	DTNB	12.83
LB	6.17	NNge	13.58
MLP	6.25	CART	13.92
LR	7.08	J48	14.42
LMT	7.92	SVM	15.67
AB	8.17	VP	15.67

After obtaining significant results using the Friedman test, we performed posthoc analysis using the Wilcoxon test. The Wilcoxon test was used to examine the statistical difference between the pairs of different ML techniques. The results of the pair-wise comparisons of the ML techniques are shown in Table 7.

The results of Wilcoxon test show that out of 18 ML techniques, the NB model is significantly better than the models predicted using 17 ML techniques such as LMT, BN, DTNB, NNge, CART, J48, SVM, and VP. Similarly, the VP model was significantly worse than models developed using NB, Bag, RBF, ADT, LB, MLP, LR, LMT, AB, RF, and VFI techniques, worse than the BN, DTNB, NNge, CART, J48 techniques and better than the SVM model.

Figure 1 shows the number of ML techniques from which the performance of a given ML technique is either superior, significantly superior, inferior or significantly inferior. For example, from the bar chart shown in Figure 1, it can be seen that the performance of the NB technique is significantly superior to 8 other ML techniques, and non-significantly superior to 9 other techniques. Similarly, the performance of the Bagging technique is significantly superior to 7 other ML techniques, non-significantly superior to 8 other techniques and non-significantly inferior to 2 other ML techniques.

The AUC values of the NB model were between 0.73-0.85 in five releases of the Android data sets. 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. The NB technique is easy to understand and interpret (linear model can be obtained as a sum of logs) and is also computationally efficient [10, 29]. The NB technique was not able to retain the results in one release of the Android data set (Android 4.2.2). 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 (DAM).

> ***RQ3:*** Which pairs of ML techniques are significantly different from each other for defect prediction?
>
> ***A3:*** There are 112 pairs of ML techniques that yield significantly different performance results in terms of AUC. The results show that the performance of the NB model is significantly better than BN, LMT, BN, DTNB, NNge, CRT, J48, SVM and VP. Similarly significant pairs of performance of the other ML techniques are given in Table 7.

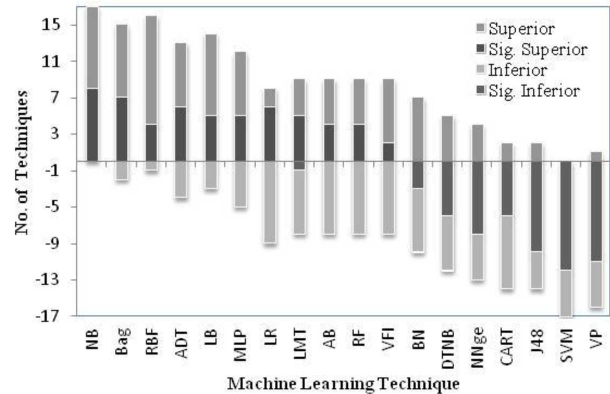

Figure 1: Results of Wilcoxon Test

To evaluate the accuracy of the predicted models, we also performed across-release cross validation. We validated the performance of the model derived from each release on the immediate subsequent release. For example, we validated the model trained using Android 2.3.7 data set on the Android 4.0.2 data set and so on. The results of the across-release cross validation in terms of AUC are shown in Table 8. The results of across-release validation show that the AUC of NB, RBF, ADT, Bagging, LMT and AB were greater than 0.7 in most of the releases of Android.

Figure 2 depicts the comparison of overall results of 18 ML techniques in terms of the average AUC using both 10-fold and across-release validation over all the Android releases. The chart shows that the overall performance results obtained from the across-release validation were better or comparable than the results obtained from the 10- fold cross validation except when Android 4.0.4 was validated using Android 4.0.2. One possible explanation to this is that the values of OO metrics in the Android releases are informative enough to predict defects in the subsequent releases. The reason for the low AUC values for across-release validation as compared to the AUC values for 10-fold cross validation in case of Android 4.0.4 could be that the defective class percentage in Android 4.0.2 is very less (5.47%) as compared to the defective class percentage in Android 4.0.4 (33.01%).

