J Inf Process Syst, Vol.14, No.3, pp.751~770, June 2018
https://doi.org/10.3745/JIPS.04.0077

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Analyzing Machine Learning Techniques for Fault Prediction Using Web Applications

Ruchika Malhotra* and Anjali Sharma*,**

Abstract

Web applications are indispensable in the software industry and continuously evolve either meeting a newer criteria and/or including new functionalities. However, despite assuring quality via testing, what hinders a straightforward development is the presence of defects. Several factors contribute to defects and are often minimized at high expense in terms of man-hours. Thus, detection of fault proneness in early phases of software development is important. Therefore, a fault prediction model for identifying fault-prone classes in a web application is highly desired. In this work, we compare 14 machine learning techniques to analyse the relationship between object oriented metrics and fault prediction in web applications. The study is carried out using various releases of Apache Click and Apache Rave datasets. En-route to the predictive analysis, the input basis set for each release is first optimized using filter based correlation feature selection (CFS) method. It is found that the LCOM3, WMC, NPM and DAM metrics are the most significant predictors. The statistical analysis of these metrics also finds good conformity with the CFS evaluation and affirms the role of these metrics in the defect prediction of web applications. The overall predictive ability of different fault prediction models is first ranked using Friedman technique and then statistically compared using Nemenyi post-hoc analysis. The results not only upholds the predictive capability of machine learning models for faulty classes using web applications, but also finds that ensemble algorithms are most appropriate for defect prediction in Apache datasets. Further, we also derive a consensus between the metrics selected by the CFS technique and the statistical analysis of the datasets.

Keywords

Empirical Validation, Fault prediction, Machine Learning, Object-Oriented Metrics, Web Application Quality

1. Introduction

Detecting defects, adopting corrective measures and providing preventive solutions are essentials of software development. When done in a coherent and methodological way, this not only improves the reliability of the software, but also helps in reducing the development costs and further enhancements [1]. However, many factors associated with software code development make defects inevitable. Thousands of lines of code, sourced by a team of coders are highly susceptible to defects. Use of third party source codes, such as functions, subroutines, libraries, etc., also adds to defect vulnerabilities. Besides, an existing code subjected to several modifications and enhancements to meet the new criteria

Manuscript received April 1, 2016; first revision March 3, 2017; accepted April 9, 2017.
Corresponding Author: Anjali Sharma (anjali1007@gmail.com)
* Dept. of Computer Science and Engineering, Delhi Technological University, Delhi, India (ruchikamalhotra2004@yahoo.com)
** Dept. of Planning Monitoring and Evaluation, CSIR-National Physical Laboratory, Delhi, India (anjali@nplindia.org)

and/or to enable new functionality also have high possibility of defect occurrence [2]. However, a significant reduction in the defects can be accomplished with the aid of defect detection solutions. While conventional approaches may favor a critical analysis of the code by segmentation, choosing an advanced programming language, improve developer training etc., there exist alternative automated ways [3]. One such is the use of metrics and machine learning techniques to build predictive models which help identify defects leading to fault prone modules with a certain level of confidence [4-8].

Along these guidelines that machine learning algorithms would suit the need, we attempt to identify defects in web applications using object oriented metrics design suite [7,9,10]. We adopt to the most popular 14 machine learning techniques and apply the methodology on to three releases of Apache Click and four versions of Apache Rave web application projects. The results are evaluated using area under curve (AUC) obtained from the receiver operating characteristics (ROC) analysis. [11].

In general, machine learning models have been extensively used across various disciplines with varying degree of successes. It has become obvious that the choice of the metrics is crucial, and an optimized metric set not only provide faster results but also provide better accuracy and reliability. For the same, we have used the filter methodology [12,13] as implemented in Weka3.7 [14], based on correlation based statistics. We note that the correlation based feature selection technique have been widely used across various disciplines [15-17], thereby finding it to be universally acceptable.

Primarily, the present work emphasizes on the statistical analysis of the metrics distributions across the various releases and, using rule and ensemble based machine learning techniques to identify fault-prone classes in Apache datasets. Of much importance, we also find that the metrics identified using the correlated feature selection renders better defect prediction models.

Beyond, minimal information exists on how a particular machine learning algorithm depends on the nature and distribution of the chosen metrics data. This is partly evident from the variations in the prediction by various machine learning algorithms on a given dataset and also by a particular algorithm on statistically different datasets. Likewise, we also note that the varying degree of performance could also well depend on the choice of the metrics, as well. In these perspectives, we have considered machine learning techniques that are based on parametric, non-parametric and ensemble algorithms.

Our discussion of the algorithm and methodology is detailed in Section 4. A priori, in Section 2 we first summarize the related work in defect detection using predictive techniques which form the motivation to the current work. In Section 3 we outline our current research work with details of the independent and dependent variables, selection of applications, procedure of the dataset collection and description, machine learning techniques and their performance indicators. Section 5 details the result with discussions, and in Section 6 we test the validity of the approach. Finally, in Section 7 we summarize our work, stating future directions.

2. Literature Review

A wide range of statistical and machine learning models exist to predict defect modules in a given software. Statistical techniques such as univariate and multivariate logistic regression (LR) and machine learning techniques such as artificial neural network (ANN), support vector machines (SVM), Bayesian network (BN) and many more have been proposed [18-20]. The correlation between software metrics

and fault-proneness had also been studied using many models [7,10,21]. Arisholm et al. [22] compared variants of Decision Tree (DT) techniques with neural networks, SVM and LR techniques on the Java Telecom system and found the DT based technique (C4.5) to yield better results. Consistent with the earlier findings of Lessmann [4], the authors suggest that the choice of the classification algorithm for fault proneness is seldom important. We note that the work of Lessmann [4] was based on the traditional McCabe [23,24] and Halstead [25] metrics and, used analysis of variance (ANOVA) for statistical comparison of classification models. Earlier, in their review on software fault prediction studies, Catal and Diri [26] emphasized on the need for more studies using class-level metrics and machine learning algorithms. Their work also emphasized that fault proneness prediction studies provide more useful information with public datasets.

