Software Quality Journal (2020) 28:1581–1614
https://doi.org/10.1007/s11219-020-09525-y

An empirical study on predictability of software maintainability using imbalanced data

Ruchika Malhotra[1] · Kusum Lata[1]

Published online: 5 August 2020

Abstract
In software engineering predictive modeling, early prediction of software modules or classes that possess high maintainability effort is a challenging task. Many prediction models are constructed to predict the maintainability of software classes or modules by applying various machine learning (ML) techniques. If the software modules or classes need high maintainability, effort would be reduced in a dataset, and there would be imbalanced data to train the model. The imbalanced datasets make ML techniques bias their predictions towards low maintainability effort or majority classes, and minority class instances get discarded as noise by the machine learning (ML) techniques. In this direction, this paper presents empirical work to improve the performance of software maintainability prediction (SMP) models developed with ML techniques using imbalanced data. For developing the models, the imbalanced data is preprocessed by applying data resampling methods. Fourteen data resampling methods, including oversampling, undersampling, and hybrid resampling, are used in the study. The study results recommend that the safe-level synthetic minority oversampling technique (Safe-Level-SMOTE) is a useful method to deal with the imbalanced datasets and to develop competent prediction models to forecast software maintainability.

Keywords Software maintainability prediction · Machine learning · Data resampling · Imbalanced learning

1 Introduction

With the progression of time, software systems are becoming substantially large and complex. The maintenance of such complex systems is becoming enormously challenging for software

✉ Kusum Lata
kusumlata@dtu.ac.in

Ruchika Malhotra
ruchikamalhotra2004@yahoo.com

[1] Department of Computer Science and Engineering, Delhi Technological University, Delhi 110042, India

professionals. Some software organizations may be unable to begin new projects since most of their assets may be devoted to maintaining the old systems. Therefore, predicting software maintainability is becoming an impending area in software engineering. It focuses on the design and development of prediction models to forecast software maintainability when the software is in the initial stages of its development.

The knowledge about high-maintainability effort classes in advance helps to allocate the limited resources of an organization optimally to these classes. It results in good quality and highly maintainable software developed within the time and budget. Over the years, there has been a debate on measuring software maintainability. Software maintainability has been seen as a software quality attribute and defined according to numerous facets. According to Coleman et al. (1994), maintainability is defined as "the ease with which software component can be modified to correct the existing faults after delivery." Aggarwal et al. (2002) suggested that maintainability is an integrated measure of software characteristics like the readability of source code, software understandability, and quality of documentation. Software maintainability is the degree of difficulty in understanding and performing changes in the software, according to Schneberger (1997). Oman and Hagemeister (1994) proposed the maintainability index (MI) to assess and quantify the maintainability. The maintainability index is a linear polynomial equation computed from the software metrics. The software metrics describe the characteristics of software like average Halstead volume of a program, average cyclomatic complexity of the program, the average number of lines of source code, and the average number of comments. The polynomial equation on evaluation gives a single number that indicates the maintainability. The lower the value of MI, the lesser would be the maintainability of the software and vice versa (Oman and Hagemeister 1994). Later, Ash et al. (1994) and Coleman et al. (1995) revised the MI proposed by Oman and Hagemeister (1994) and validated it on software written procedural programming languages like Pascal, C, Ada, and Fortran. However, the correctness of MI for software systems implemented in object-oriented (OO) languages has not been advocate much. Li and Henry (1993) defined software maintainability in the form of the lines of source code changed during the period of maintenance to correct faults. This study advocated that the software maintainability has a strong correlation with the OO metrics describing various software characteristics such as inheritance, coupling, and cohesion. Later various researchers (Van Koten and Gray 2006; Zhou and Leung 2007; Thwin and Quah 2005) measured the maintainability in the form of the lines of source code changed during the maintenance. The more changes encountered in a class in the maintenance phase means more maintainability effort is required for that class and vice versa. The ultimate goal of developing these models is to predict those software classes accurately that require high maintainability effort. In the literature, various maintainability prediction models have developed with statistical, ML, evolutionary, and hybridized techniques. The software metrics have been used as predictors for developing the maintainability prediction models (Wang et al. 2009; Dagpinar and Jahnke 2003; Kumar and Rath 2015; Malhotra and Lata 2017). The high maintainability effort classes are critical for any project because these classes must be tested cautiously to decrease the probability of occurrence of faults. Also, such classes should need to be well-documented to augment understandability to carry out future maintenance activities. Software metrics ranging from procedural metrics such as number of unique operators, number of unique operands, and cyclomatic complexity of module (McCabe 1976; Halstead 1977; Schneidewind 1979) to OO metrics (Chidamber and Kemerer 1994; Henderson-Sellers 1996; Martin 2002) characterize and quantify various aspects of the software systems and play a vital role in model development. The dataset used for training the maintainability prediction models should consist of sufficient instances of high- and low-maintainability effort classes to train the model effectively. However, in reality, during maintenance, few software classes demand complex interventions, resulting in more changes in the code lines, i.e., high maintainability effort. Therefore, there is an imbalance among the number of instances of the classes requiring high (minority class)- and low-maintainability effort (majority class), resulting in an imbalanced dataset. It is challenging to train the prediction models to predict the unseen data points of these classes with reasonable accuracy using imbalanced data.

Therefore, this study is important because it deals with the development of effective SMP models by treating imbalanced datasets to predict high maintainability effort classes accurately. Identification of high maintainability effort classes is crucial as these classes need more attention during the software maintenance and testing phase as such classes are likely to be sources of defects and future advancements (Eski and Buzluca 2011). Appropriate distribution of resources to these classes helps in enhancing the quality of the software product. However, with the imbalanced datasets, many ML techniques encounter enormous trouble (Chawla et al. 2004; Fawcett and Provost 1997; Kubat et al. 1998), and the prediction models obtain higher prediction accuracies just for the majority class rather than those for both of the classes. The software maintainability prediction models developed using imbalanced data do not have any practical significance as they may misclassify the minority class (high maintainability effort) instances. Thus, such misclassification may lead to improper resource allocation to the misclassified classes resulting in poor quality software products. Early prediction of high maintainability effort classes accurately before the software product release helps the software professionals to test these classes critically. Also, software developers can effectively refactor such classes to improve their maintainability.

In the software engineering domain, the imbalanced class problem is addressed to build competent models to predict faulty and change-prone classes (Malhotra and Khanna 2017; Choeikiwong and Vateekul 2015; Gao et al. 2015). However, no study dealt with handling the imbalanced class problem in SMP. Therefore, to treat the imbalanced data problem in SMP, this study applies various data resampling methods, including oversampling, undersampling, and hybrid resampling, before learning the SMP models to improve their performance.

Data resampling: The data resampling methods modify the training dataset in such a manner that it includes enough quantity of data points of minority and majority class. These methods include oversampling, undersampling, and hybrid resampling. In the oversampling techniques, the new data points of the rare or minority class produced so that the dataset contains the proportionate number of instances of the minority and majority class. The undersampling methods work by expelling a few data points of the majority class to make a proportionate dataset. Hybrid resampling combines the oversampling and undersampling strategy (Kotsiantis et al. 2006).

The study has the following objectives:

- To construct SMP models to predict high maintainability effort classes by treating the imbalanced datasets with data resampling techniques.
- To assess the predictive performance of the developed SMP models and validate them statistically.
- To investigate the improvement in the predicting performance of the built SMP models after data resampling.

We achieve the above-specified objectives by finding answers to the following research questions (RQs).

RQ1: What is the performance of SMP models developed using ML techniques on original imbalanced datasets?

RQ2a: What is the performance of SMP models developed using ML techniques after balancing the datasets with data resampling methods?
RQ2: Which data resampling method improves the performance of the prediction models the most?

In the interest of answering the above research questions, we build up SMP models that use OO metrics as predictors and software maintainability as the outcome. The datasets extracted from eight Apache open-source software packages are used to develop SMP models with the application of ML techniques. The stable evaluators Balance and G-mean are used in this study to evaluate the predictive performance of the SMP models. Also, the study conducts a statistical analysis of constructed models to strengthen the conclusions. The organization of the remaining paper is given below:

Section 2 presents related work. Section 3 describes the research methodology. Section 4 describes the results of the study. Section 5 presents the threats to validity, and Section 6 describes the conclusions and future work.

2 Related work

We present the related work in two sections. The first section discusses the state-of-the-art of SMP models, whereas the second section discusses the studies which have faced and handled the class imbalance problem.

2.1 Literature work related to studies predicting software maintainability

This section discusses various studies that have proposed models to predict software maintainability. Different learning techniques ranging from ML, statistical, and hybridized have been used to construct models by building up the relationship of software metrics with maintainability. An empirical analysis of the dataset extracted from two software systems written in Java language is conducted by Dagpinar and Jahnke (2003). The study revealed that coupling and size are strong maintainability predictors. The study by Elish and Elish (2009) proposed TreeNet classifier. The outcome of the study evidenced that OO metrics are good predictors of maintainability.

A non-linear model, project-pursuit regression, is given by Wang et al. (2009) to build SMP models. The study developed SMP models using OO metrics extracted from two commercial software systems. The study by Jin and Liu (2010) used clustering techniques to predict software maintainability. The study empirically validated OO metrics collected from software projects written in the C++ language. A comprehensive statistical comparison of 27 different ML techniques to develop models to forecast maintainability was conducted by Kaur and Kaur (2013). The study revealed that instance-based classifier performs best to predict maintainability. The study by Olatunji and Ajasin (2013) proposed extreme learning machines to develop SMP models using OO metrics. The study by Zhang et al. (2015) suggested a framework, SMPlearner, where they employed 44 metrics collected at different hierarchy levels and developed SMP models. They validated SMPlearner on eight datasets pertaining to open-source software systems. Kumar and Rath (2015) built the SMP model by applying hybridized techniques. This study was carried out on two commercial datasets widely used in the literature. Wang et al. (2019) proposed a fuzzy network framework to predict software maintainability using two widely used commercial datasets, and the study advocated that the proposed framework improves the accuracy of SMP models compared with standard fuzzy-based models. Kumar et al. (2019) used class-level software metrics with three different types of neural networks to train SMP models. The genetic algorithm with gradient decent approach is used to find optimal weights of neural networks. Schnappinger et al. (2019) extracted software metrics using static analysis tools and predicted software maintainability using diverse ML techniques. Thus, in this way, we see various models to predict software maintainability have been successfully developed and validated on software metrics. This study is related to the studies published in the literature in the manner that like the previous studies (Zhang et al. 2015; Kumar and Rath 2015; Wang et al. 2009; Kaur and Kaur 2013), this study also predicts software maintainability using the internal characteristics of the software systems. However, the imbalanced data problem has not been touched in any of the studies published. This problem is taken care of in this investigation to develop effective prediction models to forecast the software maintainability.

2.2 Literature work related to studies taking care of class imbalance problem

The imbalanced class problem arises when in a particular dataset, the quantity of data points of one class is far more compared to the other. With such datasets, the many ML techniques encounter serious troubles (Laurikkala 2001) . Many times, with the imbalanced datasets, ML techniques learn to predict only the dominant (majority) class instances. In contrast, the cases of the class of interest (minority class) are discarded by being treated as noise (Maloof 2003). Various resolutions are given by the researchers to address the imbalanced class problem at the algorithm level and the data level. The data level solutions employ numerous kinds of data resampling strategies to get rid of the issue of imbalanced data. The algorithmic solutions include regulating the cost of both classes to tackle the problem (Chawla et al. 2004). For predictive modeling in software engineering, the imbalanced class problem encountered prediction of classes that are likely to be change-prone and defective. This problem is solved in different ways to uplift the performance of the predictive models. Choeikiwong and Vateekul (2015) proposed an algorithm-level solution to class imbalance problem for software fault prediction. They implemented a classifier in which the separation hyperplane of the Threshold Adjustment Support Vector Machine (R-SVM) is adjusted to cut down the bias from the dominant class. This study was performed on 12 datasets. The findings of the paper showed that R-SVM improved the prediction rate of models for predicting faulty modules. Gao et al. (2015) examined four different scenarios of feature selection and data sampling to boost the predictive capability of defect prediction models developed with imbalanced datasets. This study confirmed that feature selection on resampled data improves the predictive capability of the models. Laradji et al. (2015) proposed an average probability ensemble (APE) incorporating several base classifiers to cope up with the imbalanced class problem. To further improve the prediction capability, feature selection was combined with APE in this study. Siers and Islam (2015) proposed a cost-sensitive classifier, which was an ensemble of the decision trees to tackle the problem. Pelayo and Dick (2007) investigated the synthetic minority oversampling, which balances the proportion of the defective and non-defective modules.

