Neurocomputing 343 (2019) 120–140

Contents lists available at ScienceDirect

Neurocomputing

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

An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data

Ruchika Malhotra*, Shine Kamal

Discipline of Software Engineering, Department of Computer Science & Engineering, Delhi Technological University, Delhi, India

ARTICLE INFO

Article history:
Received 22 October 2017
Revised 19 February 2018
Accepted 21 April 2018
Available online 4 February 2019

Keywords:
Defect prediction
Imbalanced data
Oversampling methods
MetaCost learners
Machine learning techniques
Procedural metrics

ABSTRACT

Software defect prediction is important to identify defects in the early phases of software development life cycle. This early identification and thereby removal of software defects is crucial to yield a cost-effective and good quality software product. Though, previous studies have successfully used machine learning techniques for software defect prediction, these techniques yield biased results when applied on imbalanced data sets. An imbalanced data set has non-uniform class distribution with very few instances of a specific class as compared to that of the other class. Use of imbalanced datasets leads to off-target predictions of the minority class, which is generally considered to be more important than the majority class. Thus, handling imbalanced data effectively is crucial for successful development of a competent defect prediction model. This study evaluates the effectiveness of machine learning classifiers for software defect prediction on twelve imbalanced NASA datasets by application of sampling methods and cost sensitive classifiers. We investigate five existing oversampling methods, which replicate the instances of minority class and also propose a new method SPIDER3 by suggesting modifications in SPIDER2 oversampling method. Furthermore, the work evaluates the performance of MetaCost learners for cost sensitive learning on imbalanced datasets. The results show improvement in the prediction capability of machine learning classifiers with the use of oversampling methods. Furthermore, the proposed SPIDER3 method shows promising results.

1. Introduction

In recent years, many studies [1–8] have successfully developed software defect prediction models. These models predict the possibility of occurrence of a defect in the future or unseen versions of a software product. A software defect may cause software failure and forbids it to produce desirable outcomes, therefore an attempt for early detection of software defects is beneficial. Defects which are identified in the initial phases of the software development life cycle can be corrected appropriately. Such defects are prevented from propagation to later phases where they may become difficult to detect and costlier to correct. Thus, an accurate software defect prediction model aids in development of good quality software product with lower testing and maintenance costs and thereby satisfied customers. Defect prediction models are beneficial in assigning resources when they are at constraint. For instance, if a software manager has resources to rigorously test only 30% of software modules, then the top 30% of the modules that are predicted as defective can be tested. However, if resources are available, it is always a good engineering practice to test all the modules whether they are predicted defective or not. Software defect prediction models rely on past data and classify modules as defective or non- defective on the basis of this historically collected data. Previous studies have used various software metrics sets [1,3] along with defect data, to build prediction models which have proved reformative for predicting defect prone modules.

Previous research studies on defect prediction demonstrate that 80% of the defects occur in very few modules (20%) [9]. This indicates that defective classes are present in minority as compared to non-defective classes, which results in imbalanced datasets. In such cases, the distribution of classes is one-sided, which may result in incorrect prediction of the minority class instances. Although, the minority class instances are low in number, but it is important to classify them correctly. Incorrect prediction of defective classes might result in escape of critical errors leading to bad quality software and higher testing costs. Therefore, it is important to address imbalanced data problem for software defect prediction to improve software quality, to reduce prediction error and for successful deployment of the software.

* Corresponding author.
E-mail addresses: ruchikamalhotra@dtu.ac.in (R. Malhotra), kamalshine93@gmail.com (S. Kamal).

https://doi.org/10.1016/j.neucom.2018.04.090
0925-2312/

As discussed, handling imbalanced datasets to obtain improved results is an important challenge in software defect prediction area. Various methods have been developed to deal with imbalanced data like data sampling methods, cost sensitive learning, and ensemble methods [10–13]. Data sampling methods as mentioned by Lopez et al. [10], sample the data either by eradicating some of the majority class samples (under sampling) or by duplicating or adding new synthetic minority class samples (oversampling). Cost sensitive learning balances the data distribution by considering the cost of misclassification. All misclassification errors may not be equal in terms of cost. A predictor tries to minimize the cost by making less number of costlier misclassification errors. In this work, we assess the performance of oversampling methods as well as MetaCost (MC) learners [14] for handling the imbalanced datasets.

Though, a number of studies in literature have explored the problem of using imbalanced datasets in software defect prediction domain [2–4,6–8,11–13] only few oversampling methods have been investigated in this domain. Amongst the explored oversampling methods, Synthetic Minority Oversampling Technique (SMOTE) is a popular method [2,4,11,15,16].

This study implements a wide variety of oversampling methods together with the MC learners for five machine learning (ML) classifiers to handle imbalanced data problem. The ML classifiers used in this study are decision trees (J48 and Random Forest (RF)), Naïve Bayes (NB), and two ensemble methods AdaboostM1 (AB) and Bagging (BG). Furthermore, in order to generalize the results, we explore twelve defect prediction public NASA datasets. We also implement five existing oversampling techniques which are: SMOTE, ADAptive SYNthetic sampling technique (ADASYN), Safe-Level-SMOTE (SL-SM), Selective Preprocessing of Imbalanced Data (SPIDER) and SPIDER2. Moreover, the study proposes and implements an improved version of SPIDER2 i.e. SPIDER3. The results are assessed using Area Under the Receiver Operating Characteristic Curve (AUC), sensitivity, specificity and precision performance measures. Furthermore, this study performs statistical comparison of the results using Friedman and Wilcoxon signed rank tests. The results of the study are then compared with previous best results indicated by literature studies.

This study investigates the following research questions (RQ):

RQ1: What is the effect of application of oversampling methods for developing defect prediction models using ML techniques?

RQ1a: Does balancing of datasets using oversampling methods improve the performance of ML techniques for defect prediction? If yes, what is the extent of improvement in the performance of ML techniques for software defect prediction?

RQ1b: Which is the best oversampling method to improve the performance of ML techniques for software defect prediction in this study?

RQ2: What is the comparative performance of the proposed version of SPIDER2 technique i.e. SPIDER3 and the original SPIDER2 and SPIDER techniques for software defect prediction?

RQ3: What is the effect of using MC learners on imbalanced datasets for software defect prediction?

RQ4: What is the comparative performance of best oversampling method and MC learners for software defect prediction?

The organization of this study is as follows: Section 2 presents the related work and Section 3 briefly explains the imbalanced data problem. Section 4 provides details about experimental design and Section 5 describes the research methodology used in the experiment. In Section 6 the obtained results are stated and analyzed using statistical tests and Section 7 states the threats to validity. Section 8 concludes the final outcome of the study.

2. Related work

There are a number of previous studies which have used NASA data sets for defect prediction. A review by Catal and Diri [17] of defect prediction studies observed wide use of method-level metrics in this domain. A recent review by Malhotra [1], surveyed both ML and statistical techniques for defect prediction. According to her survey, most of the studies used method level metrics. Moreover, the ML techniques outperformed the statistical techniques in majority of the cases for developing software defect prediction models. Therefore, our study evaluates ML methods for developing defect prediction models.

A recent study by Mahmood et al. [18] ascertained the negative effect of imbalanced data on the performance of defect prediction models. Weiss et al. [19] and Galar et al. [13] also worked on imbalanced data in defect prediction. They used cost sensitive learning, sampling methods and ensemble methods to improve the performance of defect prediction models. A study by Lin et al. [20] advocated that dynamic sampling should be used for addressing the imbalanced data issue.

The next two sections state the studies which have used NASA datasets from the year 2010 to 2017 and do not use any sampling or cost-sensitive technique for specifically handing imbalanced data (Table 1) and the ones which use a method for specifically addressing the imbalanced data issue (Table 2). Section 2.3 states the studies which have investigated sampling techniques on other datasets apart from NASA.

2.1. Defect prediction studies which do not address imbalanced data issue

As stated earlier, Table 1 reports the details of studies which have used NASA datasets but have not addressed the imbalanced issue. Though, some of the studies [3,5,21–26,28,29] depicted in Table 1 use effective performance measures for handling imbalanced data, they do not address the issue by use of specific techniques such as sampling or cost-sensitive learning.

2.2. Defect prediction studies which address imbalanced data issue

Table 2 depicts studies which have used specific methods for handling the problem of imbalanced data while using NASA datasets in the period 2010–2017.

A study by Zheng [30] used cost-sensitive boosting for handing imbalanced nature of datasets. Seliya et al. [31] proposed a roughly balanced bagging algorithm for developing defect prediction models in case of imbalanced datasets. Seliya and Khoshgoftaar [12] used cost sensitive method to analyze the performance ML techniques in case of imbalanced datasets. Rodriguez et al. [32] used subgroup discovery approaches for handling imbalanced nature of training data. Wang and Yao [33] compared the balancing techniques and proposed a dynamic version of Adaboost.NC for handling imbalanced data. A study by Rodriguez et al. [34] compared cost-sensitive, sampling methods, hybrid techniques and ensembles to deal with imbalanced datasets. Their results showed that the algorithms to deal with imbalanced datasets enhanced the performance of prediction models. To address the imbalanced data problem for software defect prediction, Liu et al. [6] proposed a two-stage cost-sensitive learning (TSCS) method. The proposed method incorporated the cost information in two stages, i.e., feature selection as well as classification stage. Their experimental results demonstrated that the TSCS methods outperformed single-stage cost-sensitive learning methods. Abdi

Table 1
Overview of literature studies using NASA datasets which do not address imbalanced data issue.

Study reference	Number of NASA datasets	Classification techniques
De Carvalho et al. [21]	5	Multi-Objective Particle Swarm Optimization, MLP, NB, BN, SVM, C4.5, RIPPER, Non-Nested Generalized Exemplars.
Liu et al. [22]	7	Decision Table, J48, Partial Decision Tree, Genetic Programming, LR, NB, RF, One Rule, Ripper Down Rules, Bagging etc.
Pendharkar [23]	1	Exhaustive Search-Probabilistic Neural Network, Simulated Annealing-Probabilistic Neural Network, CART, MLP, C4.5.
Misirh et al. [24]	4	Voting Feature Intervals, MLP, NB and two ensemble algorithms
Song et al. [25]	13	NB, J48, One Rule.
Ma et al. [26]	7	Transfer NB
Yu [27]	2	KNN, C4.5, SVM, Least Square SVM (LS-SVM), Asymmetric Weight LS-SVM, LR
Dejaeger et al. [3]	8	LR, RF, 15 different variants of NB
Czibula et al. [28]	10	Bagging, Evolutionary Decision Rules for Subgroup Discovery, One Rule, Relational Association Rules
Jin and Jin [29]	4	Quantum Particle Swarm optimization & MLP, Quantum Particle Swarm optimization & SVM, LR, NB, BN, MLP, Radial Basis Function Network, SVM and many others
Tantithamthavorn et al. [5]	12	NB, LR, RF

MLP: Multi-layer Perceptron; NB: Naïve Bayes; BN: Bayesian Network; SVM: Support Vector machine; LR: Logistic Regression; RF: Random Forests; CART: Classification and Regression Tree; KNN: K-Nearest Neighbor; RIPPER: Repeated Incremental Pruning to Produce Error Reduction.

et al. [7] divided data into two clusters namely balanced and unbalanced in order to properly learn rules for accurate defect prediction. Arar and Ayan [8] used cost-sensitive neural network for handling imbalanced data. Siers and Islam [35] proposed two techniques namely cost-sensitive decision trees and cost-sensitive voting to handle the imbalanced data problem. Furthermore, they also incorporated the oversampling methods, SMOTE and SL-SM to optimize the cost of software defect prediction using decision forest. Wang et al. [36] also addressed the issue of handling imbalanced data using kernel learning and ensemble learning. Cheng et al. [4] proposed semi-supervised learning with task-driven dictionary learning for handling imbalanced data issues along with deficient labeled samples.

2.3. Studies which use sampling methods for handling imbalanced data

Apart from the studies mentioned in Sections 2.1 and 2.2, there were other key research studies which addressed the issue of handling imbalanced data using sampling. Kamei et al. [11] experimentally evaluated the effects of sampling methods (random over sampling, SMOTE, random under sampling and one-sided selection) on defect-prediction models, developed using ML techniques. They discovered that sampling methods improved the prediction performance of the linear and logistic models, while the performance of neural network and classification tree models did not always improve by the use of sampling methods. Tan et al. [2] investigated four sampling methods to improve the performance for online change classification, one of them was SMOTE. Their results depicted that the use of resampling method (an oversampling technique) improved the classification performance. Riquelme et al. [15] evaluated the use of resampling and SMOTE on five NASA datasets. Their results supported the use of SMOTE for developing effective defect prediction models. Shatnawi [16] evaluated the use of SMOTE on Eclipse dataset for developing defect prediction models using three ML techniques. He found the application of SMOTE to be favorable for developing software defect prediction.

An analysis of literature studies reveal that oversampling methods especially SMOTE is a popular method for handling imbalanced datasets. However, it may be noted from Table 2 that only 4 studies investigated sampling approaches on NASA datasets. Furthermore, there are improved methods like SPIDER [37] and ADASYN [38] which may produce better results when compared to SMOTE. To the best of author's knowledge, no study has evaluated them on ML methods for developing software defect prediction models. Therefore, we explore all the mentioned oversampling methods (SMOTE, ADASYN, SL-SM, SPIDER and SPIDER2) to handle imbalanced data problem in the software defect prediction domain.

A previous work by Malhotra and Khanna [39] analyzed the performance of only three sampling methods (sampling, SMOTE and Spread Subsample) along with MC learners on change prediction data. However, this work is different from author's previous work as it investigates the use of five oversampling methods along with MC learners on defect prediction data. Also, the study proposes a new variant (SPIDER3) of an existing oversampling method SPIDER2. Our improved version of the SPIDER2 algorithm is evaluated to assert if it is better than its original version for handling imbalanced data. The various oversampling methods used in this work have been implemented by the authors in the MATLAB environment. This study uses twelve widely used NASA defect datasets as compared to only six data sets used in author's previous work on change prediction.

3. Imbalanced data problem

Imbalanced data problem is a common issue in many ML and data mining related domains for example, network intrusion detection [40], medical diagnosis [41], fraud detection [42], software defect prediction [18] etc. A data set is imbalanced if it has biased distributions of classes. Such a data can lead to adverse effects on the actual performance of various ML classifiers. In most of the cases, the accurate classification of minority class is more important than that of majority class as it is costly to misclassify instances from the minority class [12,39]. For example, in case of medical diagnosis, cancer disease is less common but it is important to diagnose a person with cancer correctly otherwise it may lead to a loss of life. The traditional standard classifiers are built with the assumption that the input dataset is balanced with respect to various classes but when one class dominates the other, the classifiers tend to misclassify the minority class which results in the increase of prediction error [43]. This limitation of classifiers can lead to huge losses in terms of life and money.