Table 1. Object-oriented metrics used in the study

Abbreviation	Metric name	Definition
WMC	Weighted methods per class	Sum of the complexities of all methods in a class.
DIT	Depth of inheritance tree	Sum of the count of the classes that a particular class inherits from.
NOC	Number of children	Number of immediate sub classes of a given class.
CBO	Coupling between objects	Number of classes that are coupled to a given class
RFC	Response for a class	Sum of all the internal and external methods in a given class.
LCOM	Lack of cohesion amongst methods	Count of null pair of methods not sharing common instance variables.
Ca	Afferent couplings	The number of classes outside this category that depend upon classes within this category.
Ce	Efferent couplings	The number of classes inside this category that depends upon classes outside this category.
NPM	Number of public methods	Count of public methods in a given class
LCOM3	Lack of cohesion amongst methods	Henderson-Sellers revision of LCOM to remove dependency on number of method pairs in a class.
LOC	Lines of code	Count of lines of code in a given class.
DAM	Data access metrics	Ratio of private (and/or protected) attributes to the total attributes in a given class.
MOA	Measure of aggregation	Percentage of user defined data declarations in a class
MFA	Measure of functional abstraction	Ratio of the number of inherited methods to the total number of methods of a given class.
CAM	Cohesion among methods of class	Similarity among methods of a class based upon their parameter list.
IC	Inheritance coupling	Count of number of parent classes to which a given class is coupled.
CBM	Coupling between methods	Total number of new/redefined methods coupled with all the inherited methods.
AMC	Average method complexity	The average method size for each class.

De Carvalho et al. [18] using multi-objective particle swarm optimization (MOPSO-N) technique [27,28] with six C&K design metrics (refer Table 1 for definitions) found that RFC, WMC, CBO and LCOM are the important object oriented metrics for indicating fault in a class. The results were compared with seven other machine learning methods using Wilcoxon test [29]. The authors observed that the results generated with the MOPSO-N technique was at par with the ANN and BN techniques, and the SVM algorithm yielded the lowest performance. On the other hand, Singh et al. [30] using a similar set of object oriented metrics found that SVM technique is rather a robust technique for fault prediction. Nevertheless, a consensus that emerged from either works was that the NOC metric could not be considered as a reliable feature for fault prediction. Similar conclusion on the relevance of NOC metric was also pointed out by Gyimothy et al. [5] and Olague et al. [31]. We also note that the irrelevance of NOC metric in fault proneness was found by univariate analysis [31] and not by any feature selection method.

Further, Catal et al. [19] used NASA KC1 data set to analyze the artificial immune recognition system (AIRS) and Bayesian approach, for fault prediction. Although the authors conducted no statistically significant tests, they selected the features using the popular correlation based feature selection method. The most salient finding was, that CBO was identified as an important metric for fault prediction. On the other hand, the study by Pai and Dugan [32] showed that apart from CBO, SLOC, WMC, and RFC were also equally significant, and that neither DIT nor NOC were significant. The significance of LCOM, however, appeared to be model dependent.

Kanmani et al. [33] compared ANN techniques with that of statistical techniques in the software system written in the Java language. The findings of the study revealed that neural network based fault prediction models perform better than the statistical techniques. Azar and Vybihal [34] found ant colony Optimization (ACO) technique to be better than both decision tree (C4.5) and random guessing techniques using C&K metrics. The Wilcoxon test was used for comparison. Di Martino et al. [35] configured SVM with a genetic algorithm for prediction of faulty classes on the basis of object oriented metrics and compared the results with optimization of SVM using Grid search. Their results showed that the genetic algorithm yielded better results for configuration of SVM parameters.

Okutan and Yildiz [36] used Bayesian networks to evaluate the relationship between C&K metrics and defect proneness. It was found that NOC and DIT are not effective metrics for defect prediction, but LOC, CBO, RFC and WMC play an important role in identifying faults-prone classes. Zhou et al. [37,38] utilized C&K design metrics of NASA data set to establish their relation with fault-prone classes when fault severity is taken into account. Their findings indicated that the design metrics were able to predict low severity faults better than high severity faults in fault-prone classes. D'Ambros et al. [8] evaluated various defect prediction approaches across different systems. However, the authors expressed the need for more detailed case studies on different datasets as the external validity in defect prediction was found to be difficult. Bowes et al. [39] introduced mutation-aware fault prediction models using LR, RF, NB and J48 and indicated that the best performance is obtained using a combination of both static and dynamic mutation metrics. However, the performance of the classification models was measured using Mathews correlation coefficient (MCC). In a recent work, Malhotra and Raje [40], investigated the Android dataset to predict defective classes using the object oriented metrics. Their findings showed that Ce, LOC, LCOM3, CAM and DAM to be significant predictors and that the naïve Bayes algorithm was identified as an important machine learning algorithm.

Thus, on a very general consensus, it appears that no generalization could be derived on the choice of the machine learning algorithms, choice of the features either by feature selection techniques or univariate analysis, for fault proneness. Thus, it becomes quite essential to perform more investigations on varying datasets, both public and private. To the best of our knowledge, none of the above studies have been conducted on the widely used web application framework like Apache, using algorithms that are based on statistical, rule-based and ensemble machine learning techniques. In this study, we analyze the relationship between object oriented metrics and machine learning techniques using web applications. The performance of 14 machine learning techniques (see Table 2) has been assessed and compared for defect prediction in classes of web applications. The statistical tests have been performed to obtain the statistical significant differences among the machine learning techniques on various releases of Apache Click and Rave dataset.

3. Research Background

3.1 Independent Variables

The independent variables of this study are object oriented design metrics suite computed on each Java file of the project using the defect collection and reporting (DCRS) [41] which has been developed in the Java programming language at the Delhi Technological University. The metrics used in the study are listed in Table 1.

3.2 Dependent Variable

The dependent variable analyzed in this study is the identification of fault prone classes. It represents the likelihood of defects in a class after the release of the software. The observation of classes which are found to be defective helps in the competent allocation of constraint resources during testing.

3.3 Selection of Applications

As mentioned earlier, we focus on identifying fault prone classes of web applications. In order to develop reliable predictive models, one needs multiple versions of the application with moderate number of classes. The study uses Apache Click and Apache Rave open source projects developed under the Apache Software Foundation (ASF) process. The ASF projects reliably link Git commits to closed bugs in the issue tracker, resulting in high quality data for building defect prediction models. The Apache Click and Apache Rave are large web projects developed using Java with more than three hundred classes in each release and with at least three releases. The Apache Click is a J2EE web application framework providing an easy to learn client style programming model. On the other hand, Apache Rave aims to provide a social mash-up engine to support web widgets for internet as well as intranet, and is in its early development phase.

3.4 Feature Selection

Feature selection is the process of selecting the most discriminatory features out of the available ones [42] and is considered as a crucial procedure in machine learning problems. While for an accurate and

precise predictions all features may look important, in general it may serve as an inappropriate methodology yielding poor outcomes. For instance, a large feature set will certainly make the problem computationally cumbersome. Beyond, a raw collection of features also may lead to information redundancy and increase the complexity of the prediction models. For the entire process of machine learning aided predictions, cost effectiveness demands an optimization effort in data acquisition and processing, prior subjective to the prediction models. It is now well known that features which are correlated in the input dataset not only lead to ambiguous predictions, but also affect the generalization capability of the machine learning algorithms.