The paper by Sun et al. (2012) used ensemble and coding schemes to handle class imbalance for predicting defective classes. The study first converted the unbalanced binary class data into multiclass balanced data by using coding-based schemes and then trained the defect prediction models from this multiclass data. Tan et al. (2015) employed three data resampling approaches and updatable classification techniques to boost the predictive capability of fault prediction models learned using imbalanced class datasets. Wang and Yao (2013) investigated different methods, including resampling, ensembles, and threshold moving, to improve defect prediction models. The study also proposed a dynamic version of AdaBoost to handle the imbalanced class issue in the area of defect prediction. A paper by Zheng (2010) proposed three cost-sensitive boosting techniques to improve the prediction rate of neural networks. Malhotra and Khanna (2017) developed software change prediction models from imbalanced data by employing three data resampling methods and meta cost learners. In this way, various studies in the literature have dealt with class imbalance situations in predictive modeling in the software engineering domain to improve defect prediction and change prediction models. However, the imbalanced class issue is untouched in literature in the maintainability prediction. Therefore, in this direction, this study will be the first study to deal with the imbalanced class problem for SMP.

3 Research methodology

The research methodology comprises all the components of the study, experimental design, data resampling methods, and ML techniques used for developing the SMP models.

3.1 Components of the empirical study

3.1.1 Predictor and response variable

Training a prediction model for a predictive task requires a dataset comprising predictors (independent) and response (dependent) variable. For software quality prediction models, the predictor variables are the software metrics. Software metrics quantify various aspects of the software systems and are used to predict and estimate different software characteristics (Chidamber and Kemerer 1991; Ebert and Dumke 2007; Fenton and Bieman 2014). Over the years, different software metrics (procedural and OO) are proposed, and their relationship with software maintainability is established.

3.1.2 Predictor variables

We use OO metrics as the independent variable to develop prediction models in this study as the study is carried out using OO systems developed in the Java programming. The OO metrics used in the study include Chidamber and Kemerer (C&K) metric suite (Chidamber and Kemerer 1994), Quality Model for Object-Oriented Design-QMOOD (Bansiya and Davis 2002) metric suite, and metrics proposed by Henderson-Sellers (Henderson-Sellers 1996) and Martin (Martin 2002). C&K metrics suite includes the metrics, namely WMC: weighted methods per class, DIT: depth of inheritance tree, NOC: number of children of a class, CBO: coupling between the objects, LOCM: lack of cohesion among methods, and RFC: response for class. The QMOOD metric suite includes the metrics, namely, MOA: measure of aggregation, DAM: data access metric, MFA: measure of functional abstraction, NPM: number of public methods, and CAM: cohesion among methods of a class. The Martin metrics are Ce: efferent coupling and Ca: afferent coupling. Few other metrics used in the study are IC: inheritance coupling, CBM: coupling between the class methods and AMC: average method complexity, LOC: lines of source code, and LCOM3. LCOM3 is the variation of LCOM given by Henderson-Sellers. These metrics describe different aspects, namely, cohesion, coupling, size, inheritance, composition, and encapsulation of OO systems.

The metrics WMC, NPM, LOC, DAM, and AMC are indicators of the size of a class. The metrics CBO, RFC, Ca, Ce, IC, and CBM measure the coupling. The inheritance property is measured with the help of NOC, DIT, and MFC. The metrics LCOM, CAM, and LCOM3 are indicators of class cohesion, whereas MOA measures composition. These metrics that quantify the different characteristics of a class are regarded as internal quality attributes. The class qualities such as testability, reliability, reusability, and maintainability belong to a set of quality attributes that are called external quality attributes (Al-Dallal 2013). The present study is based on Morasca's (2009) suggestion to predict software maintainability, i.e., external quality attribute, by constructing probabilistic models. In this study, the prediction models use the above OO metrics as the predictor variables to estimate the external quality attribute, namely, software maintainability. The internal quality attributes used have significant relation with software maintainability. For instance, the attributes WMC, NPM, LOC, DAM, and AMC measure the size of a class. If the size increases, the code would likely be less maintainable, i.e., likely to require high-maintainability effort (Al-Dallal 2013). Table 1 shows a brief explanation of the predictors used in this paper. As researchers for predictive modeling in the domain of software engineering have extensively used these metrics (Singh et al. 2010; Kpodjedo et al. 2011; Giger et al. 2012; Gyimothy et al. 2005; Olague et al. 2007; Elish and Al-Rahman Al-Khiaty 2013), this motivates us to use these metrics for our study. Radjenovic et al. (2013) conducted a review of 106 papers predicting defects in classes. This review revealed that C&K metrics have frequently been used for predicting faults in classes. Therefore, our study also has taken C&K metrics to validate them for predicting software maintainability. The paper Lu et al. (2012) assessed sixty-two OO metrics for estimating the change-prone classes. The study discovered in the study that LOC, CBO, LCOM, and CAM are significant metrics. The C&K metrics, combined with QMOOD metrics, are used by Eski and Buzluca (2011) to predict change-prone classes. The study advocated the combination of C&K and QMOOD metrics be the competent predictors for predicting classes that are likely to be changed in the future. Hence, our paper uses an effective combination of predictors successfully validated in literature for predictive modeling tasks in software engineering.

Let us see how we can compute the values for the few of the OO metrics used in this study. For example, let us say we want to calculate the value for WMC and DIT.

Let us see how we can compute the values for the few of the OO metrics used in this study. For example, let us say we want to calculate the value for WMC and DIT. Suppose class A has three methods M_1, M_2, and M_3 with complexities X_1, X_2, and X_3, respectively. Then, WMC for class A will be given as WMC = sum (X_1, X_2, X_3). Let us see how to calculate another metric. Consider an OO program with six classes: CL_1, CL_2, CL_3, CL_4, CL_5, and CL_6. The classes CL_2, CL_3, and CL_4 are derived from CL_1, and CL_5 and CL_6 are derived from CL_4. DIT computes the class depth in the hierarchy of inheritance. DIT for CL_1, i.e., $DIT(CL_1) = 0$,

Table 1 OO metrics studied

Metric	Definition
WMC	The sum of cyclomatic complexities of all methods of a class is called weighted methods per class.
DIT	This metric measure the depth of a class in the inheritance hierarchy.
NOC	It is defined as the number of immediately derived classes of a particular class.
CBO	This metrics shows the number of classes to which a specific class is coupled where the coupling can the data accesses, function calls inheritance, etc.
RFC	The number of functions executed in response to a message received by an object of a class is called response for class.
LCOM	This metric measures the number of methods in a class that is not related to themselves by sharing some of the class data.
Ca	This metrics measures merely the number of classes that use a particular class.
Ce	The number of classes used by a specific class is called efferent coupling.
NPM	This is the count of the number of public methods defined in a class.
DAM	It is described as the number of protected or private attributes declared in a class divided by the total number of attributes declared in that class.
MOA	This metrics is a count of the number of data fields in a class whose type is user-defined.
MFA	This metric defines the number of methods that are inherited by a specific class divided by the sum of methods accessible by that class.
CAM	This metric measures the relationship between class methods. The association is found by their list of arguments and defined as the sum of the number of unique argument types used by all of the class methods divided by product of the total number of methods in the class and total count of unique argument type in that class.
IC	The number of parent classes to which a given class is coupled is called inheritance coupling of that class.
CBM	This metric computes the total number of methods in a class to which all of the metrics that are inherited, coupled.
AMC	This average method size of each class is called average method complexity.
LOC	It is defined as the total number of lines in the binary code of a class.
LCOM3	LCOM3 is given as $LCOM3 = \frac{1}{n}\left(\sum_{i=1}^{n} f(pi)\right) - \frac{ma}{1-ma}$ Here n = number of attributes in a class; ma = number of methods in a class and $f(pi)$ = number of functions that access an attribute.

DIT(CL$_2$) = 1, DIT(CL$_3$) = 1, DIT (CL$_4$) = 1, DIT(CL$_5$) = 2, and DIT(CL$_6$) = 2. The deeper classes add up to more design complexity. Table 1 presents a brief definition of the OO metrics used.

3.2 Response variable

In this study, maintainability is estimated in change count (CC) metric. This metric is defined as $CC = LOC_{added} + LOC_{deleted} + LOC_{modified}$ where LOC_{added} is lines of source code added during the maintenance phase, $LOC_{deleted}$ is lines of source code deleted during the maintenance period, and $LOC_{modified}$ is lines code modified in the maintenance period. Many studies in the literature (Kumar and Rath 2015; Elish and Elish 2009; Kaur and Kaur 2013) measure maintainability like this. The response variable in the study is maintainability. It has two values, low maintainability effort and high maintainability effort. Maintainability is a binary variable obtained after discretizing the CC metric. The low-maintainability effort classes are those that require a few changes in lines of code in the software in the maintenance phase, whereas high maintainability effort classes require more changes in source code during maintenance. The study aims at developing efficient prediction models to predict high maintainability effort classes with good accuracy. Early prediction of such classes helps to allocate the resources to these classes in an optimal way, thereby producing high-quality and maintainable software.

3.2.1 Software system studied

We undertook this study using OO metrics extracted from open-source software. The brief explanation of the software under consideration in the present study is as follows:

- Apache Bcel (Byte Code Engineering Library) is proposed to present clients a helpful method to make, analyze, and control Java class documents that are ending with .class. An object, which is the representation of a class, contains all information like fields, methods, and particularly bytecode instructions of a particular class.
- Apache Betwixt gives a method for transforming beans into XML and producing digester rules automatically. Just like the BeanInfo system can be utilized to modify the default introspection on a Java object, the digester rules can be customized on a per-type style.
- Apache Io is a Java library that includes numerous classes. The library enables developers to do various everyday tasks efficiently with much less effort. Different functions, such as input, output, utility classes, and comparators, are included in this library.
- Apache Ivy is a robust tool for recording, tracking, resolving, and reporting various project dependencies. It is very flexible, configurable, and has strong integration with Apache Ant, a Java build tool.
- Apache Jcs is a Java Caching System written in Java language. It speeds up the applications by managing the cache data of different forms. In addition to cache data management, it provides various other features like memory management, element grouping, and remote synchronization, to name a few.
- Apache Lang provides various methods that the standard Java libraries unable to deliver. These functions include basic numerical methods, string methods, and concurrency control.
- Apache Log4j is a logging framework that can be configured through external configuration files dynamically. It provides a convenient way to direct logging information to various destinations such as console and database.
- Apache Ode (Orchestration Director Engine) is a software component that is intended to execute BPEL (Business Process Execution Language) business processes. It sends and gets messages to and from web administrations, controls information, and takes care of error handling.

3.2.2 Performance metrics

Prediction of maintainability is regarded as the classification problem in this study. Given training data points of classes labeled as low maintainability effort or high maintainability effort, a classifier can be learned from the data points and used to classify the unknown classes to be low-maintainability effort or high maintainability effort. The classifier's performance is assessed by examining the confusion matrix. Table 2 depicts the confusion matrix for a two-class classification task. The confusion matrix contains the class values in the form of positives and negatives. For the present study, a positive value corresponds to high maintainability effort instances, and the negative class value corresponds to low-maintainability effort instances.

Table 2 Confusion matrix

	Predicted positive	Predicted negative
Actual positive	True positive (TP)	False negative (FN)
Actual negative	False positive (FP)	True negative (TN)

Some widely used traditional performance metrics to evaluate a classifier like an accuracy, error rate, etc. are calculated using a confusion matrix. The performance metrics, accuracy, and error rate assume the uniform class distribution of the positive and negative class. Using these performance measures to evaluate a classifier might gives rise to misleading conclusions in the situation of imbalanced data as these performance measures are intensely inclined towards the majority class. Consider a case where in a particular dataset, 99% of the data points belong to the majority class. Let us say the accuracy of the classifier is 99%. It means the classifier will predict the label of every unseen test data point as that of the label of the majority class. Due to this fact, accuracy or error rate are not recommended for imbalanced data. The minority class is the prime attention in case of the imbalanced data. However, accuracy and error rate give equal importance to the classification error. Due to this reason, the use of these performance evaluators to judge a classifier is suspicious for class imbalance problem. The confusion matrix is used to derive the performance evaluators that assess the classifier's performance independently on positive instances as well as on negative instances. These performance metrics are sensitivity (true positive rate: TPR) and specificity (true negative rate: TNR). However, for imbalanced data, there is a trade-off among TPR and TPN. Therefore, many researchers considered a few stable performance measures to evaluate the prediction models for class imbalance situation. The paper by He and Garcia (2009) advocated that G-mean is a stable metric for assessing the prediction models developed from the imbalanced data. A robust performance evaluator, Balance, to evaluate prediction models developed on an imbalanced dataset is given by Menzies et al. (2007). So, our study assesses the maintainability prediction models using stable and strong evaluators, namely, Balance and G-mean. The formulas for performance evaluators are given in Table 3.

3.2.3 Statistical tests

In empirical research, deriving conclusions entirely from the empirical results without applying the statistical analysis can be misleading (Lessmann et al. 2008). Using statistical tests, establish confidence in the outcomes of an empirical investigation and help to validate the hypothesis formed. We apply statistical analysis at different stages in this study and confirm the corresponding hypothesis built there, e.g., in the first stage, the null hypothesis tested is "the performance of the ML techniques do not differ significantly after applying data resampling methods compared to the situation when no resampling is employed." This hypothesis is validated with the help of the Friedman test. The Friedman test is a distribution-free test independent of any presumptions about the distribution of performance measures before its application. If test statistics obtained after the Friedman test result is sufficiently substantial for the rejection of the null hypothesis, it means the variance in the ML techniques' performance after the data resampling is non-random. In such a situation, we go for a post hoc examination by Wilcoxon signed-rank test. This test is applied after Bonferroni correction. We find the pairwise difference between the best resampling method with the other resampling methods.

