Centroid and Nearest Neighbor based Class Imbalance Reduction with Relevant Feature Selection using Ant Colony Optimization for Software Defect Prediction

Kiran Kumar B.[1], Jayadev Gyani[*], Bhavani Y.[3], Ganesh Reddy P.[4], Nagasai Anjani Kumar T[5]
bkk.it@kitsw.ac.in, je.gyani@mu.edu.sa, yerram.bh@gmail.com, B18IT009@kitsw.ac.in, B18IT037@kitsw.ac.in
[1,3,4,5]Department of Information Technology,
Kakatiya Institute of Technology & Science, Warangal, INDIA
[1]https://orcid.org/0000-0002-5326-6289
[2]Department of CS, College of Computer and Information Sciences,
Majmaah University, AlMajmaah, 11952, Saudi Arabia
*Corresponding author: je.gyani@mu.edu.sa

Abstract
Nowadays software defect prediction (SDP) is most active research going on in software engineering. Early detection of defects lowers the cost of the software and also improves reliability. Machine learningtechniques are widely used to create SDP models based on programming measures. The majority of defect prediction models in the literature have problems with class imbalance and high dimensionality. In this paper, we proposed Centroid and Nearest Neighbor based Class Imbalance Reduction (CNNCIR) technique that considers dataset distribution characteristics to generate symmetry between defective andnon-defective records in imbalanced datasets. The proposed approach is compared with SMOTE (Synthetic Minority Oversampling Technique). The high-dimensionality problem is addressed using Ant Colony Optimization (ACO) technique by choosing relevant features. We used nine different classifiersto analyze six open-source software defect datasets from the PROMISE repository and seven performance measures are used to evaluate them. The results of the proposed CNNCIR method with ACO based feature selection reveals that it outperforms SMOTE in the majority of cases.

Keywords:
Ant Colony Optimization; Class imbalance; Feature selection; Oversampling

1. Introduction

Software Defect Prediction (SDP) is the most significant activity in the testing phase of the software development process. SDP identifies modules that are prone to failure and must be tested thoroughly. Although the SDP is most successful during testing, predicting which modules will fail is not always easy. The number of defects in a software module is proportional to the quality of the software. Asa result, defect prediction is crucial in determining software quality. Though testing is required to re- duce the number of defects, the drawback is it requires more number of human resources in terms of time. In the early stages of testing, accurate identification of defective modules can help reduce overalltesting time. Statistical methods have been phased out in favor of classification models, which divide the modules into two categories: defective and non-defective. After the model has been trained, it is tested on unknown modules and its performance is evaluated using performance measures. As a result of class imbalance problem, the machine learning model gets biased towards the majority class resultingin misleading accuracy. An imbalanced dataset contains fewer samples from one class than from other class. The former class is referred as minority class, and the other class is called as the majority class. The classification algorithm cannot make accurate prediction when the dataset is imbalanced due to lackof enough knowledge about minority class. The balanced datasets with equal number of defective and non-defective records is required to get accurate results. An uneven distribution of values in class la- bel characteristics could be one of the causes of poor classification algorithm accuracy with imbalanced datasets. As a result, the classifier's performance is biased toward the majority class.

In the literature, various sampling approaches are proposed to balance imbalanced datasets. The balanced datasets improve the accuracy of the classification algorithms. Undersampling, Oversampling, and Synthetic data generation are the most common methods for dealing with imbalanced data.

Undersampling reduces certain samples from the majority class in order to balance the data. Undersampling is both informative as well as random. Informative under-sampling selects samples from the majority class based on a pre-specified criterion whereas random undersampling, select samples randomly and discards them. For informative undersampling, the algorithms Easy Ensemble and Balance Cascade are popular. The disadvantage of undersampling method is the loss of information.
Oversampling is the process of reproducing samples from