There are four different characteristics that imbalanced data holds as explained by Ramyachitra and Manikandan [44]: small disjuncts, lack of density, noisy data and dataset shift. In this study, we are dealing with the fourth characteristic that is dataset shift. This is defined as the case where the dataset follow different distributions with respect to various classes and the minority class is mostly sensitive to prediction errors. This work focuses on binary class (defective and non-defective) imbalance problem.

There are many techniques to handle imbalanced datasets on various levels like data level, algorithm level, cost sensitive level, feature selection level and ensemble level [10]. These levels further encompass different methods and algorithms to handle the imbalanced data. This study explores data level and cost sensitive approach. In data level approach we specifically focus on

Table 2
Overview of literature studies using NASA datasets which address the problem of imbalanced data.

Study reference	Number of NASA datasets	Classification techniques	How is the issue of Imbalanced data addressed?
Zheng [30]	4	Three cost-sensitive neural boosting algorithms	Use of cost-sensitive boosting.
Seliya et al. [31]	7	C4.5, NB, Roughly Balanced Bagging Algorithm	By using Roughly Balanced Bagging algorithm.
Seliya and Khoshgoftaar [12]	7	C4.5, RF along with six cost-sensitive learning techniques	By using cost-sensitive learning.
Rodriguez et al. [32]	7	Evolutionary Decision Rules for Subgroup Discovery, CN2 Subgroup Discovery, Subgroup Discovery, APRIORI Subgroup Discovery	By using subgroup discovery approaches.
Wang and Yao [33]	10	NB, RF, Adaboost. NC, Random Under-sampling, Balanced random Under-sampling, Threshold Moving.	By using undersampling approaches, cost-sensitive learning and threshold moving.
Rodriguez et al. [34]	13	C4.5, NB	By using cost-sensitive methods, sampling techniques, ensembles and hybrid approaches.
Liu et al. [6]	7	Two-stage cost-sensitive learning, Cost-sensitive neural network.	By using cost-sensitive learning.
Abdi et al. [7]	4	MLP, BN, NB, RIPPER, SVM, C4.5	By dividing data into balanced and imbalanced clusters.
Arar and Ayan [8]	4	Artificial Bee Colony & MLP, NB, RF, C4.5, Artificial Immune Recognition System, Immunos.	By using cost-sensitive neural network.
Siers and Islam [35]	6	Cost-sensitive Decision trees, Cost-sensitive Voting, C4.5, SVM.	By using cost sensitive learning and oversampling methods in the proposed algorithms.
Wang et al. [36]	12	NB, C4.5, Cost-sensitive neural network boosting, asymmetric kernel principal classification, multiple kernel ensemble learning, coding based ensemble learning, Adaboost. NC	By using two-state of art class-imbalance learning methods.
Cheng et al. [4]	9	Adaboost along with four sampling techniques (Random Undersampling, SMOTE, random Committee with Undersampling, Cost-sensitive discriminative dictionary learning)	By using sampling and cost-sensitive learning methods.

MLP: Multi-layer Perceptron; NB: Naïve Bayes; BN: Bayesian Networks; SVM: Support Vector machine; RF: Random Forests; RIPPER: Repeated Incremental Pruning to Produce Error Reduction.

oversampling methods and in cost sensitive approach, the performance of MC learners is evaluated.

Various researchers in the domain of prediction using imbalanced data have ascertained that oversampling may result in overfitting as the training dataset may have multiple replicate instances [10, 45]. However, use of advanced oversampling methods such as SMOTE overcomes this limitation [10]. SMOTE creates synthetic replicated instances by interpolation of various instances belonging to the minority class that lie together, thus reducing overfitting [10]. However, SMOTE has certain drawbacks such as overlapping between classes while creating synthetic samples [46]. In order to overcome this, other adaptive sampling methods such as SL-SM [47], SPIDER [37] and SPIDER2 [48] have been proposed, which have been evaluated in this study.

4. Experimental design

This section provides the details regarding various design settings used in this study.

4.1. Dependent and independent variables

The problem of software defect prediction is considered as a binary classification problem, where a class is either “defect prone” or “not defect prone” [6]. A module can either contain faults or can be defect free (not defect prone) [49].

This work uses a set of 39 procedural metrics as independent variables. Procedural metrics consists of a set of traditional code metrics defined by Halstead [50] and McCabe [51] and lines of code (LOC) counts that are categorized under size metrics. Halstead metrics are used to measure complexities on the basis of number of operators and operands in a module while McCabe metrics set is deduced using the flow graph information of a module [52]. The dependence of defect proneness over static code metrics is considered practical as they have helped in successful detection of the defect prone nature of the class in the past [21,24,25,30–36,52]. They are also helpful in deciding whether the module should go through manual inspections or not. According to the review by Malhotra [1] procedural metrics were widely used, i.e. in more than 51% of the investigated defect prediction studies. They can be calculated at reasonably low costs for both small and large systems. Table 3 states the static code metrics, size metrics and other metrics which are a part of procedural metrics used in this study.

4.2. Datasets description

This study uses a set of 12 publicly available NASA datasets which are collected from the PROMISE repository. As observed by Malhotra [1] more than 60% of the investigated defect prediction studies in her review used NASA datasets. Moreover, Section 2 states a number of defect prediction studies which use these datasets. Each dataset comprises of procedural metrics including static code metrics and size metrics. It also contains the defect proneness as dependent variable. The value of dependent variable is set to ‘0’ or ‘no’ if the module is not defective otherwise it is set to ‘1’ or ‘yes’ in case of defective module. The datasets selected in this study are highly imbalanced to moderately balanced with minority class instances (i.e. number of defective modules) percentage in the range of 1–35.5%. The detailed description of 12 NASA datasets used in this study together with the procedural metrics used in each dataset is given in Table 3, which includes the total number of modules per dataset and percentage of defective modules.

Though NASA datasets are widely used, certain research studies have raised concerns about quality of data extracted from the PROMISE repository [53–55]. As a result of this criticism, NASA released cleaned datasets after performing corrective steps which were proposed by Shepperd et al. [54]. The datasets were released as either D′ or D″. The following cleaning operations were performed in D″ datasets [54]:

i. All data points with implausible values i.e. which may disregard certain integrity constraints were removed.
ii. All data points which have conflicting values for features which violate referential integrity constraints were removed.
iii. All data points which have identical values for all attributes as well as the dependent variable were removed.
iv. All data points which have identical values for all attributes except the dependent variable (inconsistent data points) were removed.
v. All data points which have missing values for any attribute were removed.
vi. All independent variables (features) with constant values were removed.
vii. All independent variables with identical values for all data points were removed.

We used CM1′, JM1″, KC2″, KC3″, MC1″, MC2″, MW1″, PC2′ and PC3″. Also, original uncleaned versions were used for PC1, PC4 and PC5. However, we performed all cleaning steps (i–vii) for PC1, PC4 and PC5 ourselves. Similarly, steps (iii–iv) were performed for CM1′ and PC2′, which makes them clean according to D″ standards. Furthermore, outliers were removed using interquartile range filter of WEKA for all the 12 datasets. Thus, the data used for performing empirical validation in this study is cleaned and appropriate for experimentation.

4.3. Selection of performance measures

Researchers have been involved in a controversy over the use of performance measures while dealing with imbalanced data. The use of recall, precision and accuracy performance measures have been criticized by researchers [56,57] for the evaluation of prediction models, while AUC, balance etc. are considered effective measures for defect prediction models developed using imbalanced data [52,56,57]. This work evaluates the performance of ML classifiers using four performance measures namely AUC, recall, specificity and precision. These measures are computed using confusion matrix (Table 4). It consists of four variables out of which two are predicted class labels and other two are actual class labels. Two classes used in this study are defective (positive) and non-defective classes (negative). In matrix, TN is the number of non-defective samples of the dataset which are predicted as non-defective, TP is the number of defective samples of the dataset predicted correctly as defective, FN implies to the number of defective samples predicted as non-defective and similarly FP refers to the number of non-defective samples predicted falsely as defective. Table 5 describes the definition along with formula for the used performance measures.

4.4. Statistical test selection

Lessmann et al. [52] ascertained that only few previous studies have used statistical tests for performance validation. In order to statistically evaluate the performance of oversampling methods, we use two statistical tests: Friedman test and Wilcoxon signed rank test. These tests are non-parametric in nature. The assumptions made in the non-parametric tests are not stringent and one may ignore the presence of outliers in the datasets, variance homogeneity, normal distributions etc. [59].

Friedman test assigns ranks to different methods under experiment on the basis of performance metrics used for evaluation. The

Table 3
Static code metrics description in NASA datasets.

Metric		NASA dataset											
		CM1	JM1	KC2	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
Halstead metrics	*Level*	√	√	√	√	√	√	√	√	√	√	√	√
	Program time	√	√	√	√	√	√	√	√	√	√	√	√
	Volume	√	√	√	√	√	√	√	√	√	√	√	√
	Error estimate	√	√	√	√	√	√	√	√	√	√	√	√
	Length	√	√	√	√	√	√	√	√	√	√	√	√
	Content	√	√	√	√	√	√	√	√	√	√	√	√
	Difficulty	√	√	√	√	√	√	√	√	√	√	√	√
	Effort	√	√	√	√	√	√	√	√	√	√	√	√
	Num_operands	√	√	√	√	√	√	√	√	√	√	√	√
	Num_unique_operands	√	√	√	√	√	√	√	√	√	√	√	√
	Num_operators	√	√	√	√	√	√	√	√	√	√	√	√
	Num_unique_operators	√	√	√	√	√	√	√	√	√	√	√	√
McCabe metrics	*Essential Complexity*	√	√	√	√	√	√	√	√	√	√	√	√
	Cyclomatic Complexity	√	√	√	√	√	√	√	√	√	√	√	√
	Design Complexity	√	√	√	√	√	√	√	√	√	√	√	√
	Cyclomatic Density	√	√	√	√	√	√	√	√	√	√	√	√
Size metrics	*Number of lines*	√	–	–	√	√	√	√	√	√	√	√	√
	LOC total	√	√	√	√	√	√	√	√	√	√	√	√
	LOC executables	√	√	√	√	√	√	√	√	√	√	√	√
	LOC comments	√	√	√	√	√	√	√	√	√	√	√	√
	LOC code & comments	√	√	√	√	√	√	√	√	√	√	√	√
	LOC blanks	√	√	√	√	√	√	√	√	–	√	√	√
Others	*Branch count*	√	√	√	√	√	√	√	√	√	√	√	√
	Condition count	√	–	–	√	√	√	√	√	√	√	√	√
	Decision count	√	–	–	√	√	√	√	√	√	√	√	√
	Edge count	√	–	–	√	√	√	√	√	√	√	√	√
	Parameter count	√	–	–	√	√	√	√	√	√	√	√	√
	Modified condition count	√	–	–	√	√	√	√	√	√	√	√	√
	Multiple condition count	√	–	–	√	√	√	√	√	√	√	√	√
	Node count	√	–	–	√	√	√	√	√	√	√	√	√
	Decision density	√	–	–	√	–	√	√	√	√	√	√	–
	Design density	√	–	–	√	√	√	√	√	√	√	√	√
	Essential density	√	–	–	√	√	√	√	√	√	√	√	√
	Global data density	–	–	–	√	√	√	–	–	–	–	–	√
	Call pairs	√	–	–	√	√	√	√	√	√	√	√	√
	Maintenance severity	√	–	–	√	√	√	√	√	√	√	√	√
	Global data complexity	–	–	–	√	√	√	–	–	–	–	–	√
	Normalized cyclomatic complexity	√	–	–	√	√	√	√	√	√	√	√	√
	Percent comments	√	–	–	√	√	√	√	√	√	√	√	√
Number of code attributes		37	21	21	39	38	39	37	37	36	37	37	38
Total number of modules		344	7782	522	194	1988	125	253	705	1585	1077	1458	17,186
Percentage of defective modules		12.21	21.48	20.5	18.55	2.31	35.2	10.67	8.03	1	12.44	12.20	3

Table 4
Confusion matrix.

Class	Predicted negatives	Predicted positives
Actual negative	True negatives (TN)	False positives (FP)
Actual positives	False negatives (FN)	True positives (TP)

Table 5
Performance measures.

Performance measures	Definition
Sensitivity (Recall)	It is defined as a percentage of correctly predicted defective modules. $Sensitivity = \frac{TP*100}{TP+FN}$
Specificity	It is defined as a percentage of correctly predicted non-defective modules. $Specificity = \frac{TN*100}{TN+FP}$
Precision	It is defined as the ratio of correctly predicted defective modules to the total number of modules predicted as defective. $Precision = \frac{TP*100}{TP+FP}$
Area under Receive Operating Characteristic (ROC) curve [58]	AUC is a combined measure of sensitivity and specificity. The ROC is a curve plotted between sensitivity and (1-specificity) on the x and y-coordinate axis respectively. The larger the area enclosed under the curve the better is the performance of the ML technique [56].

lower mean rank attained by an oversampling method, the better it is. If the results obtained by Friedman test are significant, we perform Wilcoxon signed rank test with Bonferroni correction. Bonferroni correction is used to remove family wise errors. The test ascertains whether the pairwise performance of two methods differs significantly or not. It compares two related scenarios (a vs b) using positive and negative ranks. Positive ranks indicate the number of times 'a' outperforms 'b' out of total number of instances while negative ranks indicate the number of times 'b' outperforms 'a'. If positive ranks are equal to negative ranks then the performance of both 'a' and 'b' is considered equal. We use α=0.05 as a decision parameter for the acceptance or rejection of null hypothesis for both the tests. We also calculate the effect size of Wilcoxon test as suggested by Pallant [60]. The effect size is given by the following formula:

$$Effect\ Size = \frac{Z}{\sqrt{N}}$$

Here, Z corresponds to the test statistic and N corresponds to twice the number matched pairs in Wilcoxon test. The computed effect size can be interpreted according to Cohen [61] as small (0.1), medium (0.3) or large (0.5) according to its obtained value.

