Predicting Defects in Imbalanced Data using Resampling Methods: An Empirical Investigation

Ruchika Malhotra [1] , **Juhi Jain** Corresp. [2]

[1] Department of Software Engineering, Delhi Technological University (former Delhi College of Engineering), Shahbad Daulatpur, Delhi, India

[2] Department of Computer Science and Engineering, Delhi Technological University (former Delhi College of Engineering), Shahbad Daulatpur, Delhi, India

Corresponding Author: Juhi Jain
Email address: juhijain_phdco2k16@dtu.ac.in

The development of correct and effective software defect prediction (SDP) models is one of the utmost needs of the software industry. Statistics of many defect-related open-source data sets depict the class imbalance problem in object-oriented projects. Models trained on imbalanced data leads to inaccurate future predictions owing to biased learning and ineffective defect prediction. In addition to this large number of software metrics degrades the model performance. This study aims at (1) identification of useful metrics in the software using correlation feature selection, (2) extensive comparative analysis of 10 resampling methods to generate effective machine learning models for imbalanced data, (3) inclusion of stable performance evaluators—AUC, GMean, and Balance and (4) integration of statistical validation of results. The impact of 10 resampling methods is analyzed on selected features of 12 object-oriented Apache datasets using 15 machine learning techniques. The performances of developed models are analyzed using AUC, GMean, Balance, and sensitivity. Statistical results advocate the use of resampling methods to improve SDP. Random oversampling portrays the best predictive capability of developed defect prediction models. The study provides a guideline for identifying metrics that are influential for SDP. The performances of oversampling methods are superior to undersampling methods.

Predicting defects in imbalanced data using resampling methods: An Empirical Investigation

7 Ruchika Malhotra[1], Juhi Jain[2]

9 [1] Department of Software Engineering, Delhi Technological University, Delhi, India
10 [2] Department of Computer Science and Engineering, Delhi Technological University, Delhi,
11 India

13 Corresponding Author:
14 Juhi Jain[2]
15 F-26/109, Sector-7, Rohini, Delhi, 110085, India
16 Email address: erjuhijain@gmail.com

Abstract

19 The development of correct and effective software defect prediction (SDP) models is one of the
20 utmost needs of the software industry. Statistics of many defect-related open-source data sets
21 depict the class imbalance problem in object-oriented projects. Models trained on imbalanced
22 data leads to inaccurate future predictions owing to biased learning and ineffective defect
23 prediction. In addition to this large number of software metrics degrades the model performance.
24 This study aims at (1) identification of useful metrics in the software using correlation feature
25 selection, (2) extensive comparative analysis of 10 resampling methods to generate effective
26 machine learning models for imbalanced data, (3) inclusion of stable performance evaluators—
27 AUC, GMean, and Balance and (4) integration of statistical validation of results. The impact of
28 10 resampling methods is analyzed on selected features of 12 object-oriented Apache datasets
29 using 15 machine learning techniques. The performances of developed models are analyzed
30 using AUC, GMean, Balance, and Sensitivity. Statistical results advocate the use of resampling
31 methods to improve SDP. Random oversampling portrays the best predictive capability of
32 developed defect prediction models. The study provides a guideline for identifying metrics that
33 are influential for SDP. The performances of oversampling methods are superior to
34 undersampling methods.

1 Introduction

36 Software Defect Prediction (SDP) deals with uncovering the probable future defects. Efficient
37 defect prediction helps in the timely identification of areas in software that can lead to defects in
38 software owing to better resource utilization (Malhotra, 2016). Source code metrics give useful

39   insights to software quality attributes like cohesion, coupling, size, inheritance, etc, and are
40   extensively used in developing software defect models (Basili, Briand & Melo, 1996; Singh,
41   Kaur & Malhotra, 2010; Radjenović et al., 2013). Effective models can be generated using the
42   object-oriented (OO) metrics. With the increasing complexity of software, early prediction of
43   defects, and assurance of good software quality of projects become difficult tasks. To achieve
44   this task of SDP, machine learning (ML) techniques have been used by several researchers from
45   the past two decades. But software defect data is mostly imbalanced (Khoshgoftaar, Gao &
46   Seliya, 2010). This issue of SDP has recently gained a high interest in researchers in the software
47   engineering community. Data is imbalanced if the number of minority classes, in our case,
48   defective classes, is much lower than the majority class, i.e., non-defective classes. This
49   imbalanced distribution of data misguides classifiers while learning the defective class correctly
50   and hence results in biased and inaccurate results. A good defect prediction model will be the
51   one that is trained on similar distribution of instances of defective and non-defective classes.
52   From this point onwards, we will refer to the imbalanced data problem as the class imbalance
53   problem (CIP) because we are dealing with an imbalanced ratio of defective and non-defective
54   classes of software.
55   CIP is very prominent in many real-life problems like medical diagnosis (Mazurowski et al.,
56   2008; Khalilia, Chakraborty & Popescu, 2011), fraud detection (Phua, Alahakoon & Lee, 2004;
57   Vasu & Ravi, 2011), text categorization (Zheng, Wu & Srihari, 2004; Moreo, Esuli & Sebastiani,
58   2016), sentiment analysis (Lane, Clarke & Hender, 2012), churn prediction (Burez & Van den
59   Poel, 2009), etc. Researchers are actively participating in finding solutions for this problem in
60   the software engineering domain for the last two decades. One of the prominent solutions is
61   using resampling methods comprising of undersampling and oversampling methods (Kotsiantis,
62   Kanellopoulos & Pintelas, 2006). A large number of software metrics adds to the problem of
63   constructing good SDP models. This curse of dimensionality is handled by selecting important
64   and distinct features using correlation feature selection (CFS). CFS is selected as it is a widely
65   accepted feature selection (FS) technique and it has emerged as an effective FS technique in a
66   benchmark study (Ghotra, McIntosh & Hassan, 2017). This study deals with the application of
67   resampling methods to handle CIP for 12 Apache datasets and performing a comparative
68   analysis of various SDP models developed using ML techniques. In this study, six oversampling
69   techniques namely Synthetic Minority Over-sampling Technique (SMT), Safe Level Synthetic
70   Minority Over-sampling Technique (SLSMT), Selective Preprocessing of Imbalanced Data
71   (SPD), ADAptive SYNthetic Sampling (ADSYN), Random OverSampling (ROS), Aglomerative
72   Hierarchical Clustering (AHC) and four undersampling techniques namely Condensed Nearest
73   Neighbor + Tomek's modification of Condensed Nearest Neighbor (CNNTL), Random Under-
74   Sampling (RUS), Neighborhood Cleaning Rule (NCL), One Sided Selection (OSS) are explored
75   on different datasets to deal with their imbalanced nature.
76   The main objective of this paper is to ascertain the importance of dealing with CIP using (1)
77   resampling methods and (2) stable performance evaluators to build correct and effective SDP

78 models. The study also identifies the important internal quality attributes (IQAs) of software on
79 which developers need to focus on.
80 Research Questions (RQs) to achieve the aforementioned objectives are designed as follows—
81 **RQ1: Which features are repeatedly selected by CFS in software engineering datasets?**
82 RQ1 finds the most important internal metrics that impact the possibility of defect(s) in the
83 software. Software metrics are recognized that software designers and developers should focus
84 on while building the software. We found that defects in software engineering datasets are
85 greatly impacted by LCOM, Ca, Ce, and RFC. LCOM is a cohesion metric whereas Ca, Ce, and
86 RFC are coupling metrics. In any software, high cohesion and low coupling are desired.
87 Therefore, as LCOM, Ca, Ce and RFC play a crucial role in SDP, their values should be
88 monitored while designing the software. This would result in software with fewer defects.
89 **RQ2: What is the performance of ML techniques on imbalanced data while building SDP**
90 **models?**
91 RQ2 summarizes the performance of 12 imbalanced datasets with 15 ML techniques in terms of
92 Sensitivity, GMean, Balance, and AUC.
93 **RQ3a: What is the comparative performance of various SDP models developed using**
94 **resampling methods?**
95 **RQ3b: Is there any improvement in the performance of SDP models developed using**
96 **resampling methods?**
97 RQ3 presents the performance of SDP models that are built on balanced datasets. The ratio of
98 defective and non-defective classes is equalized with help of different resampling methods and
99 their performance is compared with their original versions, i.e. when no resampling of classes is
100 done. We empirically proved that all resampling methods improve the performance of SDP
101 models as compared to the models that are trained on original data. The values of AUC, Balance,
102 GMean, and Sensitivity increased with models trained on resampled data, and, hence, these
103 models depict better defect prediction capabilities.
104 **RQ4: Which resampling method outperforms the addressed undersampling and**
105 **oversampling methods for building an efficient SDP model?**
106 In RQ3 it is proved that resampling methods produce better SDP models but out of the wide
107 array of resampling methods which resampling method should the developer or researcher
108 choose? RQ4 is generated to find out the resampling method(s) that outperforms the other
109 resampling methods. The results of the Friedman Test followed by the post-hoc Nemenyi Test
110 obtained on AUC, Balance, GMean, and Sensitivity values show that ML models based on ROS
111 and AHC statistically outperforms other models.
112 **RQ5: Which ML technique performs the best for SDP in imbalanced data?**
113 This study uses 15 ML techniques from different categories. RQ5 tries to explore if there is a
114 statistical difference in the performances of different ML techniques. We constricted the
115 comparison to the models developed using ROS as it was statistically better than other
116 resampling methods except for AHC. Performance values of ROS-based models were
117 comparable but greater than the performance values of AHC-based models. We concluded that

118 statistically ensemble methods and nearest neighbor methods performed better than neural
119 networks and statistical techniques.
120 The answers to these questions are explored by building ML models on CFS selected features
121 using ten-fold cross-validation. Predictive performances of developed models are evaluated
122 using stable performance evaluators like Sensitivity, GMean, Balance, and AUC. Kitchenham et
123 al. (2017) reported that in many studies results are biased because they lack statistical validation.
124 They recommend using a robust statistical test to examine if performance differences are
125 significant or not. Statistical validation is carried out using the Friedman test followed by post
126 hoc analysis that is performed using the Nemenyi test. The conducted study will acquaint
127 developers with useful resampling methods and performance evaluators that will assist them to
128 solve CIP. This study also guides developers and software practitioners about the important
129 metrics that affect the SDP potential of ML models. The result examination ascertained ROS-
130 based and AHC-based ML models as the best defect predictors for datasets related to the
131 software engineering domain. With ROS as a resampling method, nearest neighbors and
132 ensembles exhibited comparable performance in SDP. These models were statistically better than
133 other ML models.
134 The rest of the paper is organized as follows. Section 2 "Related work" deals with research work
135 done in SDP for imbalanced data. Empirical Study Design is explained in Section 3 "Materials
136 and Methods". Section 4 "Results" expounds on the empirical findings and provides answers to
137 set RQs. Next, Section 5 "Discussions" summarizes the results and provides the comparison of
138 this study with related studies. Section 6 "Validity Threats" uncovers the validity threats of this
139 study. Finally, Section 7 "Conclusions" presents the concluding remarks with potential future
140 directions.

141 2 Related Work

142 This section presents the related work done in the field of feature selection and resampling
143 solutions proposed in the SDP field.

144 2.1 Feature Selection in SDP

145 Apache datasets have 20 OO metrics and models developed using all these metrics can hamper
146 the defect prediction capabilities of ML models. The reason is the presence of redundant or
147 irrelevant metrics. This curse of dimensionality can be reduced by using feature reduction
148 strategies. This involves either feature selection—reducing the number of features, or feature
149 extraction—extracting new features from existing ones. This study focuses on feature selection
150 by using a widely acceptable CFS technique. Ghotra, McIntosh & Hassan (2017) explored 30
151 feature selection techniques and concluded CFS as the best feature predictor. They used NASA
152 datasets and PROMISE datasets with 21 ML techniques. Balogun et al. (2019) explored feature
153 selection and feature reduction methods for five NASA datasets over four ML techniques and
154 experimentally concluded that FS techniques did not show consistent behavior for the datasets or
155 ML techniques. Recent studies (Arar & Ayan, 2017; Lingden et al. 2019) have emphasized the
156 importance of feature selection and the impact of CFS in building efficient models with reduced
157 complexity and computation time. Balogun et al. (2020) empirically investigated the effect of 46

158 FS methods over 25 datasets from different sources using Naïve Bayes and decision trees. Based
159 on accuracy and AUC performance, they concluded CFS was the best performer in the FSS
160 category. Therefore, in this study, CFS is used to reveal the most relevant features.