Manuscript received October 5, 2022
Manuscript revised October 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.10.1

the minority class in order to achieve symmetry among the non-defective and defective records and balance the data. Oversampling is also both informative and random. Informative oversampling is a technique that generates synthetic minorityclass samples based on a set of criteria. Random oversampling balances data by randomly oversampling minority class samples. Oversampling eliminates the problem of data loss. It does, however, have a data duplication problem. The Synthetic Minority Oversampling Technique (SMOTE) is a benchmark technique for dealing with this imbalance problem. In this technique, new synthetic data samples will becreated by randomly selecting a minority class sample and its nearest neighbor samples. In this paper, anew method is proposed for balancing defective and non-defective samples called Centroid and Nearest Neighbor based Class Imbalance Reduction (CNNCIR). To overcome the problem of high dimensionality Ant colony optimization (ACO) technique is used which selects only the relevant features which havea high contribution to improve the classification model's performance. The models are trained using a range of machine learning classifiers, and the performance of our new method was compared to that ofSMOTE.

2. Related work

To solve the problem of class imbalance and high dimensionality in software defect prediction, many approaches have been proposed by the researchers. Re-sampling methods can be used to balance datasets by eliminating data samples from majority class or reproducing the data samples from minority class. In the literature, random undersampling, random oversampling, and variants of these methods are often used. These procedures, on the other hand, run the risk of erasing or duplicating valuable information.

GNV RamanaRao [1] reviewed the literature on developing defect prediction models in software engineering using machine learning and statistical methods. To address the issue of class imbalance, [2] suggested a novel Neighborhood based Under-Sampling (N- US) approach. This project aims to demonstrate the efficacy of this approach in predicting damaged modules with high accuracy. To reduce information loss, the technique N-US under samples the dataset to enhance the visibility of minority data points while limiting the excessive deletion of majority data points.

The Jensen-Shannon divergence was utilized by [3] to automatically determine the most comparable source project to the target project. The project's class imbalance is then improved using a grouped synthetic minority oversampling approach (SMOTE).

Parameters for the probable range of tolerable noise for baseline models are proposed in [4]. They proposed a model for SDP, which outperforms other methods and has the highest noise tolerance. Mohammad AmimulIhsan Aquil [5] assessed a variety of software defect prediction models, including ensemble, clustering, and classification techniques, and found that Ensemble strategies gave consistency in high accuracy prediction.

Comparison of the results with seven recently used bio-inspired algorithms and metaheuristics for feature selection, including Ant Colony Optimization, Multi-Objective Evolutionary Algorithm, using the Genetic Algorithm (GA), Bee Search, Harmony Search, Bat Search, Cuckoo Search,Tabu Search, and Particle Swarm Optimization (PSO) as baseline algorithms is given in [6]. MLMNB-SDP framework is presented in [7], which includes a statistical hypothesis testing method for detecting software metrics with substantial conditional dependencies.

A new approach is suggested [8], in which the error database is retrievedandthen used as input. In the next step, the clustering process clusters the collected input (data). The modified C-Mean Algorithm is used for this. As a result, the data is grouped. The clustered data is subsequently grouped using an efficient classification technique. As a result, we employ a hybrid nervous system. As a result, there are software flaws, and the MCS technique is used to reduce them. A framework for software fault prediction based on CFS and SMOTE techniques is suggested in [9]. This system was built using Decision Tree (DT) and Bayesian Networks (BN) classifiers.

The effect of data sampling on Just-In-Time fault prediction was investigated in [10]. The ratio of class imbalance dataset and the performance of the SDP model have a substantial negative relationship, according to our findings. Second, while most software fault data is not as imbalanced as one might assume, even a little amount of imbalance is enough to damage classical learning performance.

Concise information on the most recent trends and breakthrough in ensemble learning for software defect prediction is presented in [11], as well as a foundation for future improvements. Synthetic minority oversampling technique (SMOTE)to balance the dataset and predicted the severity at the system and component levels is used in [12]. To deal with the two independent projects with heterogeneous feature sets, Boosted Relief Feature Subset Selection (BRFSS) is developed [13].

Datasets are created using the Rapid Keyword Extraction (RAKE) technique to extract the topics or keywords from

developer descriptions of bug reports [14]. A systematic literature evaluation is conducted in [15], which identifies most recent research trends in field of SDP by reviewing papers published between 2016 and 2019.

