Performance Analysis of Software Defects Prediction using Over-Sampling (SMOTE) and Resampling

Mohammad Zubair Khan[1], Reyadh Alluhaibi[2]

[1,2]Department of Computer Science College of Computer Science and Engineering
Taibah University, Madinah Kingdom of Saudi Arabia

Summary
The performance of software defect prediction heavily suffers from data-imbalance. In this article, the imbalance problem using SMOTE and resampling methods has been solved. The performance of software defects prediction with imbalance data and without imbalance data has also been studied. Experiments with WEKA 3.8.3 have been conducted and the performance of different classifiers are calculated using oversampling and resampling methods. Further, for statistical analysis paired T-TEST is used to validate the results. The effectiveness of oversampling and resampling methods for different classifiers are also checked.
The results show that after oversampling and resampling, the performance of classifiers has significantly increased and the winner classifiers are bagging AdaBoost and Random Forest in many cases.

Keywords:
SDP, Software Defect Prediction, Oversampling, Resampling, SMOTE, Classifiers, NaiveBase, SVM, AdaBoost, Bagging, RandomForest.

1. Introduction

Software defect prediction is a vital issue in the field of software engineering research to help the software engineers and programmers to improve the quality and cost-effectiveness of the software package. As the data is increasing the size of software, the complexity is also increasing side by side. Software defects are costly thus finding the defects before delivering them is vital for an organization. Software defect prediction SDP is a field of machine learning for predicting. There are defects in software or not based on previously available software data. Thus, it is a learning problem over software metrics that are in numbers to reflect the complexity of software such as size and control. The software metrics dataset commonly uses Line of Code (LOC), Cyclomatic Complexity Metrics (CCM) [1], Halstead Complexity Metrics (HCM) [2] and Object-Oriented Metrics (OOM) [3]. For our experiments, we use the NASA [10] dataset for software defects prediction.
Many types of research in SDP show that there is a correlation between LOC and predicting defects [4,5]. Chidambaram and Kemerer[6] also proposed may software metrics called (CK) metrics for object oriented system. The CK metrics are weighted method per class (WMC), depth of inheritance tree (DIT) and many children (NOC) and so on. Other object oriented metrics are available in the literature [7,8,9]. In SDP, the data of interest is a defective class module, as per dataset the defective modules are very less, and the non-defective class module is in majority. Therefore, we can say that the available dataset for the experiment is class-imbalanced. A dataset is called class imbalanced if non-interest classes are in majority in compare to interest classes [2].
Since software imperfection forecast has depended on software metrics datasets, it cannot be avoided from the class- imbalance issue. Class- imbalance influences the precision of defects expectation and empowers the indicator cannot predict defective modules well. In this manner, it is essential for defect expectations to tackle the class-imbalance issue. However, the class imbalance problem is very common in classification. With the development of Data mining techniques, many research works have done in this regard to solve the class imbalance problem in SDP [11, 12].
Initially, a few scientists endeavored to utilize cost-touchy learning how to take care of the class imbalance issue. Siers and Islam [13] proposed a technique utilizing a cost delicate choice decision forest and casting a ballot to take care of the class-imbalance issue in SDP. Arara and Ayanba [14] utilized the cost-delicate neural network to set the expenses of misclassifying the positive and negative classes with applicable coefficients and improved the performance of prediction. Zhou and Liu [15] applied cost-touchy neural systems to SDP and they defended which strategies are effective in preparing cost-sensitive neural systems and presumed that threshold moving was fitting for preparing cost-sensitive neural systems.
Secondly, some researcher uses sampling techniques like oversampling, under-sampling and resampling to resolve data imbalance problem. Chawla et al. [16] proposed a SMOTE algorithm that used original instances of minority class to synthesize new samples of a minority class and then append them to datasets.

Thirdly, some researcher uses ensemble learning to improve or resolve-the-data imbalance problem. Chawla et al. [17] tried to combine SMOTE with boosting procedure

Manuscript received November 5, 2019
Manuscript revised November 20, 2019

to improve the accuracy in prediction. Wang et al. [18] proposed the AdaBoost.NC, which utilized the error correlation data into the loads of preparing information, and found that it was superior to standard AdaBoost. Wang [19] improved AdaBoost.NC and proposed another variant of AdaBoost by changing the parameters during training. Laradji et al. [17] endeavored to utilize the average probability ensemble (APE) figuring out how to take care of class imbalance issues. Khoshgoftaar et al. contemplated SMOTEBoost [17] and Random Under-Sampling (RUS) Boost [20], analyzed their performance, and demonstrated that bagging was superior to boosting for noisy and class-imbalanced information [21]. Cholmyong Pak,Tian Tian Wang and Xiao Hong Su [41] proposed a software defect prediction using SMOTE to coup up with data imbalance problem by changing different parameters in SMOTE algorithm.

Finally, researchers also use the resampling methods. These methods are the process of frequently drawing samples from imbalance dataset and refitting a class model on each sample to learn more about the fitted model. Resampling methods are costly in terms of execution because they require repeating the same statistical method on N different subset of data. It is a kind of bootstrap, Cordeiro, and Neves [23, 24], which use exponential smoothing, and bootstrap in time series for forecasting. Now as per the study, the research [19] says that data imbalance can improve at the data label and algorithm label. However, the algorithm depends on data structure and improvement of training data is more significant than the selection of classification algorithms [22]. Here, we focus on sampling methods specially resample and SMOTE. If oversampling happened by duplication and oversample data is the same as original so we use 10-fold cross-validation [25]. However, SMOTE uses the same data to synthesize the new data so we can say that it is a better option for training data and improving the performance of the algorithm.

In this article, we learned about certain issues that are looked in SDP using SMOTE and resampling. We perform tests for examining how the level of an appended minority class and the quantity of nearest neighbor, affected the performance of SDP. We used a paired t-test to analyze the statistical significance of results. We also propose the effectiveness of SMOTE and resampling. The Dataset in this experiment is from the PROMISE (NASA) repository [10]. We selected commonly used classifiers to compare the performance where the result shows that oversampling by SMOTE and resampling was effective in the case of Random Forest, Bagging, AdaBoost and J48-consolidated.

The paper organization is as follows: in Section 2, we describe the background of paired t-test along with the example, SMOTE and resample. In Section 3 the commonly used classifiers are discussed, Section 4 conducts the experiment design, whereas Section 5 analyzes of results of WEKA-3.8.3. In the end, conclusion and future scope are provided.

2. Background

Statistics is vital for analyzing the results; thus, we are using paired t-tests. "The paired t-test is used to compare two data means; here two-samples in which-observations-in-one sample can be paired-with observation in another sample [26]." For example, before and after observation on the same subject or a comparison of two algorithms of measurements or two different treatments where treatments or measurement applied on the same subject.

2.1 Procedure for paired t-test method [26]

Suppose a sample of n Dataset were given a diagnostic (algorithm) test before resampling or oversampling (SMOTE) on a particular module and again perform the test after resampling and SMOTE. To find out improvement in the performance of Algorithm results or test scores. The t-test method works as follows.