2.2 Solutions Proposed for CIP in SDP

162 Only 20% of software classes are accountable for the defects in software (Koru & Tian, 2005).
163 This principle is enough to explain the reason for the uneven distribution of minority (defective)
164 classes and majority (non-defective) classes. Areas of SDP and software change prediction
165 (Malhotra & Khanna, 2017; Tan et al., 2015) are explored to handle CIP resulting in promising
166 outcomes. Now, to solve this class imbalance issue, a variety of resampling methods have been
167 proposed in the literature, among which oversampling and undersampling techniques are most
168 widely used. Liu, An & Huang (2006) used a combination of oversampling and undersampling
169 techniques for predicting software defects using a support vector machine. The performance of
170 developed models was evaluated using F-Measure, GMean, and ROC curve. Experimentation by
171 Pelayo & Dick (2007) showed improvement in GMean values for SDP when the oversampling
172 technique SMT is used with a C4.5 decision tree classifier. Kamei et al. (2007) evaluated the
173 effect of ROS, RUS, SMT, and OSS on industrial software. The experimental analysis proved
174 that resampling methods improved the performance of LDA and LR models in terms of F1-
175 measure. Khoshgoftaar & Gao (2009) used RUS to handle CIP and also used a wrapper-based
176 feature selection technique for attribute selection. They investigated four different scenarios of
177 sampling techniques and feature selection combinations to evaluate which model has better
178 predictive capability in terms of accuracy and AUC.
179 Galar et al. (2011) performed SDP for imbalanced data using bagging— and boosting—based
180 ensemble techniques with C4.5 as the base classifier. The performance was evaluated using the
181 AUC measure. Some other studies support the application of resampling methods for handling
182 CIP (Riquelme et al.,2008; Pelayo & Dick, 2012; Seiffert et al., 2014; Rodriguez, 2014).
183 Shatwani (2012) also performed an empirical comparison of defect prediction models built using
184 oversampling techniques with three different classifiers on the eclipse dataset. Wang & Yao
185 (2013) investigated ML models built using five different resampling methods on 10 PROMISE
186 datasets and their findings confirmed that models based on resampled data result in better SDP.
187 Experimentations concluded the effective model development with AdaboostNC ensemble.
188 Wang & Yao (2013) were not able to conclude on which resampling method should be selected
189 by the software practitioners. We solved this issue by providing detailed statistical analysis and
190 experimentally proved ROS and AHC to be the better options for researchers and other
191 practitioners.
192 Jindaluang, Chouvatut & Kantabutra (2014) proposed the undersampling technique with the k-
193 centers clustering algorithm which proves to be effective in terms of Sensitivity and F-measure,
194 but they didn't use any stable metric for imbalanced data like GMean or AUC.
195 Bennin et al. (2016) concluded that ROS outperformed the SMT approach when defective
196 classes are less than 20% in the software. This study also empirically proved that ROS is
197 statistically better than SMT, hence supports the conclusions of Bennin et al. (2016).

198   Tantithamthavorn et al. (2018) included a large number of datasets but provided a comparative
199   study with only four resampling methods. They also optimized SMT and found its performance
200   comparable with RUS. We created 1980 models with 12 datasets as compared to their 4242
201   models with 101 datasets. In this study, we covered 10 resampling methods and provided the
202   breadth of resampling methods. We compared models with 15 ML techniques whereas
203   Tantithamthavorn et al. (2018) used only 7 ML techniques. The results of our study are different
204   from those of Tantithamthavorn et al. (2018). The main reasons are that the model evaluation is
205   highly dependent on the nature of the data and the behavior of ML techniques. This problem
206   amplifies by adding the third dimension of resampling methods. Furthermore, Bennin et al.
207   (2018) empirically concluded that AUC performance is not improved by the SMT and this
208   contradicts with the results of Tantithamthavorn et al. (2018).
209   Agrawal & Menzis (2018) also proposed the modified SMT by tuning the SMT parameters.
210   They emphasized on the fact that preprocessing, i.e., resampling technique is more important
211   factor than the ML technique used for the construction of prediction model. If data could be
212   better (less skewed), then results would be more reliable. Malhotra and Kamal (2019) inspected
213   the impact of oversampling techniques on ML models built with 12 NASA datasets. They
214   demonstrated the improvement in ML models with oversampling and proposed a new resampling
215   method— SPIDER3.
216   Though many studies have been conducted, still there is no particular set of resampling methods
217   that can be considered the winner of all. These techniques certainly need more replicated studies
218   with different classifiers and different datasets. More often NASA datasets are exploited by
219   researchers for investigating CIP. We have comparatively used Apache datasets to visualize the
220   effect of sampling techniques. Repetitive studies are required to be performed in the future for a
221   fair comparison. Apart from decision tree-based and ensemble-based classifiers, this study used
222   rule-based, neural network-based, and statistical-based machine learners for assessing the
223   predictive capability of models.

224   ## 3 Materials & Methods

225   This section describes the components involved in this empirical study. This section describes
226   the framework established to build a classification model for defect prediction from dataset
227   collection to model validation. Figure 1 explains the experimental setup for the study.

228   ### 3.1 Step1: Dataset Collection

229   Datasets mined from the promise repository are used for empirical predictive modeling and
230   validation. These datasets are donated by (Jureczko Madeyski, 2010) and downloaded by
231   Promise repository (http://openscience.us/repo). Datasets consist of 20 software metrics and a
232   dependent variable indicating the number of defects in a particular class. The selection of
233   projects is based on the percentage of data imbalance in them. Details of datasets are provided in
234   Table 1. #Classes denotes the total number of classes in the project, #DClasses denotes the
235   number of defective classes, and #%ageDefects represents the percentage of defective classes in
236   a project. The percentage of defective classes in addressed projects varies from 9.85% to 33.9%.

237 This low percentage represents the imbalanced ratio of defective and non-defective classes in the
238 datasets.

3.2 Step 2: Data preprocessing and Identification of Variables

240 The independent variables used in this study are different OO metrics characterizing a software
241 system from various aspects. The OO metrics used in the study include the following metrics:

242 ***Chidamber and Kemerer (CK) Metric suite (Chidamber & Kemerer, 1994):*** Six popularly used
243 metrics namely Weighted Methods of a Class (WMC), Depth of Inheritance Tree (DIT), Number
244 of Children (NOC), Lack of Cohesion in Methods (LCOM), Response For a Class (RFC) and
245 Coupling Between Objects (CBO) are incorporated in this metric suite. This metric suite has
246 been validated in many empirical studies for developing SDP models.

247 ***Quality Model for Object-Oriented Design metric suite (QMOOD) (Bansiya & Davis, 2002):***
248 The study uses few metrics from this metric suite namely Number of Public Methods (NPM),
249 Data Access Metric (DAM), Method of Functional Abstraction (MFA), Measure of Aggression
250 (MOA) and Cohesion among Methods of a class (CAM) along with CK metrics for developing
251 defect prediction models. These metrics are also well exploited in related studies to develop
252 effective software quality prediction models.

253 ***Other metrics:*** Few other metrics have also been used in this study as independent variables in
254 addition to the above metrics that are widely used by researchers. These metrics are—Efferent
255 Coupling (Ce), Afferent Coupling (Ca), Lines of Code (LOC), Coupling Between Methods of a
256 Class (CBM), Average Method Complexity (AMC), and the variant of LCOM (LCOM3). Ce and
257 Ca are proposed by Martin (1994) and LOC, CBM, AMC, and LCOM3 are proposed by
258 Henderson-Sellers (1995). Inheritance Coupling (IC), maximum Cyclomatic complexity
259 (Max_cc), and average cyclomatic complexity (Avg_cc) are also used in addition to the above
260 metrics.

261 Details of metrics used can be referred to from
262 http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html.

263 Datasets are checked for any inconsistencies like missing data or redundant data. If there are data
264 inconsistencies, the model prediction would be biased. Therefore, it is important to clean the data
265 before using it for model development. Datasets collected contain a continuous variable
266 representing the number of defects. It is converted into a binary variable by replacing '0' with
267 'No' and natural numbers with 'yes'. The binary dependent variable, 'defect' with two possible
268 values are 'yes' and 'no' reflects whether the software class is defective or not.

3.3 Step 3: Feature Selection

270 All these metrics may not be important in the concern of predicting defects in the early stages of
271 project development. This study employs CFS for the identification of significant metrics. A
272 review by Malhotra [56] has revealed that CFS is the most commonly used feature selection
273 technique. CFS is used in this study for selecting features because it is the most preferred FS
274 technique in SDP literature (Ghotra, McIntosh & Hassan, 2017; Arar & Ayan, 2017; Lingden et
275 al. 2019; Balogun et al., 2020). CFS performs the ranking of features based on information gain
276 and identifies the optimal subset of features. The features in the subset are highly correlated to

277   the class label 'defect' and are uncorrelated or less correlated with each other. CFS is
278   incorporated to minimize the multicollinearity effect.
279   A list of metrics selected by CFS for each dataset is presented in Table 2.
280   **3.4 Step 4: Handling CIP**
281   This step involves applying resampling methods for treating CIP. These methods are
282   implemented using a knowledge extraction tool based on evolutionary learning (KEEL) (Alcalá-
283   Fdez, 2011). Six oversampling methods and four undersampling methods were applied to create
284   a balance between majority and minority classes. Details of these resampling methods are given
285   in Table 3.
286   **3.5 Step 5: Performance Evaluation and Model Development**
287   ***3.5.1 Performance Evaluators***
288   When it comes to imbalanced data, the selection of appropriate performance evaluators plays a
289   critical role. These measures can be calculated by using the confusion matrix shown in Table 4.
290   TP represents the number of defective classes predicted correctly. TN represents the number of
291   non-defective classes predicted correctly. FP represents the number of non-defective classes that
292   are wrongly predicted as defective classes. FN represents the number of defective classes that are
293   wrongly predicted as non-defective classes.
294   The use of accuracy to evaluate performance is specious when data is imbalanced. Instead,
295   robust performance evaluators like AUC, GMean, and Balance should be used in the class
296   imbalance framework. The Sensitivity indicates the probability of correctly predicted defective
297   classes out of total defective classes. whereas specificity refers to the probability of identifying
298   non-defective classes correctly. Sensitivity or True Positive Rate (TPR) is defined as

299
$$Sensitivity = \frac{TP}{TP + FN} \tag{1}$$

300   GMean maintains a balance between both these accuracies (Li et al. 2012). Therefore, it is wise
301   to use GMean as an effective measure to assess imbalanced data. GMean is defined as the
302   geometric mean of Sensitivity and specificity for any classifier.

303
$$GMean = \sqrt{sensitivity * specificity} \tag{2}$$

304   where
$$Specificity = \frac{TN}{TN + FP} \tag{3}$$

305   Balance corresponds to the Euclidean distance between a pair of Sensitivity and False Positive
306   Rate (FPR) (Li et al. 2012). FPR is the probability of false alarm. It exemplifies the proportion of
307   non-defective classes that are misclassified as defective classes amongst actual non-defective
308   classes. Balance can be computed as—

309
$$Balance = 1 - \sqrt{\frac{\left(0 - \frac{FPR}{100}\right)^2 + \left(1 - \frac{Sensitivity}{100}\right)^2}{2}} \ where \tag{4}$$

310
$$FPR = \frac{FP}{TN + FP} \tag{5}$$

311   The area under the curve (AUC) is widely accepted as a consistent and robust performance
312   evaluator for predictions in imbalanced data (Fawcett, 2006; Malhotra & Khanna, 2017). It is
313   threshold independent and can handle skewed data. It is a measure to distinguish between the
314   two classes. The range of AUC is (0, 1). Higher the AUC value, the better the prediction model.

315   AUC value of 0.5 signifies that the model cannot differentiate between the two classes. AUC
316   values from 0.7 to 0.8 are considered acceptable. AUC values greater than 0.8 are considered
317   excellent.

318   **3.5.2 *Model Development using ML Techniques***

319   This study developed models based on 15 ML techniques. Ten-fold with-in project cross-
320   validation is carried out to reduce the partitioning bias. Data is divided into ten partitions. Nine
321   partitions are used for the training part and the remaining one partition is used for the testing
322   part. Then performance evaluators are averaged across ten folds. The ML parameters that were
323   used in experiments created in this study are noted in Table 5. ML techniques used in this study
324   can be divided into five major categories as described below.

325   *3.5.2.1 Statistical Techniques*

326   •   *Naive Bayes (NB) (John & Langley, 1995):* Naïve Bayes is a probability-based classifier
327   that works on the Bayes theorem. It is an instance-based learner that computes class wise
328   conditional probabilities. Features need to be conditionally independent with each other.
329   It provides fair results even in violation of this assumption. This ML technique works
330   well for both categorical and numerical variables.

331   •   *Logistic Regression (LR) (Le Cessie & Van Houwelingen, 1992*): Logistic Regression is
332   also a probabilistic classifier used for dichotomous variables and assumes that the data
333   follows Gaussian distribution. It works well in case of assumption desecration. During
334   training coefficient values are minimized by ridge estimator to solve multicollinearity and
335   this makes the model simpler. The algorithm runs until it converged.

336   •   *Simple Logistic (SL) (Sumner, Frank & Hall, 2005):* SimpleLogistic uses LogitBoost to
337   construct logistic regression models. LogitBoost uses the logit transform to predict the
338   probabilities. With each repetition, one simple regression model is added for each class.
339   The process terminates when there is no more reduction in classification error.

340   •   *LogitBoost (LB) (Friedman, Hastie & Tibshirani, 2000):* LogitBoost is an additive
341   logistic regression with a decision stump as the base classifier. It maximizes the
342   likelihood and, therefore, generalizes the linear logistic model. The base classifier taken
343   is the decision stump which considers entropy for classification.

344   *3.5.2.2 Neural Networks*

345   •   *MultiLayerPerceptron (MLP) (Rojas & Feldman, 2013):* It is a backpropagation neural
346   network that uses sigmoid function as the activation function. The number of hidden
347   layers in the network is determined by the average of the number of attributes and total
348   classes for a particular dataset. The error is backpropagated in every epoch and reduced
349   via gradient descent. The network is then learned based on revised weights.

350   *3.5.2.3 Nearest Neighbors*

351   •   *IBk (Aha, Kibler & Albert,1991):* IBk is an instance-based K-nearest neighbor learner. It
352   calculates the Euclidian distance measure of the test sample with all the training samples
353   to find its 'k' nearest neighbors. It then assigns the class label to the testing instance
354   based on the majority classification of nearest neighbors. Only one nearest neighbor is

355   determined with k=1 and the class label of that nearest neighbor is assigned to the testing
356   instance.