In general, there exist two mechanisms for feature selection. They are the wrapper and the filter based methods. While, the wrappers use the classifier at hand to select the feature subset, the filter method optimizes the features independently of the classifier. In fact, the filter methods which are independent of the classifiers either use the probability based distance approaches such as the Bhattacharyya distance [43], the Chernoff distance [44], the Patrick Fisher distance [45], or the correlation based approach [12,13]. The choice of feature selection, however, depends on the problem at hand. It has been discussed previously that since the correlation based feature selection makes use of all the training data at once, it can give better results than the wrapper on small dataset [12,13]. In other words, a feature selection method which would render high reliability in detecting defects in web applications is preferred to have the following characteristics: (i) it should not only scale, but also must lead to high predictability for a large number of web applications, (ii) it should be independent of having an explicit class labeling, (iii) since classification of the web metrics is not the goal, the feature selection process should not assume the use of a specific classifier, and (iv) it should have the good performance among the methods satisfying the above conditions.

Since, the present study of fault proneness predictions rely on Apache dataset using a variety of machine learning algorithms, we adopt to the filter methods. Further, we also note that a comparative study of 32 feature selection methods on defect prediction performance had been carried out by Xu et al. [46] using feature ranking, wrapper based and, filter based feature evaluation techniques. The authors found that CFS unequivocally yields the best performance.

3.5 Performance Indicators

A variety of performance indicators, such as confusion matrix, gain and lift chart, Kolmogorov chart, Gini coefficient, concordant-discordant ratio, ROC, root mean squared error, etc., have been used to evaluate the predictive capability of models developed using machine learning techniques. In general, the defect dataset has a disproportionate ratio of faulty and non faulty classes and is imbalanced in nature. The ROC is the commonly used performance measure to deal with the imbalanced property of the dataset. The ROC curve represents the correctly predicted faulty classes (sensitivity) on the y-coordinate versus the one minus the percentage of correctly predicted non-faulty classes (1-specificity) on the x-coordinate. The optimal cutoff point that maximizes both sensitivity and specificity is determined using the ROC curve. The comparative performance analysis of each machine learning technique is evaluated using ROC curves.

The AUC is the value of the area under the ROC curve and, its value lies between zero and one. It is a combined measure of sensitivity and specificity and, is used to compute the accuracy of the predicted models. The higher the value of AUC, better is the predictive capability of the model. The AUC is

insensitive to the effects of noise and imbalanced dataset. Hence it is advantageous to use AUC for performance evaluation of the predictive models.

3.6 Validation Methods

The practical understanding on the accuracy of the model can be predicted by applying it to the different data sets other than from which it is built. Therefore, we performed a 10 cross-validation of the models. Each dataset is randomly divided into 10 equal subsets. Each time one of the 10 subsets is used as the test set and the other 9 subsets are used to form a training set. The process is repeated 10 times and the results from all the folds are combined to produce model result [47].

3.7 Machine Learning Techniques

We have used machine learning techniques for building prediction models. A set of feature vectors (object oriented metrics described in section 'Independent Variables') and the corresponding labels (either faulty or non-faulty) are used as the training set to build the fault prediction model. The model is then applied to a different set of feature vectors called the testing set. The classification of classes with the corresponding faulty or non-faulty labels for the testing is compared with the real labels to compute the performance indicators as explained in Section 3.6.

Table 2. Machine learning techniques used in the study

		Description
Statistical classifiers	BN (Bayes Net) LR (Logistic Regression)	The classification of NB and BN depends on the Bayes theorem of attaching the prior distribution and the likelihood of the observed data in order to derive the posterior distribution. NB assumes the attributes to be conditionally independent while BN also takes into consideration the correlation among the attributes. LR is a regression model where classification is done by estimating probabilities using a cumulative logistic distribution.
Decision tree methods	DT (Decision Tree) REPT (Reduced Error Pruning Tree) RT (Random Tree) J48 (C4.5 based technique)	Decision Tree classification is based on minimization of the generalization error. DT models the complex rules and their corresponding actions. REPT is a fast decision tree learner, which uses information gain for splitting and variance reduction for pruning. RT uses regression tree logic to create multiple trees in different iterations and selects the best one from all generated trees by calculating mean square error. C4.5 is a top-down greedy algorithm based on entropy and gain ratio for growing and pruning.
Support Vector Machines	VP (Voted Perceptron) SMO (Sequential Minimal Optimization)	SVM is a hyperplane that separates positive examples from negative examples with maximum margin. Training of SVM requires solving of a large quadratic programming (QP) optimization problem. SMO solves QP problem by breaking it into smallest possible QP problems and solving them analytically. The polynomial voted method is used in VP.
Neural networks	MLP (Multilayer Perceptron)	A MLP is an artificial neural network model that consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. It maps sets of input data onto a set of appropriate classes by concatenation of weighting, aggregation and thresholding functions.

		Description
Ensemble learning	Bag (Bagging) RF (Random Forest) LB (Logistic Boost) AB (Adaptive Boosting)	Ensemble learning is meta-learning technique that uses voting process. Bagging uses different random sampling of the training set for each individual classifier in the ensemble to compensate for the increase in error rate of any individual classifier. RF is a collection of decision trees and uses voting to obtain final class prediction and hence improves the classification rate. AB and LB uses combined weighted sum to represent the final output of boosted classifier. LB applies logistic regression techniques to the AB method for minimization of weighted least-squares error of weak learners.

The performance of machine learning techniques depends on the properties of the data to be classified. In Table 2 we present the summary of machine learning techniques used. The experiments are conducted with a Weka3.7 tool to build the predictive models by using machine learning techniques implemented with the default parameter settings. Quite differently, few recent studies [35,48,49] have emphasized on the importance of parameter tuning using heuristic techniques like genetic algorithms and differential evolution. It is argued that such tuning techniques can provide better prediction results [35,48,49]. Nevertheless, it has been also stated by Fu et al. [48], that parameter tuning is required to be repeated for any change in data, and that different tuning algorithms result in different optimized parameter values. Therefore, the parameter tuning technique eventually leads the defect prediction model to be short in attaining universality. Also, it may also be noted that the tuned parameter technique as mentioned in [48] is likely to overstate the results, if the goals are improperly defined. Beyond, Arcuri and Fraser [50] have shown that parameter tuning has very sensitive effects on the external validity of the results by using search based techniques. Thus, given that parameter tuning addresses a defect detection problem on a very local scale, we adapt to the default parameters as supplemented by the Weka suite of programs, so as to have a wider applicability, reproducibility, inter-data comparison and generality in web applications.

3.8 Statistical Testing

The statistical difference between various machine learning techniques is computed using Friedman test [51]. It is a non parametric test, used to rank a set of techniques over multiple data sets. The Friedman test is based on two hypotheses:

Null Hypothesis (Ho): There is no significant difference between the performances of the compared techniques.

Alternative Hypothesis (H1): There exists a significant difference between the performances of the comparative techniques.