Let $x = test\ score\ before\ SMOTE\ or\ resampling$, and let $y = test\ score\ after\ SMOTE\ or\ resampling$

To test the hypothesis of no difference when means of before and after, we shall proceed as below:

Compute the difference $(\mathrm{d}_i = y_i - x_i)$ between above two observations, for all the value of i, the difference is positive or negative

Compute the mean difference $(\bar{d})$

Compute the standard deviation, S

Compute the standard error of mean difference $SE(\bar{d}) = \frac{s}{\sqrt{n}}$

Compute t-test as $|t| = \frac{\bar{d}}{\frac{S}{\sqrt{n}}} \sim t_{n-1}$

Where $S^2 = \frac{1}{n-1}\sum(\mathrm{d}_i - \bar{d})^2$ and n is sample size.

Compute the p-value corresponding to calculated t.

Compare p with the level of significance α (pre-assign value)

If $p \leq \alpha$ then means differ significantly

Example: let's use the calculated value of accuracy for algorithm Random Forest before SMOTE =A and after SMOTE =B

ACCURACY	
A	B
84.72	83.73
77.24	81.09
82.4	84.29
98.3	98.27
76.44	86.58
93.04	93.81
97.84	96.89
87.58	87.13
90.05	92.06
78.95	82.7

Solution:

Paired *t*-test

Paired-Samples-Statistics					
		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	A	86.66	10	8.083	2.556
	B	88.66	10	6.156	1.947

Karl Pearson Correlations

Paired_Samples_Correlations				
		N	Correlation	Sig.
Pair 1	A & B	10	.923	.000

From the above table since $p \leq \alpha = 0.05$, highly Significant

T-Test – Paired_Samples_Test

Paired_Samples_Test									
	Paired-Differences						*t*	*df*	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	90% Confidence Interval of the Difference					
				Lower	Upper				
Pair 1	A – B	-1.999	3.367	1.065	-4.408	.410	-1.877	9	.093

First post: The pair of factors being tried and the request the subtraction was completed. (On the off chance that you have indicated more than one variable pair, this table will have different columns.)

"**Mean**: The-average-of-the-difference-between-the-values-of-variables.

Standard deviation: The standard deviation of the difference scores.

The standard error means: The standard error (standard-deviation divided-by-the-square-root-of-the sample-size). Used-in-computing both-the-test statistic-and-the-upper and-lower-bounds of the confidence interval.

***t*:** The-test statistic- (denoted *t*) for-the-paired *t*-test.

df: The degrees-of-freedom for this test.

Sig. (2-tailed): The-*p*-value corresponding-to-the given-test-statistic *t*-with degrees-of-freedom-*df* [26]".

From the above analysis $p = 0.093 \leq \alpha = 0.10$

There is a significant difference between RF before SMOTE i.e. A – RF after SMOTE i.e. B at 90% Confidence. The above result shows that the performance of random forest after SMOTE is increasing with 90% confidence.

2.2 Performance Metrics for software defect prediction

Confusion Matrix is commonly utilized in assessing the classification method. It appears in Table 1. TP is True - Positive, TN is True - Negative, FP is false - Positive, and FN is false - Negative. "The bigger TP and TN, the more precise (accurate) the classifier is. All measurements are taken from the Confusion Matrix [25]". In terms of software defect prediction the TP, TN, FP, and FN define as

True Positive (TP)	A faulty software instance classified - as - defective.
True Negative (TN)	A non-defective software instance, classified as clean.
False Positive (FP)	A non-defective software instance classified - as - defective
False Negative (FN)	A defective software instance classified as non-defective

Table 1: Confusion Matrix

	Predicted: NO	**Predicted: YES**	**Total**
Actual: NO	**TN**	**FP**	**N**
Actual: YES	**FN**	**TP**	**p**
Total	**N'**	**P'**	

As per rule the accuracy (it is additionally called the acknowledgment rate) of a classifier is defined as follows

$$Accuracy = \frac{TN+TP}{TP+FP+TN+FN} \quad (1)$$

True Positive Rate is also called as sensitivity. This is defined as follows:

$$True\ Positive\ Rate = \frac{TP}{TP+FN} \quad (2)$$

False - Positive - Rate is the proportion of clean software. This is wrongly classified as faulty. The FPR can be defined as follows:

$$False\ Positive\ Rate = \frac{FP}{FP+TN} \quad (3)$$

F-score is a harmonic mean of precision and recall. This can be defined as follows

$$Precision = TP/(TP+FP) \quad (4) \qquad Recall = TP/(TP+FN) \quad (5)$$

$$F-Score = (2 * Precision * Recall)/(Precision + Recall) \quad (6)$$

AUC calculates the area under a ROC – curve. This can be defined as follows:

$$AUC = \frac{1+\mathrm{TPR}+\mathrm{FPR}}{2} \quad (7)$$

"G-measure is another measure that also used in software defects prediction. G-measure is defined as harmonic - mean of - recall and - specificity. Probability - of - False - Alarm (PF) is the - ratio - of clean instances wrongly - classified - defective (FP) among the total clean instances (FP+TN) [25]".

$$PF = FP/(FP+TN) \quad (8)$$

$$Specificity = 1 - PF = TN/FP + TN \quad (9)$$

$$G-measure = 2 * Recall * Specificity/Recall + Specificity \quad (10)$$

2.3 Types of Classifiers

There are several classifiers available in the literature. Some are vital for software defect prediction listing in the Table 2.

Table 2: Type of classifiers use in WEKA for Prediction

No.	Name	Description	Capabilities
1	-J48	"A-Class for producing a pruned or unpruned C4.5.Java implementation of C4.5 it's a kind of Tree [34]".	"-**Class** – Binary-class, Missing-class-values, Nominal-class -**Attributes** -- Binary-attributes, Date-attributes, Empty-nominal-attributes, Missing-values, Nominal-attributes, Numeric-attributes, Unary-attributes -**Interfaces** -- Draw-able, Partition-Generator, -Sourceable, Weighted-Instances-Handler. -**Additional** Minimum number of instances: 0[34]"
2	-Random Forest	"A-Class for building a forest of random-trees. It is known as an ensemble learning method [35]"	"-**Class** -- Binary-class, Missing-class-values, Nominal-class, Numeric-class -**Attributes** -- Binary-attributes, Date-attributes, Empty-nominal-attributes, Missing-values, Nominal-attributes, Numeric-attributes, Unary-attributes -**Interfaces:** -Drawable, Partition-Generator-Randomizable, Weighted-Instances-Handler Additional Minimum number of instances: 1 [35]"
3.	-SMO	"Actualizes John Platt's consecutive negligible improvement algorithm for preparing a support vector classifier. It is otherwise called Support Vector Machine for example SVM [36, 37]"	"-**Class** – Binary-class, Missing-class-values, Nominal-class -**Attributes** -- Binary-attributes, Empty-nominal-attributes, Missing-values, Nominal-attributes, Numeric-attributes, Unary-attributes -**Interfaces** -- Weighted-Instances-Handler **Additional** Minimum number of instances: 1[35]"
4.	--Naïve Bayes	"A-Class for a Naive Bayes classifier using estimator classes. It is based on the Bayes' theorem [38]."	"-Class -- Binary-class, Missing-class-values, Nominal-class Attributes -- Binary-attributes, Empty-nominal-attributes, Missing-values, Nominal-attributes, Numeric-attributes, Unary-attributes Interfaces -- Weighted-Attributes-Handler, Weighted-Instances-Handler Additional Minimum number of instances: 0 [38]"
5.	--AdaBoost	"A-Class for -boosting a nominal class classifier using the Adaboost M1 method [18]."	"-Class -- Binary-class, Missing-- values, Nominal-class -Attributes -- Binary-attributes, Date-attributes, Empty-nominal-attributes, Missing-values, Nominal-attributes, Numeric-attributes, Unary-attributes Interfaces -- -Randomizable, -Sourceable, Weighted-Instances-Handler Additional Minimum number of instances: 1 [18]"
6.	--Bagging	"A-Class for bagging a -classifier to reduce the discrepancy. It is one of the ensemble procedures, it uses a sequence of numerous base classifiers to generate a better composite classifier [39]."	"-Class -- Binary-class, Date-class, Missing-class-values, Nominal-class, Numeric-class -Attributes -- Binary-attributes, Date-attributes, Empty-nominal-attributes, Missing-values, Nominal-attributes, Numeric-attributes, Unary-attributes -Interfaces -- Partition-Generator, -Randomizable, Weighted-Instances-Handler Additional Minimum number of instances: 1 [39]"
7.	--Simple Logistic regression	"A-Classifier for construction linear logistic regression -models. It is a -statistical technique to predict the -probability of -categorical dependent-variable [40]."	"-Class -- Binary-class, Missing-class-values, Nominal-class -Attributes -- Binary-attributes, Date-attributes, Empty-nominal-attributes, Missing-values, Nominal-attributes, Numeric-attributes, Unary-attributes Interfaces -- Weighted-Instances-Handler Additional Minimum number of instances: 1 [40]"