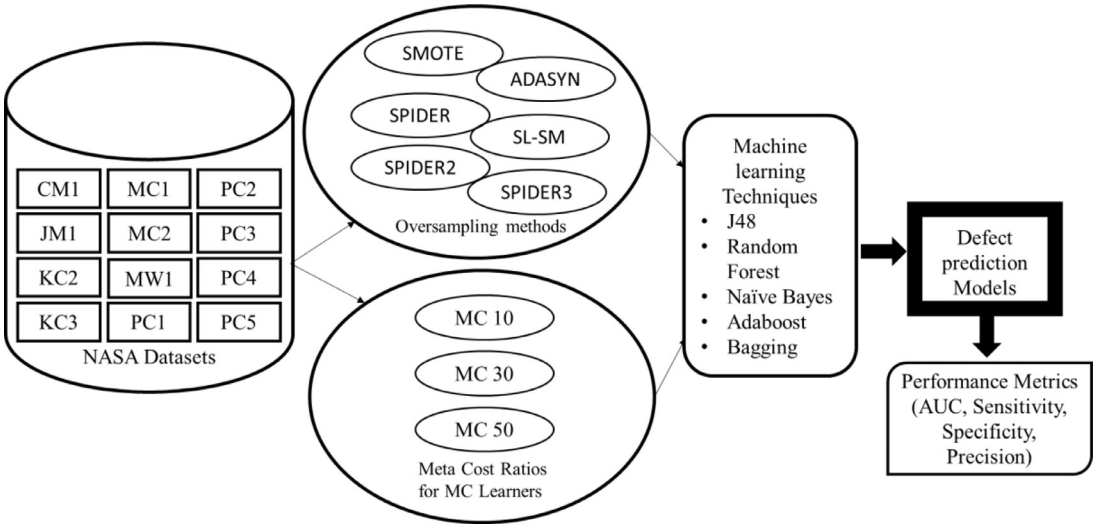

Fig. 1. Experimental setup.

4.5. Model evaluation

The study uses 10-fold cross validation method for model evaluation. The method works by randomly dividing the dataset into ten subsets. Ten iterations are performed where one subset is taken as testing set while other nine subsets are considered as training sets. All the ten subsets are used for validation exactly once. The final result is calculated by the average estimation of results generated during each iteration.

5. Research methodology

In this study, we have implemented five existing oversampling methods and have proposed an improved oversampling method. The techniques are applied on imbalanced NASA datasets along with the use of MC learners (with three cost ratios) to handle the imbalanced data problem. Furthermore, we apply five ML classifiers on the datasets in order to develop defect prediction models. Their performance is evaluated by using four performance measures described in the previous section. A diagrammatic representation of the studies experimental setup is given in Fig. 1.

The study uses WEKA (www.cs.waikato.ac.nz/ml/weka) for simulation of ML techniques with default parameters. This section describes the various methods to handle imbalanced datasets and the ML techniques for defect prediction used in this study.

It may be noted that the oversampling methods were applied only on the training folds in WEKA and not on the whole dataset in advance.

5.1. Oversampling methods

Oversampling methods attempt to balance data either by replicating the minority class samples or by generating new synthetic samples of the minority class. This study implemented five oversampling methods in the MATLAB environment whose brief explanation is stated in this section. Different parameters (k and N) are required for all the oversampling methods except ADASYN, which are stated in Table 6. The study chooses seven different values of k depending on the requirement of each of the 12 NASA datasets used in this work.

5.1.1. SMOTE

SMOTE [45] creates synthetic samples for each minority class sample using its k nearest neighbors. Selection of number of nearest neighbors depends upon the amount of oversampling (N) needed. The amount of oversampling required can be 100%, 200%, 500%, 1000% and so on. For example, 500% oversampling means five nearest neighbors are randomly chosen from k nearest neighbors. The amount of oversampling further depends on the percentage of minority class instances present in each dataset with respect to total number of instances present in the dataset. This study uses oversampling in the range of 200–9000% (Table 6). The synthetic sample for each minority class sample is generated by taking the difference between the particular minority class sample and its nearest neighbor. The difference is then multiplied by a random number which belongs to the range from 0 to 1 and then it is finally added to that particular minority class sample under consideration. The details of SMOTE can be referred from [45,46].

Table 6
Parameter selection for oversampling algorithms.

Dataset	SMOTE/SL-SM		SPIDER/SPIDER2/SPIDER3
	K	N	K
CM1	5	5	5
JM1	4	4	4
KC2	4	4	4
KC3	4	4	4
MC1	25	25	25
MC2	3	2	3
MW1	5	5	5
PC1	7	7	7
PC2	15	90	90
PC3	5	5	5
PC4	5	5	5
PC5	30	30	30

5.1.2. Safe-level-SMOTE

Safe-level-SMOTE (SL-SM) [47] is the modified version of SMOTE. It focuses on how the random number (used in SMOTE) will be chosen to generate synthetic minority samples. The minority class samples are assigned safe levels (SL) on the basis of k nearest neighbors in the dataset. The synthetic samples to be generated are set in the range of 200–9000% (Table 6). We find k nearest minority class neighbors for each minority class sample 'p'. One neighbor 'n' will be chosen randomly and then safe level ratio SL_p/SL_n (number of minority samples in k nearest neighbors of p in

the dataset to the number of minority class samples in k nearest neighbors of n) will be calculated. On the basis of safe level ratio, random number is chosen accordingly. Then, it will be used similarly, as in SMOTE to generate synthetic samples. SMOTE randomly generates equal synthetic samples for each minority class sample while it is not the case in SL-SM. Details of SL-SM can be referred from [47].

5.1.3. ADASYN

In Adaptive synthetic (ADASYN) sampling technique [62], the number of synthetic samples needed to be generated for each minority class sample is decided by the density distribution. Unlike SMOTE, ADASYN automatically calculates the number of synthetic samples. Density distribution is the measure of weights which are given to each minority class sample according to their difficulty level of learning. The procedure of generating synthetic sample is same as that of SMOTE. This study uses a completely balanced dataset after application of ADASYN method as a balance ratio of 1 is chosen. A detailed description of ADASYN can be referred from [62].

5.1.4. SPIDER

Selective pre-processing of imbalanced data (SPIDER) proposed by Stefanowski and Wilk [37] consists of two phases. In the first phase, each sample from the given dataset is flagged as noisy or safe depending on the k-nearest neighbors. The detailed description of k values is described in Table 6. In the second phase, amplification of minority samples is done in three ways, that is, weak amplification, weak amplification with relabeling and strong amplification. A detailed description of the algorithm can be referred from [37].

5.1.5. SPIDER2

SPIDER2 is a modified version of SPIDER [48]. Though, SPIDER blindly amplifies the minority class samples without taking into account the changes that arise in the dataset due to relabeling, SPIDER2 takes into considerations the changes made by relabeling option in an initial phase and on the basis of those changes it flags the minority class samples as safe or noisy. After identification of noisy examples, SPIDER2 performs amplification operation on the relabeled dataset. The detailed algorithm can be referred from [48].

5.1.6. SPIDER3: a modified version of SPIDER2 (proposed method)

To add one more method in the family of SPIDER methods, we propose SPIDER3, a modified version of SPIDER2 technique. Pseudocode of SPIDER3 is presented in Fig. 2. The original version of SPIDER algorithm uses three functions which are: *correct(data, s, k), flagged(data, c, f)* and *replicate(data, s, k, maj, f)*. The first function classifies the sample '*s*' as *safe or noisy* using its k nearest neighbors and returns a 1 if it is *safe*, otherwise returns 0. The second function generates a set of those that are the part of class c and are flagged as *f (noisy or safe)*. The third function replicates the copies of minority class sample '*s*' as many number of times as there are majority class samples in *s*'s k nearest neighbors which are flagged as *f*.

SPIDER3 modifies the third function *replicate(data, s, k, maj, f)* by adding one more parameter into it. The modified replicate function is *replicate(*min*, data, s, k, maj, f)*. This function generates new synthetic samples of minority class sample '*s*' as many number of times as there are majority class samples in *s*'s k nearest neighbors which are flagged as f. The advantage of our proposed method is that it generates new samples of minority class that is, a synthetic sample rather than just replicating the same sample several times.

SPIDER3 consists of two phases. In the first phase, we identify majority class examples as safe or noisy on the basis of k nearest neighbors. Our method uses Euclidean distance instead of heterogeneous value difference metric (HVDM) distance function to compute k nearest neighbors. We used Euclidean distance because this study focuses on defect prediction of imbalanced datasets which have all numeric independent variables. Only the dependent variable is nominal. HVDM [63] distance function is useful when we deal with heterogeneous data which has both numeric and nominal attributes. It would not make much difference in case of homogeneous data. After identification, relabeling is performed on noisy majority class samples which lie in the nearest neighborhood of each corresponding minority class sample.

In the second phase, identification of minority class samples is done with reference to changes made in dataset by relabeling. Our modifications exist in the amplification phase. Instead of replicating the same minority sample SPIDER3 calculates the synthetic samples while amplifying the data. The synthetic samples are generated by using the same method as used in SMOTE. Minority class is amplified as many numbers of times as there are safe examples of majority class in its k nearest neighbors. For each minority class sample amplification, k nearest neighbors are computed in the minority class region only and then a synthetic sample is generated. There is an exception which can arise while using SMOTE formula in SPIDER. As we are using the same value of k both for generating synthetic samples as well as for the other computations required by SPIDER3, it might be possible that the value of k exceeds the total number of minority class samples present in the dataset. For example, in case of PC2 dataset there are only 16 minority class examples. But according to the amount of oversampling needed which is 9000% (percentage of minority class is just 1% and hence, large number of samples are required to be generated to fully balance the dataset) we need to set $k = 90$ (In SPIDER family there is no N, oversampling is done on the basis of k only) which exceeds total number of minority class samples. In such a case, we simply set k for synthetic sample generation as the total number of minority class examples. Remaining amount of oversampling will be done by replicating the minority sample.

5.2. MetaCost learners

Domingos [14] proposed a procedure for making classifiers cost-sensitive called MC learner. It makes any ML classifier cost sensitive by applying cost minimizing procedure over it. They do not require any information about how the individual classifier works and can use any arbitrary cost matrix.

This method uses Bayes optimal prediction to reduce the risk of achieving high overall cost which is called the conditional risk. The conditional risk $R(r/x)$ computes the expected cost value of predicting a sample x as a part of the class 'r' when it actually belongs to the class 's'. The conditional risk is defined as the summation of product of $C(r,s)$ and $P(s/x)$ for each region 'j' where $C(r,s)$ is the cost of predicting an example as a part of 'r' when it actually is the member of class 's' and $P(s/x)$ is the probability of predicting that sample x is a member of class 's'. The conditional risk partitions the sample space into 'j' regions such that least cost prediction i.e. 's' falls into the region of its own class 's'.

In this way MC learners relabels the classes of each training instance according to their best predicted classes. MC is parallel to BG ensemble method. The difference between the two exists in choosing the size of resample. BG constructs the bootstrap resample by selecting 'n' samples with replacement from the training set of size 'n' while MC learners go well with the smaller resample size also. This nature of MC learners makes them more efficient. The classifiers are then learned on each resample followed by the collection of votes from the ensemble. The majority vote decides the label of each example and hence, relabels it to the optimal predicted class.

```
D← original dataset
C_min ← set of all samples in D which are present in minority
C_maj ← set of all samples in D which are present in majority
K← number of nearest neighbours
Step 1)  for each sample s ϵ D do
                If correct(data, s, k) then
                        type=safe
                Else
                        type=noisy
Step 2)  if relabeling==true
            For each s ϵ flagged(data, C_min, noisy) do
                For each t ϵ C_maj in k nearest neighbours of s &
                 type==noisy do
                    Change class of t from C_maj to C_min
         else
            For each s ϵ flagged(data, C_min, noisy) do
                 For each t ϵ C_maj in k nearest neighbours of s &
                  type==noisy do
                     D ← D – t
Step 3)  for each sample s ϵ C_min do
                If correct(data, s, k) then
                        type=safe
                Else
                        type=noisy
Step 4)  if amplication==weak then
            For each s ϵ flagged(data, C_min, noisy) do
                    replicate(C_min, data, s, k, maj, safe)
         else
            For each s ϵ flagged(data, C_min, safe) do
                    replicate(C_min, data, s, k, maj, safe)
             For each s ϵ flagged(data, C_min, noisy) do
                    If correct(data, s, k+2) then
                            replicate(C_min, data, s, k, maj, safe)
                    else
                            replicate(C_min, data, s, k+2, maj, safe)
```

Fig. 2. SPIDER3 Algorithm for imbalanced data.

5.3. Machine learning classifiers

The study uses five ML classifiers for developing defect prediction models.

J48 is a decision tree classifier that uses information gain [64] as the splitting criteria. Random Forests (RF) is a forest of decision trees in which each tree is made up of randomly selected features from the dataset [65]. The final outcome is obtained by majority voting of constituent trees.

Naïve Bayes (NB) is based on Bayes' theorem in which the classifier assumes that the effect of one attribute on a given class is independent of the other attributes [66].

AdaboostM1 (AB) is a boosting classifier [64] which trains various individual classifiers in an iterative manner by using the complete training dataset. Each iteration of the algorithm focuses more on the difficult instances which are misclassified in the previous iteration.

Bagging (BG) is a meta classifier which trains different classifiers using bootstrapped replicas of the original training dataset [67].

6. Empirical results and analysis

In this section, we discuss and analyze the results obtained by applying ML techniques on sampled as well as original imbalanced datasets. The investigation of results is carried out systematically by answering the research questions mentioned in Section 1.

6.1. RQ1: what is the effect of application of oversampling methods for developing defect prediction models using ML techniques?

The effect of oversampling methods is evaluated by analyzing the performance measures (AUC, Sensitivity, Specificity and Precision) of defect prediction models developed before and after application of oversampling methods using 12 NASA datasets.

6.1.1. RQ1a: does balancing of datasets using oversampling methods improve the performance of ML techniques for defect prediction? If yes, what is the extent of improvement in the performance of ML techniques for software defect prediction?

Tables 7–9 provide the results (AUC, sensitivity and precision) of defect prediction models developed using ten-fold cross validation. The models were developed using original imbalanced datasets (no sampling) as well as balanced datasets after oversampling. The balanced datasets were obtained by correspondingly applying six oversampling methods: SMOTE, SL-SM, ADASYN, SPIDER, SPIDER2 and SPIDER3 implemented in the MATLAB environment. The study uses five different ML techniques: J48, RF, NB, AB and BG. The tables also depict the average performance values of all the oversampling methods on each dataset using all the ML techniques.

From the results in Tables 7–9, it is observed that AUC values in case of oversampling methods are better than those in the case when no sampling was performed. Table 7 depicts a large percentage increase (5–122%) in AUC values in majority of the cases. In fact, in certain cases (Table 7), application of oversampling methods attained a perfect AUC value of 1. Sensitivity results also show

Table 7
AUC results of defect prediction models.