Table 3 Performance metrics

Performance measure	Definition	Formula
Sensitivity	It is defined as the percentage of correctly predicted positive classes.	$\text{Sensitivity} = \frac{TP}{TP+FN}$
Specificity	It is defined as the percentage of correctly predicted negative classes.	$\text{Specificity} = \frac{TN}{TN+FP}$
G-mean	It is defined as the geometric mean of sensitivity and specificity.	$\text{G−mean} = \sqrt{\text{Sensitivity} \times \text{Specificity}}$
Balance	It is defined as Euclidean distance between pair of (sensitivity, FPR) and an optimal value of sensitivity = 1 and FPR = 0. Here, FPR is false-positive rate.	$\text{Balance} = 1-\sqrt{(0-FPR)}\ ^2 +(1-\text{Sensitivity})\ \frac{2}{\sqrt{2}}$ where $\text{FPR} = \frac{FP}{FP+TN}$

3.3 Experimental setting

3.3.1 Data collection

This investigation is carried out using eight Apache application packages. These are Apache Bcel, Betwixt, Io, Jcs, Lang, Log4i, and Ode. We analyzed two subsequent versions of each of these software systems. Section 3.1.2 describes the systems investigated in this study. All of these are large-scaled application packages and give us sufficient data points to develop the prediction model. These software systems are OO, written in Java programming language, and OO programming has full potential these days, which has influenced the project selection for this study. We have used the DCRS tool: Data Collection and Reporting System tool (Malhotra et al. 2014) for data extraction out of software systems. Data extraction has successfully been carried out from various open-source repositories using this tool. Only the prerequisite for utilizing this tool is that the repository must use GIT as version control. DCRS tool is successfully employed for data extraction corresponding to Apache application packages because Apache uses GIT.

Two successive versions of the software products have fed as input to the DCRS tool. The change log corresponding to common classes between the two analyzed versions has extracted in the form of log records with the DCRS tool. The log records contain information like a list of modified files, CC (change count), description of changes made. The CKJM tool (http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/), open-source software for extraction of OO metrics, is embedded with the DCRS tool to collect the OO metrics corresponding to Java classes. The OO metrics extracted by the CKJM tool are the quantitative measure of cohesion, coupling, inheritance, etc. of a class. Each common class among the two versions yields one data point for our analysis. Each such data point entails a combination of OO metrics and CC metric representing the lines of code changed. The OO metrics included in the data point are used as the predictors, and CC helps to form the response variable (maintainability). Each generated data point also contains a variable named "alter" containing two values "yes" and "no." If there is a change in a class, alter is assigned yes value; otherwise, no.

3.3.2 Data pre-processing

Various data pre-processing steps carried out on the datasets before beginning with our analysis are given as follows:

- Removal of no change classes: As discussed in Section 3.1.1, OO metrics and CC metric corresponding to the common classes among two successive versions form data points, and each data point also contains the alter variable containing two values yes or no corresponding to one data point. For our analysis, we considered only those data points where the alter variable contains yes values, which means that classes were changed at least once. In this way, we only included data points of changed classes for our analysis. The detail of the software projects with their names, version analyzed, #common classes (number of common classes), and #common classes changed (number of changed common classes), is given in Table 4.
- Outlier detection and removal: Outlier analysis was necessary to develop generalized prediction models. Outliers had extreme variability from the remaining data points in the dataset (Rousseeuw and Leroy 2005), and their detection and removal are essential for building a competent prediction model. Outlier analysis was performed with the Inter-quartile filter using the WEKA tool: Waikato Environment for Knowledge Analysis tool (http://www.cs.waikato.ac.nz/ml/weka/). In this way, 26, 34, 24, 58, 22, 47, 32, and 115 outliers were detected and removed from Apache Bcel, Apache Betwixt, Apache Io, Apache Ivy, Apache Jcs, Apache Lang, Apache log4j, and Apache Ode datasets respectively.
- Data discretization: After the data extraction, the CC metric values ranged from tens of change in the line of code to thousands of modifications in all eight software projects. The classes where thousands of the lines of code are changed are very few. The dependent variable (maintainability) used in this study is formed after discretizing the CC metric into two bins: low-maintainability effort and high-maintainability high maintainability effort. The label low-maintainability effort corresponds to those classes which require less maintainability effort (fewer changes in LOC). Furthermore, the label high-maintainability effort corresponds to the classes requiring more maintainability effort (more effort in the form of LOC changed). The details of software systems after this step are shown in Table 5. The low-maintainability effort data points are regarded as majority class data points as they are a larger in number compare with high maintainability effort (minority class) data points in all the eight software systems used in the study. It is evident from Table 5 that the imbalance ratio (i.e., number of data points of low maintainability effort divided by number of data points of high maintainability effort) varies from 6.29 to 23.80.

3.3.3 Applying data resampling methods

The next step in our experimental setup involves applying data resampling methods to make uniformity in minority and majority examples. We use fourteen resampling methods consisting of oversampling, undersampling, and hybrid resampling in this study.

Table 4 Details of software projects

Name of software	Version analyzed	No. of common classes	No. of common classes changed	% change
Apache Bcel	5.0–5.1	363	360	99.17
Apache Betwixt	0.6–0.7	290	279	96.20
Apache Io	2.0.1–2.2	274	272	99.27
Apache Ivy	1.4.1–2.2.0	619	429	69.30
Apache Jcs	1.2.6.5–1.2.7.9	333	219	65.76
Apache Lang	2.4–2.6	434	267	61.52
Apache Log4j	1.2.14–1.2.15	491	437	89.00
Apache Ode	1.3.1–1.3.2	1060	1004	94.71

3.3.4 Maintainability prediction model development and evaluation

After data resampling, the next step is the development of SMP models. To develop SMP models, we apply ML techniques commonly used in the literature. The details of ML techniques used to construct the models are given in the next section. Tenfold cross-validation strategy is employed during prediction model development. With this cross-validation mechanism, the total available training instances are randomly separated into ten equivalent parts. Nine parts of training data are utilized for model development. The model is validated on the tenth partition. This process is carried out ten times to ensure a low bias of random partitioning. We appraise the performance of the SMP models through G-mean and Balance. Lastly, we do a statistical analysis to establish confidence in the results produced.

3.4 Data resampling methods used

The study executes fourteen resampling techniques for handling imbalanced datasets using the KEEL tool: Knowledge Extraction Based on Evolutionary Learning tool with default parameter settings (https://www.keel.es).

- Adaptive synthetic sampling (ADASYN) adaptively generates synthetic examples corresponding to minority class instances. It employs the weighted density distribution to decide the number of synthetic cases to be made corresponding to each minority class instance. Many examples are created corresponding to harder-to-learn cases, and a smaller number of instances are generated, corresponding to easy-to-learn examples (He et al. 2008).

Table 5 Data discretization results

Software	High maintainability effort classes (minority classes)	Low maintainability effort classes (majority classes)	Imbalance ratio (IR)
Apache Bcel	18	316	17.55
Apache Betwixt	27	218	08.07
Apache Io	10	238	23.80
Apache Ivy	28	343	12.25
Apache Jcs	27	170	06.29
Apache Lang	27	193	07.14
Apache Log4j	42	363	08.64
Apache Ode	57	832	14.59

- Synthetic minority oversampling technique (SMOTE) oversamples the rare class by introducing artificial instance in the dataset. The artificial instances are formed alongside the line joining the minority class example with its k-nearer neighbors. The requisite number of neighbors has been taken from the k-nearer neighbors for this purpose. For instance, if 200% oversampling is needed, two out of five nearer neighbors of particular minority class instances are randomly picked up, and an artificial example is generated corresponding to each. The formation of an artificial example consists of two steps. The first step involves computing the difference between the selected example and its chosen nearer neighbor. The next step is multiplying the difference by a random number between 0 and 1 and adding the resultant value in the initially selected instance (Chawla et al. 2002).
- Borderline synthetic minority oversampling technique (Border-Line-SMOTE) oversamples the instances of the minority that are at the borderline by introducing synthetic samples using SMOTE. A minority class instance is said to belong to the borderline when all its k-nearer neighbor instances belong to the majority class (Han et al. 2005).
- Safe Level-SMOTE: Unlike SMOTE, Safe-Level-SMOTE computes the safe level of each minority class instance before producing synthetic instances. The safe level of a minority class instance is the number of minority class instances from its k-nearer neighbors (Bunkhumpornpat et al. 2009)
- Synthetic minority oversampling technique + edited nearer neighbor (SMOTE-ENN) is a hybridized technique based on SMOTE. In this technique, synthetic samples are generated using SMOTE, and Wilson's Edited Nearest Neighbor rule is applied to filter out the noisy instances (Batista et al. 2004).
- Synthetic minority oversampling technique + Tomek's modification of condensed nearer neighbors (SMOTE-TL) is a hybrid method based on SMOTE. Synthetic examples are generated using SMOTE, after which the Tomek's links are detected and filtered out from the dataset. Two instances belonging to dissimilar classes are supposed to form Tomek's link if the distance between them is less than that of the distance between them to any other sample in the dataset (Batista et al. 2004).
- Random oversampling (ROS) at random generates rare class instances until both rare and dominant classes do not have the same number of examples (Batista et al. 2004).
- Selective pre-processing of imbalanced data (SPIDER) technique involves oversampling the minority class and filtering out the instances from the majority class depending on if they are safe or noisy using KNN classification. After this, the noisy samples are removed (Stefanowski and Wilk 2008).
- Selective pre-processing of imbalanced II (SPIDER-II) involves two phases to pre-process the minority and majority class examples. In the beginning, noisy instances from the dominant class are identified. Then, these instances are removed or relabeled as per the reliable option. Afterward, noisy instances from the rare class are identified and replicated as per the amphl option (Napierala et al. 2010).
- Random undersampling (RUS) at random removes the instances from the dominant class until both of the rare and the dominant class do not have the same number of samples (Batista et al. 2004).
- Condensed nearer neighbor (CNN) method is formed based on the nearer-neighbor rule. It eliminates certain instances out of the dataset without affecting the NN classification performance (Hart 1968).

- Condensed nearer neighbors with Tomek's modification (CNN-T). This technique combines Condensed nearer neighbors and TL. Based on the information given by Tomek's link, certain instances are removed from the dataset without affecting the NN classification performance (Batista et al. 2004).
- Class Purity Maximization Clustering (CPM). In this technique, two random instances from the majority and minority class are designated as initial cluster centers. The remaining samples are segregated into two subgroups according to nearer centers with a precondition that one subgroup should have more class purity. The process is recursively repeated unlit the two subsets are unable to from cluster in such a way that class purity of at least one group should be more than that of the parent cluster (Yoon and Kwek 2005).

3.5 ML techniques

In this study, we explored different categories of ML, namely neural network, instance-based learner, ensembles, and decision tree. Apart from these techniques, the study also selected one statistical technique. The predictive capability of chosen techniques is well established in the research of software quality domains for predictive modeling tasks. Neural networks (NNs) have proven their outstanding capability to derive meaning insights and extract a pattern from complex data. Many studies in the literature have used neural networks for building successful models to predict software maintainability (e.g., Thwin and Quah 2005; Zhou and Leung 2007; Ahmed and Al-Jamimi 2013). A study by Malhotra (2015) conducted a review of ML techniques to predict defective classes in software. The review revealed that C4.5 was the most popular technique in the decision tree category, and this technique has an outstanding capability to predict defect-prone classes. The C4.5 decision tree is very much simple in terms of its implementation and offers comprehensive predictive capability (Arisholm et al. 2007). In the category of instance-based learners, we have selected two techniques, namely, KStar (KS) and k-nearer neighbor. Kaur and Kaur (2013) statistically compared 27 different ML techniques to predict maintainability and discovered the outstanding performance of KS. All of the above techniques used in the study are distribution-free techniques that require no prior assumptions on the statistical distribution of the dataset. We also used two ensemble techniques in this study. Ensemble learners are a combination of several classifiers whose predictions are aggregated to obtain a single consolidated decision (Oza and Tumer 2008). Several studies have advocated the remarkable applicability of ensembles to achieve improved capability for quality modeling tasks in software engineering (Catolino and Ferrucci 2018; Xu et al. 2010; Peng et al. 2011). Thus, given the vast number of techniques that can potentially be selected for developing SMP models, we had to strike a balance between curtailing the choice of techniques by considering existing empirical studies and also covering a wide diversity of ML techniques based on their properties. Hence, the primary reason for selecting these techniques for developing SMP models was their establishment and popularity in the literature for predictive modeling in the software engineering domain. The brief description of all the techniques used is given below:

- C4.5 is the enhancement of the ID3 decision tree. ID3 algorithm has few limitations: (i) it only works for nominal attributes and (ii) the dataset should not have any missing values. Ross Quinlan, the innovator of ID3, improved ID3 to overcome these limitations and proposed a modified version of the ID3 algorithm called C4.5. This algorithm creates more generalized prediction models, and it can handle continuous-valued attributes as well as missing values (Quinlan 2014).