3. Sampling Strategies

The sampling method plays an important role in solving data imbalance problems. Sampling procedures include oversampling under-sampling and resampling. In the oversampling method, it randomly selects samples of a minority class and adds them to the dataset [27]. In the under-sampling methods, it randomly select the majority class and remove them from dataset for balance the class. Both techniques have a problem like in under-sampling if majority class remove from dataset because it may be lost the value able information from data, and in under-sampling methods, it would add many samples of minority class to the dataset. This leads to overfitting of minority class [28]. Resampling methods are usually utilized for class imbalance problem. Their importance over additional sampling procedures is that they are simply portable. Albeit such methods can be very simple to implement but tuning with them adequately is not a simple task. It is not clear which method is more effective [29, 30]. There are many versions of oversampling and under-sampling available. "Seba Susan, Amitesh Kumar says data imbalance problem can be achieved before classification by a novel three steps intelligent under-sampling of majority class monitored by oversampling of a minority class, which is again monitored by intelligent under-sampling of the minority class [29]". That has now turned into the majority class due to oversampling. Kubat at.el. [31] proposed an uneven selection under-sampling method, which especially removes only negative samples and keeps all positive samples. Andrew Estabrooks et.al.[30] tried to endeavor to consolidate an under-sampling with an over-sampling for adjusting imbalanced data. By the combination of under-sampling and oversampling could not remove their weaknesses [16]. Pelayo at.et [32] looked at a few sampling methods and found that at 0.05 of significance level, the impact of under-sampling was critical, and the impact of over-sampling was not significant. "SMOTE algorithm is an only oversampling method which uses original samples of minority class to append to dataset [16, 17]". "It is useful than random oversampling and under-sampling for solving data imbalance [33]". The main algorithm SMOTE as follow:

Algorithm SMOTE (T, N, k) (Source: [17])

$Input: Number\ of\ Minority\ class\ samples T;\ Amount\ of\ SMOTEN\%;\ Number\ of\ nearest\ neighbors\ k$

$Output: \left(\frac{N}{100}\right) * T\ synthetic\ minority\ class\ samples$

(1) $if\ N < 100$
(2) $then\ Randomize\ the\ T\ minority\ class\ samples$
(3) $T = \left(\frac{N}{100}\right) * T$
(4) $N = 100$
(5) $endif$
(6) $N = (int)(\frac{N}{100})$
(7) $k = Number\ of\ nearest\ neighbors$
(8) $numofattrs = Number\ of\ attribute$
(9) $Sample[\ ][:]array\ for\ the\ original\ minority\ class$
(10) $new\ index: Keeps\ a\ count\ of\ the\ number\ of\ synthetic\ samples\ generated, initialize\ to\ 0$
(11) $Synthetic[\ ][\ ]: array\ for\ synthetic\ samples$
(12) $for\ i = 1\ to\ T$
(13) $Populate\ (N, i, nnarray)$
(14) $endfor$
$Populate(N, i, nnarray)\ //\ Function\ togenerate\ the\ synthetic\ samples.$
(15) $while\ N \neq 0$
(16) $Choose\ a\ random\ number\ between\ 1\ and\ k\ call\ it\ nn. This\ step\ chooses\ one\ of\ the\ k\ nearest\ neighbours\ of\ i$
(17) $for\ attr = 1\ to\ numattrs$
(18) $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
(19) $gap = random\ number\ between\ 0\ and\ 1$
(20) $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
(21) $endfor$
(22) $newindex++$

(23) $N = N - 1$
(24) $endwhile$
(25) $returns\ (*\ end\ of\ populate\ *)$

Table 3: Example of production of synthetic samples (SMOTE).

$\boldsymbol{Example}$: $Let\ us\ consider\ the\ samples\ are\ (5,7), (2,8), (6,8)\ and\ (6,8)$
$Let\ (5,7) be\ the\ sample\ and\ k = 3, so\ nearest\ neighbours\ are\ (2,8), (6,8), (6,8)$
$Suppose\ that\ the\ sample\ (6,8) is\ one\ among\ three\ neighbours.$
$x_1 = 5, y_1 = 6\ , y_1 - x_1 = 6 - 5 = 1$
$x_2 = 7, y_2 = 8\ , y_2 - x_2 = 8 - 7 = 2$
$So, new\ sample\ (f_1, f_2) = (5,7) + rand(0-2) * (1,2)$
$rand(0-2) generate\ a\ reandom\ number\ betwen\ 0\ and\ 2$

4. Experiment Method

4.1 Software defect dataset Repository

The Software defect dataset MATRICS used in this article listed in Tables 4 and 5. We use the PROMISE Repository for this purpose.

Table 4: Dataset attributes from the PROMISE repository

Metric Type	Metric Names	CM1	KC1	KC2	MC1	MC2	PC1	PC2	PC3	PC4	PC5
McCabe	01. Cyclomatic-Complexity	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	02. Cyclomatic-Density	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	03. Decision Density	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	04. Design Density	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	05. Essential Complexity	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	06. Essential Density	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	07. Global Data Density	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	08. Global Data Complexity	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	09. Maintenance Severity	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	10. Module Design Complexity	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	11. Pathological Complexity	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	12. Normalized Cyclomatic Complexity	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Raw Halstead	13. Number of Operators	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	14. Number of Operands	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	15. Number of Unique Operators	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	16. Number of Unique Operands	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Derived Halstead	17. Length (N)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	18. Volume (V)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	19. Level (L)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	20. Difficulty (D)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	21. Intelligent Content (I)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	22. Programming Effort (E)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	23. Error Estimate (B)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES

	24. Programming Time (T)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
LOC COUNT	25. LOC Total	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	26. LOC Executable	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	27. LOC Comments	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	28. LOC Code and Comments	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	29. LOC Blank	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	30. Number of Lines (opening to closing bracket)	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Misc.	31. Node Count	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	32. Edge Count	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	33. Branch Count	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
	34. Condition Count	YES	YES	YES	NO	YES	YES	YES	YES	YES	YES
	35. Decision Count	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES
	36. Formal Parameter Count	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES
	37. Modified Condition Count	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES
	38. Multiple Condition Count	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES
	39. Call Pairs	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES
	40. Percent Comments	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES
Error	41. Error Count	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES
	42. Error Density	YES	YES	YES	NO	NO	YES	YES	YES	YES	YES