357   • *Kstar (Cleary & Trigg, 1995):* Like Ibk, Kstar is also an instance-based learning
358   algorithm. The difference between the two techniques is about the similarity measures
359   they use. IBk exploits Euclidian distance and Kstar uses the similarity measure based on
360   entropy. Kstar exhibits good classification competence for noisy and imbalanced data.

361   ### 3.5.2.4 Ensembles

362   • *AdaboostM1 (ABM1) (Quinlan, 1993; Freund & Schapire, 1996):* AaboostM1 is an
363   ensemble technique where numbers of weak classifiers are used iteratively to improve the
364   overall performance. It augments the performance of weak learners by adjusting the weak
365   hypothesis returned by the weak learner. The base decision tree used in ABM1 is J48. J48
366   learns from the previous trees about misclassified instances and calculates the weighted
367   average. NumIterations in parameters represent the number of classifiers involved in this
368   ensemble. This technique helps in reducing the bias in the model.

369   • *Bagging (Bag) (Quinlan, 1993; Breiman, 1996):* Bagging or bootstrap aggregation is also
370   one of the ensemble techniques that improve the predictive capability of base classifiers
371   by making bags of training data. Models work in parallel and their results are averaged.
372   Bagging reduces the variance. The number of bags used for experimentation is 10 and the
373   base classifier used is J48.

374   • *Iterative Classifier Optimizer (ICO):* LogitBoost is used as the iterative classifier in this
375   technique. Cross-validation is utilized for its optimization. In the experiments conducted,
376   it goes through 50 iterations to decide for the best cross-validation.

377   • *Logistic Model Tree (LMT) (Landwehr, Hall & Frank, 2005):* Logistic Model Tree is a
378   meta-learning algorithm that uses logistic regression at leaf nodes for classification. A
379   combination of linear logistic regression and decision tree helps in dealing with the bias-
380   variance tradeoff. This technique is robust to missing values and can handle numeric as
381   well as nominal attributes.

382   • *Random Tree (RT) (Leo, 2001; Sumner, Frank & Hall, 2005):* Random Tree is an
383   ensemble-based supervised learner where different trees are constructed from the same
384   population. Random samples of the population are generated to form different trees with
385   a random selection of features. After bags are constructed, models are developed and
386   majority voting is performed to classify the class.

387   • *Random SubSpace method (RSS) (Ho, 1998):* Random SubSpace is used to construct
388   random forests. Randomly feature subsets are selected to generate multiple trees.
389   Bagging is performed with Reptree. Reptree is faster than the basic decision tree and
390   generates multiple trees in each iteration. It then selects the tree whose performance is the
391   best.

392   ### 3.5.2.5 Decision Trees

393   • *Pruning rule-based classification tree (PART) (Frank & Witten, 1998):* PART is a rule-
394   based learning algorithm that exploits partial C4.5 decision trees and generates rules at

395   each iteration. PART stands for a pruning rule-based classification tree. The rule that
396   results in the best classification is selected. MDL is used to find the optimal split. smaller
397   the confidence factor more will be the pruning done.
398   • *J48 (Quinlan, 1993):* J48 is a JAVA implementation of the C4.5 decision tree. It follows
399   the greedy technique to build a decision tree and uses the gain ratio as splitting criteria.
400   Leaf nodes are the classification labels- defective and non-defective and rules can be
401   derived by traversing from root to leaf node. It generates a binary tree, and one-third of
402   the data is used for reduced error pruning.

403 **3.6 Step 6: Statistical Validation**

404 The results need to be statistically verified because without the involvement of statistics results
405 may be misleading (Frank & Witten, 1998). Kitcehnham et al. (2018) emphasized employing
406 robust statistical tests for validating the experimental results. For statistical validation, we can
407 use either parametric or nonparametric tests. The nonparametric Friedman statistical test
408 (Friedman, 1940) and the Nemenyi Test are exercised in this study because software data do not
409 follow a normal distribution (Demšar, 2006). The Friedman test is executed for different
410 performance evaluators for establishing the statistical difference amongst the performance of
411 developed SDP models. We need to compare several ML models built for several datasets.
412 Therefore, Friedman rankings are computed using the Friedman test. Mean ranks are determined
413 with the help of actual values of performance measure and then these ranks are exploited to
414 perform post-hoc Nemenyi test. If the Friedman test results tend to be positive, post hoc analysis
415 is carried by the Nemenyi test to find pair-wise significant differences. Nemenyi test is executed
416 to determine the technique that statistically outperforms others. The Friedman test and the
417 Nemenyi test are the non-parametric alternatives of the parametric ANOVA test and Tukey test.
418 Both tests are performed using a 95% confidence interval. Hypotheses are set for the
419 corresponding test and we need to accept or refute hypotheses at $\alpha = 0.05$.

420 **4 Results**

421 **4.1 RQ1: Which features are repeatedly selected by CFS in software engineering**
422 **datasets?**

423 20 OO metrics of datasets fundamentally define the IQAs of the software and can be grouped
424 into cohesion, coupling, size, complexity, inheritance, encapsulation, and composition metrics.
425 OO metrics corresponding to each IQA are presented in Table 6. #Selected denotes the number
426 of times a particular metric is selected by CFS for all datasets. LCOM was selected by 11
427 datasets whereas the Ce metric was selected by 10 datasets. This RQ contemplates the metrics
428 that are important for SDP. The weightage of each metric that was selected by CFS for 12
429 datasets is considered and their proportion selection was determined for each IQA.
430 In cohesion metrics, LCOM, CAM, and LCOM3 were chosen by 91.67%, 66.67%, and 41.67%
431 of datasets respectively. The cumulative proportion of the selection of cohesion metrics is 66.7%.
432 This shows that cohesion metrics are important for SDP and while developing the software,
433 developers can focus more on LCOM and CAM values. Similarly, the composition metric
434 (MOA) was selected by 66.67% of the datasets.

435 Exploring Table 6, though the cumulative proportion of coupling metrics is 55.6% its
436 significance can be judged by selecting the top three selected metrics—RFC, Ca, and Ce. RFC is
437 picked by 10 datasets whereas Ca and Ce are opted by 9 datasets each. Considering only these
438 three metrics, the proportion selection of coupling metrics raises from 55.65% to 77.78%.
439 In size metrics, LOC and AMC are more preferred software metrics for defect prediction. In all
440 datasets, the least selected metrics belong to the inheritance category. Therefore, resource
441 investment can be done wisely by developers. The number of times any metric is selected for all
442 the datasets guides developers and software practitioners in determining its worth for SDP.

4.2 RQ2: What is the performance of ML techniques on imbalanced data while
building SDP models?

445 A box and whisker diagram (boxplot diagram) graphically represents numerical data
446 distributions using five statistics: a) the smallest observation, b) lower quartile (Q_1), c) median,
447 d) upper quartile (Q_3), and e) the largest observation. The box is constructed based on the
448 interquartile range (IQR) from Q_1 to Q_3. The median is represented by the line inside the box.
449 The whiskers at both ends indicate the smallest observation and the largest observation.
450 Figure 2 presents the Boxplot diagrams to visually depict the defect predictive capability of ML
451 models on imbalanced data in terms of AUC, Balance, GMean, and Sensitivity.

452 *AUC Analysis:* The AUC of ML models of imbalanced data varies from 0.52 to 0.86 with a
453 median value of 0.76. Only 55.6% of models have AUC greater than 0.75. AUC is less than 0.85
454 in 97.8% of cases. Log4j1.1 depicted the best AUC value of 0.86 with NB. Analysis of results
455 obtained on experimentation reveals that statistical techniques, NB and SL, have performed
456 fairly well in terms of AUC, depicting the highest AUC values for Log4j1.1, Ant1.7, Jedit4.2,
457 Log4j1.0, and Tomcat6.0. Similarly, LMT models were able to predict the highest AUC values
458 for five datasets. Overall, 25% of ML models have AUC less than 0.7.

459 *Balance Analysis:* The range of Balance in No Resampling (NS) case is from 29.20 to 76.55. In
460 Tomcat6.0, only 59.01 value was achieved as the maximum value by NB. Balance values
461 achieved by RSS, PART, and LMT are comparatively lower than other ML techniques. The
462 median value for Balance for all models in the NS case is 56.35. Ivy2.0, Synape1.0, and
463 Tomcat6.0 attained the maximum Balance value with NB. IBk also resulted in maximum
464 Balance values for Jedit4.0, Synapse1.1, and Xerces1.3. 68.3% of datasets have a Balance value
465 of less than 65. Only 3.3% of datasets have a Balance value greater than 75.

466 *GMean Analysis*: When no resampling method is used, GMean ranges from 0 to 0.79. The
467 median value of GMean for all datasets is observed as 0.6. NB achieved the highest GMean
468 values for 58.33% of datasets. Considering the models for all datasets with 5 different ML
469 techniques, only 14.4% of models achieved GMean greater than 0.7. 30.6% of models have a
470 GMean value less than 0.5.

471 *Sensitivity Analysis:* Sensitivity is less than 60% in 93.9% of cases. The median value of
472 Sensitivity is only 0.39. This supports the low predictive capability of developed models when
473 CIP is not handled. Sensitivity values lie between 0 and 0.63. Only 0.6% of cases have a
474 Sensitivity greater than 70% which is not an acceptable achievement for any prediction model.

475   The maximum Sensitivity value obtained in the NS case is 0.7 by IBk, the nearest neighbor
476   technique, in the Log4j1.1 dataset.
477   Thus, the overall performance of SDP models developed using machine learning algorithms on
478   imbalanced data is not satisfactory for high-quality predictions.

479   **4.3 RQ3a: What is the comparative performance of various SDP models**
480   **developed using resampling methods?**
481   **RQ3b: Is there any improvement in the performance of SDP models developed**
482   **using ML techniques on the application of resampling methods?**

483   To answer these questions, we exploited performance evaluators— Sensitivity, GMean, Balance,
484   and AUC values that are calculated with help of a confusion matrix obtained by ten-fold cross-
485   validation-trained models developed using resampling methods. Boxplot diagrams are generated
486   and presented in Figure 3 to visually depict the defect predictive capability of ML models in
487   terms of AUC, Balance, GMean, and Sensitivity on resampled data. Median values of the best
488   resampling method and NS scenario for AUC, Balance, GMean, and Sensitivity are recorded in
489   Figure 4, Figure 5, Figure 6, and Figure 7 in form of bar graphs. We have analyzed the
490   performance of developed models based on mean and median values obtained for the cumulative
491   ML techniques of considered performance evaluators. NS cases are included to provide a fair
492   comparison with the resampling-based models.

493   *4.3.1 Comparative performance of various SDP models developed using*
494   *resampling methods*

495   For all the datasets, the median values are reported for NS and the resampling method that yields
496   the maximum median performance value in bar graphs. Comparison of boxplot diagram in
497   Figure 2 with boxplot diagram in Figure 3 shows the rise in the median line, quartiles, and the
498   highest value achieved by models that were built using resampled data.

499   *AUC Analysis:* On the application of resampling methods, 51.4% of models achieved AUC
500   greater than 0.8. From the bars in Figure 4, it is visible that ROS performance is the best amongst
501   others. ROS attained the highest mean and median value for 75% of the datasets-Ant1.7,
502   Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2, Log4j1.0, Synapse1.0, Tomcat6.0, and Xerces1.3. It also
503   showed the highest mean value for Synapse1.1. Other resampling methods like AHC, SMT, and
504   SPD also have depicted good performance in terms of AUC. SPD got the highest median value
505   for Log4j1.1 (0.92) and Synapse1.1 (0.87). In undersampling methods, only NCL results can be
506   considered progressive. NCL was able to manage to secure the highest median value for only one
507   dataset, i.e., Synapse1.2. The highest AUC value of 1 is achieved by Jedit4.2 and Tomcat6.0
508   datasets. 13.9% of models have AUC greater than 90%, which is a remarkable improvement.

509   ***Balance Analysis:*** Referring to Figure 5, according to the Balance performance evaluator values
510   achieved in predictive modeling, ROS performance seems to outperform the other resampling
511   methods. ADSYN could maximum achieve 86.24 Balance value for Synapse1.0. There were
512   only three resampling methods, ROS, AHC, and SPD, that could achieve the highest Balance
513   value greater than 90. ROS was able to generate a Balance value of 95.58 for Ivy2.0.
514   Comparatively, undersampling techniques, RUS and OSS, had a maximum Balance value of

515 only 77.36 and 80.02 respectively. The highest median and mean values for all datasets except
516 Synapse1.2 are attained by either SPD or ROS. Synapse 1.2 got the highest mean value with
517 NCL and undersampling method. 60.9% of cases have a Balance greater than 70. Therefore,
518 there is 357% of growth in median values of Balance when resampling is done as compared to
519 NS.
520 *GMean Analysis:* Similar patterns are observed in GMean also. Bar graphs in Figure 6 portray
521 the better predictive capabilities of ML techniques with ROS. The highest mean and median
522 values are attained by ROS and SPD for all datasets except Synapse1.2. NCL gave the best
523 results for mean and median values of Synapse1.2. 80.7% of cases have GMean greater than 0.65
524 for resampling methods. AHC, though not having any maximum values, but have consistent
525 performance for all the datasets. With NS, only 3.9% of cases could achieve a GMean value
526 greater than 0.75. With resampling methods, the number of cases for GMean values greater than
527 or equal to 0.75 has elevated to 42.7% with the 1182% of improvement. Comparing the
528 oversampling and undersampling methods, oversampling methods thrived in predicting defects
529 efficiently.
530 *Sensitivity Analysis:* Sensitivity values are illustrated in the bar graph presented in Figure 7.
531 CNNTL worked best for Log4j1.1 and Synapse1.2 with an average Sensitivity value of 0.85 and
532 0.83 respectively. For other datasets, ROS outperformed other resampling methods. After
533 applying resampling methods, Sensitivity increases above 0.9 for 11.9% of the developed
534 models. 53% of resampling-based models have a Sensitivity greater than 0.7 as compared to only
535 0.6% of cases in the NS scenario. This accounts for a whopping 9440% of improvement in ML
536 models having a Sensitivity value greater than 0.7.
537 Therefore, it can be concluded that the predictive capability of ML models has immensely
538 improved after treating imbalanced data properly using resampling methods.
539 **4.3.2 Comparison of Resampling-based ML models with NS models**
540 A comparison of the results of models developed without employing resampling methods and
541 models developed using resampling methods provides an answer to RQ3b. 1800 ML models
542 were constructed for 12 datasets with ten resampling methods. The results are compared based
543 on the maximum value attained and averaged median value achieved in each dataset. Percentage
544 Improvement in Performance Measures after resampling methods are used is presented in Table
545 7. Analysis of Table 7 shows that there is a positive increment in all the values of maximum or
546 median for the four performance evaluators when resampling methods are employed to build
547 SDP models. This proves that there is a definite improvement in resampling-based ML models
548 than the NS models when evaluated based on AUC, Balance, GMean, and Sensitivity.
549 For AUC, the overall percentage growth for the median value is 6.3%. The maximum AUC
550 value achieved in the NS case is 0.86 for Log4j1.1 which increases to 0.97 when resampling
551 methods are applied to it. Jedit4.2 and Tomcat6.0 were able to attain a maximum AUC value of 1
552 showing 18.2% and 22.2% of the increase. For Synapse1.0, on the application of resampling
553 methods, the maximum value depicts the incremental growth of 33.8% and the respective
554 median growth corresponds to 24.3%.