- Multilayer perceptron with conjugate learning (MLP-CG) is a feed-forward neural network for classification and prediction. MLP-CG uses conjugate gradient methods for training the weights of a multilayer perceptron. Conjugate gradient methods are characterized by the fact that their memory requirements are low and fast global and local convergence (Moller 1993).
- Radial basis function neural network (RBFNN) is a type of non-linear feed-forward neural network with a single hidden layer. It has guaranteed learning because of a single layer of weights that are adjustable and calculated by a linear optimization problem. This neural network can represent non-linear transformations (Broomhead and Lowe 1988).
- Increment radial basis function neural network (IRBFNN) is a type of NN that learns by apportioning new units and changing the parameters of existing units. If the system performs inadequately on a pattern presented to it, at that point, another unit is allocated, which remedies the reaction to the introduced pattern. If the system performs well on a displayed pattern, at that point, the system parameters are updated (Platt 1991).
- BG is an ensemble technique that creates individual subsets of the training dataset randomly with replacement. A predictor is developed corresponding to each subset. The results of individual predictors are then either averaged or combined using majority voting (Breiman 1996).
- AB is based on the boosting method. The boosting technique develops a powerful classification model by using some weak classifiers. AdaBoost improves the predictive power of weak classifiers. The learning of weak classifiers is carried out by using the weighted training data samples, and the misclassification rate of the individual classification models is determined. This algorithm includes a weight-updating process. The correctly classified data points get small weights, and the wrongly classified data points get large weights. In this manner, the AB technique focuses more on the difficult-to-learn data points (Quinlan 2014).
- KNN has higher interpretability and less calculation time. KNN classifies an instance using the majority voting of its neighboring examples. An instance is said to belong to the class, which is one of the most common among the k-nearer neighbors (Cover and Hart 1967).

KS belongs to the category of instance-based classifiers. In KS, the output label of a test example is determined based on the output label of the training examples that are similar to the given test example. KS technique uses entropy as a measure of dissimilarity (Cleary and Trigg 1995).

- LR with ridge estimator is a prominent method used for binary classification. Ridge estimators enhance the parameter estimates and lower the error when maximum-likelihood estimators cannot fit the data (Le Cessie and Van Houwelingen 1992).

4 Results and analysis

We developed the SMP models using original imbalanced datasets and after applying data resampling methods. The performance of SMP models is compared concerning performance measures: G-mean and Balance.

4.1 RQ1: What is the performance of SMP models developed using ML techniques on original imbalanced datasets?

To answer this research question, we develop SMP models with original imbalanced datasets using ML techniques discussed in Section 3.4. Tables 6 and 7 show the predictive performance of ML techniques for SMP models based on G-mean and Balance, respectively. In this experiment, we notice that ML techniques without applying resampling methods have inferior performance regarding G-mean and Balance. On analyzing Table 6, we see that in 61% of the cases, G-mean values are even less than 50%. Similarly, in 66.66% of the cases (Table 7), the Balance value is less than even 50%. Figure 1 shows the boxplots for the performance of maintainability prediction models regarding Balance and G-mean in case of imbalanced data. It is evident from Fig. 1 that G-mean values are even 0% for a few of the cases, whereas Balance has 29% as its lowest value. Also, the median of G-mean and Balance for all ML techniques is approximately 40%. These trends of the poor performance of ML techniques for maintainability prediction are because the datasets are imbalanced in nature, and the prediction model is unable to learn the minority class instances (class = high maintainability effort) properly, i.e., for training the model, very few minority class instances are presented to the classifier. Therefore, such kind of prediction models cannot be utilized for making future predictions for unknown instances.

4.2 RQ2a: What is the performance of SMP models developed using ML techniques after balancing the datasets with data resampling methods?

In this section, we assess ML techniques' performance for predicting software maintainability after applying various data resampling methods to balance the datasets. Tables 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, and 23 show SMP models' performance concerning performance measures G-mean and Balance, respectively, after applying data resampling methods. It is evident from Tables 8, 9, 10, 11, 12, 13, 14, and 15 that G-mean values are higher than 50% in 85%, 72%, 84%,73%, 95%, 95%, 95%, and 83% of the cases respectively for Apache Bcel, Apache Betwixt, Apache Io, Apache Ivy, Apache Jcs, Apache Lang, Apache Log4j, and Apache Ode datasets. Similarly, Balance values are higher than 50% in 81%, 68%, 83%, 66%, 95%, 95%, 94%, and 87% of the cases for Apache Bcel, Apache Betwixt, Apache Io, Apache Ivy, Apache Jcs, Apache Lang, Apache Log4j, and Apache Ode datasets respectively after data resampling as shown in Tables 16, 17, 18, 19, 20, 21, 22, and 23.

Figures 2 and 3 show the boxplots for the performance of maintainability prediction models regarding G-mean and Balance after data resampling. It is evident from Fig. 2 that G-mean

Table 6 G-mean results of SMP model developed using the imbalanced datasets

Dataset	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
Apache Bcel	46.46	46.46	51.86	22.66	0.00	46.77	23.27	51.52	40.70
Apache Betwixt	18.62	36.59	51.74	35.93	0.00	0.00	46.27	55.30	45.49
Apache Io	0.00	43.87	44.06	62.58	0.00	0.00	44.06	43.68	62.85
Apache Ivy	42.07	41.82	31.86	26.37	41.76	32.68	49.71	49.04	40.75
Apache Jcs	70.94	70.73	69.42	72.76	0.00	66.08	72.98	74.62	76.25
Apache Lang	77.90	62.49	56.83	65.88	0.00	67.38	62.83	64.56	81.46
Apache Log4j	56.69	50.68	69.42	61.18	0.00	0.00	50.61	59.65	41.99
Apache Ode	39.44	39.18	34.59	42.53	0.00	41.60	41.75	48.59	34.59

Table 7 Balance results of SMP model developed using imbalanced datasets

Dataset	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
Apache Bcel	44.97	44.97	48.88	33.00	29.29	44.99	33.19	48.84	41.07
Apache Betwixt	31.76	44.50	49.78	39.08	29.29	34.52	44.94	52.50	44.79
Apache Io	29.00	43.37	43.39	57.55	29.29	29.29	44.06	43.34	57.56
Apache Ivy	41.91	41.90	36.76	34.38	41.89	36.87	46.96	46.90	41.71
Apache Jcs	65.89	65.86	65.59	68.4	29.29	60.70	68.44	73.71	73.26
Apache Lang	73.69	57.99	52.81	62.68	29.29	63.11	58.04	60.47	78.66
Apache Log4j	52.79	42.72	65.59	58.61	29.29	58.32	47.79	55.98	42.52
Apache Ode	40.45	40.42	37.95	42.76	29.29	41.69	41.69	46.59	37.95

reaches up to 80% in most of the datasets. Also, Balance reaches up to 70 to 80% in all eight datasets, as shown in Fig. 3. It is also quite evident from Figs. 2 and 3 that the median of G-mean and median of Balance are even higher after data resampling.

The median of G-mean is greater than 60% for Apache Jcs, Apache Lang, and Apache Log4j datasets (Fig. 3).

Median of G-mean is approximately equal to 60% in Apache Bcel, Apache Betwixt, Apache Io, Apache Ivy, and Apache Ode datasets. Also, the median of Balance is higher than 60% for Apache Jcs, Apache Lang, and Apache Log4j datasets. The median of Balance is nearly 60% for Apache Bcel, Apache Betwixt, Apache Io, Apache Ivy, and Apache Ode datasets after applying data resampling methods (Fig. 3). To conclude, we say that there is a vast improvement in the performance of SMP models after applying data resampling methods as compared with the situation when no data resampling is used.

4.3 RQ2b: Which data resampling method improve the performance of the prediction models the most?

To assess the performance of data resampling methods used in the study, we perform Friedman's test concerning performance metrics G-mean and Balance for all eight datasets used in the study along with the scenario when no data resampling is used. In this direction, the following hypotheses are formed and tested.

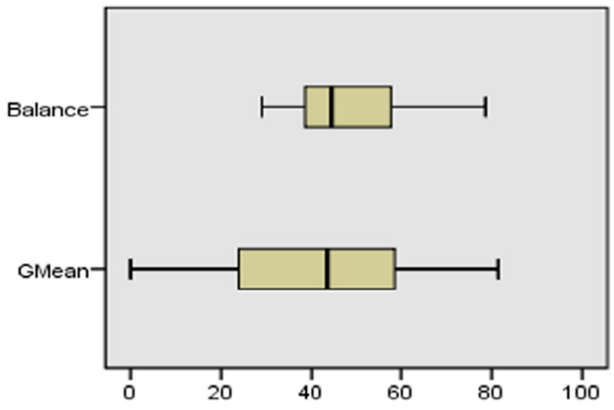

Fig. 1 Boxplots G-mean and Balance of original imbalanced data

Table 8 Performance of SMP models based on G-mean for Apache Bcel dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	60.58	52.97	64.63	52.87	55.30	71.18	66.03	61.05	69.67
Border-Line-SMOTE	61.26	56.72	60.66	31.54	48.17	73.22	55.02	50.83	76.42
ROS	58.91	55.11	57.12	22.65	73.13	63.42	59.88	57.53	54.63
Safe-Level-SMOTE	65.96	66.74	58.86	22.58	75.64	69.81	67.27	59.93	59.53
SMOTE	63.09	64.19	61.96	43.00	60.12	63.09	66.82	61.62	60.93
SMOTE-ENN	62.41	63.20	63.72	37.79	63.46	70.66	70.90	57.32	63.58
SMOTE-TL	67.86	61.73	63.64	50.46	70.26	70.22	68.12	66.82	63.39
SPIDER	61.16	51.77	59.74	22.65	33.33	69.46	51.60	59.84	52.95
SPIDER-II	63.64	56.97	64.63	22.65	40.04	67.85	56.06	54.63	59.32
CNN	73.84	65.49	61.16	53.82	58.49	75.75	51.86	65.22	73.77
CNN-T	68.56	65.81	66.94	44.29	65.96	69.92	62.30	56.62	67.66
CPM	54.50	51.09	53.54	37.50	22.69	43.57	21.09	53.36	54.54
NCL	65.49	64.95	60.96	52.26	23.57	56.72	39.84	50.31	46.69
RUS	80.13	79.55	65.63	58.32	70.41	75.99	60.47	47.06	69.50

H_0: Null hypothesis—There is no significant difference in the predictive performance of SMP models developed with original imbalanced datasets and after applying data resampling methods concerning performance measures G-mean and Balance.

H_a: Alternate hypothesis—There is a significant difference in the performance of SMP models developed with original imbalanced datasets and after applying data resampling methods concerning performance measures G-mean and Balance.

The above-stated hypotheses are tested at a confidence level of 95% ($\alpha = 0.05$) by extracting the values of the performance metrics Balance and G-mean of all datasets used in the study. Tables 24 and 25 show the Friedman test results for G-mean and Balance, respectively. The mean rank attained by each data resampling method is shown in parenthesis. The higher the rank obtained by the resampling method, the better would be that method.

On conducting Friedman's test for different data resampling methods concerning G-mean measure on all eight datasets used in the study, the p value obtained is 0.00 ($p < 0.05$), which means the results of the Friedman test are significant. It is evident from Table 24 that Safe-Level-SMOTE achieves the best rank. The worst rank is obtained for no resampling. Similarly, on conducting Friedman's test for different data resampling methods for Balance measure on

Table 9 Performance of SMP models based on G-mean for Apache Betwixt dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	52.32	62.17	53.58	55.70	55.19	60.91	69.51	57.29	71.87
Border-Line-SMOTE	40.60	51.48	44.81	46.52	36.50	54.18	61.25	52.32	54.60
ROS	43.07	47.54	54.31	35.93	46.45	58.16	66.36	45.10	71.82
Safe-Level-SMOTE	54.74	59.07	62.24	39.75	52.54	68.77	70.28	68.18	58.26
SMOTE	59.04	52.17	49.46	55.19	31.77	62.65	71.25	54.68	64.04
SMOTE-ENN	58.87	55.91	54.37	60.55	25.87	62.35	72.54	54.68	70.28
SMOTE-TL	58.32	55.02	59.66	60.21	37.44	65.74	68.18	53.12	64.49
SPIDER	50.01	44.93	53.11	43.78	26.96	47.66	65.29	56.37	65.07
SPIDER-II	53.67	54.46	48.35	50.28	25.14	62.40	68.10	24.37	62.70
CNN	50.98	58.89	54.99	45.43	25.34	47.16	51.35	48.07	49.80
CNN-T	65.16	61.46	63.05	60.83	50.33	54.81	56.67	34.04	60.13
CPM	42.66	46.27	57.37	40.05	17.78	50.31	50.88	0.00	57.59
NCL	66.51	68.77	59.56	64.05	19.20	71.11	61.58	34.83	55.85
RUS	57.51	68.77	72.34	64.49	51.91	69.40	58.06	34.83	56.40