The Friedman measure is defined as follows:

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

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 is distributed over (k-1) degrees of freedom. If the value of Friedman measure is in the critical region (obtained from χ^2 with a specific level of

significance, i.e., 0.01 or 0.05 and (k-1) degrees of freedom), then the Null hypothesis is rejected and it is concluded that there is a difference between the performance of comparable techniques, else Null hypothesis is accepted. If the Null hypothesis is rejected after applying the Friedman test, we perform post-hoc analysis using Nemenyi test [52]. It is a non-parametric test that performs pairwise comparisons of the difference in performance of the techniques. The critical difference (CD) is calculated using the following formula. The SPSS version 16 for Windows (SPSS Inc., Chicago, IL, USA) is used for applying Friedman and Nemenyi tests.

$$CD = q\sqrt{\frac{k(k+1)}{6n}}$$

3.9 Data Description

The class-defect characteristics of the three releases of Apache Click and four releases of Apache Rave web applications are provided in Tables 3 and 4, respectively. The respective tables, lists the number of classes of each version, size, number of faults, faulty class percentage and the name of the software along with the release of the software under which fault was fixed to the immediate subsequent release.

Table 3. Apache Click data set characteristics

Software	Total classes	Total LOC	No. of faults	Faulty class (%)
Click 2.0–2.1	336	20151	65	13.3
Click 2.1–2.2	389	23816	315	40.3
Click 2.2–2.3	402	24506	110	20.9

Table 4. Apache Rave data set characteristics

Software	Total classes	Total LOC	No. of faults	Faulty class (%)
Rave 0.12–0.13	435	21955	634	47.1
Rave 0.16–0.17	509	27308	126	17.49
Rave 0.20.1–0.21.1	642	36480	4604	96.26
Rave 0.22–0.23	685	37928	295	29.34

3.10 Data Collection Method

In order to collect data points from each software project, the source code of different releases of Apache Click and Rave applications developed in Java language has been obtained from GitHub repository https://github.com/apache/click and https://github.com/apache/rave, respectively. The faults were collected from the defect logs by using DCRS [40], which mines the change logs of two pre-determined consecutive releases of software. In this study, defects incurred from the immediate previous release and the subsequent ones are taken. The collected faults are then mapped to the classes in the source code. We also collected a binary variable named "FAULTY" which is true ("YES") if the count of the total number of faults in the class is non-zero, or false ("NO") otherwise.

4. Research Methodology

In this section, we elaborate on the approach that has been used in this work in order to achieve the prediction of fault in a class using object oriented metrics. Following are the necessary steps (depicted in Fig. 1), which we incorporate in our approach for model prediction:

- The change logs maintained by different software repositories corresponding to different software are analyzed.
- The object oriented metrics and fault data is extracted from the reports using DCRS module.
- The faults are associated with the corresponding classes of the software module.
- The fault prediction models are built by applying various machine learning techniques in order to conduct an extensive empirical study for prediction of faulty classes.
- The models are validated using 10-fold cross method.
- The proposed models are evaluated using appropriate performance evaluation measures.

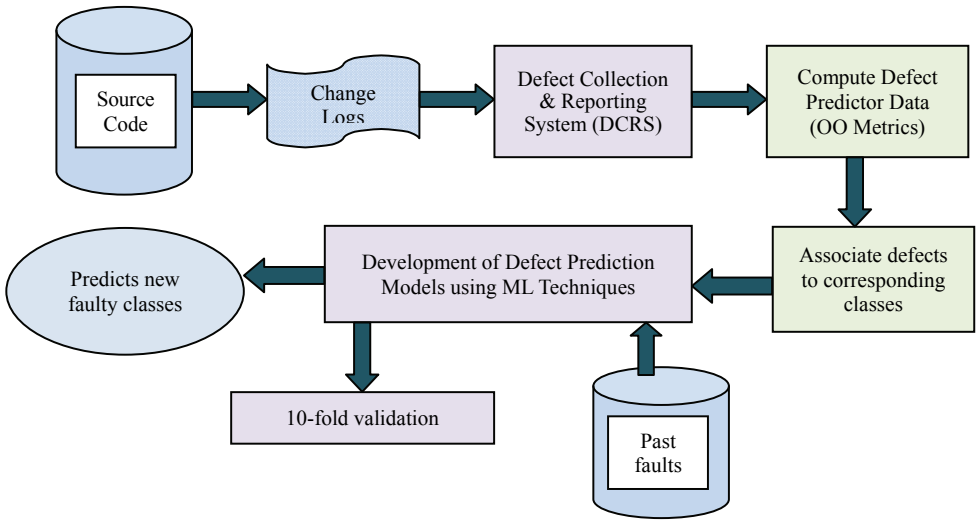

Fig. 1. Schematic representation of the research methodology adopted in this work.

4.1 Research Questions

We investigate the following research questions:

- RQ.1: Which object oriented metrics serve as good indicators of faults in a class?
- RQ.2: What is the overall performance of the statistical and machine learning techniques for the prediction of fault prone classes on Apache Click and Apache Rave datasets?
- RQ.3: Which is the best predictive technique for identifying fault prone classes?
- RQ.4: Which pair of machine learning techniques is significantly different from one another for prediction of fault prone classes in web applications?

4.2 Descriptive Statistics

The maximum (max), minimum (min) and mean values for each object oriented metric from the selected versions of Apache Click and Apache Rave projects, are shown in Tables 5 and 6, respectively.

We attempt to make a qualitative inference on the nature and impact of the object oriented metrics from the data shown in Tables 5 and 6. In general, a high value associated with WMC has been anticipated to yield more faults [53]. It may be noted that there exists no well defined WMC limit values for fault predictions. However, it is evident from Tables 5 and 6, that with regard to WMC metric, Apache Click is anticipated to have a lesser fault proneness in comparison to Apache Rave. For Apache Click and Apache Rave, the WMC data suggest a spread over the range 0–95 and 0–142, respectively, although with a comparable mean value.

The values associated with DIT are found to be less than the recommended value of 5 [54]. A high DIT is anticipated to increase faults. From the dataset, we find a maximum (minimum) of 3 (2) for Apache Click, while for Apache Rave it has been determined as 4. These values empirically suggest that DIT metric may not be quite detrimental to this case study. Apart from DIT, which measures the depth of inheritance, an important and closely associated metric is the NOC, the latter which measures the breadth of the class hierarchy. The dataset shows that Apache Click has larger NOC (11), in comparison to Apache Rave (3). In general, high NOC is found to indicate fewer faults.