555   The increase in median values of Balance and GMean for all the datasets is 29.1% and 21.2%
556   respectively with resampling methods. Synapse1.0, Tomcat6.0, Ivy2.0, and Camel1.6 has
557   illustrated more than 60% of improvement in Balance median values and more than 70%
558   improvement in GMean median values. The maximum Balance value gained by models with
559   resampling methods is 95.58 for Ivy2.0 which was earlier 58.88. Similarly, the Sensitivity
560   median values have shown a remarkable improvement of 84.6%. The median value of NS was
561   0.39 when all datasets were considered together. This value was raised to 0.71 on the application
562   of resampling methods.
563   Answer to RQ3b— The results verify that there is an improvement in the performance of SDP
564   models developed using ML techniques on the application of resampling methods.
565   **4.4 RQ4: Which resampling method outperforms the addressed undersampling**
566   **and oversampling techniques for building an efficient SDP model?**
567   This RQ addresses the effectiveness of 10 resampling methods that are investigated in this study
568   for constructing good SDP models. For the experiments conducted, is there any particular
569   resampling method that can be considered the best? In this direction, we conducted Friedman
570   tests on performance evaluators to provide rankings to models built using resampling methods.
571   The case when no resampling method was used is also included.
572   The Friedman test is used to find the difference between techniques statistically. Four hypotheses
573   are formed for four different performance evaluators. The hypothesis formed to achieve this
574   objective is stated as:
575   H_i0 (Null Hypothesis): There is no significant statistical difference between the performance of
576   any of the defect prediction models developed after using resampling methods and models
577   developed using original data, in terms of PM_j.
578   H_ia (Alternate Hypothesis): There is a significant statistical difference between the performance
579   of any of the defect prediction models developed after using resampling methods and model
580   developed using original data, in terms of PM_j.
581   where i = 1 to 4 denoting H_1, H_2, H_3 and H_4 hypothesis and j = 1 to 4. PM_1 = AUC, PM_2 =
582   Balance, PM_3 = GMean, and PM_4 = Sensitivity.
583   Table 8 provides the desired ranking of SDP models developed in this study for AUC, Balance,
584   GMean, and Sensitivity. The mean ranks of each resampling method and NS are shown in
585   parentheses. As discussed, NS represents the scenario when no sampling technique is used, so, it
586   represents the performance with original data. We evaluated the hypothesis at the 0.05 level of
587   significance, i.e., 95% of the confidence interval. Rank 1 is the best rank and rank 11 is the worst
588   rank. The p-values achieved for all performance evaluators are 0.000 and recorded in Table 8.
589   As p-values are less than 0.05, we reject the null hypothesis and declare that there is a significant
590   difference between resampling methods applied to developed SDP models.
591   Table 8 shows that ROS and AHC have unanimously scored Rank 1 and Rank 2 respectively for
592   all the performance evaluators. Out of four undersampling methods, only one, i.e., NCL can
593   make its space in the first seven positions for the reliable performance evaluators— AUC,
594   Balance, and GMean. OSS, CNNTL, and RUS have acquired positions in the last four ranks with

595 AUC, Balance, and GMean. These rankings clearly state the supremacy of oversampling
596 methods over undersampling methods.
597 For Balance, GMean, and Sensitivity, NS case is ranked last. Therefore, results statistically
598 approved the visualization in RQ3 that usage of resampling methods improved the predictive
599 power of SDP models.
600 We have used Kendall's coefficient of concordance to assess the effect size. It is a quantitative
601 measure of the magnitude of the experimental effect. Kendall in Table 8 shows the value for
602 Kendall's coefficient of concordance. Its value ranges from 0 to 1. It reflects the degree of
603 agreement. The Kendall value is 0.656, 0.589, 0.587, and 0.538 for AUC, Balance, GMean, and
604 Sensitivity respectively. As the values are greater than 0.5 but less than 0.7, therefore, its effect is
605 moderate.
606 Friedman test tells whether there is an overall difference or not in the model performances, but if
607 there is a difference, it fails to further identify the pairwise difference, i.e, exactly which
608 technique is significantly different from other. For this, a post-hoc analysis was conducted using
609 Nemenyi Test on overall datasets and the comparative pairwise performance of all the
610 resampling methods was evaluated with ROS. The Nemenyi test was carried out at the $\alpha = 0.05$
611 level of significance. Table 9 summarizes the Nemenyi test results for AUC, GMean, Balance,
612 and Sensitivity. 'S+' represents 'significantly better' results. If the difference between the mean
613 ranks of the two techniques is less than the value of critical distance (CD), there is no significant
614 difference in the 95% confidence interval. If the difference is greater than the CD value, the
615 technique with a higher rank is considered statistically better. The computed CD value for the
616 Nemenyi test conducted for resampling methods was 1.135. It can be inferred from Table 9 that
617 ROS has comparable performance with AHC and exhibits statistically better performance than
618 all other compared scenarios based on AUC, Balance, GMean, and Sensitivity.
619 Answer to RQ4: Oversampling methods resulted in better SDP models than undersampling
620 methods. ROS and AHC emerged as the statistically better resampling method in terms of AUC,
621 Balance, GMean, and Sensitivity.
622 **4.5 RQ5: Which ML technique performs the best for SDP based on resampled**
623 **data?**
624 To answer this RQ, we performed the Friedman test for 15 ML techniques by considering ROS-
625 based models. The Friedman rankings are recorded in Table 10 with Rank 1 as the best rank. The
626 Friedman test was held at the 0.05 significance level with a degree of freedom of 14. The null
627 hypothesis is set as there is no difference between comparative performances of ROS-based
628 models for different ML techniques. The p-value for each performance evaluator was 0.000.
629 Therefore, the results are considered 95% significant. We refute the null hypothesis as there is a
630 significant difference amongst performances of models built using different ML techniques. The
631 Kendall value for AUC, Balance, GMean, and Sensitivity for ML techniques with ROS-based
632 models is 0.876, 0.835, 0.844, and 0.838. All the four performance evaluators have a Kendall
633 value greater than or equal to 0.835. This signifies that rankings for different datasets are

634 approximate 83.5% similar and hence, increases the reliability and credibility of the Friedman
635 statistical results. The impact of differences in results is high.
636 This can be observed from Table 10 that Kstar, IBk, ABM1, RT, and RSS techniques incited
637 better prediction models than other ML techniques.
638 Kstar and IBk are the variants of the nearest neighbor techniques. These ML techniques provide
639 good results when there is a large number of samples irrespective of the data distribution.
640 Nearest neighbors are instance-based fast learners. Mean ranks are inscribed in brackets. The
641 mean rank of KStar is 14.41 for AUC which is the highest in the pool. The rank of IBk is second
642 when the models are evaluated based on Balance, GMean, and Sensitivity. The mean rank of IBk
643 is 12.66, 12.91, and 12.33 for Balance, GMean, and Sensitivity respectively.
644 ABM1, RT, Bag, and RSS have also attained high ranks. ABM1 is the first ranker in Balance
645 and GMean with a mean rank of 13.41 and 13.5 respectively. ABM1 got 2nd rank with AUC and
646 4th rank with sensitivity. For the stable performance evaluators, ABM1, Bag, RT, and RSS
647 appeared in the first six ranks, proving their competency in the ML world. These four techniques
648 are ensembles that are considered robust in dealing with imbalanced data. The crux of ensemble
649 techniques is to cover the weakness of the base ML technique and combine them to reduce bias
650 or variance and enhance its predictive capability.
651 The last three ranks for all the performance evaluators are grabbed by the statistical learners:
652 Naïve Bayes, Simple Logistic, and Logistic Regression. These ML techniques, in contrast, were
653 able to generate good prediction models when no resampling method was involved in model
654 construction. The balancing of both the classes boosted the performance of other classifiers,
655 especially ensembles and nearest neighbors, and resulted in their unacceptable performance.
656 The post-hoc analysis is carried out using the Nemenyi Test and results are computed with CD =
657 6.191. Pairwise Comparison of ML Techniques using Nemenyi Test for AUC, Balance, GMean,
658 and Sensitivity are presented in Table 11. ABM1 was paired with other ML techniques and it
659 was found comparable with IBk, Kstar, RT, BAG, RSS, PART, LMT, and J48. ABM1 was
660 found statistically significant than MLP, ICO, LB, SL, LR, and NB.
661 Answer to RQ5: Ensembles and nearest neighbors performed the best for SDP in imbalanced
662 data with a random oversampling method.

5 Discussions

664 Feature Selection is an important activity in software quality predictive modeling. The most
665 widely accepted feature selection technique in literature is CFS. Through this study, we provided
666 the important subset of features to the software practitioners. We identified that coupling and
667 cohesion are the most important internal quality attributes that developers should focus on while
668 building software to avoid defects. Defects in software were found to be highly correlated to
669 LCOM, RFC, Ca, and Ce.
670 Median values of Sensitivity have improved by 45.7%, 341.9%, 250%, 72.2%, 154.8%, 66.7%,
671 18.9%, 330%, 46.5%, 32.8%,293.3%, and 88.8% for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0,
672 Jedit4.2, Log4j1.0, Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3
673 respectively after applying resampling methods.

674 Median values of Balance have improved by 16.4%, 61.3%, 74.7%, 23.6%, 49.5%, 26%, 2.2%,
675 86.8%, 13.1%, 7.2%,77.7%, and 31.9% for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2,
676 Log4j1.0, Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3
677 respectively after applying resampling methods.
678 Median values of G-Mean have improved by 13.1%, 73.7%, 73.5%, 16.9%, 44.7%, 20.1%,
679 0.4%, 88.3%, 9%, 5.5%, 83.5%, and 24.2% for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2,
680 Log4j1.0, Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3
681 respectively after applying resampling methods.
682 Median values of ROC-AUC have improved by 1.9%, 3.8%, 14.3.7%, 1.9%, 1.3%, 8.7%, 0.6%,
683 24.3%, 0.7%, 3.3%, 4.7%, and 5.8% for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2, Log4j1.0,
684 Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3 respectively after
685 applying resampling methods.
686 For the datasets considered, ROS and AHC were significantly better than other resampling
687 methods. Better model predictions can be achieved by incorporating oversampling methods than
688 the undersampling methods. Our findings are in the agreement with the Wang & Yao (2013)
689 conclusions. They proved that resampling methods improve defect prediction and, in their
690 settings, the AdaBoost ensemble gave the best performance.
691 Resampling methods did not improve the defect predictive capability of statistical techniques.
692 Developers and researchers should prefer ensemble methods for software quality predictive
693 modeling.
694

695 6 Validity Threats

696 6.1 Conclusion Validity

697 Conclusion validity threat is a threat to statistical validity and this indicates that results of the
698 empirical study are not properly analyzed and validated. To avoid this threat, ten-fold cross-
699 validation is performed. Friedman test and Wilcoxon signed-rank test during post hoc analysis
700 strengthens conclusion validity further. In this study, both the statistical validation techniques are
701 nom parametric in nature. Non-parametric tests are not based on any assumptions for underlying
702 data and, therefore, applicable to the selected datasets. This reinforces the analysis of the
703 relationship between independent variables and dependent variables and hence enhancing
704 conclusion validity.

705 6.2 Internal Validity

706 Applying resampling methods results in the change in the original ratio of defective and non-
707 defective classes. This would be affecting the causal relationship between independent and
708 dependent variables resulting in internal validity bias in our study. However, we have used stable
709 performance evaluators – GMean and AUC to assess the performance of different SDP that help
710 to reduce this threat. We have also used Sensitivity that takes care of the proper classification of
711 defective classes which is one of the major requirements of our problem domain. So, judicious
712 selection of performance evaluators may have reduced some effect of internal validation threat.