Table 10 Performance of SMP models based on G-mean for Apache Io dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	66.26	43.29	52.30	72.24	42.90	59.55	71.71	42.11	62.85
Border-Line-SMOTE	53.96	43.77	43.77	69.51	44.25	62.31	53.14	43.48	61.63
ROS	53.37	43.68	41.40	78.64	58.41	61.63	51.57	43.39	77.27
Safe-Level-SMOTE	70.11	65.47	65.14	69.66	69.75	70.29	75.54	63.68	71.36
SMOTE	59.83	53.26	42.11	72.24	59.27	53.26	71.01	52.90	58.98
SMOTE-ENN	52.30	53.14	61.77	73.79	42.31	66.89	81.79	52.66	58.55
SMOTE-TL	66.42	53.26	60.81	73.28	48.29	53.26	86.31	60.53	57.39
SPIDER	60.94	53.49	52.54	76.64	52.54	53.49	52.90	53.14	67.83
SPIDER-II	61.22	53.49	52.30	62.44	52.54	53.49	52.54	52.66	59.83
CNN	57.39	50.96	59.55	65.14	30.67	50.96	63.51	40.17	76.70
CNN-T	78.22	73.96	57.69	64.69	30.67	73.96	80.65	38.35	77.57
CPM	75.93	64.98	65.14	66.10	30.74	64.98	65.47	55.15	65.79
NCL	62.44	61.77	62.04	67.98	31.62	61.77	52.42	61.36	87.82
RUS	65.84	57.10	63.01	78.64	71.12	57.10	64.69	37.96	66.80

all eight datasets used in the study, the p value obtained is 0.00 ($p < 0.05$), which means, again, the results of the Friedman's test are significant. Concerning Balance, the mean rank obtained after the Friedman's test for different data resampling methods along with no resampling scenario is shown in Table 25. Again, Safe-Level-SMOTE yielded the best rank concerning Balance measure after the Friedman's test is applied, and the worst rank is obtained for the no-resampling situation. As the test statistics of the Friedman test are significant for both G-mean and Balance, this leads to the rejection of the null hypothesis (H_0) and acceptance of the alternate hypothesis (H_a). Therefore, in this way, we observe a significant improvement in the performance of SMP models developed after applying data resampling methods on imbalanced datasets. It is observed that the enhanced version of SMOTE, namely, Safe-Level-SMOTE, and hybrid resampling methods, SMOTE-TL, SMOTE-ENN, are among the four ranked methods as per ranking obtained after Friedman's test concerning G-mean and Balance measures. The Safe-Level-SMOTE method emerges as the best technique to improve the performance of prediction models. To further extend our analysis, i.e., to get insight into whether the Safe-Level-SMOTE method is statistically better than other resampling methods used in the study or not, we apply Wilcoxon's signed-rank test at 95% level of confidence

Table 11 Performance of SMP models based on G-mean for Apache Ivy dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	56.43	35.82	35.58	58.19	41.76	59.36	72.95	65.77	65.37
Border-Line-SMOTE	27.11	41.01	40.75	39.12	25.37	36.68	70.78	59.02	55.82
ROS	56.54	52.19	62.97	35.99	45.88	60.26	69.63	61.22	67.60
Safe-Level-SMOTE	67.45	60.90	60.13	39.38	47.88	64.94	65.18	56.00	72.90
SMOTE	63.72	54.76	62.36	46.18	38.92	59.39	72.95	63.41	72.29
SMOTE-ENN	56.08	51.14	50.90	49.83	57.03	50.32	66.69	60.61	67.58
SMOTE-TL	56.43	58.52	48.14	57.35	60.97	54.95	73.44	63.89	66.94
SPIDER	57.63	45.20	56.72	47.46	47.15	48.75	60.92	64.57	59.78
SPIDER-II	57.72	54.76	52.76	47.07	43.65	54.59	59.30	62.54	69.77
CNN	40.17	56.33	58.40	55.22	24.99	51.14	61.12	49.49	54.36
CNN-T	58.40	45.40	61.61	59.67	58.75	67.16	54.94	49.74	57.87
CPM	26.18	50.16	42.49	54.47	57.46	40.57	52.04	61.637	49.49
NCL	50.82	55.61	67.44	62.74	26.72	51.47	58.44	49.66	51.21
RUS	63.62	62.68	58.86	64.85	64.22	62.43	71.83	52.54	63.25

Table 12 Performance of SMP models based on G-mean for Apache Jcs dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	74.62	76.44	71.34	77.47	66.17	80.03	80.44	74.97	80.52
Border-Line-SMOTE	64.88	78.47	66.90	76.74	72.79	79.21	80.36	77.21	73.51
ROS	72.98	74.68	71.83	72.76	69.92	76.74	83.94	62.74	73.03
Safe-Level-SMOTE	80.14	80.89	82.55	74.91	81.02	81.62	77.54	78.63	78.94
SMOTE	77.46	77.96	60.16	80.85	72.52	83.37	79.17	68.60	78.33
SMOTE-ENN	80.30	82.84	70.94	73.54	62.83	84.22	77.26	65.68	76.67
SMOTE-TL	79.21	80.58	77.21	75.31	69.97	81.45	80.03	66.57	77.28
SPIDER	79.84	77.71	75.28	75.52	26.89	79.06	76.44	76.40	85.24
SPIDER-II	81.11	76.95	81.17	81.65	55.17	85.35	79.21	76.41	79.49
CNN	75.77	77.21	70.25	62.48	60.35	78.72	75.31	10.84	80.03
CNN-T	83.67	77.26	76.67	56.09	63.85	80.44	75.02	10.84	78.63
CPM	71.10	68.54	63.45	71.58	49.55	73.98	67.16	63.09	66.96
NCL	87.07	84.95	83.08	74.48	27.13	82.79	78.66	78.01	80.89
RUS	74.08	74.10	72.21	74.56	73.63	78.32	75.32	10.84	75.93

($\alpha = 0.05$) by doing Bonferroni correction. Using the Wilcoxon signed-rank test, a pairwise comparison among the Safe-Level-SMOTE method and other resampling methods is computed concerning G-mean and Balance measures of all ML techniques for all datasets. The test statistics of Wilcoxon signed-rank are reported in Table 26 both for G-mean and Balance. In Table 26, S+ means a significant difference in the performance of two corresponding pairs of resampling methods, and NS signifies that there is no significant difference. The results depict that Safe-Level-SMOTE significantly outperforms ADASYSN, SMOTE, Border-Line-SMOTE, SPIDER, SPIDER-II, ROS, CNN, CNN-T, CPM, NCL, and no resampling by both G-mean and Balance. Also, the test results depict that Safe-Level-SMOTE do not significantly outperform SMOTE-TL, SMOTE-ENN, and RUS. The performance of SMOTE-TL, SMOTE-ENN, and RUS is comparable with Safe-Level-SMOTE.

4.4 Discussion on results

For the imbalanced datasets, SMP models' performance is not good in terms of both of the performance measures G-mean and Balance. An analysis of Table 6 indicates that for the

Table 13 Performance of SMP models based on G-mean for Apache Lang dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	70.74	68.76	67.58	67.82	70.36	81.41	77.38	65.12	81.41
Border-Line-SMOTE	78.85	67.57	67.01	68.92	46.28	79.29	77.97	69.15	85.41
ROS	73.93	64.92	75.27	66.45	67.18	77.97	76.18	55.13	80.78
Safe-Level-SMOTE	80.55	82.04	74.45	66.45	72.92	83.01	77.33	75.95	78.85
SMOTE	82.77	72.58	52.57	69.87	55.91	83.98	77.56	69.65	80.30
SMOTE-ENN	77.75	74.55	73.09	69.33	59.10	82.65	80.06	66.80	79.58
SMOTE-TL	79.09	80.79	73.81	68.99	74.25	83.01	79.09	68.54	80.63
SPIDER	83.26	74.14	75.34	65.31	19.25	83.12	73.30	77.33	79.42
SPIDER-II	76.86	73.09	72.67	72.25	37.58	82.42	76.40	77.63	79.58
CNN	81.48	69.43	68.32	61.98	44.37	81.25	73.17	61.33	74.08
CNN-T	76.18	69.25	73.30	60.70	61.95	72.96	65.31	61.02	69.99
CPM	78.89	67.58	68.03	61.58	35.05	79.34	72.78	58.64	73.21
NCL	79.51	75.99	66.26	66.80	19.20	74.14	75.99	70.77	81.01
RUS	76.88	73.04	76.63	66.49	62.78	80.79	68.54	60.22	77.43

Table 14 Performance of SMP models based on G-mean for Apache Log4j dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	71.29	62.75	58.79	72.35	53.29	68.96	69.42	68.15	62.65
Border-Line-SMOTE	59.65	61.66	57.24	73.48	51.96	61.30	67.83	62.32	66.70
ROS	67.83	54.71	61.66	62.75	64.80	59.87	68.70	62.75	65.35
Safe-Level-SMOTE	71.13	69.10	64.77	62.36	70.32	72.35	78.73	70.16	69.20
SMOTE	63.87	64.89	61.27	72.48	47.78	67.42	73.82	72.59	69.18
SMOTE-ENN	68.05	73.11	71.53	71.17	42.79	71.53	75.42	73.11	65.80
SMOTE-TL	69.47	71.65	64.71	73.59	60.84	64.52	73.27	70.25	69.84
SPIDER	67.85	58.85	62.52	65.77	21.76	66.09	55.30	62.04	55.93
SPIDER-II	65.13	63.67	65.69	64.28	37.59	64.38	63.85	67.10	60.76
CNN	55.58	67.57	61.55	68.63	52.48	64.28	66.03	63.71	60.07
CNN-T	70.42	68.98	73.87	70.04	66.67	72.22	66.67	62.64	52.36
CPM	47.52	59.12	59.03	64.51	54.69	57.50	65.42	51.22	57.50
NCL	62.32	71.31	59.56	72.60	21.79	69.42	65.50	71.64	57.50
RUS	72.98	72.97	70.69	66.09	72.19	69.29	66.05	71.41	54.08

Apache Bcel dataset, the G-mean values of SMP models developed by ML techniques ranged from 0 to 51.86%, and except for IRBFNN and MLP-CG, G-mean results of all techniques are less than 50%. A similar trend is observed for the imbalanced Apache Betwixt dataset. For Apache Betwixt dataset, in the imbalanced scenario, the SMP models' G-mean values ranged from 0 to 55.30%. For the Apache Io dataset, the lowest G-mean values were reported to be 29.29% for SMP models developed with KS and BAGG techniques, and the G-mean results ranged from 29.29 to 57.56%. For this dataset, except for KNN and RBFNN techniques, the G-mean values for the remaining techniques were reported to be far less than 50%. For the Apache Ivy dataset, the G-mean values ranged from 26.37 to 49.71%. It is worth noting that for Ivy datasets, SMP models' performance is inferior in terms of G-mean, and none of the techniques could achieve G-mean value of even 50%. For the Apache Log4j dataset, in case of imbalanced scenario, the G-mean results of SMP models developed with the application of different ML techniques ranged from 0 to 69.42%, and for Apache Ode dataset, the G-mean values ranged from 0 to 24.56%. On analyzing the Apache Jcs results, the G-mean values were reported to be in the range of 0–76.25%. For the Apache Lang dataset, the SMP models developed with the help of different ML techniques gave G-mean values in the range of 0–

Table 15 Performance of SMP models based on G-mean for Apache Ode dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	50.59	61.36	51.00	48.42	50.31	61.69	69.48	59.25	71.24
Border-Line-SMOTE	44.71	51.69	49.41	52.33	43.29	42.89	64.97	49.64	59.65
ROS	55.36	51.36	61.42	42.53	65.06	57.17	72.28	52.49	72.04
Safe-Level-SMOTE	69.63	60.66	65.20	42.48	70.29	65.68	72.94	63.33	67.49
SMOTE	56.03	53.10	70.58	60.62	47.18	61.74	71.98	61.73	69.16
SMOTE-ENN	70.71	65.30	74.06	60.97	51.43	65.20	70.71	65.66	71.42
SMOTE-TL	71.76	65.30	70.67	64.26	60.86	63.68	71.76	70.43	72.52
SPIDER	58.30	43.08	58.07	42.45	48.87	53.32	58.30	55.83	66.14
SPIDER-II	62.63	51.20	61.23	45.76	18.72	49.51	62.59	45.49	59.49
CNN	51.69	50.47	45.97	53.41	64.43	55.61	51.69	50.11	66.98
CNN-T	64.26	52.81	61.53	54.54	13.21	65.89	64.26	55.17	65.41
CPM	50.84	50.09	50.40	46.51	18.69	52.02	50.84	52.18	55.71
NCL	50.46	53.71	56.58	56.23	65.30	61.23	50.46	62.06	45.05
RUS	68.53	66.99	66.69	61.23	65.30	70.71	68.53	59.75	66.91

Table 16 Performance of SMP models based on Balance for Apache Bcel dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	58.83	51.55	62.78	51.50	52.50	69.96	65.08	59.09	68.89
Border-Line-SMOTE	56.72	52.80	56.62	36.71	47.62	68.48	52.41	48.69	72.32
ROS	56.12	52.44	56.45	33.00	73.12	60.15	58.41	55.53	52.28
Safe-Level-SMOTE	65.78	65.59	58.51	32.96	74.91	69.71	66.94	59.42	58.19
SMOTE	60.03	60.38	59.56	43.74	58.56	60.03	65.65	59.39	59.03
SMOTE-ENN	59.76	60.07	63.27	40.21	62.58	67.77	69.78	55.42	63.15
SMOTE-TL	66.33	59.45	62.18	49.99	70.23	69.30	67.67	65.65	62.02
SPIDER	56.70	48.87	56.40	33.00	37.15	64.56	48.85	56.43	50.65
SPIDER-II	60.22	52.82	62.78	33.00	41.01	64.20	52.69	52.28	56.27
CNN	71.47	64.68	59.15	53.64	55.96	74.46	50.92	65.11	73.09
CNN-T	67.92	64.95	66.53	44.36	65.78	69.70	62.21	56.38	66.94
CPM	54.51	51.10	52.97	39.19	33.02	44.24	31.75	53.36	54.50
NCL	60.64	60.55	56.67	51.16	33.32	52.80	40.98	48.55	44.99
RUS	78.85	79.31	64.79	58.04	70.27	75.30	60.36	47.23	69.43

81.56%. The performance of SMP models developed from the imbalanced datasets in terms of Balance performance measure is reported to be very poor.