In the C&K metric suite, the number of classes to which a class is coupled is determined by the CBO metric. High CBO is found to be undesirable, as excessive coupling between classes prevent reuse. From Tables 5 and 6, we find the maximum value of CBO associated with the Apache Click to be 13, while for Apache Rave versions 0.12–0.13 and 0.16–0.17 as 9, and 7 for 0.20.1–0.21.1 releases. A high value of 27 is determined for the latest version 0.22.1–0.23, suggesting it to be highly fault prone with regard to the CBO metric. However, in comparison to the data shown in Tables 5 and 6, we find that the fault class percentage associated with Apache Rave version 0.20.1–0.21.1 is highest (96.26%), which is in contrast with the empirical predictions.

Table 5. Statistical description of Apache Click dataset

Metrics	Click 2.0–2.1			Click 2.1–2.2			Click 2.2–2.3		
	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean
WMC	93	0	9.64	93	0	9.81	97	0	9.74
DIT	2	0	0.70	2	0	0.70	3	0	0.72
NOC	11	0	0.13	11	0	0.13	11	0	0.13
CBO	13	0	0.77	13	0	0.82	13	0	0.82
RFC	94	0	10.64	94	0	10.80	98	0	10.73
LCOM	4278	0	128.86	4278	0	127.35	4656	0	130.92
Ca	12	0	0.30	12	0	0.34	12	0	0.33
Ce	5	0	0.51	7	0	0.53	7	0	0.53
NPM	87	0	8.42	87	0	8.44	91	0	8.38
LCOM3	2	0	1.54	2	0	1.50	2	0	1.50
LOC	581	0	59.97	581	0	61.22	607	0	60.96
DAM	1	0	0.47	1	0	0.62	1	0	0.67
MOA	3	0	0.05	4	0	0.07	4	0	0.07
MFA	0.38	0	0	0.383	0	0	0.72	0	0
CAM	1	0	0.68	1	0	0.65	1	0	0.66
IC	1	0	0	1	0	0	1	0	0
CBM	9	0	0.03	9	0	0.02	9	0	0.03
AMC	5	0	3.38	5	0	3.40	5	0	3.42

Table 6. Statistical description of Apache Rave dataset

Metrics	Rave 0.12–0.13			Rave 0.16–0.17			Rave 0.20.1–0.21.1			Rave 0.22.1–0.23		
	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean
WMC	142	0	8.2	142	0	9.27	142	0	9.81	142	0	9.65
DIT	4	0	1.02	4	0	1.02	4	0	1.01	4	0	1.01
NOC	3	0	0.01	3	0	0.02	3	0	0.02	3	0	0.01
CBO	9	0	0.27	9	0	0.34	7	0	0.27	27	0	0.34
RFC	143	0	9.19	143	0	10.26	143	0	10.79	143	0	10.6
LCOM	10011	0	85.44	10011	0	95.08	10011	0	119.8	10011	0	114.5
Ca	8	0	0.11	8	0	0.15	7	0	0.12	27	0	0.15
Ce	3	0	0.16	3	0	0.19	3	0	0.16	3	0	0.18
NPM	124	0	7.25	124	0	8.31	124	0	8.84	124	0	8.69
LCOM3	2	0	1.47	2	0	1.42	2	0	1.41	2	0	1.41
LOC	858	0	50.47	858	0	53.65	858	0	56.82	858	0	55.37
DAM	1	0	0.62	1	0	0.64	1	0	0.63	1	0	0.62
MOA	4	0	0.09	4	0	0.08	6	0	0.06	6	0	0.05
MFA	1	0	0.021	1	0	0.02	1	0	0.02	1	0	0.01
CAM	1	0	0.68	1	0	0.63	1	0	0.64	1	0	0.62
IC	1	0	0.048	1	0	0.07	1	0	0.07	1	0	0.06
CBM	1	0	0.048	1	0	0.07	1	0	0.07	1	0	0.06
AMC	5	0	4.42	5	0	4.24	5	0	4.31	5	0	4.22

Studies also reveal that the number of public methods (NPM) used, also effectively serve as a good indicator to fault predictions. From the works of Shah et al. [55], it has been found that for medium and large softwares categorized by its size, NPM plays a significant role. Our dataset shows that NPM varies between 87 and 91 for Apache Click versions, while being 124 among the Apache Rave versions. These high values of NPM are suggestive that the respective class may be split for optimal performance [55]. Among the other metrics, proposed by Bansiya and Davis [9], for fault proneness are DAM and, CAM which also serve as good indicators. In case of DAM, which are in the range [0, 1], a high value is generally desired. We find that the average value of DAM for the latest two versions of Apache Click and Apache Rave is approximately 0.6 or above. Similarly the average value of CAM, the statistical mean is determined to be 0.6 or above for both Apache Click and Apache Rave. It may be noted that the preferred value of CAM is close to 1.

Few other metrics also indicate that Apache Rave is relatively more fault prone than Apache Click. For example, the RFC, which represents the response function of a class is found to be 143 for Apache Rave, while 94–98 in the Apache Click. In general, classes with high RFC pose complexity in reading, testing and debugging. Although, no value makes a quantitative judgment on fault proneness with respect to the RFC metric [53], in the present case the high RFC values associated with Apache Rave certainly indicates to its instability with respect to Apache Click application.

LCOM is yet another metric that help in determining the fault proneness. Based on the nature and applicability of the object oriented suite of programs, four variants of LCOM have been proposed. Here, we emphasize on LCOM and LCOM3. Following Tables 5 and 6, we find the LCOM to be as high as 4000 or more for Apache Click, and more than 10000 for Apache Rave. However, when one considers the average value, LCOM shows a higher value for Apache Click (approximately 130) than for Apache Rave. The latter shows an increase in the mean value varying from (LCOM) mean = 85 for version

0.12–0.13 and (LCOM) mean = 114 for the 0.22.1– 0.22 versions. Here also, a high LCOM indicate to greater fault proneness. However, it may be noted that the validity of LCOM to be used as an indicator metric for fault proneness has been criticized previously [32]. For instance, it has been argued that for classes which use data that are generated by its own properties is likely to show high LCOM values. Such situations certainly are not problematic. A work around was to redefine the LCOM metric, which originally was based on the method-data interaction. The expression to calculate LCOM3 is given as,

$$LCOM3 = (m - \sum (mA)/a) / (m - 1),$$

where m and a are the number of procedures (methods) and variables (attributes) in a class. The quantity "mA" represent the number of methods that access a variable. In the above expression, mA is summed over all attributes of a given class. It is seen that LCOM3, for both Apache Click and Apache Rave, varies between 0–2. For LCOM3 = 0, it suggest to cases where each method access all variables, indicating highest possible cohesion and LCOM3 = 1 is suggestive of lack of cohesion of methods.

5. Analysis and Results

In this section, we present the results of the empirical comparison of machine learning techniques in terms of the AUC. The classifier models have been developed using independent variables described in Section 3.1. The independent variables were selected through the CFS technique to obtain better results. Table 7 presents the relevant metrics found in each release of Apache Click and Apache Rave datasets after applying the CFS technique. The results show that LCOM3, WMC, NPM and DAM were the most commonly selected object oriented metrics over the various releases of the Apache Click and Apache Rave data sets.