Dataset	CM1						JM1						KC2						KC3					
Classifier	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG
N. Samp.	0.594	0.763	0.694	0.717	0.727	**0.699**	0.616	0.701	0.633	0.669	0.689	**0.662**	0.704	0.825	0.832	0.784	0.825	**0.794**	0.653	0.736	0.661	0.573	0.729	**0.670**
SMOTE	0.894	0.955	0.927	0.965	0.960	**0.940**	0.903	0.938	0.745	0.927	0.937	**0.890**	0.927	0.965	0.841	0.947	0.961	**0.928**	0.887	0.943	0.885	0.913	0.944	**0.914**
SL-SM	0.893	0.938	0.895	0.952	0.945	**0.925**	0.892	0.931	0.755	0.920	0.930	**0.886**	0.924	0.963	0.858	0.946	0.963	**0.931**	0.874	0.940	0.852	0.902	0.918	**0.897**
ADASYN	0.920	0.962	0.943	0.966	0.967	**0.952**	0.880	0.918	0.704	0.901	0.914	**0.863**	0.906	0.955	0.786	0.941	0.951	**0.908**	0.860	0.934	0.862	0.898	0.935	**0.898**
SPIDER	0.937	1.000	0.711	0.736	0.994	**0.876**	0.872	0.983	0.558	0.612	0.944	**0.794**	0.838	0.930	0.707	0.745	0.885	**0.821**	0.908	0.979	0.661	0.760	0.947	**0.851**
SPIDER2	0.873	0.984	0.708	0.781	0.976	**0.864**	0.798	0.949	0.614	0.647	0.905	**0.783**	0.862	0.937	0.763	0.773	0.904	**0.848**	0.879	0.963	0.708	0.767	0.925	**0.848**
SPIDER3	0.881	0.951	0.904	0.946	0.946	**0.926**	0.856	0.918	0.705	0.882	0.907	**0.854**	0.886	0.952	0.766	0.920	0.943	**0.893**	0.828	0.937	0.833	0.917	0.922	**0.887**
Dataset	**MC1**						**MC2**						**MW1**						**PC1**					
Classifier	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**
N. Samp.	0.521	0.883	0.708	0.842	0.860	**0.763**	0.698	0.717	0.702	0.616	0.676	**0.682**	0.449	0.716	0.728	0.711	0.705	**0.662**	0.719	0.844	0.768	0.793	0.820	**0.789**
SMOTE	0.984	0.997	0.936	0.992	0.992	**0.980**	0.736	0.905	0.807	0.863	0.899	**0.842**	0.906	0.956	0.896	0.949	0.954	**0.932**	0.953	0.982	0.912	0.974	0.978	**0.960**
SL-SM	0.980	0.998	0.939	0.992	0.993	**0.980**	0.761	0.898	0.807	0.859	0.865	**0.838**	0.862	0.936	0.863	0.934	0.927	**0.904**	0.935	0.976	0.902	0.962	0.967	**0.948**
ADASYN	0.989	0.998	0.947	0.995	0.995	**0.985**	0.681	0.832	0.738	0.761	0.798	**0.762**	0.932	0.974	0.930	0.969	0.966	**0.954**	0.963	0.987	0.935	0.984	0.985	**0.971**
SPIDER	0.992	1.000	0.754	0.889	0.999	**0.927**	0.786	0.974	0.678	0.764	0.922	**0.825**	0.947	1.000	0.764	0.836	0.985	**0.906**	0.960	0.995	0.783	0.831	0.990	**0.912**
SPIDER2	0.992	1.000	0.754	0.889	0.999	**0.927**	0.714	0.894	0.720	0.726	0.848	**0.780**	0.920	0.992	0.760	0.868	0.972	**0.902**	0.956	0.993	0.785	0.858	0.985	**0.915**
SPIDER3	0.983	0.998	0.941	0.993	0.993	**0.982**	0.764	0.878	0.769	0.818	0.837	**0.813**	0.908	0.958	0.884	0.961	0.946	**0.931**	0.946	0.986	0.917	0.976	0.980	**0.961**
Dataset	**PC2**						**PC3**						**PC4**						**PC5**					
Classifier	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**
N. Samp.	0.448	0.836	0.877	0.914	0.828	**0.781**	0.616	0.831	0.766	0.791	0.824	**0.766**	0.777	0.945	0.836	0.913	0.920	**0.878**	0.817	0.977	0.937	0.959	0.975	**0.933**
SMOTE	0.993	0.999	0.995	0.999	0.998	**0.997**	0.894	0.969	0.891	0.959	0.963	**0.935**	0.948	0.991	0.916	0.986	0.988	**0.966**	0.989	0.999	0.944	0.999	0.999	**0.986**
SL-SM	0.975	0.994	0.985	0.995	0.990	**0.988**	0.894	0.967	0.891	0.951	0.961	**0.933**	0.940	0.989	0.908	0.982	0.985	**0.961**	0.988	0.999	0.945	0.999	0.999	**0.986**
ADASYN	0.994	0.999	0.996	0.999	0.997	**0.997**	0.911	0.977	0.895	0.965	0.973	**0.944**	0.955	0.992	0.930	0.988	0.989	**0.971**	0.988	0.999	0.945	0.999	0.999	**0.986**
SPIDER	0.990	1.000	0.896	0.976	1.000	**0.972**	0.940	0.999	0.760	0.777	0.988	**0.893**	0.955	0.999	0.840	0.917	0.990	**0.940**	0.986	0.999	0.913	0.945	0.996	**0.968**
SPIDER2	0.990	1.000	0.896	0.976	0.999	**0.972**	0.920	0.987	0.751	0.781	0.976	**0.883**	0.937	0.990	0.818	0.925	0.981	**0.930**	0.986	0.999	0.921	0.950	0.996	**0.970**
SPIDER3	0.993	0.999	0.996	0.999	0.998	**0.997**	0.890	0.966	0.869	0.951	0.962	**0.928**	0.944	0.986	0.887	0.967	0.982	**0.953**	0.985	0.999	0.933	0.997	0.999	**0.983**

N.Sap→ No Sampling.

Table 8
Sensitivity results of defect prediction models.

Dataset	CM1						JM1						KC2						KC3					
Classifier	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG
N. Samp.	26.2	0.0	33.3	0.0	0.0	**11.9**	22.5	20.5	18.5	0.0	17.2	**15.7**	49.5	47.7	42.1	43.9	43.0	**45.2**	33.3	13.9	38.9	36.1	13.9	**27.2**
SMOTE	84.5	83.3	84.5	84.1	83.7	**84.0**	84.3	83.7	24.4	79.1	83.5	**71.0**	89.7	89.5	46.2	86.7	89.2	**80.3**	87.2	84.4	83.9	82.2	81.1	**83.8**
SL-SM	84.2	79.8	74.9	80.8	79.8	**79.9**	82.2	81.8	26.6	76.7	81.2	**69.7**	87.5	88.7	46.9	82.9	88.3	**78.9**	83.9	79.2	65.1	79.2	79.9	**77.5**
ADASYN	87.7	85.6	89.7	86.3	86.0	**87.1**	78.4	77.9	18.4	72.2	77.4	**64.9**	84.4	86.8	34.7	80.1	86.8	**74.6**	84.0	78.8	80.8	79.5	79.5	**80.5**
SPIDER	99.6	100.0	31.4	92.7	100	**84.7**	92.3	95.4	12.5	59.0	91.7	**70.2**	77.7	84.9	24.3	61.0	77.7	**65.1**	91.2	92.6	27.9	62.5	90.4	**72.9**
SPIDER2	95.1	95.1	27.5	82.2	95.5	**79.1**	89.4	91.4	18.4	83.8	91.7	**74.9**	79.9	87.8	38.4	77.4	80.8	**72.9**	86.5	89.0	31.0	74.8	84.5	**73.2**
SPIDER3	83.6	82.8	86.6	81.3	82.1	**83.3**	82.6	82.1	18.4	74.5	80.8	**67.7**	85.6	89.2	39.0	83.5	87.7	**77.0**	76.4	84.5	44.7	77.0	83.9	**73.3**
Dataset	**MC1**						**MC2**						**MW1**						**PC1**					
Classifier	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**
N. Samp.	8.7	19.6	32.6	0.0	0.0	**12.2**	52.3	38.6	38.6	40.9	38.6	**41.8**	14.8	18.5	55.6	29.6	11.1	**25.9**	24.6	13.1	36.1	0.0	8.2	**16.4**
SMOTE	96.8	97.1	97.4	96.3	96.3	**96.8**	83.3	80.3	42.4	75.8	79.5	**72.3**	88.3	87.0	67.9	84.0	84.6	**82.4**	90.4	90.4	46.9	87.5	88.7	**80.8**
SL-SM	96.0	96.4	95.7	95.9	95.9	**96.0**	83.6	83.6	43.8	75.0	79.7	**73.1**	82.9	82.9	65.8	81.2	79.5	**78.5**	85.9	85.6	48.6	83.4	84.8	**77.7**
ADASYN	98.1	98.1	96.2	97.7	97.8	**97.6**	67.9	66.7	25.6	60.3	65.4	**57.2**	90.6	91.5	70.1	89.3	88.8	**86.1**	93.1	91.9	55.8	91.4	92.2	**84.9**
SPIDER	100.0	100.0	37.8	69.5	100.0	**81.5**	92.5	95.0	29.2	83.3	97.5	**79.5**	100.0	100.0	56.7	73.0	97.8	**85.5**	100.0	100.0	34.8	90.4	99.4	**84.9**
SPIDER2	100.0	100.0	37.8	69.5	100.0	**81.5**	84.5	87.3	43.6	86.4	87.3	**77.8**	96.6	96.6	57.1	73.7	95.4	**83.9**	98.8	98.8	34.9	91.7	98.8	**84.6**
SPIDER3	96.5	97.0	98.2	96.4	96.4	**96.9**	79.0	80.6	36.3	75.0	79.0	**70.0**	87.8	86.2	69.6	84.5	84.0	**82.4**	90.3	90.6	47.1	88.4	89.9	**81.3**
Dataset	**PC2**						**PC3**						**PC4**						**PC5**					
Classifier	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**
N. Samp.	0.0	0.0	31.3	6.3	0.0	**7.5**	26.1	11.2	91.8	0.0	11.2	**28.1**	48.9	37.6	38.2	23.0	46.1	**38.8**	46.3	43.4	44.8	14.5	39.7	**37.7**
SMOTE	99.0	98.9	90.2	98.9	98.9	**97.2**	87.8	85.7	95.5	83.3	85.0	**87.5**	91.3	90.0	80.0	90.7	91.6	**88.7**	98.2	98.1	63.6	97.3	98.0	**91.0**
SL-SM	95.3	95.3	95.9	95.3	95.3	**95.4**	86.1	83.2	93.9	80.9	82.8	**85.4**	88.8	87.4	77.6	88.7	88.3	**86.2**	98.4	98.0	64.8	97.6	98.0	**91.4**
ADASYN	99.0	99.0	90.9	99.0	99.0	**97.4**	89.3	88.1	94.9	86.2	87.4	**89.2**	93.2	91.8	86.8	92.3	91.9	**91.2**	98.2	98.3	63.8	97.0	98.1	**91.1**
SPIDER	100.0	100.0	37.4	97.4	100.0	**87.0**	98.9	99.1	94.2	81.6	98.7	**94.5**	99.2	99.8	44.8	89.0	99.3	**86.4**	97.8	98.4	47.2	86.2	97.7	**85.5**
SPIDER2	100.0	100.0	37.4	97.4	100.0	**87.0**	94.3	94.2	94.1	80.8	95.7	**91.8**	95.3	96.1	43.9	92.5	95.9	**84.7**	97.1	97.6	46.7	85.3	97.5	**84.8**
SPIDER3	98.9	98.9	89.1	98.9	98.9	**96.9**	86.2	83.5	94.5	80.4	83.4	**85.6**	91.6	89.5	74.6	80.9	90.1	**85.3**	97.5	97.4	52.7	95.5	97.1	**88.0**

N.Sap→ No Sampling.

Table 9
Precision results of defect prediction models.

Dataset	CM1						JM1						KC2						KC3					
Classifier	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG	J48	RF	NB	AB	BG	AVG
N. Samp.	36.7	0.0	29.8	0.0	0.0	**13.3**	41.0	55.3	48.1	0.0	51.3	**39.1**	55.2	62.2	65.2	56.0	65.7	**60.9**	42.9	50.0	42.4	54.2	33.3	**44.6**
SMOTE	88.8	97.7	86.6	100.0	100.0	**94.6**	93.8	96.4	82.1	99.4	96.3	**93.6**	93.4	93.7	88.5	93.9	92.8	**92.5**	90.8	95.6	89.3	96.1	95.4	**93.4**
SL-SM	87.2	97.0	81.7	97.6	98.2	**92.3**	92.1	96.2	82.2	99.1	95.8	**93.1**	93.6	93.7	88.4	95.2	93.1	**92.8**	88.0	93.7	82.9	94.4	96.7	**91.1**
ADASYN	91.8	98.0	88.8	98.8	98.0	**95.1**	91.2	94.8	72.7	99.2	94.8	**90.5**	90.1	90.8	82.4	93.3	90.5	**89.4**	87.3	94.6	86.9	94.7	92.5	**91.2**
SPIDER	97.2	93.9	71.3	63.7	86.7	**82.6**	79.5	91.0	63.4	56.8	82.0	**74.5**	75.9	80.5	69.6	67.7	77.5	**74.2**	81.6	88.7	67.9	64.4	83.7	**77.3**
SPIDER2	82.5	91.1	69.4	63.2	87.1	**78.7**	79.9	85.9	82.6	63.9	80.2	**78.5**	81.1	82.5	85.1	70.4	81.0	**80.0**	87.0	85.7	76.2	75.3	82.4	**81.3**
SPIDER3	88.5	96.5	85.6	96.5	96.5	**92.7**	86.7	90.4	83.4	92.4	89.3	**88.4**	86.6	88.7	85.5	85.0	88.2	**86.8**	84.2	89.5	81.8	87.9	87.7	**86.2**
Dataset	**MC1**						**MC2**						**MW1**						**PC1**					
Classifier	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**
N. Samp.	28.6	47.4	7.3	0.0	0.0	**16.7**	60.5	63.0	68.0	60.0	60.7	**62.4**	28.6	35.7	30.0	40.0	50.0	**36.9**	39.5	38.1	30.6	0.0	55.6	**32.8**
SMOTE	99.1	99.2	70.7	99.6	99.7	**93.7**	84.0	93.8	87.5	90.9	91.3	**89.5**	94.7	95.9	81.5	99.3	98.6	**94.0**	95.9	96.9	80.4	99.8	98.0	**94.2**
SL-SM	99.3	99.3	72.5	99.5	99.6	**94.0**	81.7	91.5	84.8	90.6	85.7	**86.9**	91.5	93.3	76.2	97.9	93.4	**90.5**	94.2	95.4	76.9	98.4	97.5	**92.5**
ADASYN	99.4	99.6	83.4	99.7	99.8	**96.4**	71.6	83.9	71.4	82.5	79.7	**77.8**	96.2	96.7	89.2	98.5	100	**96.1**	96.9	98.2	89.4	99.4	98.8	**96.5**
SPIDER	96.6	99.1	70.3	73.4	97.6	**87.4**	81.0	86.4	77.8	73.5	78.5	**79.4**	87.7	92.2	72.1	75.6	91.6	**83.8**	89.9	96.1	79.0	67.6	91.3	**84.8**
SPIDER2	96.6	99.0	70.3	73.4	97.1	**87.3**	73.8	79.3	82.8	69.9	72.7	**75.7**	84.1	91.8	71.9	77.2	88.4	**82.7**	89.8	96.6	79.0	67.3	90.5	**84.6**
SPIDER3	99.8	99.5	71.5	99.6	99.6	**94.0**	84.5	87.0	81.8	83.0	82.4	**83.7**	91.9	95.7	84.0	95.0	95.6	**92.4**	93.9	97.6	81.5	99.0	98.6	**94.1**
Dataset	**PC2**						**PC3**						**PC4**						**PC5**					
Classifier	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**	J48	RF	NB	AB	BG	**AVG**
N. Samp.	0.0	0.0	7.6	25.0	0.0	**6.5**	35.7	39.5	15.3	0.0	40.5	**26.2**	51.5	73.6	46.3	66.1	64.1	**60.3**	60.1	67.3	41.1	60.0	65.9	**58.9**
SMOTE	99.7	100.0	99.7	100.0	100.0	**99.9**	89.8	96.8	53.7	99.7	97.3	**87.5**	92.7	97.5	84.1	93.7	95.1	**92.6**	98.9	99.3	95.3	98.9	99.3	**98.3**
SL-SM	98.8	100.0	97.6	99.7	100.0	**99.2**	89.0	95.9	52.8	99.5	95.4	**86.5**	91.8	96.8	81.3	91.7	95.3	**91.4**	98.7	99.3	95.3	98.3	99.3	**98.2**
ADASYN	99.8	99.9	99.6	100.0	100.0	**99.9**	94.1	97.9	58.2	99.8	97.6	**89.5**	94.5	98.4	85.8	94.5	96.6	**94.0**	99.0	99.3	94.9	99.5	99.4	**98.4**
SPIDER	97.6	99.7	88.5	90.1	98.8	**94.9**	87.4	93.1	48.3	68.2	85.3	**76.5**	90.2	94.0	83.6	78.0	89.4	**87.0**	95.7	96.4	84.3	83.8	95.1	**91.1**
SPIDER2	97.6	99.8	88.5	90.1	98.8	**95.0**	85.7	91.9	47.6	68.2	85.1	**75.7**	90.1	93.5	79.3	78.5	89.7	**86.2**	95.9	96.7	87.6	84.2	94.9	**91.9**
SPIDER3	99.9	99.9	99.6	100.0	100.0	**99.9**	89.1	95.5	53.1	99.7	95.0	**86.5**	92.6	97.1	81.9	97.4	94.1	**92.6**	98.3	99.0	90.2	99.3	99.2	**97.2**