For Apache Bcel dataset, the Balance values of SMP models developed by ML techniques ranged from 29.29 to 48.88%, and for all techniques, Balance results are less than 50%. For the Apache Betwixt dataset, in the imbalanced scenario, the Balance values of SMP models ranged from 29.29 to 52.50%, and for all the techniques except MLP-CG, the Balance results are less than 50%. For the Apache Io dataset, the lowest Balance value was reported to be 29.00% for SMP models developed with C4.5 techniques, and the G-mean results ranged from 29.29 to 57.56%. For this dataset, except for KNN and RBFNN techniques, the Balance values for the remaining techniques were reported to be far less than 50%. For the Apache Io dataset, a similar trend was observed for G-mean results. For the Apache Ivy dataset, the Balance values ranged from 34.38 to 46.96%. For Ivy dataset, the performance of SMP models is very poor in terms of Balance, and none of the techniques could achieve a Balance value of even 50%. For the Apache Log4j dataset, the Balance results of the SMP models developed using different ML techniques ranged from 29.29 to 65.59%. For the Apache Ode dataset, the Balance results ranged from 29.29 to 46.59%. On analyzing the Apache Jcs dataset results, the

Table 17 Performance of SMP models based on Balance for Apache Betwixt dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	51.19	60.77	52.73	54.83	54.46	59.91	69.43	56.58	71.84
Border-Line-SMOTE	41.86	49.68	44.58	46.34	39.34	52.11	59.20	51.19	52.27
ROS	43.78	46.84	54.30	39.08	46.58	56.41	65.94	45.45	70.93
Safe-Level-SMOTE	54.46	58.49	61.98	41.46	52.52	68.73	70.28	68.16	58.05
SMOTE	57.86	51.11	48.74	54.46	36.81	61.86	71.02	54.07	64.03
SMOTE-ENN	57.74	54.13	53.25	59.64	34.17	60.89	72.48	54.07	70.28
SMOTE-TL	57.87	54.33	58.27	59.85	39.87	65.41	68.14	52.80	64.48
SPIDER	49.03	44.62	52.39	44.15	34.51	46.89	62.45	55.30	63.57
SPIDER-II	52.79	52.22	48.05	49.17	33.71	59.75	66.47	25.27	61.11
CNN	50.35	58.34	53.63	45.71	33.85	46.67	49.64	48.14	49.72
CNN-T	64.25	61.11	62.98	59.24	49.8	54.69	56.42	37.95	59.74
CPM	43.54	46.5	57.04	40.6	31.12	50.17	50.86	29.29	57.58
NCL	64.41	67.85	58.20	63.54	31.91	69.38	59.37	38.28	55.42
RUS	57.21	67.85	72.30	64.46	51.90	68.83	58.05	38.28	55.87

Table 18 Performance of SMP models based on Balance for Apache Io dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	63.61	43.26	50.11	70.25	43.15	56.82	71.71	42.87	61.65
Border-Line-SMOTE	50.46	43.35	43.35	64.56	43.41	57.52	53.14	43.30	57.42
ROS	50.37	43.34	42.54	78.61	56.32	57.42	51.57	43.28	76.38
Safe-Level-SMOTE	68.96	63.23	63.06	64.58	68.72	69.09	75.53	62.20	69.76
SMOTE	56.93	50.35	42.87	70.25	56.71	50.35	71.01	50.27	56.59
SMOTE-ENN	50.11	50.33	57.45	70.97	42.95	63.87	81.79	50.21	56.39
SMOTE-TL	63.68	50.35	57.24	70.76	48.06	50.35	86.31	57.16	55.78
SPIDER	57.27	50.39	50.18	71.68	50.18	50.39	52.89	50.33	64.20
SPIDER-II	57.34	50.39	50.11	57.54	50.18	50.39	52.54	50.21	56.93
CNN	55.78	49.60	56.82	63.06	36.22	49.60	63.51	41.81	75.97
CNN-T	78.16	73.76	55.94	64.54	36.22	73.76	80.64	39.69	77.48
CPM	75.39	62.98	63.06	63.54	36.24	62.98	65.46	54.32	63.39
NCL	57.54	57.45	57.49	64.24	36.36	57.45	52.420	57.37	84.28
RUS	65.58	55.61	61.76	78.61	71.11	55.61	64.69	40.00	66.41

Balance values were reported to be in the range of 29.29–73.71%, and for the Apache Lang dataset, the SMP models developed with the help of different ML techniques gave Balance values in the range of 29.29–78.66%.

Therefore, if we look at SMP models' performance for imbalanced datasets, we see that the performance of the SMP models is inferior in terms of both the G-mean and Balance. The low performance is due to the skewed distribution of high maintainability effort and low maintainability effort data points in the datasets.

As the data sets have insufficient data points for the high maintainability effort classes, the SMP models may not be able to learn the prediction of high-maintainability effort classes competently, resulting in low sensitivity values (true positive rate) that resulted in low G-mean and Balance results.

However, the use of data resampling techniques (RQ2) enhanced the performance of the ML techniques for building SMP models. For the Apache Bcel dataset, the G-mean values ranged from 50.32 to 80.14%, and Balance ranged from 50.65 to 79.31%, respectively, for the majority of the cases after data resampling. For Apache Betwixt dataset, the G-mean values ranged from 50.01 to 72.54% and Balance ranged from 50.17 to 72.48%, respectively, for

Table 19 Performance of SMP models based on Balance for Apache Ivy dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	55.03	38.96	38.86	56.95	41.89	57.68	72.63	64.46	65.33
Border-Line-SMOTE	34.64	41.77	41.71	41.04	33.97	39.25	69.59	55.73	54.64
ROS	53.94	49.38	62.76	39.03	44.43	56.74	67.86	58.63	67.60
Safe-Level-SMOTE	67.05	59.54	59.58	41.17	45.77	63.89	64.68	55.36	72.56
SMOTE	61.10	51.78	61.88	46.40	40.94	56.47	72.63	60.95	72.21
SMOTE-ENN	53.76	49.14	49.06	49.40	55.38	48.86	65.88	58.35	67.16
SMOTE-TL	55.03	56.12	46.72	56.87	60.26	53.25	73.39	62.33	66.07
SPIDER	54.27	44.34	54.01	46.51	46.39	46.85	56.89	61.44	57.92
SPIDER-II	54.3	51.78	51.09	46.36	43.89	51.74	56.43	59.12	68.88
CNN	41.52	54.96	57.09	54.73	33.74	49.14	58.59	49.20	53.63
CNN-T	58.29	45.74	61.23	59.59	58.64	67.16	54.76	48.69	57.87
CPM	34.28	48.79	43.34	53.00	55.61	41.65	50.74	61.46	48.49
NCL	49.04	53.56	64.07	61.58	34.34	49.23	54.44	48.57	49.16
RUS	63.62	62.66	58.09	64.6	64.14	62.27	71.73	52.48	63.21

Table 20 Performance of SMP models based on Balance for Apache Jcs dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	73.71	74.86	69.51	76.61	65.19	79.32	80.42	74.96	79.32
Border-Line-SMOTE	60.54	75.82	63.00	73.41	70.23	76.07	78.25	76.42	70.52
ROS	68.44	70.89	69.77	68.40	69.41	73.41	83.78	61.13	72.51
Safe-Level-SMOTE	80.11	80.69	82.40	70.95	80.69	81.62	77.35	78.5	78.83
SMOTE	75.39	75.62	57.36	79.87	71.34	83.28	79.14	67.72	78.17
SMOTE-ENN	79.51	82.22	67.88	71.97	58.04	84.02	77.10	64.82	76.03
SMOTE-TL	78.72	79.69	76.42	74.95	69.45	81.16	79.93	66.57	77.11
SPIDER	78.02	75.51	72.87	72.98	34.51	77.61	74.86	75.82	85.24
SPIDER-II	80.04	75.14	80.92	80.35	51.87	84.92	78.72	76.33	78.92
CNN	73.08	75.27	67.59	62.42	57.43	75.91	74.95	29.29	79.32
CNN-T	82.77	77.10	76.03	55.64	63.64	80.42	74.92	29.29	78.50
CPM	65.91	65.31	60.16	69.64	47.49	70.69	65.87	62.07	64.64
NCL	86.97	84.94	83.02	74.25	34.53	82.75	78.28	78.01	80.69
RUS	74.09	74.10	71.85	74.00	73.50	78.17	75.24	29.29	75.88

most of the cases after data resampling. On analyzing the SMP models' results for the Apache Io dataset after data resampling, we observed that for most of the cases, G-mean and Balance values ranged from 50.96 to 87.82% and from 50.11 to 86.31% respectively. The G-mean and Balance values ranged from 50.16–73.44% to 50.74–73.39%, respectively, for most of the cases after data resampling for the Apache Ivy dataset. In the case of the Apache Jcs dataset, the G-mean and Balance values ranged from 55.17 to 87.07% and from 60.54 to 86.97%, respectively, for the majority of the cases after data resampling. The range of G-mean and Balance values was 60.70–85.41% and 60.01–83.68%, respectively, for the majority of the cases after data resampling for the Apache Lang dataset. For Apache Log4j dataset, the G-mean values ranged from 60.07 to 78.73%, and Balance values ranged from 60.02 to 78.39%, respectively, for the majority of the cases after data resampling. For the Apache Ode dataset, the G-mean and Balance values were observed in the range from 50.09 to 74.06% and from 50.11 to 72.69%, respectively, for most of the cases after data resampling. Therefore, the G-mean and Balance results showed improvement for all datasets when data resampling techniques were used. The improvement in G-mean and Balance after data resampling is due to an increase in sensitivity and specificity. When the datasets were imbalanced, SMP models gave

Table 21 Performance of SMP models based on Balance for Apache Lang dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	69.17	65.39	64.93	67.61	67.64	81.41	77.20	62.38	81.41
Border-Line-SMOTE	75.95	63.15	63.03	66.94	44.94	76.09	75.62	65.51	83.68
ROS	70.67	60.55	74.15	62.87	64.74	75.62	74.71	52.45	79.82
Safe-Level-SMOTE	80.39	81.62	74.22	62.87	71.6	82.34	76.51	74.58	78.43
SMOTE	82.17	68.36	50.01	68.61	52.65	82.96	76.67	68.47	79.51
SMOTE-ENN	75.52	70.86	70.36	67.16	55.30	80.86	79.34	65.63	78.99
SMOTE-TL	78.63	78.43	72.13	68.00	72.37	82.34	78.63	68.51	80.62
SPIDER	82.50	70.74	72.90	62.46	31.91	81.06	70.45	76.51	77.80
SPIDER-II	75.09	70.36	70.18	69.98	39.67	80.75	74.84	77.42	78.99
CNN	80.26	68.32	67.51	60.65	44.42	80.12	72.63	60.81	74.08
CNN-T	76.14	68.66	73.3	60.01	61.31	72.92	65.15	60.34	68.18
CPM	73.80	67.4	68.02	60.38	38.56	78.81	72.76	58.64	72.98
NCL	76.15	73.16	62.81	65.63	31.91	70.74	73.16	67.81	78.51
RUS	76.76	73.03	76.53	66.49	61.16	80.61	68.51	60.12	77.18

Table 22 Performance of SMP models based on Balance for Apache Log4j dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	70.73	60.21	56.66	71.96	50.83	68.04	69.25	66.93	61.95
Border-Line-SMOTE	55.98	57.69	54.17	72.87	49.34	57.61	66.08	59.05	63.80
ROS	66.08	52.31	60.79	60.21	63.52	57.14	68.66	60.21	63.17
Safe-Level-SMOTE	71.09	68.58	64.71	60.02	69.54	71.96	78.39	70.03	68.21
SMOTE	61.60	62.07	58.65	72.47	46.05	65.05	73.82	71.08	67.59
SMOTE-ENN	66.20	70.49	69.70	70.63	42.69	69.70	75.41	70.49	65.18
SMOTE-TL	68.88	70.48	62.82	73.51	57.48	63.22	72.96	68.94	69.17
SPIDER	65.25	55.76	60.86	63.38	32.66	62.5	52.5	58.96	53.71
SPIDER-II	63.06	61.5	65.58	61.8	39.39	61.85	62.31	64.89	60.09
CNN	53.55	66.52	59.61	68.46	50.56	61.8	65.37	63.66	59.54
CNN-T	69.03	68.78	73.81	69.06	66.67	71.82	66.63	62.06	52.35
CPM	46.91	57.54	58.66	64.46	53.66	56.48	65.16	51.21	57.29
NCL	59.05	68.71	59.53	72.17	32.66	65.86	62.3	69.77	55.24
RUS	72.98	72.63	70.69	65.9	72.08	69.25	65.99	71.19	54

lower sensitivity values as models were having a smaller number of instances of the high maintainability effort classes to learn the positive examples properly. However, the sensitivity increased after data resampling that has increased the G-mean of SMP models as G-mean is the geometric mean of specificity and sensitivity. After data resampling, the rise in sensitivity led to a decrease in the false-positive rate, which also improved the Balance.