Table 7. Wilcoxon 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	⤈	⤈	⤈	⤈	⤈	⤈	⤈	⤈	⤈	⤈	⤈	↓	↓	↓	↓	↓	↑	

⤉ implies performance of ML tech. is significantly better than the compared ML tech., ↑ implies performance of ML tech. is better than the compared ML tech., ⤈ implies performance of ML tech. is significantly worse than the compared ML tech., ↓ implies performance of ML tech. is worse than the compared ML tech., = implies performance Equal

Table 8. Across-Release Validation Results of 18 ML Techniques with respect to AUC

	Android					
ML 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.**
LR	0.81	0.80	0.80	0.66	0.58	0.73
NB	0.82	0.80	0.79	0.68	0.70	0.76
BN	0.85	0.50	0.79	0.63	0.50	0.66
MLP	0.84	0.82	0.81	0.66	0.60	0.75
RBF	0.82	0.76	0.78	0.72	0.80	0.78
SVM	0.68	0.50	0.71	0.50	0.50	0.58
VP	0.72	0.50	0.58	0.50	0.50	0.56
CART	0.80	0.50	0.70	0.63	0.50	0.63
J48	0.84	0.50	0.77	0.63	0.50	0.65
ADT	0.83	0.69	0.81	0.74	0.74	0.77
Bag	0.85	0.73	0.79	0.72	0.81	0.78
RF	0.84	0.57	0.80	0.71	0.65	0.72
LMT	0.81	0.77	0.80	0.74	0.58	0.74
LB	0.85	0.69	0.81	0.69	0.80	0.77
AB	0.82	0.78	0.78	0.66	0.80	0.77
NNge	0.78	0.54	0.71	0.65	0.56	0.65
DTNB	0.80	0.51	0.77	0.63	0.50	0.65
VFI	0.85	0.79	0.73	0.59	0.78	0.75

RQ4: What is the performance of ML techniques when across-release validation is used for predicting post-release defects?

A4: The performance of the ML techniques when across-release validation is used is comparable (and even better) to the performance of the ML techniques when 10-fold cross validation is used for defect prediction.

6. ACKNOWLEDGEMENTS

This work has been supported by grants from the University Grants Commission of India under Raman Fellowship for Post Doctoral Research.

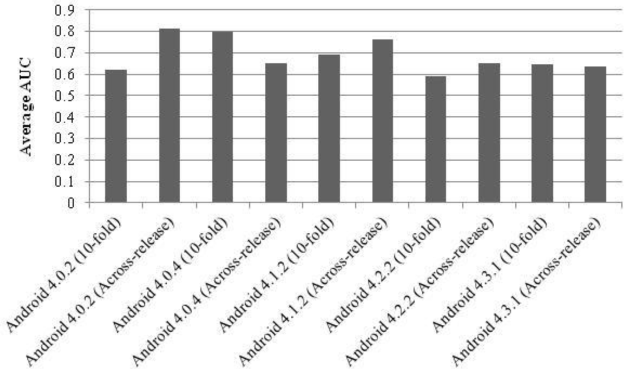

Figure 2: Comparison between AUC results of 10-fold and Across-release validation across five Releases of Android Data

7. CONCLUSIONS

The aim of this work was to comprehensively compare the performance of 18 ML techniques for defect prediction. Based on the data collected from six different releases of the 'MMS' application package of the Android OS, we used these ML techniques to investigate the relationship between the OO metrics and defect prediction.

The performance of the ML techniques are compared using the AUC measure. The main findings of the work are summarized below:

1) The Ce, LOC, LCOM3, CAM and DAM metrics were found to be significant predictors over the six releases of the Android data set by the CFS method.
2) The work confirms the overall predictive ability of the NB, AB, RBF, Bag, ADT, MLP, LB and RF techniques (AUC greater than 0.7 in most cases) for defect prediction.
3) The results of the Friedman test show the superiority of the NB technique in defect prediction.

4) The statistical test, Wilcoxon signed rank, show that there is significant difference between the performance of 112 pairs (out of 306) of the ML techniques, i.e., about 37%.
5) The results of across-release validation were comparable to the 10-fold validation. This confirms the generalizability of the models developed for defect prediction in this study.

Hence, we can conclude that ML models developed for defect prediction in this work can be used for identifying defective classes in the subsequent releases of the Android data set. The ML models can also be applied in future to different projects that are similar in nature. In future, we plan to predict severity levels of security related defects to provide practitioners an in depth knowledge about the problematic areas in software development.

8. REFERENCES

[1] Arisholm, E., Briand, L.C., Johannessen, E.B. 2010. A Systematic And Comprehensive Investigation Of Methods To Build And Evaluate Fault Prediction Models. *Journal of System and Software*. 83, 1, 2–17.

[2] Azar, D., Vybihal, J. 2011. An ant colony optimization algorithm to improve software quality prediction models: Case of class stability. *Information and Software Technology*, 53, 4, 388–393.