N.Sap→ No Sampling.

vast improvement (6–1500%) in its values in the case of application of oversampling methods as compared to the no sampling scenario (Table 8). It may be noted that in certain scenarios the sensitivity values were 0 when an imbalanced dataset was used. This indicates an inappropriate defect prediction model which cannot correctly identify any defective modules. According to Table 9, precision values also showed an improvement of 13–950% after application of oversampling methods.

The improvement in the values of various performance metrics is due to the balancing of datasets using oversampling, which increases the number of defective examples. In case of imbalanced datasets, very few instances of defective modules were present, so they were difficult to learn by the ML techniques. This is because ML techniques are based on the assumption that the dataset used to train a particular classifier is balanced. Increase in number of defective examples, makes them easy to learn by the ML techniques. This can be observed from the increased sensitivity values in Table 8.

We also analyzed the specificity values of all the models ("no sampling" and oversampling methods) and found a visible decrease of 1–25% in the specificity values using oversampling methods. Due to balancing of datasets, the defective and non-defective examples become equally important to learn, which may lead to decrease in specificity values. The decrease in specificity can be considered a concerning factor as it would lead to the testing of some of the non-defective examples which would be a complete wastage of resources, time and effort. In order to develop a good classifier, a balance between sensitivity and specificity should be achieved. With balancing of datasets, balanced results have been obtained between sensitivity and specificity. This can be observed from the improved AUC values.

6.1.2. RQ1b: which is the best oversampling method to improve the performance of ML techniques for software defect prediction in this study?

We perform Friedman test using AUC, sensitivity and precision performance measures to statistically compare the performance of different oversampling methods with the scenario when imbalanced datasets are used (no sampling). The Friedman test allocates mean ranks. A lower mean rank of a sampling method indicates better comparative performance of that method. We do not use specificity for evaluation as it is more biased towards majority class instances and this work focuses more on minority class instances (defective modules), as chances of misclassifying them are high. Misclassification of defective modules can lead to project failure and high cost to company. We first state the null and alternate hypothesis investigated by the Friedman test.

Null Hypothesis (H1, H2, H3): The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, ABM1 and BG) show no improvement with balanced datasets (achieved by application of different oversampling methods: ADASYN, SL-SM, SMOTE, SPIDER, SPIDER2 and SPIDER3) as compared to the scenario where imbalanced datasets are used.

Alternate Hypothesis (H1a, H2a, H3a): The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, ABM1 and BG) are better with balanced datasets (achieved by application of different oversampling methods: ADASYN, SL-SM, SMOTE, SPIDER, SPIDER2 and SPIDER3) as compared to the scenario where imbalanced datasets are used.

Table 10 shows the results of Friedman test using AUC, sensitivity and precision performance measures. The last column in the table denotes the *p*-value. The test results show that in 35 out of 36 cases, the scenario where no sampling is done (i.e. the case of imbalanced datasets) shows worst results. The Friedman results were found significant in 34 cases. Furthermore, the test results depict that ADASYN outperforms all other oversampling methods in majority of the cases (20 out of 36). This is due to the adaptive nature of ADASYN method. It adapts itself according to the need to generate synthetic minority samples and automatically chooses the value of *k* (nearest neighbor) and n (amount of oversampling) on the basis of position of each minority sample in the dataset. Unlike SMOTE method and its variants, ADASYN does not generate equal number of synthetic samples for each minority sample. It focuses more on those minority samples that lie in the safe region and ignores those which are noise. According to Table 10, SMOTE is also an effective oversampling method as it achieves the best rank in ten out of 36 cases. The application of oversampling methods significantly outperformed the scenario where imbalanced datasets are used. It can also be ascertained that our proposed method SPIDER3 which is an enhancement of SPIDER2, significantly outperforms SPIDER2 in twenty-two cases. Hence, it can be used as a balancing filter for imbalanced datasets. As in majority of the datasets, the tests show significant results, we can safely reject the null hypothesis H1, H2 and H3.

As the Friedman test yields significant results, we perform post-hoc Wilcoxon signed rank test to ascertain the pairwise comparisons amongst ADASYN and "No sampling" scenario in all the datasets. We found the results of ADASYN were significantly better at a cut-off of 0.05 for AUC, sensitivity as well as precision values. According to the guidelines discussed in Section 4.4, the computed effect size was found to be large (0.5–0.6). These results advocate the use of ADASYN.

We also performed pairwise comparison of ADASYN with other investigated sampling methods and found it to be comparable. The pairwise statistical comparison indicates that though the performance of AUC, sensitivity and precision was noticeably better with ADASYN in majority of cases according to Friedman test, the results were not significant pairwise. Lessmann et al. [52] reported a similar result when they performed pairwise comparisons of different ML classifiers for defect prediction. The results indicate that oversampling leads to vast improvement in performance of ML classifiers as it helps in learning of both defective and non-defective instances. However, the choice of oversampling method is not very crucial as all oversampling methods exhibit improved results. We favor the performance of ADASYN as it does not need manual feeding of K and n parameters which are required in other oversampling methods.

6.2. RQ2: what is the comparative performance of the proposed version of SPIDER2 technique i.e. SPIDER3 and the original SPIDER2 and SPIDER techniques for software defect prediction?

Although it can be noticed from the above stated Friedman results (Table 10) that SPIDER3 outperforms SPIDER2 and SPIDER in majority of the cases but still to further justify the result we apply Wilcoxon signed rank test with Bonferroni correction. The hypothesis H4 for Wilcoxon test is stated below:

Null Hypothesis H4: The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, AB and BG) do not differ significantly when SPIDER, SPIDER2 and SPIDER3 oversampling methods are used to balance the imbalanced datasets.

Alternate Hypothesis H4a: The (AUC, sensitivity or precision) results of the defect prediction models developed using five different ML classifiers (J48, RF, NB, AB and BG) are better when SPIDER3 oversampling method is used as compared to SPIDER and SPIDER2 methods to balance the imbalanced datasets.

The Wilcoxon signed rank test performs pairwise comparison of SPIDER and SPIDER2 with SPIDER3 of the defect prediction models developed by all the investigated ML techniques together on all the

Table 10
Friedman test results.

Datasets	PM	Rank1	Rank2	Rank3	Rank4	Rank5	Rank6	Rank7	*p*-value
CM1	***AUC***	ADASYN	SPIDER	SMOTE	SPIDER2	SPIDER3	SL-SM	N.Sap	0.009
	Sens.	SPIDER	ADASYN	SPIDER2	SMOTE	SPIDER3	SL-SM	N.Sap	0.008
	Prec.	ADASYN	SMOTE	SL-SM	SPIDER3	SPIDER	SPIDER2	N.Sap	0.001
JM1	***AUC***	SMOTE	SL-SM	ADASYN	SPIDER	SPIDER3	SPIDER2	N.Sap	0.020
	Sens.	SPIDER2	SMOTE	SPIDER	SL-SM	SPIDER3	ADASYN	N.Sap	0.027
	Prec.	SMOTE	SL-SM	ADASYN	SPIDER3	SPIDER2	SPIDER	N.Sap	0.001
KC2	***AUC***	SMOTE	SL-SM	ADASYN	SPIDER3	SPIDER2	N.Sap	SPIDER	<0.001
	Sens.	SMOTE	SL-SM	SPIDER3	ADASYN	SPIDER2	SPIDER	N.Sap	<0.001
	Prec.	SL-SM	SMOTE	ADASYN	SPIDER3	SPIDER2	SPIDER	N.Sap	<0.001
KC3	***AUC***	SMOTE	SPIDER	SPIDER2	SL-SM	ADASYN	SPIDER3	N.Sap	0.026
	Sens.	SMOTE	SPIDER	SPIDER2	SPIDER3	ADASYN	SL-SM	N.Sap	0.092
	Prec.	SMOTE	SL-SM	ADASYN	SPIDER3	SPIDER2	SPIDER	N.Sap	<0.001
MC1	***AUC***	ADASYN	SPIDER	SPIDER2	SPIDER3	SL-SM	SMOTE	N.Sap	0.016
	Sens.	ADASYN	SPIDER	SPIDER2	SPIDER3	SMOTE	SL-SM	N.Sap	0.015
	Prec.	ADASYN	SPIDER3	SL-SM	SMOTE	SPIDER	SPIDER2	N.Sap	<0.001
MC2	***AUC***	SMOTE	SL-SM	SPIDER	SPIDER3	SPIDER2	ADASYN	N.Sap	0.005
	Sens.	SPIDER2	SPIDER	SL-SM	SMOTE	SPIDER3	ADASYN	N.Sap	0.001
	Prec.	SMOTE	SL-SM	SPIDER3	SPIDER	ADASYN	SPIDER2	N.Sap	<0.001
MW1	***AUC***	ADASYN	SPIDER	SPIDER3	SPIDER2	SMOTE	SL-SM	N.Sap	0.008
	Sens.	ADASYN	SPIDER	SPIDER2	SMOTE	SPIDER3	SL-SM	N.Sap	0.011
	Prec.	ADASYN	SMOTE	SPIDER3	SL-SM	SPIDER	SPIDER2	N.Sap	<0.001
PC1	***AUC***	ADASYN	SPIDER	SPIDER3	SPIDER2	SMOTE	SL-SM	N.Sap	0.006
	Sens.	ADASYN	SPIDER	SPIDER2	SPIDER3	SMOTE	SL-SM	N.Sap	0.013
	Prec.	ADASYN	SMOTE	SPIDER3	SL-SM	SPIDER	SPIDER2	N.Sap	<0.001
PC2	***AUC***	ADASYN	SPIDER3	SMOTE	SPIDER	SPIDER2	SL-SM	N.Sap	0.009
	Sens.	ADASYN	SPIDER	SPIDER2	SMOTE	SPIDER3	SL-SM	N.Sap	0.012
	Prec.	SMOTE	SPIDER3	ADASYN	SL-SM	SPIDER2	SPIDER	N.Sap	<0.001
PC3	***AUC***	ADASYN	SPIDER	SMOTE	SPIDER2	SL-SM	SPIDER3	N.Sap	0.093
	Sens.	SPIDER	ADASYN	SMOTE	SPIDER2	SPIDER3	SL-SM	N.Sap	0.001
	Prec.	ADASYN	SMOTE	SPIDER3	SL-SM	SPIDER	SPIDER2	N.Sap	<0.001
PC4	***AUC***	ADASYN	SMOTE	SPIDER	SL-SM	SPIDER3	SPIDER2	N.Sap	0.001
	Sens.	ADASYN	SPIDER	SPIDER2	SMOTE	SPIDER3	SL-SM	N.Sap	0.003
	Prec.	ADASYN	SMOTE	SPIDER3	SL-SM	SPIDER	SPIDER2	N.Sap	<0.001
PC5	***AUC***	ADASYN	SMOTE	SL-SM	SPIDER3	SPIDER2	SPIDER	N.Sap	0.003
	Sens.	SL-SM	ADASYN	SMOTE	SPIDER	SPIDER3	SPIDER2	N.Sap	<0.001
	Prec.	ADASYN	SMOTE	SL-SM	SPIDER3	SPIDER2	SPIDER	N.Sap	<0.001

PM: Performance Measure, Sens.→ Sensitivity, Prec.→ Precision, N.Sap→ No Sampling.

Table 11
Wilcoxon test results.

Pair	Based on AUC values	Based on sensitivity values	Based on precision values
SPIDER3 vs SPIDER	S+ (0.017) W = 607	NS- (0.178) W = 789.5	S+ (<0.001) W = 51
SPIDER3 vs SPIDER2	S+ (0.001) W = 475	NS- (0.058) W = 677	S+ (<0.001) W = 22.5

NS+: SPIDER3 is better than SPIDER2/SPIDER; NS-: SPIDER3 is worse than SPIDER2/SPIDER;.
S+: SPIDER3 is significantly better than SPIDER2/SPIDER.

datasets used in the study. The results of the test are depicted in Table 11 with *p*-values in parenthesis and W statistic. The test was performed at α=0.05 with Bonferroni correction.