713 6.3 Construct Validity

714   Construct validity ensures the correctness of the way of measuring the independent and
715   dependent variables of the study. It also emphasizes whether the variables are correctly mapped
716   to the concept that they are representing. The Independent variables of this study are object-
717   oriented software metrics and the dependent variable of this study is 'Defect' representing the
718   absence or presence of a defect in the software module. Chosen independent variables and
719   dependent variables are widely used in defect prediction studies and thus builds confidence in the
720   removal of construct validity threats from our study. Hence, any construct validity threats may
721   not exist in our study.

6.4 External Validity

723   External validity refers to the extent to which the results of the study are widely applicable.
724   Whether the conclusions of the concerned study can be generalized or not? The datasets used in
725   this study are available publicly. Concerned software are written in the JAVA language. Thus,
726   results validity holds for similar situations only. Results may not be valid for proprietary
727   software. Resampling methods are implemented with default parameter settings in KEEL
728   (Alcalá-Fdez et al., 2011), and ML parameter settings are provided. Therefore, this study can be
729   reproduced without any complications. This minimizes the external validity threat in this study.

7 Conclusions

731   This study evaluates the effect of resampling methods on various machine learning models for
732   defect prediction using Apache software. In total 1980 models were built and the performances
733   of models were empirically compared using stable performance evaluators like GMean, Balance,
734   and AUC. Important features were selected using CFS and ten-fold cross-validation was
735   executed while training the model. Six oversampling and four undersampling methods were
736   analyzed with 15 ML techniques and results were statistically validated by using the Friedman
737   test followed by Nemenyi post hoc analysis. The use of statistical tests reinforces the correctness
738   of results. CFS is incorporated to minimize the multicollinearity effect whereas resampling
739   methods reduces the model bias and assures that predictions are not affected by majority class
740   label. This study reinsures that SDP models developed with resampling methods enhance their
741   predictive capability as compared to models developed without resampling methods. The results
742   of the Friedman test strongly advocate the use of the random oversampling method for the
743   improved predictive capability of SDP classifiers for imbalanced data. Apart from ROS, AHC
744   and SMT also demonstrated good predictive capability for uncovering defects. Nemenyi test
745   eradicates family-wise error and concluded ROS and AHC to be significantly and statistically
746   better than other resampling methods. Models developed using oversampling methods illustrated
747   better defect prediction capability than that of undersampling methods. Handling CIP using ROS
748   and AHC will aid developers and software practitioners in detecting defects effectively in the
749   early stages of software development reducing testing cost and effort. Resampling methods
750   greatly improved the performance of ensemble methods.
751   Future direction involves the inclusion of more imbalanced datasets and investigates the impact
752   of resampling methods on them. Datasets can be taken of any prevalent language different than
753   JAVA like C#. There is a dire need for a benchmark study that compares all the existing

754 resampling solutions in the literature with the common experimental settings. Optimized versions
755 of base resampling methods like RUS or ROS can be proposed. Further, we will also like to
756 explore the consequences of resampling methods with search-based techniques for the
757 classification of software defects. Instead of predicting defects, a framework can besides be
758 utilized to envisage defect severity.
759
760
761
762
763 **References**
764

765 Agrawal A, Menzies T. 2018. Is" Better Data" Better Than" Better Data Miners"?. *In IEEE/ACM*
766 *40th International Conference on Software Engineering*:1050-1061.
767 Aha DW, Kibler D, Albert MK. 1991. Instance-based learning algorithms. *Machine Learning*
768 6(1):37-66.
769 Alcalá-Fdez J, Fernández A, Luengo J, Derrac J, García S, Sánchez L, Herrera F. 2011. Keel
770 data-mining software tool: data set repository, integration of algorithms and experimental
771 analysis framework. *Journal of Multiple-Valued Logic & Soft Computing* 1(6):17.
772 Arar ÖF, Ayan K. 2017. A feature dependent naive Bayes approach and its application to the
773 software defect prediction problem. *Applied Soft Computing* 59:197-209.
774 Balogun AO, Basri S, Abdulkadir SJ, Hashim AS. 2019. Performance analysis of feature
775 selection methods in software defect prediction: a search method approach. *Applied Sciences*
776 9(13):2764.
777 Balogun AO, Basri S, Mahamad S, Abdulkadir SJ, Almomani MA, Adeyemo VE, Al-Tashi Q,
778 Mojeed HA, Imam AA, Bajeh AO. 2020. Impact of feature selection methods on the predictive
779 performance of software defect prediction models: An extensive empirical study. *Symmetry*
780 12(7):1147.
781   Bansiya J, Davis CG. 2002. A hierarchical model for object-oriented design quality assessment.
782 *IEEE Transactions on Software Engineering* 28(1):4-17.
783 Basili VR, Briand LC, Melo WL. 1996. A validation of object-oriented design metrics as quality
784 indicators. *IEEE Transactions on Software Engineering* 22(10):751-761.
785 Batista GE, Prati RC, Monard MC. 2004. A study of the behavior of several methods for
786 balancing machine learning training data. *ACM SIGKDD explorations newsletter* 6(1):20-9.
787 Bennin KE, Keung J, Monden A, Kamei Y, Ubayashi N. 2016. Investigating the effects of
788 balanced training and testing datasets on effort-aware fault prediction models. *In IEEE 40th*
789 *annual Computer software and applications conference* 1: 154-163).
790 Breiman L. 1996. Bagging predictors. *Machine Learning* 24(2):123-140.
791 Bunkhumpornpat C, Sinapiromsaran K, Lursinsap C. 2009. Safe-level-smote: Safe-level-
792 synthetic minority over-sampling technique for handling the class imbalanced problem. *In*
793 *Pacific-Asia Conference on Knowledge Discovery and Data Mining* 27(4):475-482.

794   Burez J, Van den Poel D. 2009. Handling class imbalance in customer churn prediction. *Expert*
795   *Systems with Applications* 36(3):4626-4636.
796   Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. 2002. SMOTE: synthetic minority over-
797   sampling technique. *Journal of Artificial Intelligence Research* 16(6):321-357.
798   Chidamber SR, Kemerer CF. 1994. A metrics suite for object oriented design. *IEEE*
799   *Transactions on Software Engineering* 20(6):476-493.
800   Cleary JG, Trigg LE. 1995. K*: An instance-based learner using an entropic distance measure. *In*
801   *Machine Learning Proceedings* 108-114.
802   Cohen G, Hilario M, Sax H, Hugonnet S, Geissbuhler A. 2006. Learning from imbalanced data
803   in surveillance of nosocomial infection. *Artificial Intelligence in Medicine* 37(1):7-18.
804   Demšar J. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine*
805   *Learning Research* 7(1):1-30.
806   Fawcett T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27(8):861-874.
807   Frank E, Witten IH. 1998. Generating Accurate Rule Sets Without Global Optimization. *In*
808   *Fifteenth International Conference on Machine Learning* :144-151.
809   Freund Y, Schapire RE. 1996. Experiments with a new boosting algorithm. *In 13th International*
810   *Conference on Machine Learning* 96:148-156.
811   Friedman J, Hastie T, Tibshirani R. 2000. Additive logistic regression: a statistical view of
812   boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics* 28(2):337-407.
813   Friedman M. 1940. A comparison of alternative tests of significance for the problem of m
814   rankings. *The Annals of Mathematical Statistics* 11(1):86-92.
815   Galar M, Fernandez A, Barrenechea E, Bustince H, Herrera F. 2011. A review on ensembles for
816   th se class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE*
817   *Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42(4):463-
818   484.
819   Ghotra B, McIntosh S, Hassan AE. 2017. A large-scale study of the impact of feature selection
820   techniques on defect classification models. *In IEEE/ACM 14th International Conference on*
821   *Mining Software Repositories (MSR)* 20(5):146-157.
822   Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. 2009. The WEKA data
823   mining software: an update. *ACM SIGKDD Explorations Newsletter* 11(1):10-8.
824   He H, Bai Y, Garcia EA, Li S. 2008. ADASYN: Adaptive synthetic sampling approach for
825   imbalanced learning. *In IEEE International Joint Conference on Neural Networks (IEEE world*
826   *congress on computational intelligence)* 1(6) :1322-1328.
827   Henderson-Sellers B. 1995. *Object-oriented metrics: measures of complexity.* New Jersey:
828   Prentice-Hall.
829   Ho TK. 1998. The random subspace method for constructing decision forests. *IEEE*
830   *Transactions on Pattern Analysis and Machine Intelligence* 20(8):832-844.
831   Jindaluang W, Chouvatut V, Kantabutra S. 2014. Under-sampling by algorithm with
832   performance guaranteed for class-imbalance problem. *In International Computer Science and*
833   *Engineering Conference* 30(7) :215-221.

834   John GH, Langley P. 1995. Estimating Continuous Distributions in Bayesian Classifiers. *In*
835   *proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence* 338-345.
836   Jureczko M, Madeyski L. 2010. Towards identifying software project clusters with regard to
837   defect prediction. *In Proceedings of the 6th International Conference on Predictive Models in*
838   *Software Engineering* 1-10.
839   Kamei Y, Monden A, Matsumoto S, Kakimoto T, Matsumoto KI. 2007. The effects of over and
840   under sampling on fault-prone module detection. *In First International Symposium on Empirical*
841   *Software Engineering and Measurement* 196-204.
842   Khalilia M, Chakraborty S, Popescu M. 2011. Predicting disease risks from highly imbalanced
843   data using random forest. *BMC Medical Informatics and Decision Making* 11(1):51.
844   Khoshgoftaar TM, Gao K. 2009. Feature selection with imbalanced data for software defect
845   prediction. *In International Conference on Machine Learning and Applications* 13(12):235-240.
846   Khoshgoftaar TM, Gao K, Seliya N. 2010. Attribute selection and imbalanced data: Problems in
847   software defect prediction. *In 22nd IEEE International Conference on Tools with Artificial*
848   *Intelligence* 1:137-144.
849   Khoshgoftaar TM, Yuan X, Allen EB. 2000. Balancing misclassification rates in classification-
850   tree models of software quality. *Empirical Software Engineering* 5(4):313-30.
851   Kitchenham B, Madeyski L, Budgen D, Keung J, Brereton P, Charters S, Gibbs S, Pohthong A.
852   2017. Robust statistical methods for empirical software engineering. Empirical Software
853   Engineering. 22(2):579-630.
854   Koru AG, Tian J. 2005. Comparing high-change modules and modules with the highest
855   measurement values in two large-scale open-source products. *IEEE Transactions on Software*
856   *Engineering* 31(8):625-642.
857   Kotsiantis S, Kanellopoulos D, Pintelas P. 2006. Handling imbalanced datasets: A review.
858   *GESTS International Transactions on Computer Science and Engineering* 30(1):25-36.
859   Kubat M, Matwin S. 1997. Addressing the curse of imbalanced training sets: one-sided selection.
860   *In International Conference of Machine Learning* 97:179-186.
861   Landwehr N, Hall M, Frank E. 2005. Logistic model trees. *Machine learning* 59(1-2):161-205.
862   Lane PC, Clarke D, Hender P. 2012. On developing robust models for favourability analysis:
863   Model choice, feature sets and imbalanced data. *Decision Support Systems*. 53(4):712-718.
864   Laurikkala J. 2001. Improving identification of difficult small classes by balancing class
865   distribution. *In Conference on Artificial Intelligence in Medicine in Europe* 63-66.
866   Le Cessie S, Van Houwelingen JC. 1992. Ridge estimators in logistic regression. *Journal of the*
867   *Royal Statistical Society: Series C (Applied Statistics)* 41(1):191-201.
868   Leo B. 2001. Random forests. *Machine learning* 45(1):5-32.
869   Lessmann S, Baesens B, Mues C, Pietsch S. 2008. Benchmarking classification models for
870   software defect prediction: A proposed framework and novel findings. *IEEE Transactions on*
871   *Software Engineering* 34(4):485-496.
872   Li M, Zhang H, Wu R, Zhou ZH. 2012. Sample-based software defect prediction with active and
873   semi-supervised learning. *Automated Software Engineering* 19(2):201-230.