On analyzing the results of the study, it was discovered that models developed after resampling with Safe-Level-SMOTE performed well on all the datasets. The Safe-Level-SMOTE technique improved the performance of the models in terms of G-mean and Balance. According to statistical analysis carried out with the Friedman test, the same result is obtained, i.e., the Safe-Level-SMOTE technique achieved the highest rank in terms of G-mean and Balance whereas the no-resampling situation has attained the worst rank. Therefore, these results support the use of Safe-Level-SMOTE. We also did a pairwise comparison of the performance of Safe-Level-SMOTE with all other resampling methods used in the study, and the performance of Safe-Level-SMOTE was better than all other resampling methods except SMOTE-TL, SMOTE-ENN, and RUS. The performance of SMOTE-TL, SMOTE-ENN, and RUS is comparable with the top-ranked technique, Safe-Level-SMOTE. The Safe-Level-

Table 23 Performance of SMP models based on Balance for Apache Ode dataset after resampling

Resampling technique	C4.5	AB	IRBFNN	KNN	KS	BAGG	LR	MLP-CG	RBFNN
ADASYN	49.38	58.38	48.87	47.82	49.24	59.22	69.47	57.87	71.23
Border-Line-SMOTE	44.06	49.02	47.65	50.04	42.90	42.84	62.68	47.70	56.23
ROS	52.52	48.96	61.27	42.76	62.72	53.85	71.90	50.79	71.89
Safe-Level-SMOTE	68.92	58.7	64.8	42.75	68.47	64.16	72.69	62.52	67.33
SMOTE	54.13	51.03	67.20	58.68	46.25	59.24	71.83	59.82	68.98
SMOTE-ENN	70.66	63.91	70.90	59.38	48.97	63.85	70.66	64.92	70.61
SMOTE-TL	71.63	63.91	67.23	63.79	58.19	62.78	71.63	70.28	72.43
SPIDER	54.05	42.87	55.67	42.74	47.86	50.27	54.05	52.64	63.2
SPIDER-II	58.78	48.92	60.32	45.11	46.62	47.67	58.77	45.03	56.19
CNN	49.02	49.72	46.07	52.83	31.77	52.58	49.02	50.11	66.86
CNN-T	63.79	52.80	61.37	54.54	64.27	65.37	63.79	54.75	65.4
CPM	48.83	49.48	49.95	46.73	30.53	49.95	48.83	52.04	53.97
NCL	47.85	51.23	53.68	54.23	31.77	57.52	47.85	58.62	44.12
RUS	68.41	66.99	66.69	61.22	63.91	70.54	68.41	59.69	66.89

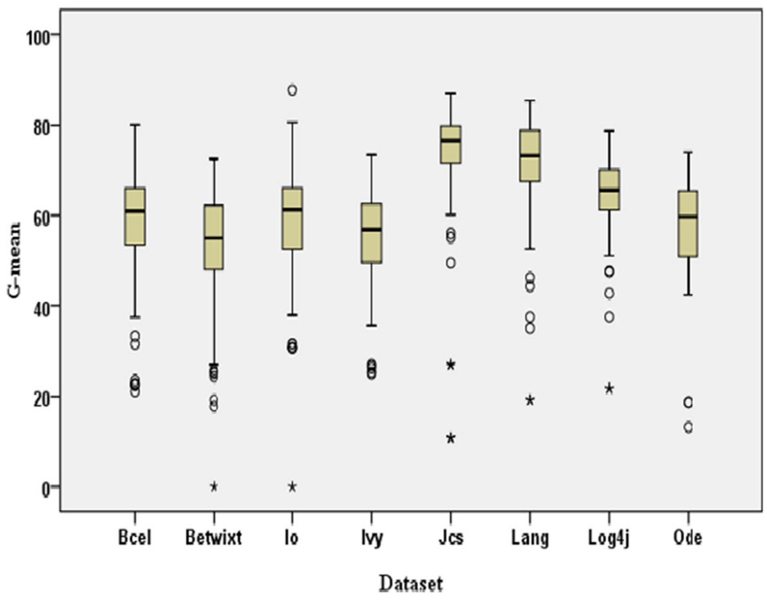

Fig. 2 Boxplots for G-mean results after data resampling

SMOTE technique does not create the same number of synthetic instances for each minority instance; instead, it emphasizes on the instances that fall in the safe region and discounts the instances that are noise. The Safe-Level-SMOTE technique's superiority denotes that a data resampling technique should properly apply a method for generating the synthetic instances which evade noise and redundancy.

The results of this work signify the importance of balanced data with an appropriate number of instances of low maintainability effort and high maintainability effort classes for constructing competent SMP models using ML techniques.

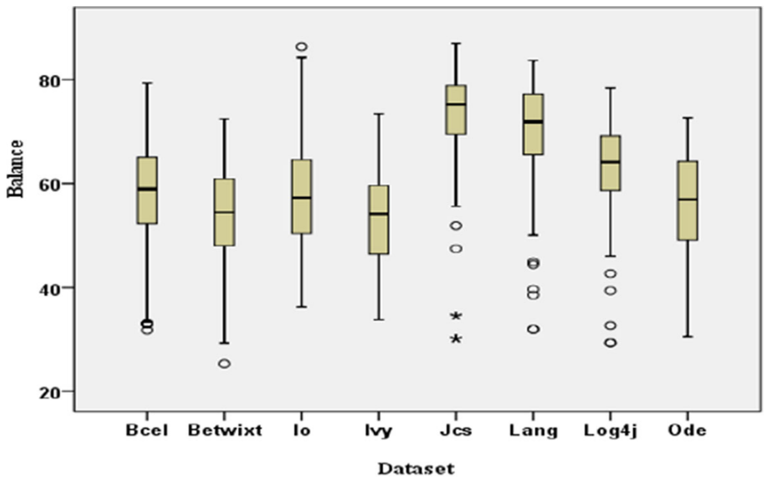

Fig. 3 Boxplots for Balance results after data resampling

Table 24 Friedman's test results for G-mean

Resampling technique	Mean rank
Safe-Level-SMOTE	11.23
SMOTE-TL	11.08
RUS	10.06
SMOTE-ENN	10.00
SMOTE	9.49
ADASYN	8.94
CNN-T	8.65
NCL	8.45
SPIDER-II	7.60
ROS	7.22
SPIDER	7.16
CNN	6.62
Border-Line-SMOTE	6.42
CPM	4.64
No resampling	2.47

5 Threats to validity

The predictor variable used for the development of prediction models in this study has already been analyzed and validated as useful in the software quality domain. The response variable used in this is formed after the discretization of a continuous variable, "change," which is extracted after scanning the change logs with the help of the DCRS tool. The DCRS tool has successfully been used for data collection in many empirical studies. Therefore, there is a threat to construct validity concerning predictors, and the response variable is not present in this study. Threats to the generalizability of the results abolish the external validity of empirical research. All eight datasets used in this study are extracted from application packages of Apache open-source software. Therefore, there exists an external validity threat that the results may vary for proprietary software and software written in a programming language other than Java. However, for developing prediction models, the ML techniques are used with their default parameter setting in this study, which minimize the threat to the generalizability of the

Table 25 Friedman's test results for Balance

Resampling technique	Mean rank
Safe-Level-SMOTE	11.35
SMOTE-TL	11.27
RUS	10.44
SMOTE-ENN	9.69
SMOTE	9.42
CNN-T	9.10
ADASYN	9.00
NCL	7.80
SPIDER-II	7.29
ROS	7.06
SPIDER	6.47
CNN	6.33
Border-Line-SMOTE	5.83
CPM	4.72
No resampling	4.24

Table 26 Wilcoxon's signed-rank test results

Pairwise resampling technique	G-mean	Balance
Safe-Level-SMOTE vs. SMOTE-TL	NS (p value = 0.0.404)	NS (p value = 0.287)
Safe-Level-SMOTE vs. RUS	NS (p value = 0.161)	NS (p value = 0.375)
Safe-Level-SMOTE vs. SMOTE-ENN	NS (p value = 0.079)	NS (p value = 0.013)
Safe-Level-SMOTE vs. SMOTE	S+ (p value = 0.001)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. CNN-T	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. ADASYN	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. NCL	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. SPIDER-II	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. ROS	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. SPIDER	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. CNN	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. Border-Line-SMOTE	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. CPM	S+ (p value = 0.000)	S+ (p value = 0.000)
Safe-Level-SMOTE vs. no resampling	S+ (p value = 0.000)	S+ (p value = 0.000)

results. The degree to which the conclusions drawn after conducting research are believable or credible is called construct validity. It is also referred to as statistical validity. The threat to conclusion validity does not exist in this study, as the results of the study are supported by appropriate statistical analysis.

6 Conclusions and future work

Early prediction of software classes needing high maintainability effort or low maintainability is an essential activity in the software development to design such classes in a better manner. In many software projects, classes requiring high maintainability effort are the majority, resulting in an imbalanced dataset. Therefore, in this direction, the study assesses the impact of applying data resampling methods to balance the class distribution in datasets and compare the performance of maintainability prediction models before and after applying data resampling methods.

Fourteen data resampling methods, including oversampling, undersampling, and hybrid resampling, are used in the study. The SMP models are developed using nine ML techniques on original imbalanced datasets and after employing data resampling methods. Tenfold cross-validation is used to partition the data for training and testing the maintainability prediction models. The results of the developed prediction models are assessed by stable and vigorous performance evaluators, Balance, and G-mean. The study uses statistical tests to strengthen the conclusions and enhance the credibility of the results.

The experimental results on all eight datasets showed that the performance of the ML techniques for predicting software maintainability is significantly improved after employing data resampling methods. The Safe-Level-SMOTE method outperformed with the performance measures G-mean and Balance compared with all the other data resampling methods used in this study. Safe-Level-SMOTE is the enhanced version of SMOTE, which determines the safe level of each minority class instance before generating the synthetic samples. The performance of two hybrid resampling techniques, SMOTE-ENN, SMOTE-TL, and an undersampling method, RUS, is also comparable with the Safe-Level-SMOTE as indicated by the results of the Wilcoxon signed-rank test. The study advocates the use of the Safe-Level-SMOTE method to handle imbalanced data and to improve the performance of ML techniques to develop efficient maintainability prediction models to forecast high maintainability effort classes at the early stages of software development that are crucial for any software project.

Thus, the study results would help in the accurate identification of the classes which have low maintainability and involve a large share of maintenance effort. The accurate identification of such classes would enable software practitioners to improve the design and code of such classes. Also, software developers can devote extra time to the testing phase for testing the low maintainability classes that would lessen the chances of discovering faults in these classes during software maintenance. Early prediction of low maintainability classes in advance would also assist software developers in strategically utilizing their resources, enhancing process efficiency, and optimizing the associated maintenance costs. Lastly, software practitioners are encouraged to document low maintainability in a better manner in order to reduce the time to comprehend the code and undertake the essential modifications during the software maintenance phase.

We plan to replicate this work to examine the effectiveness of data resampling methods used in this study with evolutionary and search-based learning techniques to predict software maintainability.

Compliance with ethical standards

The authors certify that this manuscript has not been submitted to more than one journal for simultaneous consideration, and it has not been published previously (partly or in full).

This article does not contain any studies with human participants or animals performed by any of the authors.

Conflict of interest The authors declare that they have no conflict of interest.

References

Aggarwal, K. K., Singh, Y., & Chhabra, J. K. (2002). An integrated measure of software maintainability. In Proceeding of annual reliability and maintainability symposium. (cat. no. 02CH37318), IEEE, 235-241.

Ahmed, M. A., & Al-Jamimi, H. A. (2013). Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model. *IET Software, 7*(6), 317–326.

Al Dallal, J. (2013). Object-oriented class maintainability prediction using internal quality attributes. *Information and Software Technology, 55*(11), 2028–2048.

Arisholm, E., Briand, L. C., & Fuglerud, M. (2007, November). Data mining techniques for building fault-proneness models in telecom java software. In 18th IEEE international symposium on software reliability (ISSRE'07), 215-224.