The test depicts that SPIDER3 outperforms SPIDER2 significantly in case of AUC and precision while in case of sensitivity the results of SPIDER3 is lower but not significantly. When compared with SPIDER, the SPIDER3 method significantly outperforms SPIDER method in terms of AUC and precision values. On the basis of sensitivity values, the SPIDER outperforms SPIDER3 but not significantly. The results show that SPIDER3 has improved the performance of two important performance measures namely AUC and precision. The effect size of significantly better cases ranged from medium to large (0.3–0.6) except in the case of SPIDER3 vs SPIDER with AUC values (effect size = 0.2). The improvement in SPIDER3 is due to the use of SMOTE method to find synthetic samples for each minority class sample. On the contrary, SPIDER2 simply replicates the existing minority class samples. Hence, the Wilcoxon test results reject null hypothesis H4. Our proposed method SPIDER3 exhibited better results when compared to the original methods (SPIDER and SPIDER2).

6.3. RQ3: what is the effect of using MC learners on imbalanced datasets for software defect prediction?

This study also uses cost sensitive learning to handle the imbalanced data problem in software defect prediction. We use three different cost ratios: 10, 30 and 50 (MC10, MC30 and MC50) to cost sensitize the various ML classifiers used in this study and compare them with original ML classifiers. Tables 12 and 13 state the results obtained by using different cost ratios with MC classifiers (MC learners) as well as those obtained by original ML classifiers. According to the tables, the average performance of ML techniques improved (1–47%) in 6 out of 12 datasets in terms of AUC, when MC learners were used in comparison to the original ML classifiers. However, the average performance of ML techniques decreased in the remaining datasets. The results show that the MC learners with cost ratio 10 outperform the other MC learners in majority of the cases when AUC was used. Similarly, MC learners with cost ratio 50 outperformed the other MC learners and original ML classifiers when evaluated using sensitivity values in majority of cases. In fact, MC learners with all the cost ratios outperformed the

Table 12
AUC and sensitivity results of MC learners.

	Dataset	CM1					JM1					KC2					KC3				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
AUC	**Original**	0.594	0.763	0.694	0.717	0.727	0.616	0.701	0.633	0.669	0.689	0.704	0.825	0.832	0.784	0.825	0.653	0.736	0.661	0.573	0.729
	MC (10)	0.581	0.780	0.686	0.754	0.760	0.623	0.689	0.650	0.500	0.682	0.728	0.812	0.830	0.804	0.811	0.670	0.692	0.663	0.624	0.630
	MC (30)	0.657	0.738	0.684	0.753	0.711	0.535	0.650	0.650	0.500	0.587	0.418	0.723	0.831	0.570	0.548	0.692	0.644	0.656	0.528	0.498
	MC (50)	0.678	0.727	0.688	0.736	0.671	0.483	0.613	0.650	0.500	0.500	0.471	0.702	0.832	0.499	0.500	0.517	0.646	0.652	0.506	0.500
	Dataset	**MC1**					**MC2**					**MW1**					**PC1**				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
	Original	0.521	0.883	0.708	0.842	0.86	0.698	0.717	0.702	0.616	0.676	0.449	0.716	0.728	0.711	0.705	0.719	0.844	0.768	0.793	0.82
	MC (10)	0.682	0.917	0.704	0.793	0.787	0.575	0.576	0.676	0.588	0.502	0.583	0.726	0.711	0.668	0.745	0.691	0.848	0.717	0.802	0.816
	MC (30)	0.604	0.877	0.713	0.816	0.853	0.575	0.534	0.674	0.510	0.500	0.661	0.709	0.705	0.688	0.700	0.801	0.850	0.716	0.804	0.792
	MC (50)	0.643	0.868	0.708	0.812	0.834	0.566	0.498	0.674	0.499	0.500	0.562	0.733	0.703	0.678	0.500	0.803	0.842	0.718	0.81	0.776
	Dataset	**PC2**					**PC3**					**PC4**					**PC5**				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
	Original	0.448	0.836	0.877	0.914	0.828	0.616	0.831	0.766	0.791	0.824	0.777	0.945	0.836	0.913	0.92	0.817	0.977	0.937	0.959	0.975
	MC (10)	0.652	0.814	0.867	0.826	0.849	0.710	0.819	0.564	0.809	0.806	0.859	0.918	0.728	0.881	0.900	0.871	0.971	0.93	0.960	0.966
	MC (30)	0.662	0.892	0.862	0.880	0.860	0.744	0.807	0.548	0.787	0.768	0.850	0.890	0.730	0.855	0.858	0.908	0.965	0.929	0.959	0.968
	MC (50)	0.620	0.927	0.860	0.880	0.858	0.733	0.795	0.531	0.789	0.774	0.857	0.877	0.734	0.874	0.833	0.906	0.964	0.929	0.956	0.963
Sensitivity	**Dataset**	**CM1**					**JM1**					**KC2**					**KC3**				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
	Original	26.2	0.0	33.3	0.0	0.0	22.5	20.5	18.5	0.0	17.2	49.5	47.7	42.1	43.9	43.0	33.3	13.9	38.9	36.1	13.9
	MC (10)	59.5	71.4	52.4	90.5	78.6	75.2	79.4	27.8	100.0	90.0	79.4	84.1	57.0	85.0	86.0	69.4	72.2	50.0	75.0	75.0
	MC (30)	59.5	90.5	64.3	95.2	92.9	94.6	97.2	28.8	100.0	99.8	92.5	92.5	57.0	96.3	96.3	75.0	94.4	52.8	97.2	97.2
	MC (50)	73.8	95.2	64.3	97.6	95.2	99.7	99.1	28.8	100.0	100.0	100.0	92.5	57.0	98.1	100	86.1	97.2	52.8	97.2	100.0
	Dataset	**MC1**					**MC2**					**MW1**					**PC1**				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
	Original	8.7	19.6	32.6	0.0	0.0	52.3	38.6	38.6	40.9	38.6	14.8	18.5	55.6	29.6	11.1	24.6	13.1	36.1	0.0	8.2
	MC (10)	30.4	39.1	58.7	26.1	19.6	75.0	93.2	47.7	97.7	100.0	40.7	55.6	59.3	55.6	63.0	55.7	49.2	42.6	85.2	63.9
	MC (30)	30.4	63.0	65.2	63	56.5	88.6	100.0	47.7	100.0	100.0	59.3	74.1	63.0	85.2	88.9	72.1	93.4	45.9	95.1	93.4
	MC (50)	37.0	71.7	71.7	76.1	76.1	97.7	100.0	47.7	100.0	100.0	92.6	81.5	63.0	88.9	100	88.5	98.4	47.5	96.7	96.7
	Dataset	**PC2**					**PC3**					**PC4**					**PC5**				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
	Original	0.0	0.0	31.3	6.3	0.0	26.1	11.2	91.8	0.0	11.2	48.9	37.6	38.2	23.0	46.1	46.3	43.4	44.8	14.5	39.7
	MC (10)	18.8	63.0	62.5	37.5	18.8	68.7	78.4	94.8	85.8	85.8	84.8	91.6	44.4	97.2	94.4	77.1	88.2	70.5	87.2	88.4
	MC (30)	25.0	37.5	62.5	68.8	43.8	77.6	94.8	94.8	93.3	91.8	88.2	98.9	47.2	97.8	97.8	83.1	94.4	70.9	91.9	93.4
	MC (50)	18.8	56.3	62.5	75.0	62.5	82.8	96.3	95.5	96.3	97.8	91.0	99.4	47.8	97.8	97.8	85.3	96.3	70.9	92.1	95.3

Table 13
Precision results of MC learners.

	Dataset	CM1					JM1					KC2					KC3				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
Precision	**Original**	36.7	0.0	29.8	0.0	0.0	41.0	55.3	48.1	0.0	51.3	55.2	62.2	65.2	56	65.7	42.9	50.0	42.4	54.2	33.3
	MC (10)	20.2	25.9	20.2	22.4	23.4	28.5	28.3	45	21.5	25.3	42.9	41.7	58.1	42.1	43.6	28.7	28.6	28.6	24.5	23.1
	MC (30)	20.5	17.3	20.1	17.9	17.0	22.3	22.5	45	21.5	21.6	21.3	30.4	57.5	22.2	22.3	27.3	20.9	27.9	18.6	18.3
	MC (50)	18.7	16.1	20	15.8	14.6	21.5	21.7	44.8	21.5	21.5	20.5	26.8	57.5	20.6	20.5	18.0	19.8	27.5	18.4	18.6
	Dataset	**MC1**					**MC2**					**MW1**					**PC1**				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
	Original	28.6	47.4	7.3	0.0	0.0	60.5	63.0	68.0	60.0	60.7	28.6	35.7	30.0	40.0	50.0	39.5	38.1	30.6	0.0	55.6
	MC (10)	15.7	32.7	4.9	15.8	21.4	36.7	38.3	56.8	38.4	35.2	17.7	27.3	21.1	26.8	27.4	20.5	27.3	15.8	18.4	20.7
	MC (30)	16.3	14.8	4.7	7.5	11.7	39.8	35.2	53.8	35.8	35.2	18.2	16.0	21.5	16.3	11.9	22.2	18.8	15.6	16.0	15.7
	MC (50)	17.2	10.7	4.8	6.3	9.0	36.8	35.2	52.5	35.2	35.2	11.5	13.1	21.5	13.8	10.7	21.3	16.4	15.7	15.4	14.3
	Dataset	**PC2**					**PC3**					**PC4**					**PC5**				
	Classifier	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG	J48	RF	NB	AB	BG
	Original	0.0	0.0	7.6	25	0.0	35.7	39.5	15.3	0.0	40.5	51.5	73.6	46.3	66.1	64.1	60.1	67.3	41.1	60.0	65.9
	MC (10)	7.7	7.1	3.9	14	27.3	25.8	30.3	13.0	25.2	28.0	42.1	39.4	27.6	32.8	36.8	40.6	39.3	26.1	31.7	38.4
	MC (30)	8.2	6.5	3.4	9.5	8.8	27.2	22.5	12.6	19.7	20.8	39.8	29.9	27.0	30.3	30.9	39.2	32.0	26.0	24.2	32.0
	MC (50)	5.7	6.0	3.4	8.1	7.8	26.2	19.5	12.7	17.5	18.3	40.1	26.1	26.8	28.0	30.3	36.7	29.2	25.9	22.2	29.7

original dataset in case of sensitivity with a percentage increase of 10–600%. This outcome is because the cost values of MC learners were set in such a way so as to decrease the number of false negative predictions. The lower the number of FNs, the higher will be the chance of correctly classifying defective modules. Penalizing the classifier for false negatives, leads to increase in sensitivity in all the datasets. However, in order to develop effective predictive models, a balance between specificity and sensitivity values must be achieved. Table 13 shows that the precision values decreased in majority of the cases (except CM1 and PC2). The overall decrease in precision values is a concerning factor and it should be a researcher's aim to achieve balance among all the performance measures.

We analyzed the best MC cost ratio on a specific dataset by evaluating the AUC, sensitivity and precision values obtained by all MC learners on it. An MC learner which gave good values for all the three performance measures was chosen as the best. We found that MC10 obtained the best results on eight datasets (CM1, JM1, KC2, KC3, MC2, PC3, PC4 and PC5), while MC50 (PC1 and MC1) and MC30 (MW1, PC2) exhibited best results on two datasets each. It may be noted that the best values achieved on a specific dataset by an MC learner depends on cost ratio as MC learners with different cost ratios obtained best results on each dataset.

The study also performed Wilcoxon signed rank test with Bonferroni correction in order to make a pairwise comparison between MC learners and the original ML classifiers. The test was performed on AUC, sensitivity and precision values of the defect prediction models developed by the application of five ML techniques (J48, RF, NB, AB and BG) on all the datasets (original as well as MC learners) of the study. The results of the test are reported in Table 14 with p-values in parenthesis and W statistic. It was observed that original ML classifiers outperformed MC learners in case of AUC and precision values, while in case of sensitivity values MC learners significantly outperformed the original ML classifiers. The effect size of statistically significant cases ranged from medium to large (0.3–0.6). As the defect prone classes are important to learn correctly, MC learners help improve the prediction of defective modules. However, they may not always yield improved overall results.

6.4. RQ4: what is the comparative performance of best oversampling methods and MC learners for software defect prediction?

According to the findings in RQ2, ADASYN is the best oversampling method to handle imbalanced datasets. This RQ compares ADASYN with MC learners using Wilcoxon signed rank test. The result of the pairwise comparison of ADASYN and MC learners was evaluated on AUC, sensitivity and precision using all the datasets of the study together where models were developed by the application of five ML techniques (J48, RF, NB, AB, BG). It may be noted that we compared oversampling results with the best MC cost ratio on a corresponding dataset i.e. MC10 for CM1, JM1, KC2, KC3, MC2, PC3, PC4 and PC5, MC50 for PC1, MC1 and MC30 for MW1, PC2. The results of Wilcoxon test are reported in Table 15.

According to the results in Table 15, the ADASYN significantly outperformed the MC learners in all the three cases (AUC, Sensitivity and Precision). The effect size for all the cases ranged from medium to large (0.3–0.6). ADASYN is effective because of its property to balance the data using density distribution method. This property makes ADASYN work automatically while in other oversampling methods as well as in case of MC learners, we are supposed to set one or more parameters (k, n or cost ratios) manually. As MC learners are primarily designed to address the asymmetric loss of two types of misclassification errors (false positives and false negatives), they are unable to solve the imbalanced data problem.

Table 14
Wilcoxon test results for comparing original scenario vs MC learners.

Pair	Based on AUC values	Based on sensitivity values	Based on precision values
Original ML classifiers vs MC10	NS+ (0.096) W = 738.5	S- (<0.001) W = 0	S+ (<0.001) W = 385.5
Original ML classifiers vs MC30	S+ (0.002) W = 508.5	S- (<0.001) W = 0	S+ (<0.001) W = 226.5
Original ML classifiers vs MC50	S+ (<0.001) W = 411	S- (<0.001) W = 0	S+ (<0.001) W = 182

NS+: Original ML classifiers are better than MC10/MC30/ MC50; S-: Original ML classifiers are significantly worse than MC10/MC30/MC50; S+: Original ML classifiers are significantly better than MC10/MC30/MC50.

Table 15
Wilcoxon test results for comparing ADASYN with MC learners.

Pair	Based on AUC values	Based on sensitivity values	Based on precision values
ADASYN vs MC Learners	S+ (<0.001) W = 5	S+ (0.001) W = 514	S+ (<0.001) W = 0

S+: ADASYN is significantly better than MC learner.

6.5. Discussion of results

As investigated in the research questions, the use of oversampling methods improves the performance of ML techniques for developing software defect prediction models. According to the analysis of AUC, sensitivity and precision performance metrics, ADASYN performed well on majority of datasets. The SMOTE technique also performed well. According to sensitivity values, the SPIDER technique and its variants also performed well. Furthermore, it was evaluated that the proposed modified version of SPIDER2 i.e. SPIDER3 was effective in developing defect prediction models as compared to original SPIDER and SPIDER2 oversampling methods. A recent study by Lin et al. [20] mentioned that redundant datasets may result in over-learning and will decrease the generalized performance of a technique. They dynamically select examples during training to reduce redundancy [20]. Our proposed oversampling method SPIDER3, attempts to reduce the redundancy in oversampled datasets by creating synthetic samples rather than just replicating original defective examples. This leads to its improved performance than original SPIDER and SPIDER2 methods.