874   Lingden P, Alsadoon A, Prasad PW, Alsadoon OH, Ali RS, Nguyen VT. 2019. A novel modified
875   undersampling (MUS) technique for software defect prediction. Computational Intelligence.
876   35(4):1003-20.
877   Liu Y, An A, Huang X. 2006. Boosting prediction accuracy on imbalanced datasets with SVM
878   ensembles. *In Pacific-Asia Conference on Knowledge Discovery and Data Mining* 9(4):107-118.
879   Malhotra R. 2015. A systematic review of machine learning techniques for software fault
880   prediction. *Applied Soft Computing*. 27:504-18.
881   Malhotra R, Kamal S. 2019. An empirical study to investigate oversampling methods for
882   improving software defect prediction using imbalanced data. *Neurocomputing* 343:120-40.
883   Malhotra R, Khanna M. 2017. An empirical study for software change prediction using
884   imbalanced data. *Empirical Software Engineering* 22(6):2806-2851.
885   Malhotra R. 2016. *Empirical research in software engineering: concepts, analysis, and*
886   *applications.* New York: Chapman and Hall/CRC.
887   Martin R. 1994. OO design quality metrics. *An Analysis of Dependencies* 12(1):151-170.
888   Mazurowski MA, Habas PA, Zurada JM, Lo JY, Baker JA, Tourassi GD. 2008. Training neural
889   network classifiers for medical decision making: The effects of imbalanced datasets on
890   classification performance. *Neural Networks* 21(2-3):427-436.
891   Moreo A, Esuli A, Sebastiani F. 2016. Distributional random oversampling for imbalanced text
892   classification. *In Proceedings of the 39th International ACM SIGIR conference on Research and*
893   *Development in Information Retrieval* 7(7):805-808.
894   Pelayo L, Dick S. 2007. Applying novel resampling strategies to software defect prediction. *In*
895   *NAFIPS Annual meeting of the North American Fuzzy Information Processing Society* 24(6):69-
896   72.
897   Pelayo L, Dick S. 2012. Evaluating stratification alternatives to improve software defect
898   prediction. *IEEE Transactions on Reliability* 61(2):516-25.
899   Phua C, Alahakoon D, Lee V. 2004. Minority report in fraud detection: classification of skewed
900   data. *ACM SIGKDD Explorations Newsletter* 6(1):50-59.
901   Quinlan JR. 1993. *C4.5: Programs for Machine Learning.* San Mateo: Morgan Kaufmann.
902   Radjenović D, Heričko M, Torkar R, Živkovič, A. 2013. Software fault prediction metrics: A
903   systematic literature review. *Information and Software Technology* 55(8):1397-1418.
904   Riquelme JC, Ruiz R, Rodríguez D, Moreno J. 2008. Finding defective modules from highly
905   unbalanced datasets. *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de*
906   *Datos* 2(1):67-74.
907   Rodriguez D, Herraiz I, Harrison R, Dolado J, Riquelme JC. 2014. Preliminary comparison of
908   techniques for dealing with imbalance in software defect prediction. *In Proceedings of the 18th*
909   *International Conference on Evaluation and Assessment in Software Engineering* 13(5):1-10.
910   Rojas R, Feldman J. 2013. *Neural Networks: A Systematic Introduction.* Berlin: Springer-Verlag.
911   Seiffert C, Khoshgoftaar TM, Van Hulse J, Folleco A. 2014. An empirical study of the
912   classification performance of learners on imbalanced and noisy software quality data.
913   *Information Sciences* 259:571-595.

914 Shatnawi R. 2012. Improving software fault-prediction for imbalanced data. *In International*
915 *Conference on Innovations in Information Technology* 18(3) :54-59.
916 Singh Y, Kaur A, Malhotra R. 2010. Empirical validation of object-oriented metrics for
917 predicting fault proneness models. *Software Quality Journal* 18(1):3.
918 Stefanowski J, Wilk S. 2008. Selective pre-processing of imbalanced data for improving
919 classification performance. *In International Conference on Data Warehousing and Knowledge*
920 *Discovery* 2(9) 283-292.
921 Sumner M, Frank E, Hall M. 2005. Speeding up logistic model tree induction. I*n European*
922 *Conference on Principles of Data Mining and Knowledge Discovery* 675-683.
923 Tan M, Tan L, Dara S, Mayeux C. 2015. Online defect prediction for imbalanced data. *In*
924 *IEEE/ACM 37th IEEE International Conference on Software Engineering* 2: 99-108.
925 Tantithamthavorn C, Hassan AE, Matsumoto K. 2018. The impact of class rebalancing
926 techniques on the performance and interpretation of defect prediction models. *IEEE Transactions*
927 *on Software Engineering* 46(11):1200-19.
928 Tomek I. 1976. Two modifications of CNN. *IEEE Transactions on Systems, Man, and*
929 *Cybernetics: Systems* 6:769-772.
930 Vasu M, Ravi V. 2011. A hybrid under-sampling approach for mining unbalanced datasets:
931 applications to banking and insurance. *International Journal of Data Mining, Modelling and*
932 *Management* 3(1):75-105
933 Wang S, Yao X. 2013. Using class imbalance learning for software defect prediction. *IEEE*
934 *Transactions on Reliability* 62(2):434-443.
935 Wilcoxon F, Katti SK, Wilcox RA. Critical values and probability levels for the Wilcoxon rank
936 sum test and the Wilcoxon signed rank test. *Selected Tables in Mathematical Statistics* 1:171-
937 259.
938 Zheng Z, Wu X, Srihari R. 2004. Feature selection for text categorization on imbalanced data.
939 *ACM SIGKDD Explorations Newsletter* 6(1):80-89.

Dataset Details

Dataset	#Classes	#DClasses	#%ageDefects
Tomcat6.0	858	77	9.85
Synapse1.0	157	16	10.19
Ivy2.0	352	40	11.4
Jedit4.2	367	48	13.1
Xerces1.3	453	69	15.2
Camel1.6	965	188	19.48
Ant1.7	745	166	22.3
Jedit4.0	306	75	24.5
Log4j1.0	135	34	25.2
Synapse1.1	222	60	27
Synapse1.2	256	86	33.6
Log4j1.1	109	37	33.9

OO Features selected by CFS

Dataset	Features selected by CFS
Tomcat6.0	CBO, RFC, LCOM, LOC, MOA, AMC, Max_CC, Avg_CC
Synapse1.0	RFC, LCOM, Ce, LOC, DAM, MFA, CAM, IC, AMC, Max_CC, Avg_CC
Ivy2.0	RFC, Ce, LOC, MOA, AMC, NPM, CBO, WMC, LCOM, LCOM3, CAM
Jedit4.2	CBO, RFC, Ca, Ce, NPM, LCOM3, MOA, LOC, CAM, CBM, LCOM, AMC, Max_CC
Xerces1.3	WMC, Ce, LCOM, MOA, DAM, IC, CBM, AMC
Camel1.6	DIT, NOC, CBO, LCOM, Ca, NPM, LCOM3, CAM, IC, CBM, AMC, Max_CC, Avg_CC
Ant1.7	CBO, RFC, LCOM, Ce, LOC, MOA, CAM, AMC, Max_CC
Jedit4.0	WMC, DIT, CBO, RFC, LCOM, Ca, Ce, NPM, LCOM3, LOC, DAM, MOA, CBM, Max_CC, Avg_CC
Log4j1.0	WMC, CBO, RFC, LCOM, Ce, Ca, NPM, LOC, DAM, CAM, Avg_CC
Synapse1.1	DIT, CBO, RFC, LCOM, Ce, DAM, MFA, CAM, Max_CC
Synapse1.2	WMC, CBO, RFC, Ca, Ce, LOC, MOA, CBM, CAM, AMC, Max_CC
Log4j1.1	WMC, RFC, LCOM, Ce, NPM, LCOM3, MOA, MFA

Resampling Methods used in the study

Resampling Methods		Description
Oversampling Methods	ADAptive SYNthetic Sampling (ADSYN) (He et al., 2008)	In ADASYN, synthetic samples are generated by finding the density distribution of minority classes. Density distribution is computed using a k-nearest neighbor with Euclidian distance. It is an extension of the Synthetic Minority Oversampling Technique. It focuses on the samples that are hard to classify.
	Synthetic Minority Over-sampling Technique (SMT) (Chawla et al., 2002)	The number of minority class samples is increased by generating artificial samples in direction of k nearest neighbors of minority class samples. If one neighbor is selected, then one synthetic sample is generated corresponding to that original minority sample resulting in 100% oversampling of minority classes. In this study, k=5 results in 500% oversampling of minority classes.
	Safe Level Synthetic Minority Oversampling Technique (SLSMT) (Bunkhumpornpat, Sinapiromsaran & Lursinsap, 2009)	Unlike the SMT version, where synthetic samples are generated randomly, in SLSMT first safe levels are calculated that helps in determining safe positions for generating the synthetic samples. If the safe level value is close to 0, it is considered noise and if the safe level value is close to k for k nearest neighbor implementation, then it is considered safe resulting in producing synthetic minority samples there.
	Selective Preprocessing of Imbalanced Data (SPD) (Stefanowski & Wilk, 2008)	This technique categorizes the sample as either safe or noise based on the nearest neighbor rule where distance measurement is done using a heterogeneous value distance metric. If an instance is accurately classified by its k nearest neighbors, it is considered safe otherwise it is considered noise and then discarded.
	Random OverSampling (ROS) (Batista, Prati & Monard, 2004)	ROS is a very simple oversampling technique in which minority class instances are replicated at random with the sole aim of creating a balance between majority and minority class instances.
	Agglomerative Hierarchical Clustering (AHC) (Cohen et al., 2006)	In AHC, each class is decomposed into sub-clusters and synthetic samples are generated corresponding to cluster prototypes. Since artificial samples are created as centroids of sub-clusters of classes, they, therefore extract the characteristics of that class and represent better samples than randomly generated samples.
Undersampling Methods	Random UnderSampling (RUS) (Batista, Prati & Monard, 2004)	RUS, like ROS, is a non-heuristic technique. But in this instead of replicating minority class instances, majority class instances are removed with aim of creating a balance between majority and minority class instances. The problem with this technique is that some important or useful data may be rejected as it is based on random selection.
	Condensed Nearest Neighbor(CNN) + Tomek's modification of Condensed Nearest Neighbor (CNNTL) (Cohen et al., 2006)	First CNN is applied to find a consistent subset of samples that helps to eliminate majority class samples that are far from the decision border. Then Tomek links [46] are made between samples. If it exists between any two samples, then either both are borderline samples or one of them is noise. Samples with Tomek links that fit in majority classes are removed.
	Neighborhood Cleaning Rule (NCL) (Laurikkala, 2001)	NCL uses the edited nearest-neighbor (ENN) rule to remove majority class samples. For each training sample S_i, first it finds three nearest neighbors. If S_i= majority class sample, then discard it if more than two nearest neighbors incorrectly classifies it. If S_i = minority class sample, then discard the nearest neighbors if they incorrectly classify it.
	One Sided Selection (OSS) (Kubat & Matwin, 1997)	OSS and CNNTL have similar working. The difference lies in the order of the application of CNN and the determination of Tomek links. OSS identifies unsafe samples using Tomek links and then applying condensed nearest neighbor (CNN). Noisy and borderline samples are considered unsafe. Small noise may result in the flipping of the decision border of the borderline samples; therefore, they are also considered unsafe. CNN eliminates the majority of samples that are far away from decision boundaries.

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Parameters used in ML techniques

Category	ML technique	Parameter Settings
Statistical ML techniques	NB	useKernelEstimator = false, displayModelInOldFormat = false, useSupervisedDiscretization = false
	LR	Ridge: 1.0E-8, useConjugateGradientDescent = false, maxIts = -1
	SL	Heuristic Stop = 50, Max Boosting Iterations = 500, useCrossValidation = True, weightTrimBeta = 0.0
	LB	Zmax = 3.0, likelihoodThreshold = -1.7976931348623157E308, numIterations = 10, numThreads = 1, poolSize = 1, seed = 1, shrinkage = 1.0, useResampling = False, weightThreshold = 100
Neural Networks	MLP	Hidden layer = a, Learning rate = 0.3, Momentum = 0.2, Training time = 500, Validation threshold =20
Nearest Neighbour Methods	IBk	KNN = 1, nearestNeighbourSearchAlgorithm = LinearNNSearch,
	K*	globalBlend = 20, entropicAutoBlend = False
Ensemble Methods	ABM1	numIterations = 10, weightThreshold = 100, seed =1, classifier = J48: confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, seed = 1, subtreeRaising = True, useMDLcorrection = True
	Bag	numIterations = 10, numExecutionSlots = 1, seed =1, bagSizePercent = 100, classifier = J48: confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, seed = 1, subtreeRaising = True, useMDLcorrection = True
	ICO	evaluationMetric = RMSE, lookAheadIterations = 50, numFolds = 10, numRuns = 1, numThreads = 1, poolsize = 1, seed = 1, stepSize = 1, iterativeClassifier = LogitBoost
	LMT	errorOnProbabilities = False, fastRegression = True, minNumInstances = 15, numBoostingIterations = -1, weightTrimBeta = 0.0
	RT	KValue = 0, breakTiesRandomly = False, maxDepth = 0, minNum = 1, minVarianceProp = 0.001, numFolds = 0, seed = 1
	RSS	numExecutionSlots = 1, numIterations = 10, seed = 1, subSpaceSize = 0.5, classifier = Reptree: initialCount = 0.0, maxDepth = -1, minNum = 2.0, minVarianceProp = 0.001, numFolds = 3, seed = 1
Decision Tree	PART	confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, reducedErrorPruning = False, seed = 1, useMDLcorrection = True
	J48	confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, seed = 1, subtreeRaising = True, useMDLcorrection = True

Proportion Selection of IQAs and CFS selected metrics

IQA	OO Metric	#Selected	Proportion Selection
Cohesion	LCOM	11	0.667
	CAM	8	
	LCOM3	5	
Composition	MOA	8	0.667
Size	WMC	6	0.583
	NPM	6	
	LOC	8	
	AMC	8	
Coupling	Ca	9	0.556
	Ce	10	
	CBO	5	
	RFC	9	
	CBM	3	
	IC	4	
Complexity	Avg_CC	4	0.5
	Max_CC	8	
Encapsulation	DAM	5	0.462
Inheritance	NOC	1	0.194
	MFA	3	
	DIT	3	

Percentage improvement in performance measures after resampling methods are used

Datasets	AUC Growth		Balance Growth		GMean Growth		Sensitivity Growth	
	Max	Median	Max	Median	Max	Median	Max	Median
Ant1.7	18.1%	1.3%	30.9%	16.4%	27.8%	12.1%	67.2%	44.9%
Camel1.6	34.2%	4.4%	69.8%	61.3%	68.5%	72.2%	180.0%	353.8%
Ivy2.0	19.3%	13.7%	62.3%	74.7%	54.8%	75.0%	130.2%	250.0%
Jedit4.0	24.1%	1.3%	34.9%	23.6%	32.9%	17.7%	75.0%	73.2%
Jedit4.2	19.0%	1.2%	66.7%	49.5%	57.4%	45.1%	147.5%	155.6%
Log4j1.0	14.1%	9.2%	34.9%	26.0%	31.0%	20.6%	75.0%	68.2%
Log4j1.1	12.8%	0.0%	16.1%	2.2%	15.2%	1.4%	41.4%	19.4%
Synapse1.0	33.8%	23.9%	38.7%	86.8%	33.8%	86.0%	76.8%	326.3%
Synapse1.1	21.8%	0.0%	35.9%	13.1%	36.4%	9.4%	72.7%	44.7%
Synapse1.2	14.6%	3.9%	21.5%	7.2%	21.1%	5.9%	47.6%	33.3%
Tomcat6.0	23.5%	3.8%	70.8%	77.7%	62.7%	85.0%	160.5%	288.2%
Xerces1.3	16.9%	5.2%	45.8%	31.9%	38.8%	23.7%	93.9%	88.9%
Overall	16.3%	5.3%	24.9%	29.1%	21.5%	21.7%	41.4%	82.1%