Table 5: Dataset from the PROMISE repository

No.	Dataset	Instance	Defective Instance	Non defective Instance	$\frac{Defective}{Non\ Defective}$	Data Imbalance Rate (%)
1.	CM1	327	42	285	0.147	12.8440
2.	KC1	1177	309	868	0.355	26.2531
3.	KC3	216	58	158	0.367	26.8518
4.	MC1	2002	86	1916	0.044	04.2957
5.	MC2	173	93	80	1.162	53.7572
6.	PC1	732	108	624	0.173	14.7540
7.	PC2	787	81	706	0.114	10.2922
8.	PC3	1095	172	923	0.186	15.7077
9.	PC4	1296	202	1094	0.184	15.5864
10.	PC5	1748	512	1236	0.414	29.2906

4.2 Experiment Design

In this article, three experiments have been performed on DATASET, SMOTED-DATASET, and Resampled-DATASET showed in Figures 4-6. For experiments, we use WEKA-3.8.3 tools for the given Dataset in Table 5. “In these experiments, we use the 10-fold cross-validation to estimate the performance of software defect prediction [25]”. For all the classifiers, all their parameters are set by defaults in the WEKA tool. Here we are using 6 classification algorithms like NaiveBase, SVM, J48, AdaBoost, Bagging and Random Forest on raw Dataset (-CM1, -KC1, -KC3, -MC1, -MC2, -PC1, -PC2, -PC3, -PC4, -PC5), SMOTED dataset and Resampled Dataset, as shown in Figures 1 and 3.