The results of the study indicate the importance of providing datasets with appropriate number of instances for both defective as well as non-defective modules for developing effective defect prediction models using ML techniques. Figs. 3 and 4 depict the boxplots of AUC and sensitivity values using imbalanced data ("no sampling"), ADASYN, SMOTE and SPIDER3 methods. According to the figures, the AUC and sensitivity values increased in majority of the cases with all ML classifiers when oversampling methods were used as compared to the case when no sampling technique was used.

However, the results of RQ3 indicate that the performance of defect prediction models using MC learners decreased in a number of cases than the original ML technique in terms of AUC and precision values. Although, the sensitivity values improved consistently using MC learners. It may also be noted that the meta-cost ratio is important for a specific dataset as the best result on a specific dataset may be achieved by a specific meta-cost ratio [12,39].

This study advocates the use of oversampling methods as compared to MC learners for developing defect prediction models using imbalanced datasets. The statistical analysis of the results also indicated the same (RQ4). This result is in accordance with our previous study [39], which advocated the use of resampling technique as compared to MC learners.

We also compared the results of our study with 10 previous literature studies [4,6,12,30–36], found closest to our work as they used sampling or cost-sensitive techniques for handling imbalanced nature of NASA datasets. However, out of the 10 close literature studies we could compare our results with only those studies which used at least one performance measure used in our study i.e. AUC, sensitivity, specificity or precision and which used one of our ML techniques (J48, RF, NB, AB or BG) as base learners for application of any imbalanced method. This criteria resulted in only five studies [4,12,31,33,34], out of which only two studies [33,34] could be compared as values of individual performance measures could not be obtained from three studies [4,12,31].

The comparison on the basis of AUC and sensitivity values with literature studies is depicted in Tables 16 and 17 respectively. The top 3 methods in a particular scenario are depicted in bold. We compared the results of literature studies with our top two oversampling techniques (ADASYN and SMOTE) along with our proposed SPIDER3 technique. According to Table 16, the AUC and sensitivity values obtained after application of ADASYN, SMOTE and SPIDER techniques are better than the ones obtained by Wang and Yao [33]. Similarly, the results in Table 17 support the use of investigated oversampling techniques, as better AUC values were obtained in comparison to Rodriguez et al. [34] in majority of cases. We also performed Wilcoxon signed rank test with Bonferroni correction to pairwise compare our sampling methods (ADASYN, SMOTE and SPIDER3) with the ones proposed by authors in [33,34]. ADASYN, SMOTE and SPIDER3 significantly outperformed Dynamic Adaboost.NC using both AUC and sensitivity values. Also, our three oversampling methods significantly outperformed the AUC results of all the imbalanced learning methods investigated by Rodriguez et al. [34]. The effect size of Wilcoxon test in all cases was large (0.6).

Summarizing, we state certain research guidelines for researchers in order to develop defect prediction models with ML techniques using imbalanced datasets.

- It is important to provide effective training in imbalanced datasets using appropriate number of training examples of defective as well as non-defective modules. The superiority of ADASYN indicates that the oversampling methods should appropriately choose a method for replicating or creating synthetic samples which avoids redundancy and noise.
- Researchers should first evaluate the application of oversampling methods such as ADASYN, SMOTE and the proposed method in the study SPIDER3 on unbalanced datasets as they show a vast improvement in the performance of developed software defect prediction models.

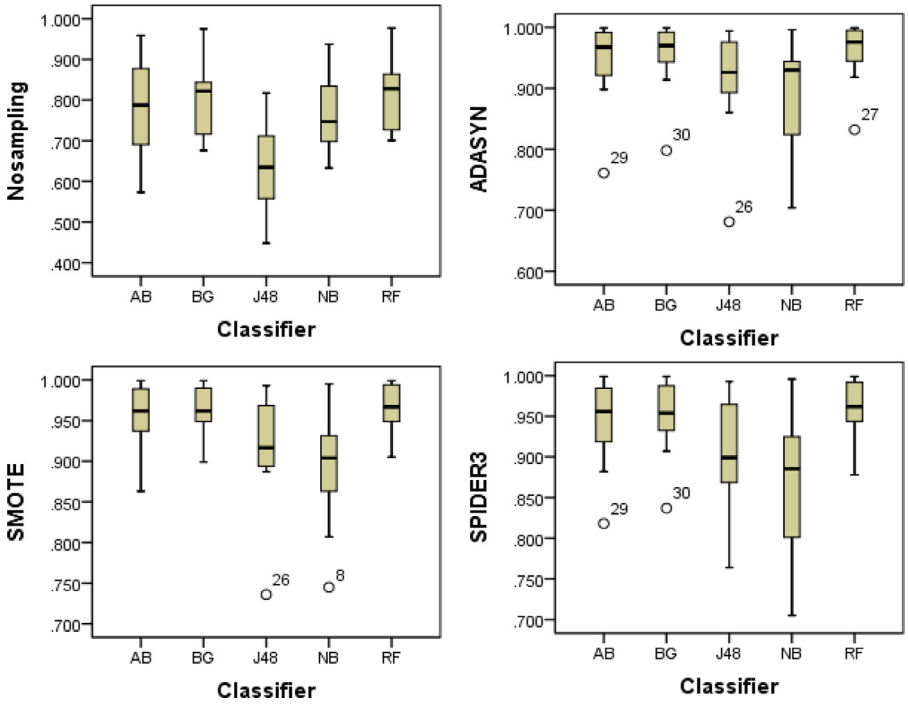

Fig. 3. AUC values after application of No sampling, ADASYN, SMOTE and SPIDER methods.

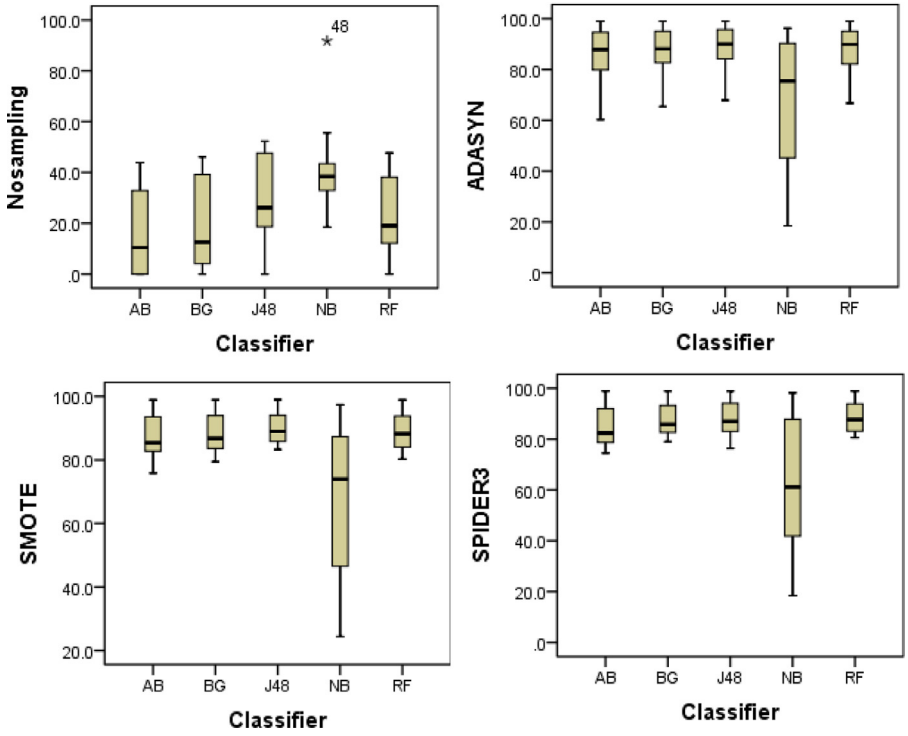

Fig. 4. Sensitivity values after application of No sampling, ADASYN, SMOTE and SPIDER methods.

- In case the researchers want to apply MC learners, it is important to evaluate several cost ratios on a dataset and choose the best cost ratio for a corresponding dataset.
- The results of the study indicate that application of oversampling methods yield better results than application of MC learners. However, this may be specific to the evaluated datasets as application of MC learners lead to large improvement in sensitivity values and therefore may be evaluated.

7. Threats to validity

This section states the various possible threats to the validity of the study.

7.1. Conclusion and internal validity

Conclusion validity primarily addresses itself with the non-evaluation of statistical analysis of the study. However, this study

Table 16
Comparison of our results with Wang and Yao [33].

Dataset	AUC results using J48				Sensitivity results using J48			
	Our results			Wang and Yao [22]	Our Results			Wang and Yao [22]
	ADASYN	SMOTE	SPIDER3	Dynamic Adaboost. NC	ADASYN	SMOTE	SPIDER3	Dynamic Adaboost. NC
CM1	**0.92**	**0.89**	**0.88**	0.79	**87.7**	**84.5**	**83.6**	59.0
JM1	**0.88**	**0.90**	**0.86**	0.77	**77.9**	**84.3**	**82.6**	66.0
KC2	**0.91**	**0.93**	**0.87**	0.83	**84.4**	**89.7**	**85.6**	77.1
KC3	**0.86**	**0.89**	**0.83**	0.79	**84.0**	**87.2**	**86.5**	57.9
MC2	**0.68**	**0.74**	**0.76**	0.65	**67.9**	**83.3**	**79.0**	52.1
MW1	**0.93**	**0.91**	**0.91**	0.71	**90.6**	**88.3**	**87.8**	48.6
PC1	**0.96**	**0.95**	**0.95**	0.87	**93.1**	**90.4**	**90.3**	57.0
PC3	**0.91**	**0.89**	**0.89**	0.82	**89.3**	**87.8**	**86.2**	70.3
PC4	**0.95**	**0.95**	**0.94**	0.92	**93.2**	**91.3**	**91.6**	88.7

Table 17
Comparison of our results with Rodriguez et al. [34].

Dataset	AUC results using J48										
	Our results			Rodriguez et al. [34]							
	ADASYN	SMOTE	SPIDER3	RUS	ROS	SMOTE	SB	RUSBoost	MC	CSC-R	CSC-MC
CM1	**0.92**	**0.89**	**0.88**	0.58	0.56	0.56	0.73	0.71	0.64	0.60	0.55
JM1	**0.88**	**0.90**	**0.86**	0.63	0.59	0.63	0.67	0.67	0.63	0.65	0.55
KC2	**0.91**	**0.93**	**0.87**	0.61	0.61	0.62	0.70	0.67	0.63	0.64	0.54
KC3	**0.86**	**0.89**	**0.83**	0.58	0.60	0.61	0.71	0.68	0.67	0.68	0.59
MC1	**0.99**	**0.99**	**0.98**	0.67	0.61	0.62	0.83	0.79	0.60	0.63	0.52
MC2	0.68	**0.74**	**0.76**	0.62	0.62	0.58	**0.75**	**0.74**	0.62	0.57	0.57
MW1	**0.93**	**0.91**	**0.91**	0.66	0.57	0.59	0.71	0.73	0.69	0.60	0.61
PC1	**0.96**	**0.95**	**0.95**	0.70	0.61	0.70	0.84	0.82	0.73	0.67	0.64
PC3	**0.91**	**0.89**	**0.89**	0.66	0.57	0.62	0.80	0.79	0.69	0.66	0.67
PC4	**0.95**	**0.95**	**0.94**	0.76	0.69	0.73	0.92	0.91	0.83	0.81	0.79
PC5	**0.99**	**0.99**	**0.99**	0.66	0.66	0.68	0.79	0.78	0.69	0.73	0.66
	AUC results using NB										
CM1	**0.94**	**0.93**	**0.90**	0.71	0.71	0.71	0.57	0.61	0.70	0.70	0.61
JM1	**0.70**	**0.74**	**0.70**	0.69	0.69	0.69	0.58	0.60	0.69	0.69	0.58
KC2	0.77	**0.84**	0.77	**0.79**	**0.79**	**0.79**	0.73	0.63	0.78	**0.79**	0.65
KC3	**0.86**	**0.89**	**0.83**	0.68	0.67	0.68	0.66	0.65	0.67	0.68	0.65
MC1	**0.95**	**0.93**	**0.94**	0.91	0.91	0.91	0.86	0.75	0.90	0.91	0.78
MC2	**0.74**	**0.81**	**0.77**	0.73	0.73	0.73	0.72	0.65	0.69	0.73	0.64
MW1	**0.93**	**0.89**	**0.88**	0.75	0.75	0.76	0.75	0.71	0.72	0.75	0.71
PC1	**0.94**	**0.91**	**0.92**	0.78	0.79	0.78	0.67	0.58	0.73	0.78	0.64
PC2	**0.99**	**0.99**	**0.99**	0.86	0.83	0.81	0.59	0.79	0.87	0.83	0.60
PC3	**0.89**	**0.89**	**0.87**	0.77	0.76	0.76	0.51	0.63	0.61	0.77	0.57
PC4	**0.93**	**0.92**	**0.89**	0.82	0.83	0.84	0.73	0.56	0.74	0.83	0.67
PC5	**0.95**	**0.94**	0.93	**0.94**	**0.94**	**0.94**	0.90	0.75	**0.94**	**0.94**	0.73

RUS → Random Under-Sampling, ROS → Random Over-Sampling, SB: SMOTEBoost, MC: MetaCost, CSC-R: Cost Sensitive Classifier with Resampling, CSC-MC: Cost sensitive Learning with minimum cost.

performed statistical evaluation using two non-parametric tests, i.e. Friedman and Wilcoxon signed rank test which reduces this threat. The use of ten-fold cross validation for developing defect prediction models reduces validation bias [26] and strengthens the analysis of developed defect prediction models.

With respect to internal validity, it may be noted that oversampled training set is not a representative of the original dataset. It needs to be ascertained if a "balanced" distribution is the most desirable one for training a classification model. This is dependent on the domain and the classification method and is in general guided by empirical results [68,69]. It may be the case that the balanced distribution used while training may not generalize to test set. This may lead to bias in learned concepts. This threat exists in our study. However, since in the scenario of defect prediction, defective classes are of utmost important, we have used oversampling methods and MC methods to correctly predict maximum number of defective classes. Furthermore, the use of a robust performance measure such as AUC, is a popular choice for evaluating the effectiveness of developed models [21,57] as it maintains a proper balance between both specificity and sensitivity values. This reduces the threat to bias.

7.2. Construct validity and external validity

As the method level metrics used in the study have been investigated by a number of researchers [3,8,30–36] for developing effective defect prediction models in literature, they have been properly verified and there is no threat to construct validity of independent variables. Similarly, the datasets used in the study are publicly available and verified. Thus, reducing the threat to construct validity.