Table 7. Sub features selected

Data set	Features selected	
Click 2.0–2.1	NOC, LCOM3, CAM	
Click 2.1–2.2	WMC, DIT, CBO, NPM, LCOM3, DAM, AMC	*RQ1: Which Object Oriented metrics serve as good indicators of faults in a class?*
Click 2.2–2.3	WMC, CBO, Ce, NPM, DAM, CAM	
Rave 0.12–0.13	WMC, DIT, Ce, NPM, LCOM3, DAM, MFA, CAM	*A1:* ***LCOM3, WMC, NPM and DAM*** *are the most commonly selected OO metrics over the various releases of the Apache Click and Rave data sets.*
Rave 0.16–0.17	NPM	
Rave 0.20.1–0.21.1	WMC	
Rave 0.22–0.23	WMC, NPM, LCOM3, LOC, MFA	

As discussed earlier, the machine learning classifiers were empirically evaluated using the AUC, which is capable of dealing with noise and unbalanced data [4]. Table 8 lists the 10-fold cross-validation results of 14 machine learning techniques on three and four releases of Apache Click and Apache Rave, respectively. The machine learning technique yielding relatively better AUC values, for a given version, is highlighted in bold. The results show that the prediction efficiency of the model using the MLP, LR, Bagging, and AB techniques have AUC greater than 0.6, corresponding to most of the releases of the Apache dataset. That, the statistical and ensemble based methods perform well in fault proneness

predictions also have been emphasized by Ghotra et al. [56]. The results demonstrated the findings using the NASA and PROMISE corpus dataset. Overall, this level of accuracy is also consistent with the findings of Menzies et al. [6], which reports that defect predictors are useful for identifying fault prone modules.

We note that various machine learning techniques predict the fault proneness of the Apache Click versions, quite well. The poor fault proneness rendered to the intermediate versions of the Apache Rave accounts to the limited number of features selected by the CFS scheme. Note that only NPM and WMC were found prominent for fault proneness by the CFS for Apache Rave versions 0.16–0.17 and 0.20.1–0.21.1, respectively. Thus, the results which indicate to only one feature selection for these intermediate versions of Apache Rave suggest a strong correlation between the features, which are problematic and harder to judge.

As evident from the results listed in Table 8, one finds that the relative performance of the machine learning algorithms is small across various versions of the Apache dataset. In order to verify that the observed performance differences between predictive models are not random, we choose Friedman test. Note that, the null hypothesis for Friedman test states that all machine learning classifiers are equivalent and hence their ranks should be equal. However, the Friedman test resulted in χ^2 value of 38.13 and FF value of 4.32 for 14 machine learning algorithms (k = 14) on the seven Apache dataset (N = 7). For a two-tailed test at the 0.05 level of significance, the critical value of $F_{k-1, \ldots (k-1)(N-1)}$ is determined to be 1.848. Thus, the null hypothesis is rejected. The average rank of each machine learning classifier is provided in Table 9. It suggests that MLP is the best technique for the development of fault prediction models for Apache dataset. Our findings are consistent with the works of Gyimothy et al. [5]. The models developed using rule based algorithms, such as DT and J4.8, were found to perform relatively poor.

Table 8. The 10-fold cross-validation results with respect to AUC of 14 machine learning techniques

	Click			Rave				
	2.0–2.1	**2.1–2.2**	**2.2–2.3**	**0.12–0.13**	**0.16–0.17**	**0.20.1–0.21.1**	**0.22.1–0.23**	
BN	0.672	0.75	0.72	0.632	0.495	0.449	0.614	*RQ2: What is the overall performance of the statistical and Machine Learning techniques for the prediction of fault prone classes on Apache Click and Apache Rave datasets?* *A2:* ***The AUC of most of the classifier models is 0.6, highlighting the predictive capability of machine learning techniques.***
NB	0.686	0.743	0.718	0.642	0.575	0.507	0.579	
LR	0.682	0.759	0.761	0.669	0.598	0.597	0.602	
MLP	0.688	0.768	0.761	0.661	0.605	0.573	0.635	
VP	0.498	0.726	0.574	0.633	0.5	0.498	0.498	
SMO	0.5	0.713	0.533	0.63	0.5	0.5	0.5	
DT	0.536	0.754	0.598	0.619	0.495	0.461	0.566	
RandomTree	0.517	0.711	0.677	0. 522	0.51	0.6	0.622	
RepTree	0.517	0.711	0.677	0.594	0.51	0.6	0.622	
RandomForest	0.603	0.756	0.725	0.605	0.495	0.449	0.495	
Bagging	0.709	0.766	0.805	0.62	0.493	0.47	0.671	
J48	0.517	0.737	0.76	0.605	0.495	0.449	0.495	
AB	0.646	0.755	0.805	0.618	0.606	0.51	0.606	
LB	0.65	0.74	0.794	0.609	0.576	0.584	0.631	

Table 9. Friedman test results of 14 machine learning techniques

	Average rank
MLP	2.71
LR	3.71
Bag	5.21
AB	5.21
LB	5.57
NB	6.29
BN	7.57
RandomTree	8.36
RepTree	8.93
DT	9.57
RF	9.71
VP	10.1
SMO	10.4
J48	11.57

RQ3: Which is the best predictive technique for identifying fault prone classes?

A3: The Friedman test result indicates that the MLP technique is relatively best among the 14 machine learning techniques for fault prediction models in Apache Click and Rave web applications. The LR, Bagging and AB techniques are among the next best performing techniques among fourteen machine learning techniques selected in this study.

In general, our findings corroborate with those of Ambros et al. [8]. In the latter, the authors using a regression model on Apache Lucene find that LR when applied to CK metric set gives an AUC value of 0.721. Consistently, our LR analysis on the Apache Click dataset gives an average of 0.734, while for Apache Rave the average AUC value across the four versions was determined to be 0.617. However, our results spans over 14 machine learning techniques, of which we find MLP yields the best performance. That, network based MLP is suited best for fault prediction has also been emphasized by Malhotra and Raje [40].

Following, we proceed with Nemenyi post-hoc test to detect fault prediction classifiers which differ significantly. As mentioned above, the Nemenyi post-hoc test compares all pairs of different classifiers and checks which model's performance differs significantly, i.e., exceed the CD) The Nemenyi test CD came out to be 5.353 at the significance level of 0.05. The results of the pairwise comparisons of the 14 ML techniques are shown in Table 10. The values which exceed the CD are highlighted in bold.