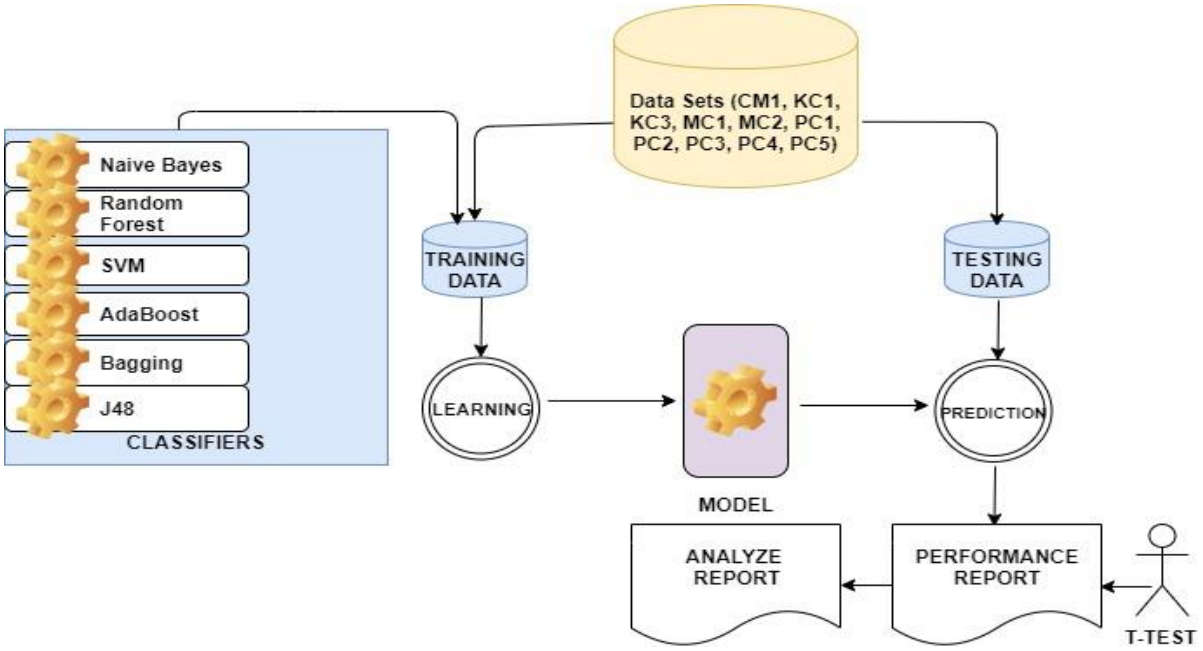

Fig. 1 Experiment on DATASET

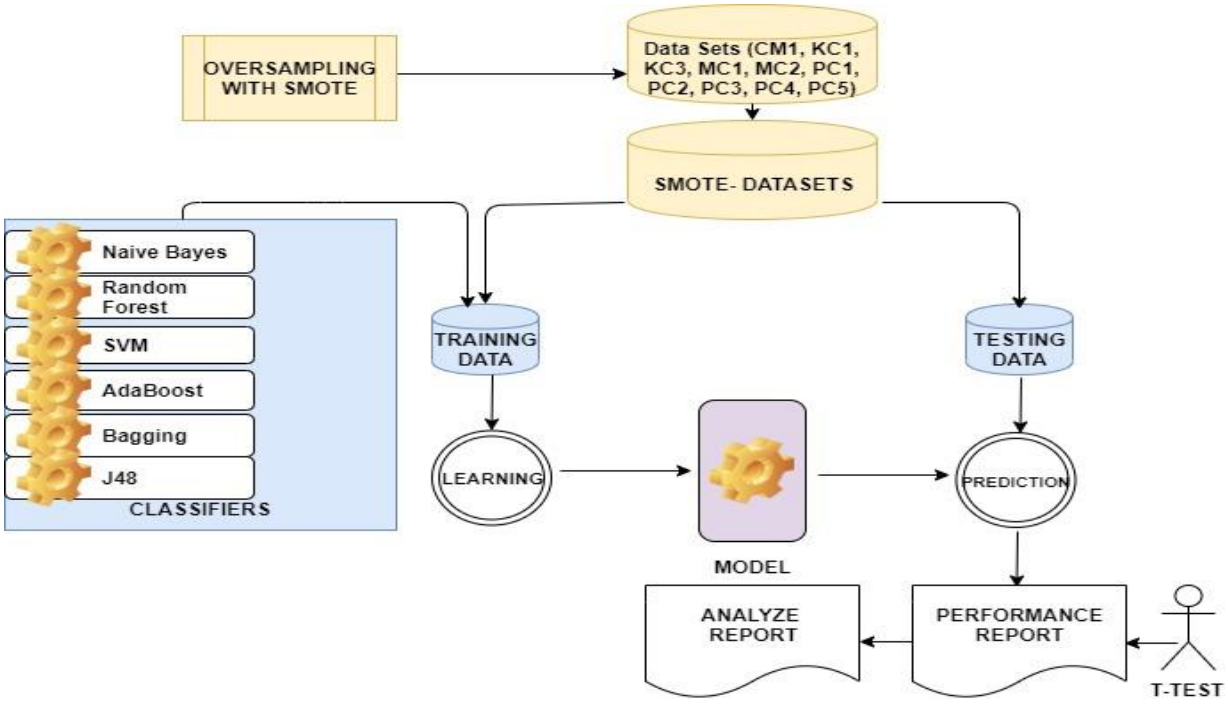

Fig. 2 Experiment on SMOTED-DATASET

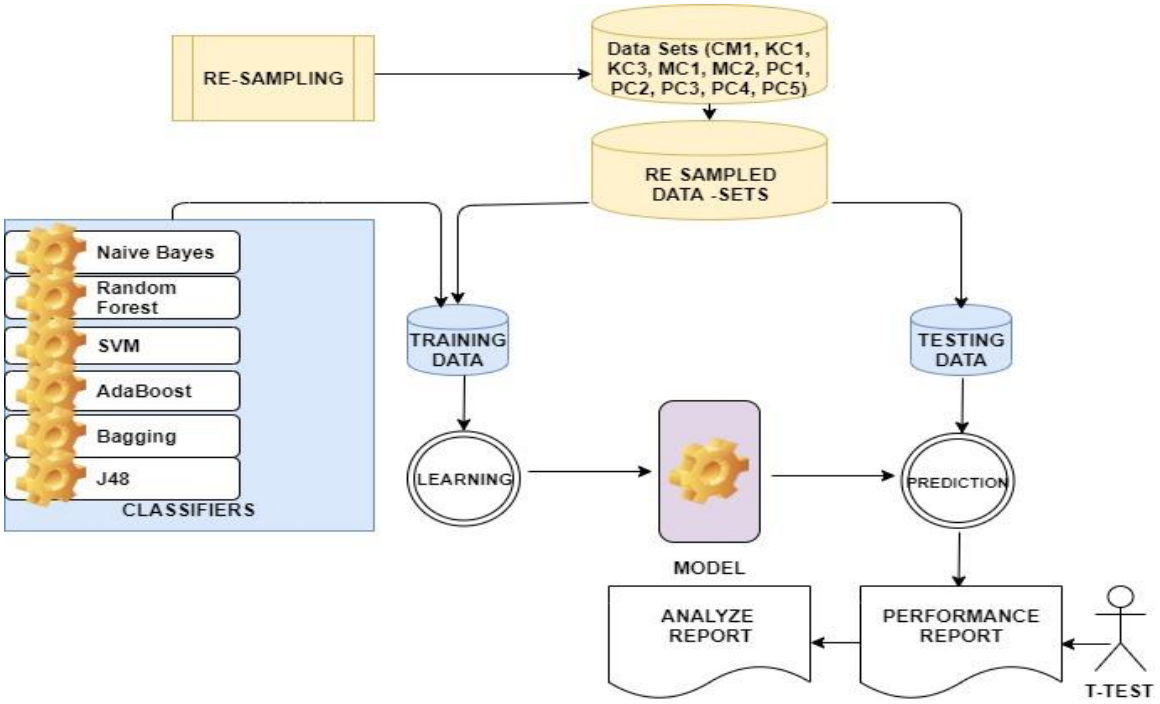

Fig. 3 Experiment2 With Resampling

5. Results Analysis

The experiment results shows in Tables 7-12. Here in the experiment set the level of significance $\alpha = 5\%$ or 0.05. The sign "v" represents that p cost is lesser than 0.05, which means the null hypothesis is rejected. The sign "*" represents that p cost is not smaller than 0.05, which means the hypothesis is accepted. In Tables 6-11, we show the Accuracy and F-measure score of ten datasets.

5.1 A t-test on imbalanced dataset

We can see that NaiveBase, our base for a correlation set apart as (1) has unique accuracy of 10 datasets on the problem. This outcome is compared to the other five algorithms and indicated with a number and mapped in a legend at the base under the table7 of results.

Note the "*" alongside J48 on PC1 dataset results. This demonstrates the outcomes are altogether not quite the same as the NaiveBase results, yet the score is lower. SVM, J48, AdaBoost, Bagging and Random Forest do not have any character alongside their outcomes in Table 6, demonstrating that the outcomes are essentially not quite the same as NaiveBase. On the off chance that an algorithm had results bigger than the base algorithm and the difference was significant, a bit "v" would show up beside the outcomes.

Therefore, in Table 6, there is 2 victory of SVM over NaiveBase and 8. However, we cannot say they are significantly accepted or rejected on p-value 0.05. We also found that in J48 has one hypothesis of no difference is rejected on the PC3 dataset and one hypothesis of no difference is accepted on the PC1 dataset and 8 again cannot say hypothetically accepted or rejected. In Adaboost 3 victory achieved mean null hypothesis is rejected on dataset PC3, PC4, PC5, no accepted and 7 places. We again cannot say that hypothetically accepted or rejected. In bagging, the algorithm is a winner over NaiveBase at 4places mean null hypothesis is rejected at 0.05 and we can't say the hypothesis is accepted or rejected at 6 places. In Random Forest the algorithm again winner at 3 places which means a null hypothesis is rejected at 0.05 and we cannot say whether it is accepted or rejected at 7 places.

Table 6: Accuracy on imbalanced Dataset

ACCURACY						
Dataset	NB (1)	SVM (2)	J48Consolidated(3)	AdaBoost (4)	Bagging (5)	RF (6)
CM1	81.34	87.16	72.84	83.79	85.33	84.72
KC1	73.4	74.09	67.55	72.89	79.02	77.24
KC3	82.4	83.74	81.97	84.31	82.86	82.4
MC1	85.76	98.2	90.61	98.25	98.2	98.3

MC2	78.2	76.93	76.34	74.02	77.61	76.44
PC1	90.16	92.49	82.25*	92.49	92.49v	93.04
PC2	96.7	97.84	94.02	96.95	97.84V	97.84
PC3	28.13	88.13v	77.54v	86.85v	87.95	87.58V
PC4	85.03	89.12v	83.34	86.85v	89.43V	90.05V
PC5	72.71	72.14	73.74	86.85v	78.77V	78.95V
	(v/ /*)	**(2/8/0)**	**(1/8/1)**	**(3/7/0)**	**(4/6/0)**	**(3/7/0)**

Now in Table 7 of F-measure again NaiveBase is our base algorithm and we are going to perform a paired T-test one by one and find the different results. The SVM is tested with NaiveBase and we found that SVM performs well on 2 places while on 2 places it also loses i.e. hypothesis is accepted at the value 0.05. The Algorithm NaiveBase now tested with J48 here we found 4 places null hypothesis is rejected. A p-value is showing with sign "v" and one rejection also showing, at 5 places the hypothesis of no difference that cannot say it is accepted or rejected at 0.05. Now NaïveBase is compared with AdaBoost, Bagging and Random Forest respectively tests found that in every 4 places all three algorithms are winner mean the hypothesis with no difference is rejected and at 6 places can not say that the hypothesis is accepted or rejected.

. Table 7:F-measure on imbalanced Dataset

***F*-measure**						
Dataset	NB (1)	SVM (2)	J48Consolidated(3)	AdaBoost (4)	Bagging (5)	RF (6)
CM1	0.29	0.4	0.16	0	0	0
KC1	0.38	0.08 *	0.47	0.43	0.46	0.45
KC3	0.59	0.57	0.68	0.67	0.55	0.56
MC1	0.11	0.72v	0.41v	0.75v	0.72v	0.74v
MC2	0.76	0.74	0.77	0.74	0.75	0.76
PC1	0.67	0.65	0.56 *	0.72	0.68	0.71
PC2	0.84	0.88	0.75	0.85	0.88	0.88
PC3	0.29	0.39	0.52v	0.52v	0.42v	0.45v
PC4	0.35	0.48v	0.61v	0.63v	0.54v	0.60v
PC5	0.3	0.12*	0.60v	0.57v	0.55v	0.58v
	(v/ /*)	**(2/5/2)**	**(4/5/1)**	**(4/6/0)**	**(4/6/0)**	**(4/6/0)**

5.2 A T-test on SMOTED dataset

Further, in Table 8, the accuracy results are shown, and in this experiment using NaiveBase, our base for comparison to rest 5 algorithms has different accuracy of 10 SMOTED datasets in the problem. This outcome is compared to the other five algorithms, indicated with a number, and mapped in a legend at the base under the Table 8 of results. After SMOTE the Dataset result of accuracy is improved as we see there is no acceptance of the null hypothesis and rejection of the null hypothesis is increases.