Friedman Rankings for Resampling methods with AUC, Balance, GMean, and Sensitivity

	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9	Rank 10	Rank 11
AUC	**ROS (10.11)**	AHC (9.13)	SPD (8.58)	NCL (8.04)	SMT (7.06)	ADSYN (5.55)	NS (4.3)	SLSMT (4.05)	RUS (3.37)	CNNTL (3.08)	OSS (2.68)
Balance	**ROS (10.08)**	AHC (9.01)	SPD (7.93)	SMT (7.56)	NCL (6.6)	ADSYN (6.23)	SLSMT (5.09)	RUS (4.42)	CNNTL (3.85)	OSS (3.41)	NS (1.77)
GMean	**ROS (10.15)**	AHC (9.01)	SPD (7.91)	SMT (7.46)	NCL (6.95)	ADSYN (6.11)	SLSMT (4.88)	RUS (4.36)	CNNTL (3.66)	OSS (3.3)	NS (2.16)
Sensitivity	**ROS (9.89)**	AHC (8.83)	SMT (7.06)	SPD (7.06)	ADSYN (7.01)	CNNTL (6.48)	NCL (5.29)	SLSMT (5.2)	RUS (4.01)	OSS (3.9)	NS (1.22)

Pairwise Comparison of Resampling methods using Nemenyi Test for AUC, Balance, GMean, and Sensitivity

Pair	AUC	Balance	GMean	Sensitivity
ROS-AHC	NS	NS	NS	NS
ROS-SPD	S+	S+	S+	S+
ROS-SMT	S+	S+	S+	S+
ROS-NCL	S+	S+	S+	S+
ROS-ADSYN	S+	S+	S+	S+
ROS-SLSMT	S+	S+	S+	S+
ROS-RUS	S+	S+	S+	S+
ROS-CNNTL	S+	S+	S+	S+
ROS-OSS	S+	S+	S+	S+
ROS-NS	S+	S+	S+	S+

Friedman Rankings for ML techniques with AUC, Balance, GMean, and Sensitivity

	AUC	Balance	GMean	Sensitivity
Rank 1	**Kstar (14.41)**	**ABM1 (13.41)**	**ABM1 (13.5)**	**RT (12.62)**
Rank 2	ABM1 (13.75)	IBk (12.66)	IBk (12.91)	IBk (12.54)
Rank 3	BAG (13.37)	RT (12.16)	RT (12.58)	Kstar (12.33)
Rank 4	RSS (12.29)	BAG (12.12)	BAG (11.75)	ABM1 (12.2)
Rank 5	IBk (9.83)	Kstar (11.08)	Kstar (11.25)	LMT (10.95)
Rank 6	RT (9.2)	RSS (10.33)	RSS (9.58)	PART (9.95)
Rank 7	LMT (8.58)	PART (9.04)	PART (9.04)	J48 (9.83)
Rank 8	J48 (7.62)	LMT (9)	LMT (9)	BAG (9.54)
Rank 9	PART (7.58)	J48 (8.58)	J48 (8.7)	RSS (8.75)
Rank 10	LB (6.08)	MLP (5.45)	MLP (5.75)	LB (5.25)
Rank 11	ICO (5.54)	ICO (4.87)	ICO (4.87)	MLP (4.91)
Rank 12	MLP (4.75)	LB (4.83)	LB (4.7)	ICO (4.83)
Rank 13	LR (2.79)	SL (2.75)	SL (2.66)	SL (2.66)
Rank 14	SL (2.5)	LR (2.41)	LR (2.33)	LR (2.41)
Rank 15	NB (1.66)	NB (1.25)	NB (1.33)	NB (1.16)
p-value	**0.000**	**0.000**	**0.000**	**0.000**
Kendall	**0.876**	**0.835**	**0.844**	**0.838**

Pairwise Comparison of ML Techniques using Nemenyi Test for AUC, Balance, GMean, and Sensitivity

Pair	AUC	Balance	GMean	Sensitivity
ABM1-IBk	NS	NS	NS	NS
ABM1-RT	NS	NS	NS	NS
ABM1-BAG	NS	NS	NS	NS
ABM1-Kstar	NS	NS	NS	NS
ABM1-RSS	NS	NS	NS	NS
ABM1-PART	NS	NS	NS	NS
ABM1-LMT	NS	NS	NS	NS
ABM1-J48	NS	NS	NS	NS
ABM1-MLP	**S+**	**S+**	**S+**	**S+**
ABM1-ICO	**S+**	**S+**	**S+**	**S+**
ABM1-LB	**S+**	**S+**	**S+**	**S+**
ABM1-SL	**S+**	**S+**	**S+**	**S+**
ABM1-LR	**S+**	**S+**	**S+**	**S+**
ABM1-NB	**S+**	**S+**	**S+**	**S+**

Figure 1

Experimental framework for SDP in imbalanced data
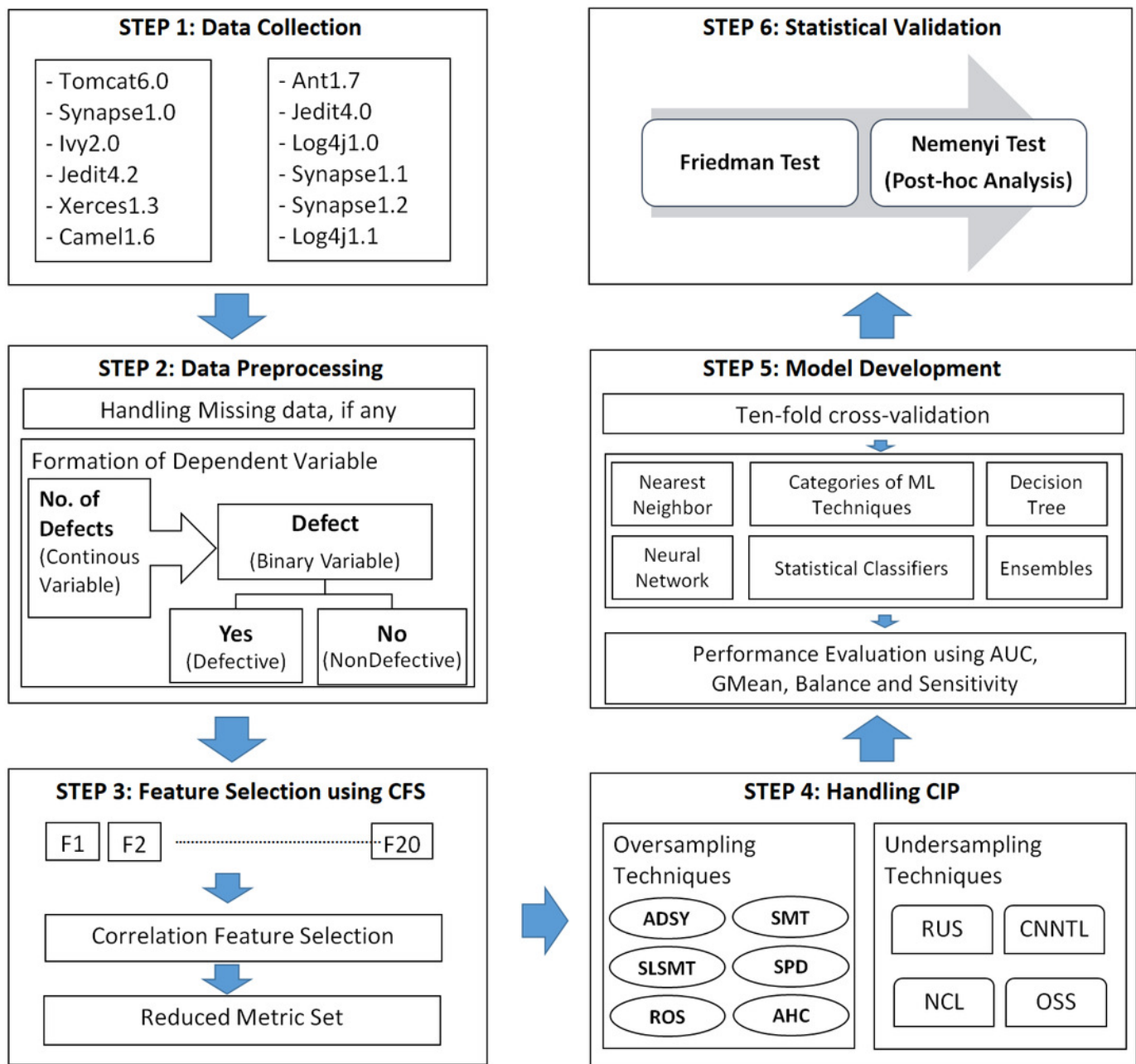
Figure 2

Boxplot Diagram representing Performance Measures using Original Data (NS Case)
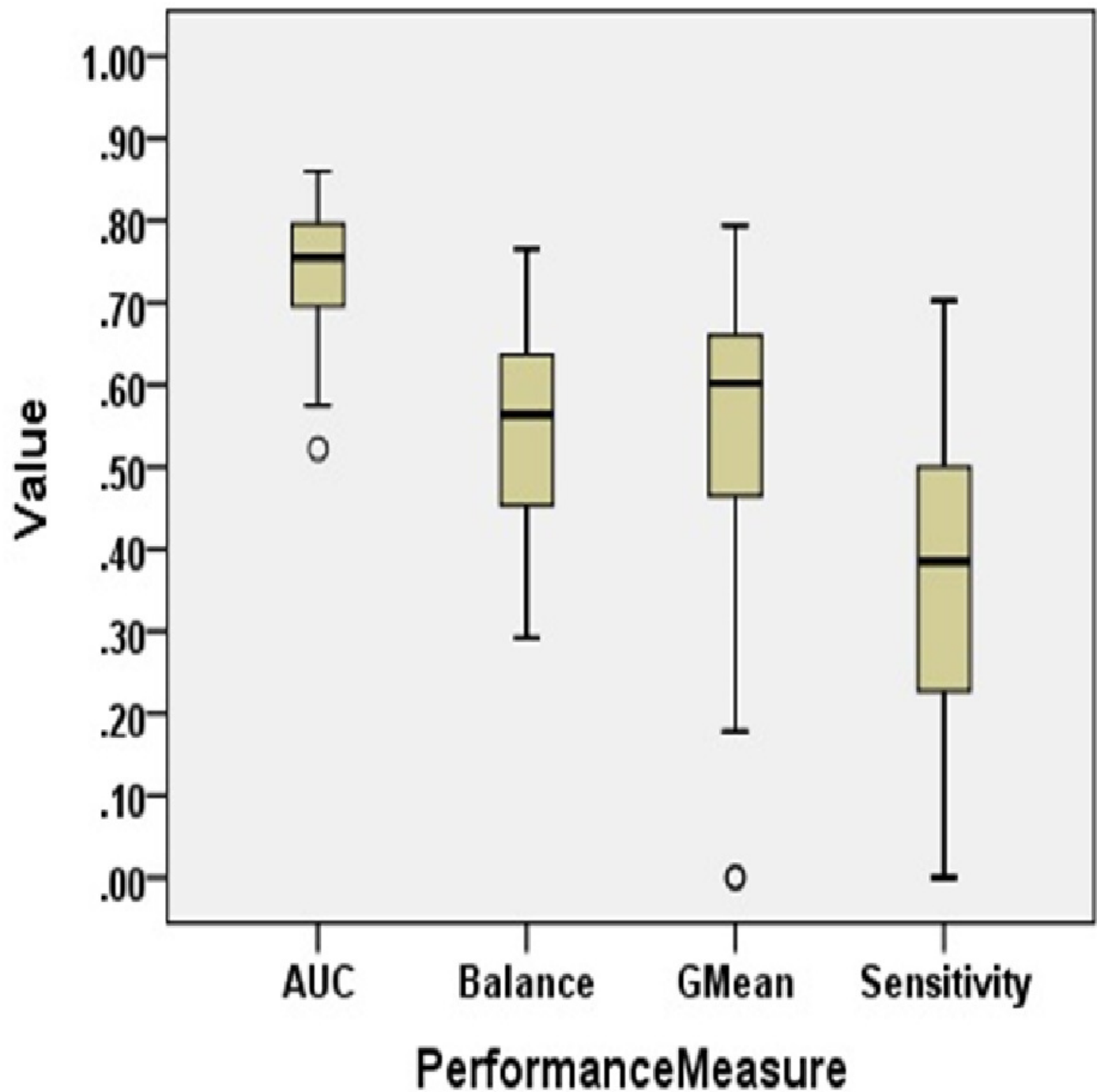
Figure 3