Table 10. Nemenyi post-hoc test results of 14 machine learning techniques

	NB	LR	MLP	VP	SMO	DT	RandomTree	RepTree	RF	Bag	J48	AB	LB
BN	1.28	3.86	4.86	2.53	2.83	2	0.79	1.36	2.14	2.36	4	2.36	2
NB	×	2.58	3.58	3.81	4.11	3.28	2.07	2.64	3.42	1.08	5.28	1.08	0.72
LR	×	×	1	**6.39**	**6.6**	**5.86**	4.65	5.22	6	1.5	**7.86**	1.5	1.86
MLP	×	×	×	**7.39**	**7.69**	**6.86**	**5.65**	**6.22**	**7**	2.5	**8.86**	2.5	2.86
VP	×	×	×	×	0.3	0.53	1.74	1.17	0.39	4.89	1.47	4.89	4.53
SMO	×	×	×	×	×	0.83	2.04	1.47	0.69	5.19	1.17	5.19	4.83
DT	×	×	×	×	×	×	1.21	0.64	0.14	4.36	2	4.36	4
RandomTree	×	×	×	×	×	×	×	0.57	1.35	3.15	3.21	3.15	2.79
RepTree	×	×	×	×	×	×	×	×	0.78	3.72	2.64	3.72	3.36
RF	×	×	×	×	×	×	×	×	×	4.5	1.86	4.5	4.14
Bag	×	×	×	×	×	×	×	×	×	X	**6.36**	0	0.36
J48	×	×	×	×	×	×	×	×	×	×	×	**6.36**	**6**
AB	×	×	×	×	×	×	×	×	×	×	×	×	0.36
LB	×	×	×	×	×	×	×	×	×	×	×	×	×

The results of Nemenyi test show that out of 14 machine learning techniques used in the study, the performance of J48 is significantly poor than LR, MLP, AB, LB and Bagging. Also, we find poor performance of VP, SMO, DT with LR, MLP, AB and LB. Therefore, we identify that statistical, MLP, and ensemble base approaches performed significantly better than rule based machine learning algorithms like J48, VP, SMO and DT. However, we find that the experimental data is not sufficient to reach any conclusion regarding the RandTree, REPTree, RF, NB, BN algorithms.

6. Threats to Validity

It is important to be conscious of the threats to the validity of the results obtained by conducting an empirical study in software engineering. The results obtained cannot be generalized as they depend on large number of project and environment specific context variables. In this study, we have analyzed seven releases of web applications with 14 machine learning techniques. One possible source of bias is the data used in the study. The data has been collected using DCRS tool and is placed on web for replication and comparison with other experiments. The set of object oriented metrics selected for this study is based on previous experiments [5,31,40]. The researchers may select different metrics collection for their studies.

The selection of applications for study considered number of classes and size of the code, which may be different for other researchers. The selection of classifiers is another possible source of bias. We have considered 14 machine learning techniques and there are still others that could have been studied. Our selection is guided by the aim of finding a meaningful balance between established techniques and novel approaches. We believe that the most important representatives of different domains (statistics, machine learning, and so forth) are included.

7. Conclusion and Future Directions

The underlying objectivity of the research is to comprehensively compare the performance of 14 machine learning techniques for fault prediction in web application, associated with the Apache Click and Apache Rave projects using object oriented metrics. En-route to the prediction of the defect proneness using various machine learning algorithms, which are based on parametric, non-parametric and ensemble based, an independent basis set was first refined using correlation based feature selection method. The models were thereafter validated using 10-fold cross method and were evaluated using the AUC performance measures. The main findings of the work are summarized below:

1) The LCOM3, WMC, NPM, and DAM object oriented metrics are found to be the significant predictors by using CFS, over the three and four releases of the Apache Click and Apache Rave data sets, respectively.
2) With its AUC being greater than 0.6, the work affirms the overall predictive ability of the MLP, LR, Bagging and AB techniques for fault prediction.
3) Following the Friedman test results, MLP appears as the most qualified methodology towards a quality fault prediction for Apache Click and Apache Rave dataset. Furthermore, the statistical post hoc Nemenyi test, indeed validates a significant pair wise difference between the performances of MLP with other machine learning techniques.

Hence, we conclude that machine learning models developed for fault prediction in this work can be successfully used for identifying faults in the subsequent releases of the Apache web application dataset. It is anticipated that these models could be also applied to different projects that are similar in nature. So as to derive an universality in default prediction across various dataset, we plan to carry out model predictions on search based techniques with different language environments, in future. A selection and detailed investigation of inter-project training data for cross project validation, is also proposed.

References

[1] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170-190, 2015.

[2] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, Cary, NC, 2012.

[3] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed. Boston, MA: Addison-Wesley, 2003.

[4] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: a proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485-496, 2008

[5] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897-910, 2005.

[6] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2-13, 2007.

[7] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1-30, 2006.

[8] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531-577, 2012.

[9] J. Bansiya, and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4-17, 2002.

[10] S. R. Chidamber and C. F. Kemerer "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.

[11] T. Fawcett, "ROC graphs: notes and practical considerations for researchers," *Machine Learning*, vol. 31, no. 1, pp. 1-38, 2004.

[12] M. A. Hall, "Correlation-based feature selection for machine learning," Ph.D dissertation, Department of Computer Science, The Waikato University, Hamilton, New Zealand, 1998.

[13] M. A. Hall and L. A. Smith, "Practical feature subset selection for machine learning," in *Proceedings of the 21st Australasian Computer Science Conference*, Perth, Australia, 1998, pp. 181-191.

[14] M. A. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10-18, 2009.

[15] Q. Song, J. Ni, and G. Wang, "A fast clustering-based feature subset selection algorithm for high-dimensional data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 1-14, 2013.

[16] V. Bolon-Canedo, N. Sanchez-Marono, A. Alonso-Betanzos, J. M. Benitez, and F. Herrera, "A review of microarray datasets and applied feature selection methods," *Information Sciences*, vol. 282, pp. 111-135, 2014.

[17] D. Liu, D. W. Sun, and X. A. Zeng, "Recent advances in wavelength selection techniques for hyperspectral image processing in the food industry," *Food and Bioprocess Technology*, vol. 7, no. 2, pp. 307-323, 2014.

[18] A. B. de Carvalho, A. Pozo, S. Vergilio, and A. Lenz, "Predicting fault proneness of classes through a multiobjective particle swarm optimization algorithm," in *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence*, Dayton, OH, 2008, pp. 387-394.

[19] C. Catal, B. Diri, and B. Ozumut, "An artificial immune system approach for fault prediction in object-oriented Software," in *Proceedings of the 2nd International Conference on Dependability of Computer System*, Szklarska, Poland, 2007, pp. 238-245.

[20] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Upper Saddle River, NJ: Prentice-Hall, 1996.

[21] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward comprehensible software fault prediction models using Bayesian network classifiers," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 237-257, 2013.

[22] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of System and Software*, vol. 83, no. 1, pp. 2-17, 2010.