As we see that SVM performance is checked with our base Algorithm i.e. NaïveBase the t-test show that SVM is a winner at 4 places the sign “v” is appearing on same, mean the null hypothesis is rejected and also 6 places hypothesis is accepted or rejected condition appears at p-value 0.05. The performance of AdaBoost, Bagging and Random Forest is very much Improved in accuracy they are winners respectively at 7 places mean null hypothesis is rejected and each has 3 places which cannot say the hypothesis is accepted or rejected.

Table 8: Accuracy on SMOTED dataset

ACCURACY						
Dataset	NB(1)	SVM(2)	J48Consolidated(3)	AdaBoost(4)	Bagging(5)	RF(6)
CM1	75.36	76.43	76.71	83.46 v	82.37	83.73 v
KC1	64.4	65.82	74.36 v	78.13 v	80.69 v	81.09 v
KC3	76.53	75.46	81.76	84.34	86.08 v	84.29
MC1	78.54	93.63 v	94.16 v	98.01 v	97.79 v	98.27 v
MC2	82.62	85.03	80.25	85.42	86.2	86.58
PC1	86.67	87.02	86.19	91.55 v	92.74 v	93.81 v
PC2	95.74	96.31	92.98	97.12	96.54	96.89
PC3	37.17	80.42 v	80.19 v	85.55 v	87.37 v	87.13 v
PC4	78.17	85.72 v	87.72 v	90.26 v	92.53 v	92.06 v
PC5	61.24	66.95 v	76.64 v	81.02 v	82.61 v	82.70 v
	(v/ /*)	**(4/6/0)**	**(5/5/0)**	**(7/3/0)**	**(7/3/0)**	**(7/3/0)**

Further, in Table 9, the experiment is checking the F-measure of all 6 algorithms and again taking NaiveBase as our base algorithm. Here we found that SVM performance is 3 places null hypothesis is rejected and at 2 places hypothesis of no difference is accepted while 5 places cannot say that condition appear at p-value on 0.05 or 5 % level of confidence. if we check the J48 and AdaBoost with NaiveBase we found that 6 places null hypothesis is rejected, and 4 places cannot say whether it rejected or accepted respectively. The F-measure score of Bagging and Random Forest again winner at 7 places mean the hypothesis of no difference is rejected and 3the condition is cannot say the hypothesis is accepted or rejected at p-value 0.05. After SMOTED data, the overall performance of algorithms is improved.

Table 9: F_measure on SMOTED dataset

F_measure						
Dataset	NB (1)	SVM (2)	J48Consolidated(3)	AdaBoost (4)	Bagging (5)	RF (6)
CM1	0.36	0.12 *	0.56 v	0.59	0.49	0.55
KC1	0.44	0.39 *	0.68 v	0.73 v	0.75 v	0.76 v
KC3	0.65	0.6	0.78	0.82	0.82 v	0.80 v
MC1	0.53	0.73 v	0.83 v	0.93 v	0.92 v	0.94 v
MC2	0.74	0.73	0.72	0.78	0.77	0.78
PC1	0.73	0.67	0.75	0.83 v	0.84 v	0.86 v
PC2	0.88	0.89	0.83	0.92	0.9	0.91
PC3	0.45	0.51	0.67 v	0.72 v	0.74 v	0.74 v
PC4	0.45	0.67 v	0.80 v	0.82 v	0.86 v	0.85 v
PC5	0.34	0.65 v	0.74 v	0.79 v	0.80 v	0.81 v
	(v/ /*)	**(3/5/2)**	**(6/4/0)**	**(6/4/0)**	**(7/3/0)**	**(7/3/0)**

5.3 T-test on Resampled Dataset

Here in the final experiment, using resampling on the raw dataset, and making NaiveBase, our base for comparison to rest 5 algorithms have different accuracy scores and F-measure score shown in Tables 11 and 12 respectively. After resampling the dataset there is no accepted of null hypothesis on all compared algorithms if we check SVM with base Algorithm NaiveBase we found that at 7 places the null hypothesis is rejected while at 3 places the hypothesis is cannot say it is accepted or rejected hypothesis. Table [10] shows that AdaBoost, Bagging and Random Forest performance is increasing and, on all datasets, they are the winner, here in all three algorithms the hypothesis of no difference is rejected and no accepted or cannot say condition does not appear. We found that the algorithms AdaBoost, Bagging and Random Forest is performing well in nature mean null hypothesis is rejected on all datasets as victory shows in Table 10.

Table 10: Accuracy on Resampled Dataset

ACCURACY						
Dataset	NB (1)	SVM(2)	J48Consolidated(3)	AdaBoost(4)	Bagging(5)	RF (6)
CM1	82.85	86.55	80.42	94.82 v	93.86 v	94.49 v
KC1	71.54	74.60 v	76.55 v	89.88 v	89.97 v	90.56 v
KC3	81.54	86.67 v	94.42 v	97.23 v	97.25 v	97.71 v
MC1	92.06	98.15 v	91.76	99.35 v	99.00 v	99.35 v
MC2	75.13	79.8	84.44	90.29 v	87.91 v	90.26 v
PC1	90.15	92.89 v	89.06	96.72 v	96.17 v	97.13 v
PC2	78.52	97.71 v	94.79 v	99.49 v	99.11 v	99.49 v
PC3	34.91	87.67 v	85.11 v	93.34 v	93.25 v	94.16 v
PC4	84.72	89.28 v	87.66	95.22 v	95.06 v	95.06 v
PC5	71.68	71.57	84.04 v	89.88 v	90.22 v	90.91 v
-	**(v/ /*)**	**(7/3/0)**	**(5/5/0)**	**(10/0/0)**	**(10/0/0)**	**(10/0/0)**

Further experiment checks the F-measure score shown in Table 11 we found that SVM has 3 places null hypothesis is accepted and 2 places null hypothesis is rejected rest of the algorithms like AdaBoost, Bagging and Random forest performance is excellent on resample data. All 3 algorithms are rejecting the hypothesis of no difference at 5%, on all ten datasets.