With respect to external validity, this study performed empirical validation on 12 NASA industrial datasets. Though, the results of this study are generalized to the domain of NASA datasets, they may be different for other domains. Thus, the threat to external validity exists. The evaluated NASA datasets have been developed using C, C++ and Java language. Use of datasets developed using different languages increases the generalizability of the study as the

results may be applicable to other datasets of these languages. Furthermore, the study uses appropriately sized and appropriate number of datasets contributing to the external validity of the study's results.

8. Conclusions and future work

This study ascertains if balancing the datasets improves the performance of ML techniques in software defect prediction. The study uses five ML classifiers namely J48, RF, NB, AB and BG to develop defect prediction models. In order to handle imbalanced data, the study uses five existing oversampling methods: SMOTE, ADASYN, Safe-Level-SMOTE, SPIDER and SPIDER2. We implemented all the above-mentioned oversampling methods in order to perform our analysis. We also proposed a modified version of SPIDER2 i.e. SPIDER3 and implemented the same. Furthermore, MC learners were also evaluated to ascertain their effectiveness in improving the results of the developed defect prediction models on imbalanced datasets. Moreover, a comparative analysis between MC learners and oversampling methods was also performed. The empirical validation was done using AUC and three traditional metrics: sensitivity, precision and specificity and the outcomes of the study were statistically assessed. The conclusions of the study are as follows:

- *Oversampling methods significantly improved classification performance:* A significant improvement was observed in the performance of ML techniques when oversampling methods were used to handle imbalanced nature of datasets. With the application of various oversampling methods, average AUC values computed cumulatively on all the 12 datasets using all ML techniques were as high as 0.94, average precision values as high as 93.65% and average sensitivity values as high as 83.48% were obtained. ADASYN was observed to be the best oversampling method amongst other investigated sampling methods due to its adaptive nature and capability to balance the data automatically using density distributions. In addition, SMOTE and other oversampling methods also performed well.
- *SPIDER3 performed well as compared to SPIDER2 and SPIDER methods:* The proposed method SPIDER3 i.e. the modified version of SPIDER2 significantly outperforms SPIDER2 and SPIDER methods in case of AUC and precision, while it showed comparative results when evaluated using sensitivity measure. The improved performance of the modified version is attributed to generation of synthetic samples for each defective class sample rather than just their replication.
- *MC learners improved only sensitivity values:* MC learners were also investigated to handle the imbalance data problem. They cost sensitize the classifier in order to improve performance of the predictive models by using different cost ratios for various misclassification errors. However, the MC learners only yielded improved results for sensitivity measure. AUC and precision results were not effective as compared to the defect prediction models developed using the original ML techniques.
- *ADASYSN performed better than MC learners:* A pairwise comparison between ADASYN and MC learners with best cost ratio concluded that ADASYN is a better method for handling imbalanced data problem as compared to MC learners. ADASYN significantly outperformed the MC learners in case of AUC, sensitivity and precision performance metrics. The favorable nature of ADASYN is due to effective density distribution function.

Hence, the analysis performed in this study can be used to develop efficient defect prediction models in case of imbalanced training data. Our future work will focus on another category of sampling methods and that is undersampling methods. Other cost-sensitive approaches and ensemble methods may also be investigated in the domain of software defect prediction for developing efficient models using imbalanced data.

References

[1] R. Malhotra, A systematic review of machine learning techniques for software fault prediction, Appl. Soft Comput. 27 (2015) 504–518.
[2] M. Tan, L. Tan, S. Dara, C. Mayeux, Online defect prediction for imbalanced data, in: Proceedings of the 37th International Conference on Software Engineering, 2, IEEE Press, 2015, pp. 99–108.
[3] K. Dejaeger, T. Verbraken, B. Baesens, Toward comprehensible software fault prediction models using bayesian network classifiers, IEEE Trans. Softw. Eng. 39 (2) (2013) 237–257.
[4] M. Cheng, G. Wu, M. Yuan, H. Wan, Semi-supervised software defect prediction using task-driven dictionary learning, Chin. J. Electron. 25 (6) (2016) 1089–1096.
[5] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, An empirical comparison of model validation techniques for defect prediction models, IEEE Trans. Softw. Eng. 43 (1) (2017) 1–18.
[6] M. Liu, L. Miao, D. Zhang, Two-stage cost-sensitive learning for software defect prediction, IEEE Trans. Reliab. 63 (2) (2014) 676–686.
[7] Y. Abdi, S. Parsa, Y. Seyfari, A hybrid one-class rule learning approach based on swarm intelligence for software fault prediction, Innov. Syst. Softw. Eng. 11 (4) (2015) 289–301.
[8] Ö.F. Arar, K. Ayan, Software defect prediction using cost-sensitive neural network, Appl. Soft Comput. 33 (2015) 263–277.
[9] T. Zimmermann, N. Nagappan, A. Zeller, Predicting bugs from history, Softw. Evol. 4 (1) (2008) 69–88.
[10] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics, Inf. Sci. 250 (2013) 113–141.
[11] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, K.I. Matsumoto, The effects of over and under sampling on fault-prone module detection, in: *Proceedings of the First International Symposium on* Empirical Software Engineering and Measurement, *ESEM*, IEEE, 2007, pp. 196–204.
[12] N. Seliya, T.M. Khoshgoftaar, The use of decision trees for cost-sensitive classification: an empirical study in software quality prediction, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 1 (5) (2011) 448–459.
[13] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches, IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) 42 (4) (2012) 463–484.
[14] P. Domingos, Metacost: a general method for making classifiers cost-sensitive, in: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 1999, pp. 155–164.
[15] J.C. Riquelme, R. Ruiz, D. Rodríguez, J. Moreno, Finding defective modules from highly unbalanced datasets, Actas Talleres Jorn. Ing. Softw. Bases Datos 2 (1) (2008) 67–74.
[16] R. Shatnawi, Improving software fault-prediction for imbalanced data, in: *Proceedings of the 2012 International Conference on* Innovations in Information Technology (IIT), IEEE, 2012, pp. 54–59.
[17] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert Syst. Appl. 36 (4) (2009) 7346–7354.
[18] Z. Mahmood, D. Bowes, P.C. Lane, T. Hall, What is the impact of imbalance on software defect prediction performance? in: Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, ACM, 2015, October, p. 4.
[19] G.M. Weiss, K. McCarthy, B. Zabar, Cost-sensitive learning vs. sampling: which is best for handling unbalanced classes with unequal error costs? DMIN 7 (2007) 35–41.
[20] M. Lin, K. Tang, X. Yao, Dynamic sampling approach to training neural networks for multiclass imbalance classification, IEEE Trans. Neural Netw. Learn. Syst. 24 (4) (2013) 647–660.
[21] A.B. De Carvalho, A. Pozo, S.R. Vergilio, A symbolic fault-prediction model based on multiobjective particle swarm optimization, J. Syst. Softw. 83 (5) (2010) 868–882.
[22] Y. Liu, T.M. Khoshgoftaar, N. Seliya, Evolutionary optimization of software quality modeling with multiple repositories, IEEE Trans. Softw. Eng. 36 (6) (2010) 852–864.
[23] P.C. Pendharkar, Exhaustive and heuristic search approaches for learning a software defect prediction model, Eng. Appl. Artif. Intell. 23 (1) (2010) 34–40.
[24] A.T. Mısırlı, A.B. Bener, B. Turhan, An industrial case study of classifier ensembles for locating software defects, Softw. Qual. J. 19 (3) (2011) 515–536.
[25] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A general software defect-proneness prediction framework, IEEE Trans. Softw. Eng. 37 (3) (2011) 356–370.
[26] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Inf. Softw. Technol. 54 (3) (2012) 248–256.
[27] L. Yu, An evolutionary programming based asymmetric weighted least squares support vector machine ensemble learning methodology for software repository mining, Inf. Sci. 191 (2012) 31–46.
[28] G. Czibula, Z. Marian, I.G. Czibula, Software defect prediction using relational association rule mining, Inf. Sci. 264 (2014) 260–278.
[29] C. Jin, S.W. Jin, Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization, Appl. Soft Comput. 35 (2015) 717–725.

[30] J. Zheng, Cost-sensitive boosting neural networks for software defect prediction, Expert Syst. Appl. 37 (6) (2010) 4537–4543.
[31] N. Seliya, T.M. Khoshgoftaar, J. Van Hulse, Predicting faults in high assurance software, in: Proceedings of the 2010 IEEE 12th International Symposium on High-Assurance Systems Engineering (HASE), IEEE, 2010, November, pp. 26–34.
[32] D. Rodríguez, R. Ruiz, J.C. Riquelme, J.S. Aguilar–Ruiz, Searching for rules to detect defective modules: a subgroup discovery approach, Inf. Sci. 191 (2012) 14–30.
[33] S. Wang, X. Yao, Using class imbalance learning for software defect prediction, IEEE Trans. Reliab. 62 (2) (2013) 434–443.
[34] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, J.C. Riquelme, Preliminary comparison of techniques for dealing with imbalance in software defect prediction, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2014, May, p. 43.
[35] M.J. Siers, M.Z. Islam, Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem, Inf. Syst. 51 (2015) 62–71.
[36] T. Wang, Z. Zhang, X. Jing, L. Zhang, Multiple kernel ensemble learning for software defect prediction, Autom. Softw. Eng. 23 (4) (2016) 569–590.
[37] Stefanowski, J., & Wilk, S. (2008). Selective Pre-processing of Imbalanced Data for Improving Classification Performance. Lecture Notes on Computer Science, 5182, 283–292.
[38] H. He, Y. Bai, E.A. Garcia, S. Li, ADASYN: adaptive synthetic sampling approach for imbalanced learning, in: Proceedings of IEEE International Joint Conference on Neural Networks, IJCNN (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 1322–1328.
[39] R. Malhotra, M. Khanna, An empirical study for software change prediction using imbalanced data, Empir. Softw. Eng. 22 (6) (2017) 1–46.
[40] Z. Miao, L. Zhao, W. Yuan, R. Liu, Multi-class imbalanced learning implemented in network intrusion detection, in: Proceedings of the 2011 International Conference on Computer Science and Service System (CSSS), IEEE, 2011, pp. 1395–1398.
[41] M. Zięba, Service-oriented medical system for supporting decisions with missing and imbalanced data, IEEE J. Biomed. Health Inform. 18 (5) (2014) 1533–1540.
[42] T.M. Padmaja, N. Dhulipalla, R.S. Bapi, P.R. Krishna, Unbalanced data classification using extreme outlier elimination and sampling techniques for fraud detection, in: Proceedings of International Conference on Advanced Computing and Communications, ADCOM, IEEE, 2007, pp. 511–516.
[43] P. Jeatrakul, K.W. Wong, C.C. Fung, Classification of imbalanced data by combining the complementary neural network and SMOTE algorithm, in: Proceedings of International Conference on Neural Information Processing, Springer, Berlin, Heidelberg, 2010, November, pp. 152–159.
[44] D. Ramyachitra, P. Manikandan, Imbalanced dataset classification and solutions: a review, Int. J. Comput. Bus. Res. 5 (4) (2014).
[45] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[46] B.X. Wang, N. Japkowicz, Imbalanced data set learning with synthetic samples, in: Proceedings of IRIS Machine Learning Workshop, 19, 2004.
[47] C. Bunkhumpornpat, K. Sinapiromsaran, C. Lursinsap, Safe-level-smote: safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem, Adv. Knowl. Discov. Data Min. 5476 (2009) 475–482.
[48] K. Napierała, J. Stefanowski, S. Wilk, Learning from imbalanced data in presence of noisy and borderline examples, in: Rough Sets and Current Trends in Computing, Springer, Berlin/Heidelberg, 2010, pp. 158–167.
[49] I. Gondra, Applying machine learning to software fault-proneness prediction, J. Syst. Softw. 81 (2) (2008) 186–195.
[50] M.H. Halstead, Elements of Software Science, 7, Elsevier, New York, 1977, p. 127.
[51] T.J. McCabe, A complexity measure, IEEE Trans. Softw. Eng. SE-2 (4) (1976) 308–320.
[52] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (4) (2008) 485–496.
[53] J. Petrić, D. Bowes, T. Hall, B. Christianson, N. Baddoo, The jinx on the NASA software defect data sets, in: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2016, p. 13.
[54] M. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: some comments on the NASA software defect datasets, IEEE Trans. Softw. Eng. 39 (9) (2013) 1208–1215.
[55] D. Gray, D. Bowes, N. Davey, Y. Sun, B. Christianson, The misuse of the NASA metrics data program data sets for automated software defect prediction, in: Proceedings of 15th Annual Conference on Evaluation & Assessment in Software Engineering, EASE, IET, 2011, pp. 96–103.
[56] T. Menzies, A. Dekhtyar, J. Distefano, J. Greenwald, Problems with precision: a response to comments on 'data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (9) (2007) 637–640.
[57] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[58] T. Fawcett, An introduction to ROC analysis, Pattern Recognit. Lett. 27 (8) (2006) 861–874.
[59] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.
[60] J. Pallant, SPSS Survival Manual, McGraw-Hill Education„ UK, 2013.
[61] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, 2nd ed., Lawrence Erlbaum Associates, 1988.
[62] H. He, Y. Bai, E.A. Garcia, S. Li, ADASYN: adaptive synthetic sampling approach for imbalanced learning, in: Proceedings of IEEE International Joint Conference on Neural Networks, IJCNN (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 1322–1328.
[63] D.R. Wilson, T.R. Martinez, Reduction techniques for instance-based learning algorithms, Mach. Learn. 38 (3) (2000) 257–286.
[64] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, Data Mining: Practical machine Learning Tools and Techniques, Morgan Kaufmann, 2016.
[65] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[66] K.P. Murphy, Naive Bayes Classifiers, University of British Columbia, 2006.
[67] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[68] N.V. Chawla, Data mining for imbalanced datasets: an overview, Data Mining and Knowledge Discovery Handbook, Springer US, 2009, pp. 875–886.
[69] G.M. Weiss, F. Provost, Learning when training data are costly: the effect of class distribution on tree induction, J. Artif. Intell. Res. 19 (2003) 315–354.

Dr. Ruchika Malhotra is Associate Head and Assistant Professor at the Department of Software Engineering, Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She was awarded with prestigious Raman Fellowship for pursuing Post-doctoral research in Indiana University Purdue University Indianapolis 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. Her h-index is 21 as reported by Google Scholar. She is the author of book titled "Empirical Research in Software Engineering" published by CRC press and co-author of a book on Object Oriented Software Engineering published by PHI Learning. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics and the definition and validation of software metrics. She has published more than 140 research papers in international journals and conferences.

Shine Kamal has completed her post-graduation from Delhi Technological University (formerly Delhi College of Engineering), Department of Software Engineering. Delhi, India. She has completed her graduation from Punjabi University, Patiala, India. Her research interests are in empirical research in software engineering, improving, statistical and adaptive prediction models, software metrics and software testing.