Boxplot Diagram representing Performance Measures using Resampled Data
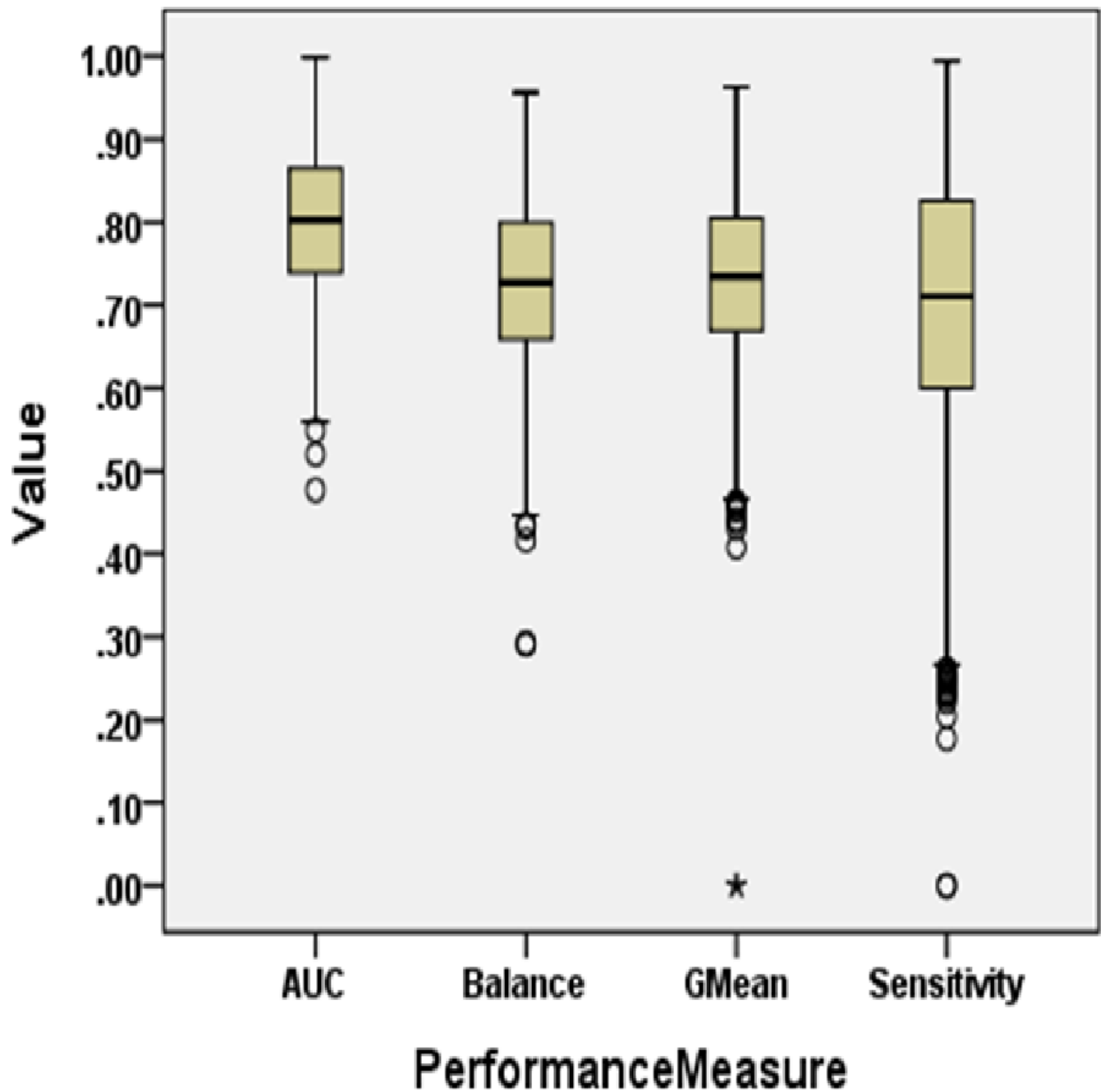
Figure 4

Dataset-wise Comparison of median AUC values of NS and RS
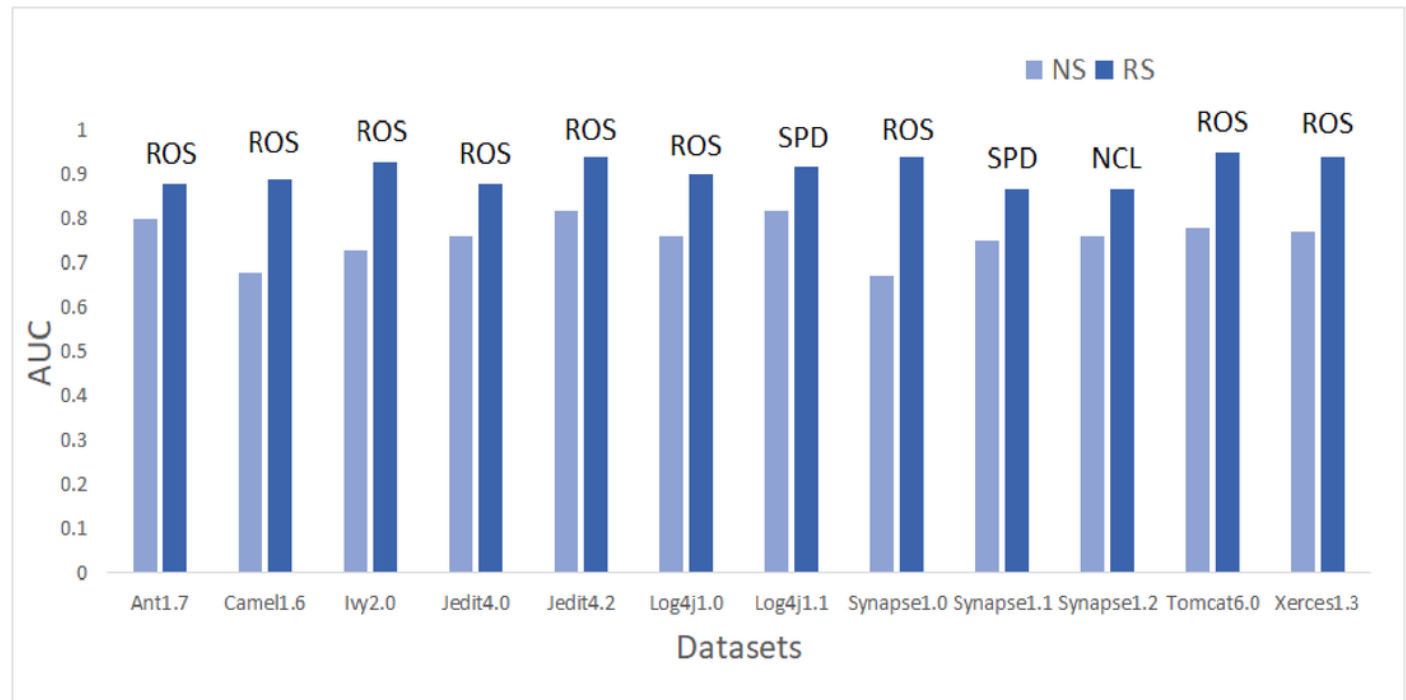
Figure 5

Dataset-wise Comparison of median Balance values of NS and RS
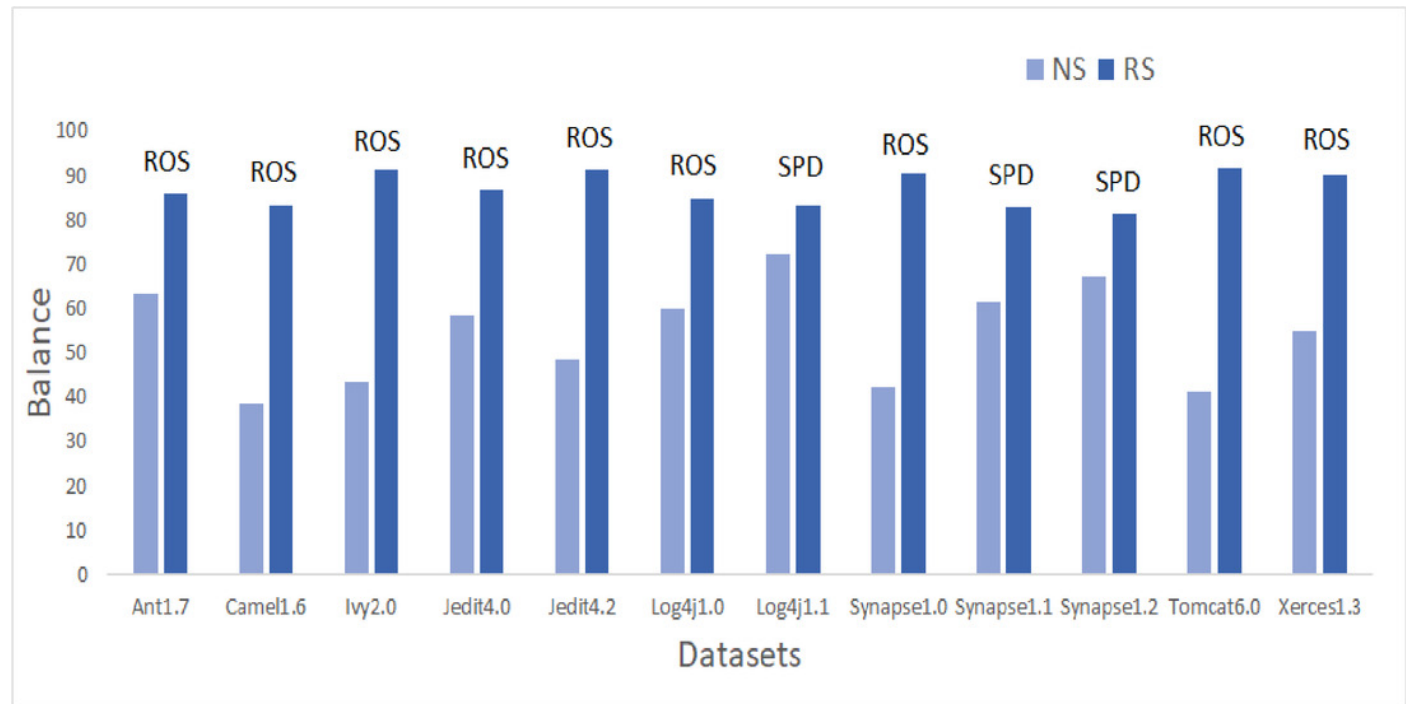
Figure 6

Dataset-wise Comparison of median GMean values of NS and RS
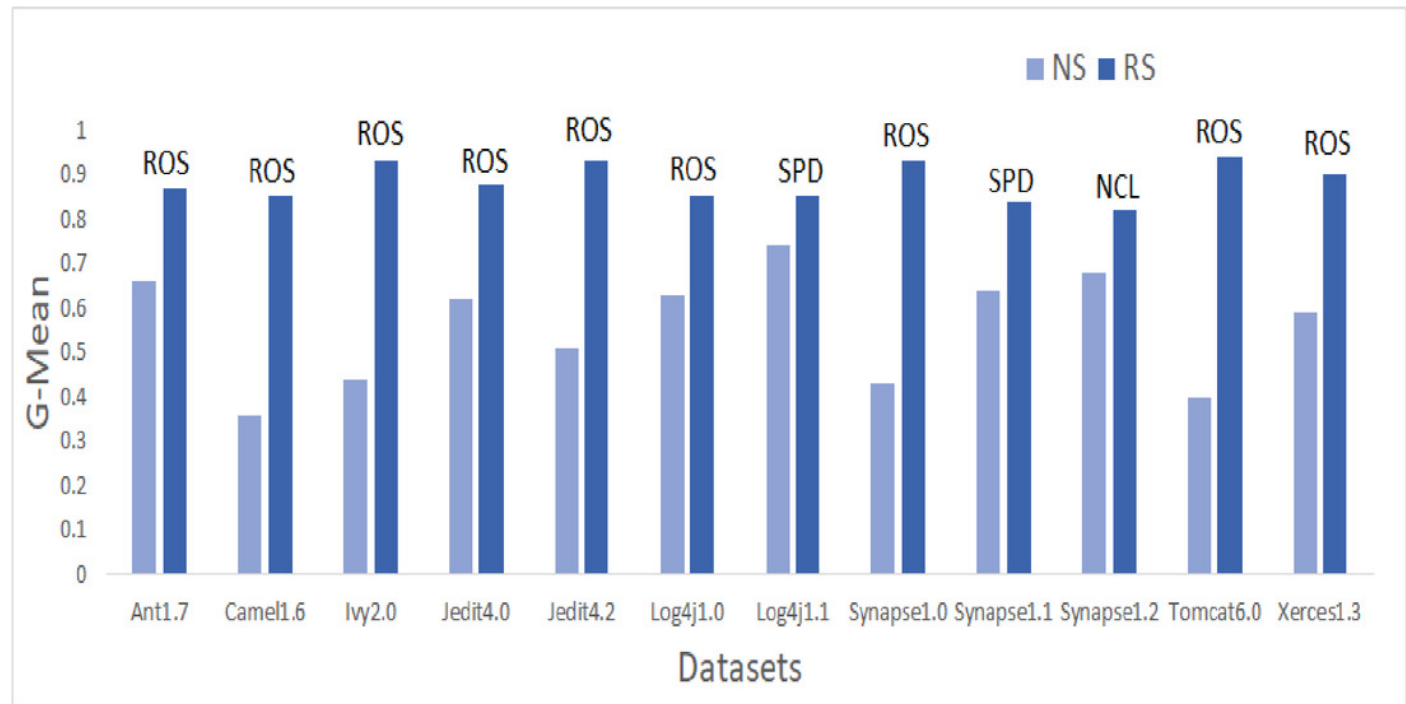
Figure 7

Dataset-wise Comparison of median sensitivity values of NS and RS