Table 11: F-measure on Resampled Dataset

F-measure						
Dataset	NB (1)	SVM (2)	J48Consolidated(3)	AdaBoost(4)	Bagging(5)	RF (6)
CM1	0.38	0.00 *	0.5	0.79 v	0.73 v	0.73 v
KC1	0.36	0.10 *	0.61 v	0.80 v	0.78 v	0.81 v

KC3	0.57	0.65	0.90 v	0.95 v	0.95 v	0.95 v
MC1	0.12	0.72 v	0.50 v	0.92 v	0.87 v	0.92 v
MC2	0.71	0.79	0.85	0.91 v	0.88 v	0.91 v
PC1	0.67	0.67	0.71	0.88 v	0.85 v	0.89 v
PC2	0.43	0.86 v	0.79 v	0.97 v	0.95 v	0.97 v
PC3	0.32	0.35	0.64 v	0.77 v	0.74 v	0.79 v
PC4	0.38	0.49	0.70 v	0.84 v	0.82 v	0.82 v
PC5	0.26	0.09 *	0.75 v	0.82 v	0.82 v	0.83 v
	(v/ /*)	(2/5/3)	(7/3/0)	(10/0/0)	(10/0/0)	(10/0/0)

6. Conclusion

In this article, we studied software defects predictions using SMOTE and resample methods of different datasets. The core contribution of this article is to check the performance of classifiers after SMOTE, resample, and validate the results using a paired t-test. Further, we check the evaluation criteria whether SMOTE or resampled is effective or not. The experiment found that the performance of AdaBoost, Bagging and Random Forest improved after SMOTE and resampled dataset in many cases. SMOTE and resample alone cannot solve the data imbalanced problem.

References:

[1]. T. J. McCabe, A complexity measure, IEEE Trans. Software. Eng. 2(4) (1976) 308–320.

[2]. M. H. Halstead, Elements of Software Science (Elsevier Science, 1977).

[3]. S. R. Chidamber and C. F. Kemerer, A metrics suite for object-oriented design, IEEE Trans. Softw. Eng. 20(6) (1994) 476–493.

[4]. Zhang, H. An investigation of the relationships between lines of code and defects. in 2009 IEEE International Conference on Software Maintenance. 2009.

[5]. Mende, T. and R. Koschke. Revisiting the evaluation of defect prediction models. in Proceedings of the 5th International Conference on Predictor Models in Software Engineering. 2009. ACM.

[6]. Chidamber, S.R., and C.F. Kemerer, A metrics suite for object-oriented design. IEEE Transactions on software engineering, 1994. 20(6): p. 476-493.

[7]. Jureczko, M. and D. Spinellis, Using object-oriented design metrics to predict software defects. Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, 2010: p. 69-81.

[8]. Gupta, D.L. and K. Saxena, Software bug prediction using object-oriented metrics. Sādhanā, 2017. 42(5): p. 655-669.

[9]. Singh, A., R. Bhatia, and A. Singhrova, Taxonomy of machine learning algorithms in software fault prediction using object-oriented metrics. Procedia Computer Science, 2018. 132: p. 993-1001.

[10]. http://openscience.us/repo/defect/mccabehalseted/

[11]. H. He and E. A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21(9) (2009) 1263–1284.

[12]. D. Rodriguez, I. Harraiz and R. Harrison, Preliminary comparison of techniques for dealing with an imbalance in software defect prediction, in Int. Conf. Evalua. Assessment of Softw. Eng. 43 (2014) 1–10.

[13]. M. J. Siers and Md. Z. Islam, Software defect prediction using a cost-sensitive decision forest and voting, and a potential solution to the class imbalance problem, Inf. Syst. 51 (2015) 62–71.

[14]. Ö . F. Arara and K. Ayanba, Software defect prediction using the cost-sensitive neural network, Appl. Soft Comput. 33 (2015) 263–277.

[15]. Z.-H. Zhou and X.-Y. Liu, Training cost-sensitive neural networks with methods addressing the class imbalance problem, IEEE Trans. Knowl. Data Eng. 18(1) (2006) 63–77.

[16]. N. V. Chawla, K. W. Bowyer, L. O. Hell, and W. P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, J. Arti¯. Intell. Res. 16 (2002) 321–357.

[17]. N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, SMOTEBoost: Improving prediction of the minority class in boosting, Knowledge Discovery in Databases: PKDD 2003, LNAI Vol. 2838, 2003, pp. 107–119.

[18]. S. Wang, H. Chen and X. Yao, Negative correlation learning for classification ensembles, in Proc. Int. Joint Conf. Neural Netw., WCCI, 2010, pp. 2893–2900.

[19]. S. Wang and X. Yao, Using class imbalance learning for software defect prediction, IEEE Trans. Reliabil. 62(2) (2013) 434–443.

[20]. C. Seirert, T. M. Khoshgoftaar, J. V. Hulse and A. Napolitano, RUSBoost: A hybrid approach to alleviating class imbalance, IEEE Trans. Syst. Man, Cybern. Part A, Syst. Humans 40(1) (2010) 185–197

[21]. T. M. Khoshgoftaar, J. V. Hulse and A. Napolitano, Comparing boosting and bagging techniques with noisy and imbalanced data, IEEE Trans. Syst. Man, and Cybern. Part A, Syst. Humans 41(3) (2011) 552–568.

[22]. T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic and Y. Jiang, Implications of ceiling erects in defect predictors, in Int. Workshop on Predictor Models in Software Engineering, 2008, pp. 47–54.

[23]. C. Cordeiro and M.M. Neves, The Bootstrap methodology in time series forecasting, In “Proceedings of CompStat2006” (J. Black and A. White, Eds.), Springer Verlag (2006), 1067–1073.

[24]. C. Cordeiro and M.M. Neves, Bootstrap and exponential smoothing working together in forecasting time series, In ”Proceedings in Computational Statistics (COMPSTAT 2008)” (Paula Brito, Editor), Physica-Verlag (2008), 891–899.

[25]. J. Han, M. Kamber, and J. Pei, Data Mining Concepts and Techniques, 3rd edn. (Morgan Kaufmann, 2012).

[26]. Kent State University Libraries. (2019, March 27). SPSS tutorials: Independent samples t-test. Retrieved March 25, 2019, from http://libguides.library.kent.edu/SPSS/IndependentTTest

[27]. H. He and E. A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21(9) (2009) 1263–1284.

[28]. N. V. Chawla, Data mining for imbalanced datasets: An overview, Data Mining and Knowledge Discovery Handbook (Springer Science & Business Media, 2010), pp. 875–886.
[29]. Seba Susan and Amitesh Kumar,-SMOTE-SSO: Three-step intelligent pruning of majority and minority samples for learning from imbalanced datasets, Applied Soft Computing, 10.1016/j.asoc.2019.02.028, (2019).
[30]. Andrew Estabrooks, Taeho Jo, Nathalie Japkowicz,- A Multiple Resampling Method Learning from Imbalanced Data Sets. Computational Intelligence, Volume 20, Number 1, 2004
[31]. M. Kubat and S. Matwin, Addressing the curse of imbalanced training sets: One-sided selection, in Proc. Fourteenth Int. Conf. Machine Learning, 1997, pp. 179–186.
[32]. L. Pelayo and S. Dick, Evaluating stratification alternatives to improve software defect prediction, IEEE Trans. Reliabil. 61 (2012) 516–525.
[33]. G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, A study of the behavior of several methods for balancing machine learning training data, SIGKDD Explore. 6(1) (2004) 20–29.
[34]. J. Ross, Quinlan.C4.5: Programs for Machine Learning (Morgan Kaufmann, 1993).
[35]. L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[36]. S. S. Keerthi, S. K. Shevade, C. Bhattacharyya and K. R. K. Murthy, Improvements to Platt's SMO algorithm for SVM classifier design, Neural Comput. 13(3) (2001) 637–649.
[37]. J. Platt, Fast training of support vector machines using sequential minimal optimization, Advances in Kernel Methods (Philomel Books, 1999), pp. 185–208.
[38]. G. H. John and P. Langley, Estimating continuous distributions in Bayesian classifiers, Proc. Eleventh Conference on Uncertainty in Artificial Intelligence, 1995, pp. 338–345.
[39]. L. Breiman, Bagging predictors, Mach. Learn. 24(2) (1996) 123–140
[40]. S. Le Cessie and J. C. Van Houwelingen, Ridge estimators in logistic regression, Appl. Stat. 41(1) (1992) 191–201
[41]. Cholmyong Pak, TianTianWang and Xiao Hong Su An Empirical Study on Software Defect Prediction Using Over-Sampling by SMOTE, International Journal of Software Engineering and Knowledge Engineering Vol. 28, No. 6 (2018) 811–830 #. DOI: 10.1142/S0218194018500237.