[23] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, 1976.

[24] A. H. Watson, D. R. Wallace, and T. J. McCabe, *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Gaithersburg, MD: National Institute of Standards and Technology, 1996.

[25] M. H. Halstead, *Elements of Software Science*. New York, NY: North-Holland, 1977.

[26] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Systems Applications*, vol. 36, no. 4, pp. 7346-7354, 2009.

[27] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of 4th IEEE International Conference on Neural Networks*, Perth, Australia, 1995, pp. 1942-1948.

[28] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *Proceedings of IEEE International Conference on Evolutionary Computation*, Anchorage, AK, 1998, pp. 69-73.

[29] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80-83, 1945.

[30] Y. Singh, A. Kaur, and R. Malhotra, "Application of support vector machine to predict fault prone classes," *ACM SIGSOFT Software Engineering Notes*, vol. 34, no. 1, pp. 1-6, 2009.

[31] H. M. Olague, L. H. Etzkorn, S. Gholston, and 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 Transactions on software Engineering*, vol. 33, no. 6, pp. 402-419, 2007.

[32] G. J. Pai and J. B. Dugan, "Empirical analysis of software fault content and fault proneness using Bayesian methods," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 675-686, 2007.

[33] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," *Information and Software Technology*, vol. 49, no. 5, pp. 483-492, 2007.

[34] D. Azar and J. Vybihal, "An ant colony optimization algorithm to improve software quality prediction models: case of class stability," *Information and Software Technology*, vol. 53, no. 4, pp. 388-393, 2011.

[35] S. Di Martino, F. Ferrucci, C. Gravino, and F. Sarro, "A genetic algorithm to configure support vector machines for predicting fault-prone components," in *Product-Focused Software Process Improvement*. Heidelberg: Springer, 2011, pp. 247-261.

[36] A. Okutan and O. T. Yildiz, "Software defect prediction using Bayesian networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154-181, 2014.

[37] Y. Zhou, B. Xu, and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object oriented systems," *Journal of Systems and Software*, vol. 83, no. 4, pp. 660-674, 2010.

[38] Y. Zhou and H. Leung, "Empirical analysis of object oriented design metrics for predicting high severity faults," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771-784, 2006.

[39] D. Bowes, T. Hall, M. Harman, Y. Jia, F. Sarro, and F. Wu, "Mutation-aware fault prediction," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, Saarbrucken, Germany, 2016, pp. 330-341.

[40] R. Malhotra and R. Raje, "An empirical comparison of machine learning techniques for software defect prediction," in *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, Boston, MA, 2014, pp. 320-327.

[41] R. Malhotra, N. Pritam, K. Nagpal, and P. Upmanyu, "Defect collection and reporting system for Git based open source software," in *Proceedings of the International Conference on Data Mining and Intelligent Computing*, New Delhi, India, 2014, pp. 1-7.

[42] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[43] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society*, vol. 35, pp. 99-109, 1943.

[44] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *The Annals of Mathematical Statistics*, vol. 23, no. 4, pp. 493-507, 1952.

[45] E. Patrick and F. Fisher, "Nonparametric feature selection," *IEEE Transactions in Information Theory*, vol. 15, no. 4, pp. 577-584, 1969.

[46] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: an empirical comparison," in *Proceedings of the 27th International Symposium on Software Reliability Engineering*, Ottawa, Canada, 2016, pp. 309-320.

[47] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society Series B (Methodological)*, vol. 36, no. 2, pp. 111-114, 1974.

[48] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: is it really necessary?," in *Information and Software Technology*, vol. 76, pp. 135-146, 2016.

[49] F. Sarro, S. Di Martino, F. Ferrucci, and C. Gravino, "A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, Trento, Italy, 2012, pp. 1215-1220.

[50] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Search Based Software Engineering*. Heidelberg: Springer, 2011, pp. 33-47.

[51] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86-92, 1940.

[52] P. B. Nemenyi, "Distribution-free multiple comparisons," Ph.D. dissertation, Princeton University, Princeton, NJ, 1963.

[53] S. C. Misra and V. C. Bhavsar, "Relationships between selected software measures and latent bug-density: guidelines for improving quality," in *Computational Science and Its Applications*. Heidelberg: Springer, 2003, pp. 724-732.

[54] D. Glasberg, K. El Emam, W. Melo, and N. Madhavji, *Validating Object-Oriented Design Metrics on a Commercial Java Application*. Ottawa, Canada: National Research Council of Canada, 2000.

[55] S. M. A. Shah, M. Morisio, and M. Torchiano, "An overview of software defect density: a scoping study," in *Proceedings of the 19th Asia-Pacific Software Engineering Conference*, Hong Kong, China, 2012, pp. 406-415.

[56] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of the 37th International Conference on Software Engineering*, Florence, Italy, 2015, pp. 789-800.

Ruchika Malhotra https://orcid.org/0000-0003-3872-6213

She is Associate Head and Assistant Professor at the Department of Software Engineering, Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She has been awarded the prestigious UGC Raman Postdoctoral Fellowship by the Indian government for pursuing postdoctoral research from Department of Computer and Information Science, Indian University-Purdue University, Indianapolis, IN, USA. She was an assistant professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She received her Master's and Doctorate degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She has received IBM Faculty Award 2013. She has received Best Presenter Award in Workshop on Search Based Software Testing, ICSE 2014, Hyderabad, India. Her h-index is 21 as reported by Google Scholar. She is Executive Editor of *Software Engineering: An International Journal*. She is author of book titled "*Empirical Research in Software Engineering*" published by CRC Press, USA and co-author of a book on *Object Oriented Software Engineering* published by PHI Learning. Her research interests are in empirical research in software engineering, improving software quality, statistical and adaptive prediction models, software metrics, the search-based software engineering and software testing. She has published more than 140 research papers in international journals and conferences.

Anjali Sharma https://orcid.org/0000-0003-0389-046X

She is working at CSIR-National Physical Laboratory, Delhi, India as senior scientist and is currently pursuing her Doctoral degree from Delhi Technological University, Delhi, India. She has also worked in India Software Labs of IBM India Private Ltd. She has more than 12 years of experience in software, including program management, web applications development, ERP implementation, database administration, predictive modeling and data analysis. She completed her M.Tech. and B.E. degrees in computer science from Banadthali University, Rajasthan, India and Dr. Bhim Rao Ambedkar University, India, respectively. She has been conferred with individual 'CSIR-ERP Project Champion' Award in 2013 for exemplary performance by Department of Scientific & Industrial Research under Ministry of Science & Technology, Government of India. Her research interests are in software quality improvement, applications of machine learning techniques in defect prediction of web applications, and empirical analysis. She is a co-author of granted patents in the United States, Malaysia, Taiwan, and Singapore and has also published articles in international conferences and journals.