[3] Bansiya, J., Davis, A. 2002. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transactions on Software Engineering*. 28, 1, 4-17.

[4] De Carvalho, A.B., Pozo, A., Vergilio, S., Lenz, A. 2008. Predicting fault proneness of classes through a multiobjective particle swarm optimization algorithm. In *20th IEEE Int. Conf. Tools with Artif. Intell*. (2008) 387–394.

[5] Catal, C., Diri, B., Ozumut, B. 2007. An artificial immune system approach for fault prediction in object-oriented Software. In *2nd Int. Conf. Dependability Comput. Syst.* (2007) 1–8.

[6] Henderson-Sellers, B. 1996. *Object-oriented metrics: measures of complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

[7] Chidamber, S., Kemerer, C. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*. 20, 6, 476-493.

[8] Demšar, J. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*. 7, 1-30.

[9] Dejaeger, K., Verbraken, T., Baesens, B. 2013. Toward comprehensible software fault prediction models using Bayesian network classifiers. *IEEE Transactions on Software Engineering*. 39, 2, 237–257.

[10] Friedman, M. 1940. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics,* 11, 1, 86–92.

[11] Gyimothy, T., Ferenc, R., Siket, I. 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*. 31, 10, 897-910.

[12] Hall, M. 2007. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the 17th Int. Conference on Machine Learning*. 359–366.

[13] Hanley, J., McNeil, B.J. 1982. The meaning and use of the area under a Receiver Operating Characteristic ROC curve. *Radiology*. 143, 1, 29-36.

[14] Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V., Thambidurai, P. 2007. Object-oriented software fault prediction using neural networks. *Information and Software Technology*. 49, 5, 483–492.

[15] Lessmann, S., Baesens, B., Mues, C., Pietsch, S. 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*. 34, 4, 485–496.

[16] Malhotra, R. 2014. Comparative Analysis of statistical and machine learning methods for predicting faulty modules. *Appl. Soft Comput.* 21, 286-297.

[17] Malhotra, R., Kaur, A., Singh, Y. 2010. Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. *Int. J. Systems Assurance Eng. and Management.* 1, 3, 269-281.

[18] Malhotra, R., Nagpal, K., Upmanyu, P., Pritam, N. 2014. Defect Collection and Reporting System for Git Based Open Source Software. *In ICACCI* 2014.

[19] Martino, S., Ferrucci, F., Gravino, C., Sarro, F. 2011. A genetic algorithm to configure support vector machine for predicting fault prone components. *Product Focus Software Process Improvement*. 6759, 247-261.

[20] Menzies, T., Greenwald, J., Frank, A. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Trans. Software Eng*. 33, 1, 2-13.

[21] Myrtveit, I., Stensrud, E., Shepperd, M. 2005. Reliability and Validity in Comparative Studies of Software Prediction Models. *IEEE Trans. Software Eng*. 31, 5, 380-391.

[22] Okutan, A., Yıldız, O.T. 2012. Software defect prediction using Bayesian networks. *Empirical Software Engineering*. 19, 1, 154-181.

[23] Pai, G.J., Dugan, J.B. 2007. Empirical analysis of software fault content and fault proneness using Bayesian methods. *IEEE Transactions on Software Engineering*. 33, 10, 675–686.

[24] Singh, Y., Kaur, A., Malhotra, R. 2009. Application of Support Vector Machine to Predict Fault Prone Classes. *ACM SIGSOFT Software Engineering Notes.* 34, 1, 1-6.

[25] Singh, Y., Kaur, A., Malhotra, R. 2010. Empirical validation of object-oriented metrics for predicting fault proneness models. *Software Quality Journal*. 18, 1, 3-35.

[26] Stone, M., 1974. Cross-validatory choice and assessment of statistical predictions. *J. Royal Society Series* B. 36, 2 (December 1973), 111-114.

[27] Wilcoxon, F. 1945. *Individual comparisions by ranking methods*. Biometrics. 1:80-83.

[28] Yu, L., Mishra, A. 2012. Experience in Predicting Fault-Prone Software Modules Using Complexity Metrics. *Quality Technology & Quantitative Management*. 9, 4, 421-433.

[29] Zhou, Y., Leung, H. 2006. Empirical analysis of object-oriented design metrics for predicting high severity faults. *IEEE Transactions on Software Engineering*. 32, 10, 771-784.

[30] Zhou, Y., Xu, B., Leung, H. 2010. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. *The Journal of Systems and Software*. 83, 4, 660-674.