Mohammad Zubair Khan got the Ph.D. degree in Computer Science and Information Technology from Faculty of Engineering, M.J.P. Rohilkhand University, Bareilly India, and the Master of Technology in Computer Science and Engineering in 2006 from U.P. Technical University, Lucknow, India. He is currently working as Associate Professor in the Department of Computer Science, College of computer science and engineering Taibah University. Past he has worked as head and Associate professor, in the Department of Computer Science and Engineering, Invertis University, Bareilly India. He has published more than 40 journals and conference papers. Mohammad Zubair Khan is a member of Computer Society of India since 2004. His current research interests are data mining, big data, parallel and distributed computing, theory of computations, and computer networks. He has more than 15 years teaching and research experience

Reyadh Alluhaibi received the B.E degree from Taibah University in 2005. He received M.Sc. degree from Tulsa University in 2009. After working as a lecturer (from 2009 to 2012) in the Dept. of Computer Science, Taibah University. He received the PhD degree from Manchester University in 2017. After working as an assistant professor (from 2017) in the Dept. of Computer Science, Taibah University. His research interest includes Machine Learning, Natural Language Processing, Computational Linguistics, Computational Semantics, Knowledge Representation, and Temporal Logics.

Appendix-I

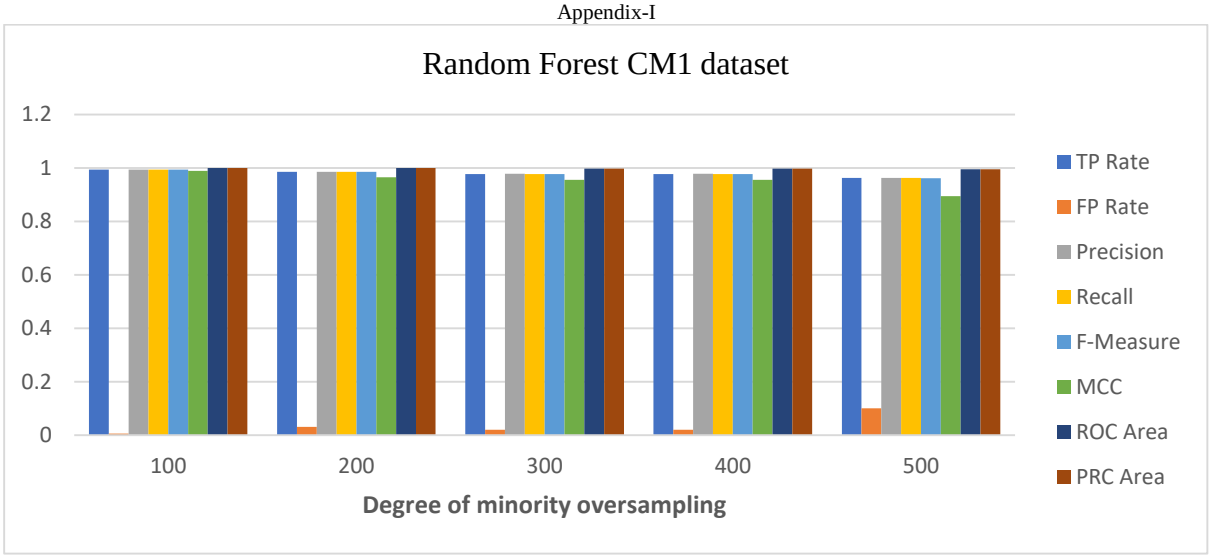

Figure 4: Random Forest results after SMOTE at different Degree label

Table 12: Random Forest results in WEKA After SMOTE for CM1 dataset

Degree of minority oversampling	TP- Rate	FP-Rate	Precision-	-Recall	-F-Measure	-MCC	ROC- Area	PRC-Area
100	0.994	0.006	0.994	0.994	0.994	0.989	1	1
200	0.986	0.031	0.986	0.986	0.986	0.966	1	1
300	0.978	0.02	0.979	0.978	0.978	0.956	0.998	0.998
400	0.978	0.02	0.979	0.978	0.978	0.956	0.998	0.998
500	0.963	0.101	0.963	0.963	0.962	0.895	0.995	0.995

Table 13: Random Forest results in WEKA After Resample for CM1 dataset

Sample Size in %	TP- Rate	FP-Rate	-Precision	-Recall	-F-Measure	-MCC	ROC- Area	PRC-Area
50	0.944	0.49	0.942	0.944	0.935	0.625	0.832	0.936
60	0.973	0.259	0.974	0.973	0.971	0.833	0.813	0.939
70	0.98	0.203	0.981	0.98	0.979	0.873	0.864	0.96
80	0.975	0.261	0.976	0.975	0.973	0.834	0.93	0.975
90	1	1	1	1	1	1	1	1

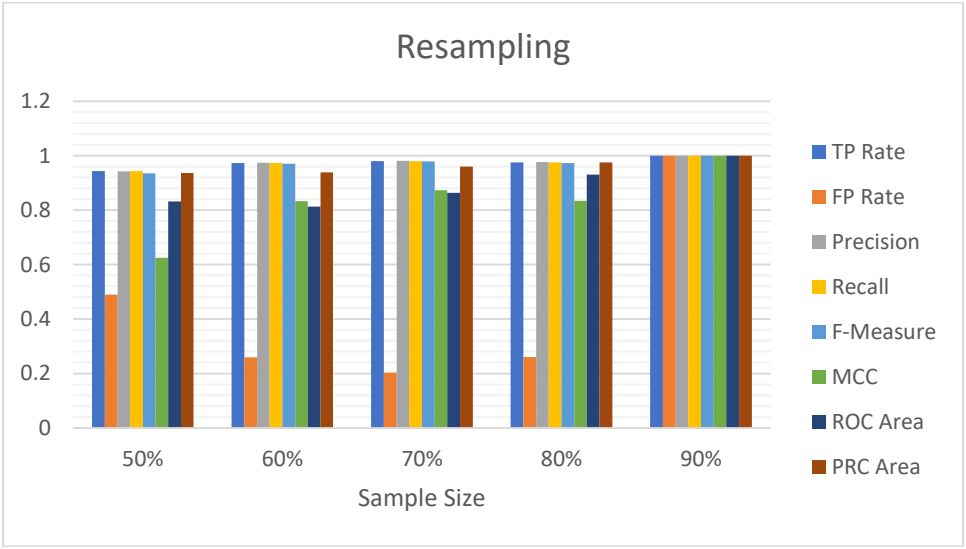

Figure 5: Random Forest results after Resample at different sample size for CM1 dataset