Ash, D., Alderete, J., Oman, P. W., & Lowther, B. (1994, September). Using software maintainability models to track code health. In *In proceedings of international conference on software maintenance, 94* (pp. 154–160).

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering, 28*(1), 4–17.

Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter, 6*(1), 20–29.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123–140.

Broomhead, D. S., & Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks (No. RSRE-MEMO-4148). Royal Signals and Radar Establishment Malvern (United Kingdom).

Bunkhumpornpat, C., Sinapiromsaran, K., & Lursinsap, C. (2009, April). Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In Pacific-Asia conference on knowledge discovery and data mining. Springer, Berlin, Heidelberg, 475-482.

Catolino, G., & Ferrucci, F. (2018, March). Ensemble techniques for software change prediction: a preliminary investigation. In 2018 IEEE workshop on machine learning techniques for software quality evaluation, IEEE, 25-30.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321–357.

Chawla, N. V., Japkowicz, N., & Kotcz, A. (2004). Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter, 6*(1), 1–6.

Chidamber, S. R., & Kemerer, C. F. (1991, November). Towards a metrics suite for object oriented design. In proceedings of conference on object-oriented programming systems, languages, and applications, 197-211.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476–493.

Choeikiwong, T., & Vateekul, P. (2015). Software defect prediction in imbalanced data sets using unbiased support vector machine. In *Information science and applications* (pp. 923–931). Berlin, Heidelberg: Springer.

Cleary, J. G., & Trigg, L. E. (1995). K*: An instance-based learner using an entropic distance measure. In Machine learning proceedings. Morgan Kaufmann, 108-114.

Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer, 27*(8), 44–49.

Coleman, D., Lowther, B., & Oman, P. (1995). The application of software maintainability models in industrial software systems. *Journal of Systems and Software, 29*(1), 3–16.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*(1), 21–27.

Dagpinar, M., & Jahnke, J. H. (2003, November). Predicting maintainability with object-oriented metrics-an empirical comparison. In 10th working conference on reverse engineering, 2003. WCRE 2003. IEEE, 155-164.

Ebert, C., & Dumke, R. (2007). *Software measurement: establish – extract – evaluate – execute*. Springer.

Elish, M. O., & Al-Rahman Al-Khiaty, M. (2013). A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software. *Journal of Software: Evolution and Process, 25*(5), 407–437.

Elish, M.O., & Elish, K.O. (2009). Application of treenet in predicting object-oriented software maintainability: a comparative study. In 13th European conference on software maintenance and reengineering, CSMR 2009. IEEE, 69-78.

Eski, S., & Buzluca, F. (2011, March). An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes. In *In 2011 fourth international conference on software testing, verification and validation workshops, IEEE* (pp. 566–571).

Fawcett, T., & Provost, F. (1997). Adaptive fraud detection. *Data Mining and Knowledge Discovery, 1*(3), 291–316.

Fenton, N., & Bieman, J. (2014). *Software metrics: a rigorous and practical approach*. CRC press.

Gao, K., Khoshgoftaar, T. M., & Napolitano, A. (2015). Combining feature subset selection and data sampling for coping with highly imbalanced software data. In *In Proceedings of 27th international conference on software engineering and knowledge engineering, Pittsburgh* (pp. 439–444).

Giger, E., Pinzger, M., & Gall, H. C. (2012, June). Can we predict types of code changes? An empirical analysis. In 9th IEEE working conference on mining software repositories (MSR), IEEE, 217-226.

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

Halstead, M. H. (1977). *Elements of software science. 7, p. 127*. New York: Elsevier.

Han, H., Wang, W. Y., & Mao, B. H. (2005). Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *In International conference on intelligent computing, springer* (pp. 878–887).

Hart, P. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory., 14*(3), 515–516.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering, 21*(9), 1263–1284.

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). Adasyn: adaptive synthetic sampling approach for imbalanced learning. In *In IEEE international conference on neural networks (IEEE world congress on computational intelligence)* (pp. 1322–1328).

Henderson-Sellers, B. (1996). Object-oriented metrics, measures of complexity. Prentice Hall.

Jin, C., & Liu, J. A. (2010). Applications of support vector machine and unsupervised learning for predicting maintainability using object-oriented metrics. In *In 2010 IEEE second international conference on multi-media and information technology (MMIT)* (pp. 24–27).

Kaur, A., & Kaur, K. (2013). Statistical comparison of modeling methods for soft-ware maintainability prediction. *International Journal of Software Engineering and Knowledge Engineering, 23*(06), 743–774.

Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Handling imbalanced datasets: a review. *GESTS International Transactions on Computer Science and Engineering, 30*(1), 25–36.

Kpodjedo, S., Ricca, F., Galinier, P., Guéhéneuc, Y. G., & Antoniol, G. (2011). Design evolution metrics for defect prediction in object-oriented systems. *Empirical Software Engineering, 16*(1), 141–175.

Kubat, M., Holte, R. C., & Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Machine Learning, 30*(3), 195–215.

Kumar, L., & Rath, S.K. (2015). Predicting object-oriented software maintainability using a hybrid neural network with parallel computing concept. In Proceedings of the 8th India software engineering conference, ACM, 100-109.

Kumar, L., Lal, S., & Murthy, L. B. (2019). Estimation of maintainability parameters for object-oriented software using hybrid neural network and class level metrics. *International Journal of System Assurance Engineering and Management, 10*(5), 1234–1264.

Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology, 58*, 388–402.

Laurikkala, J. (2001). Improving identification of difficult small classes by balancing class distribution. In Conference on Artificial Intelligence in Medicine, Springer, Berlin, Heidelberg, 63–66.

Le Cessie, S., Van Houwelingen, J.C. (1992). Ridge estimators in logistic regression. Applied statistics, 91-201.

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

Li, W., & Henry, S. (1993). Object-oriented metrics that predict maintainability. *Journal of Systems and Software, 23*(2), 111–122.

Lu, H., Zhou, Y., Xu, B., Leung, H., & Chen, L. (2012). The ability of object-oriented metrics to predict change-proneness: a meta-analysis. *Empirical Software Engineering, 17*(3), 200–242.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 4*, 308–320.

Malhotra, R. (2015). A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing, 27*, 504–518.

Malhotra, R., & Khanna, M. (2017). An empirical study for software change prediction using imbalanced data. *Empirical Software Engineering, 22*(6), 2806–2851.

Malhotra, R., & Lata, K. (2017). An exploratory study for predicting maintenance effort using hybridized techniques. In Proceedings of the 10th innovations in software engineering conference, ACM, 26-33.

Malhotra, R., Pritam, N., Nagpal, K., & Upmanyu, P. (2014). Defect collection and reporting system for git based open source software. In *In 2014 international conference on data mining and intelligent computing (ICDMIC), IEEE* (pp. 1–7).

Maloof, M. A. (2003). Learning when data sets are imbalanced and when costs are unequal and unknown. In *In International conference on machine learning*. Workshop on Learning from: Imbalanced Data Sets II.

Martin, R. C. (2002). Agile software development: principles, patterns, and practices. Prentice Hall.

Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 1*, 2–13.

Moller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks, 6*(4), 525–533.

Morasca, S. (2009, October). A probability-based approach for measuring external attributes of software artifacts. In Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement, IEEE Computer Society, 44-55.

Napierala, K., Stefanowski, J., & Wilk, S. (2010). Learning from imbalanced data in the presence of noisy and borderline examples. In *In International conference on rough sets and current trends in computing, springer* (pp. 158–167).

Olague, H. M., Etzkorn, L. H., Gholston, S., & Quattlebaum, S. (2007). Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on Software Engineering, 33*(6), 402–419.

Olatunji, S. O., & Ajasin, A. (2013). Sensitivity-based linear learning method and extreme learning machines compared for software maintainability prediction of object-oriented software systems. *ICTACT Journal of Soft Computing, 3*(3), 514–523.

Oman, P., & Hagemeister, J. (1994). Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software, 24*(3), 251–266.

Oza, N. C., & Tumer, K. (2008). Classifier ensembles: select real-world applications. *Information Fusion, 9*(1), 4–20.

Pelayo, L., & Dick, S. (2007). Applying novel resampling strategies to software defect prediction. In NAFIPS 2007 Annual meeting of the North American fuzzy information processing society, IEEE, 69–72.

Peng, Y., Kou, G., Wang, G., Wu, W., & Shi, Y. (2011). Ensemble of software defect predictors: an AHP-based evaluation method. *International Journal of Information Technology & Decision Making, 10*(01), 187–206.

Platt, J. (1991). A resource-allocating network for function interpolation. *Neural Computation, 3*(2), 213–225.

Quinlan, J.R. (2014). C4.5: programs for machine learning. Elsevier.

Rousseeuw, P. J., & Leroy, A. M. (2005). *Robust regression and outlier detection* (Vol. 589) John Wiley & sons.

Radjenović, D., Heričko, M., Torkar, R., & Živkovič, A. (2013). Software fault prediction metrics: a systematic literature review. *Information and software technology, Information and Software Technology., 55*(8), 1397–1418.

Schnappinger, M., Osman, M. H., Pretschner, A., & Fietzke, A. (2019, May). Learning a classifier for prediction of maintainability based on static analysis tools. In *In 2019 IEEE/ACM 27th international conference on program comprehension (ICPC)* (pp. 243–248).

Schneberger, S. L. (1997). Distributed computing environments: effects on software maintenance difficulty. *Journal of Systems and Software, 37*(2), 101–116.

Schneidewind, N. F. (1979). Application of program graphs and complexity analysis to software development and testing. *IEEE Transactions on Reliability, 28*(3), 192–198.

Siers, M. J., & Islam, M. Z. (2015). Software defect prediction using a cost-sensitive decision forest and voting, and a potential solution to the class imbalance problem. *Information Systems, 51*, 62–71.

Singh, Y., Kaur, A., & Malhotra, R. (2010). Empirical validation of object-oriented metrics for predicting fault proneness models. *Software Quality Journal, 18*(1), 13–35.

Stefanowski, J., & Wilk, S. (2008). Selective pre-processing of imbalanced data for improving classification performance. In *In International conference on data warehousing and knowledge discovery, springer* (pp. 283–292).

Sun, Z., Song, Q., & Zhu, X. (2012). Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42*(6), 806–1817.

Tan, M., Tan, L., Dara, S., & Mayeux, C. (2015). Online defect prediction for imbalanced data. In Proceedings of the 37th IEEE Conference on Software Engineering, 2, 99-108.

Thwin, M. M. T., & Quah, T. S. (2005). Application of neural networks for software quality prediction using object-oriented metrics. *Journal of Systems and Software, 76*(2), 147–156.

Van Koten, C., & Gray, A. (2006). An application of Bayesian network for predicting object-oriented software maintainability. *Information and Software Technology, 48*(1), 59–67.

Wang, L., Hu, X., Ning, Z., & Ke, W. (2009). Predicting object-oriented software maintainability using projection pursuit regression. In *In 2009 first international conference on information science and engineering, IEEE* (pp. 3827–3830).

Wang, S., & Yao, X. (2013). Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability, 62*(2), 434–443.

Wang, X., Gegov, A., Arabikhan, F., Chen, Y., & Hu, Q. (2019). Fuzzy network based framework for software maintainability prediction. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 27*(05), 841–862.

Xu, Y., Cao, X., & Qiao, H. (2010). An efficient tree classifier ensemble-based approach for pedestrian detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 41*(1), 107–117.

Yoon, K., & Kwek, S. (2005). An unsupervised learning approach to resolving the data imbalanced issue in supervised learning problems in functional genomics. In *In fifth international conference on hybrid intelligent systems, IEEE* (pp. 303–308).

Zhang, W., Huang, L., Ng, V., & Ge, J. (2015). SMPLearner: learning to predict software maintainability. *Automated Software Engineering, 22*(1), 111–141.

Zheng, J. (2010). Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications, 37*(6), 4537–4543.

Zhou, Y., & Leung, H. (2007). Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software, 80*(8), 1349–1361.

Ruchika Malhotra is Associate Head and Associate Professor in the Discipline of Software Engineering, Department of Computer Science and Engineering, Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She has been awarded the prestigious UGC Raman Postdoctoral Fellowship by the Indian government for pursuing postdoctoral research from Department of Computer and Information Science, Indiana University-Purdue University, Indianapolis, Indiana, USA. She was an Assistant Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She received her Master's and doctorate degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She has received IBM Faculty Award 2013. She has received Best Presenter Award in Workshop on Search Based Software Testing, ICSE, 2014, Hyderabad, India. Her h-index is 26 as reported by Google Scholar. She can be contacted by e-mail at ruchikamalhotra2004@yahoo.com.

Kusum Lata is currently working as Assistant Professor in University School of Management and Entrepreneurship, Delhi Technological University, and pursuing her doctoral degree from Delhi Technological University. She completed her master's degree in Computer Technology and Applications in 2010 from the Delhi College of Engineering, University of Delhi, India. She received her bachelor degree in Computer Science and Engineering in 2003 from Beant College of Engineering and Technology, Punjab Technical University, India. Her research interests are in software quality improvement, applications of machine learning techniques in maintainability prediction, and validation of software metrics. She can be contacted by e-mail at kusumlata@dtu.ac.in