Stable SMOTE-based oversampling ways to reduce the unpredictability in each stage of SMOTE-based oversampling techniques is presented in [16]. In this approach, defective instances are chosen and K-neighbor instances are selected using distance measure and uniformly distributed interpolation.

EMWS is created in [17], using the whale optimization algorithm (WOA), a meta-heuristic search-based feature selection algorithm, which successfully identify less number of and closely related representative features. Second, they built WSHCKE, a unified defect prediction predictor that uses CNN, a kernel extreme learning machine (KELM), a hybrid deep neural network method, for integration of the features that are selected into CNN abstract deep semantic features. This method enhances prediction performance by leveraging KELM's strong classification capacity. A filter-based feature selection research and synthesis using a variety of search methods and algorithms is given in [18]. In addition, five filter-based feature selection strategies are investigated using five different classifiers on datasets acquired from NASA's repository.

A high-dimensional sampling space to generate new minority class samples is used in [19]. They used scaling constraint and random over-sampling scope constraint by differentiating m-class samples to synthetically generate new samples.

The state of the art in software defects using Machine Learning algorithms is presented in [20]. In [21], the authors discovered that prior to applying deep learning for defect prediction, oversampling is both effective and necessary.

The study of [22] tries to determine the best classifier for classification of faults. In their study, they considered five categories of classification methods from which they used twenty one classifiers on five open source applications. MATLAB's classification LearnerApp was used to test multiple classification models. Over KNN, Regression, and Tree, the work proposes the usage of Ensemble and SVM algorithms. Among the twenty-one classifiers, ensemble methods and SVM performed well compared to other classifiers. A feature selection method based on clustering technique named Relief-based clustering (RFC), was proposed [23]. Two unique ways for learning from imbalanced data, one for severely imbalanced data and the other for moderately imbalanced data, with the goal of improving predicting accuracy over the minority class is introduced in [24].

The work of [25], attempts to give a comparative study of several techniques for dealing with class imbalances. They conducted a comprehensive experimental investigation which compares the efficiency of various class imbalance procedures. They evaluated the classification algorithms using various performance metrics like AUC, Precision, K-fold Cross-validation, recall and execution time. In this work, they discovered that for ensemble classification models like XGBoost, AdaBoost and Random Forest with oversampling followed by undersampling performs well.

A comprehensive overview of machine learning and the problem of unbalanced data is provided in [26]. In addition, they devised a method for predicting heart disease, cervical cancer, and chronic kidney disease which combines a softmax regression algorithm with an upgraded sparse auto encoder. To tackle the problem of class imbalance, [27] suggested a solution utilizing a Machine Learning approach for Prediction of Software Defects. A novel resampling approach called kernel density estimation-based variation sampling (DVS) is proposed by generating new minority defective samples [28]. More than half of the Eigen values in each new sample are directly inherited from the current faulty samples, but some will be altered.

OS-CCD, a new oversampling method suggested by [29], is a new notion called classification contribution degree. OS-CCD eliminates oversampling from noisy points while following the spatial distribution characteristics of original data on the class boundary. Soft computing based machine learning techniques to develop an efficient approach is used in [30]. This approach helps to predict, optimize the features and efficiently learn the features.

3. Methodology

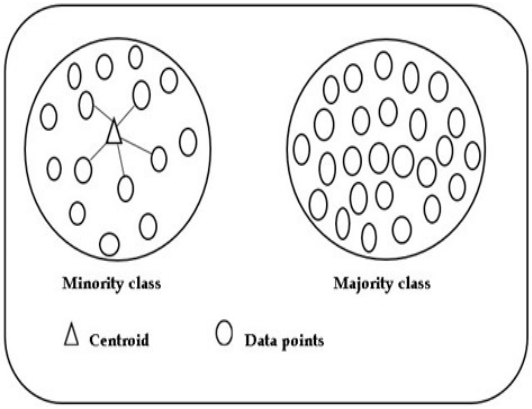
Our proposed method, Centroid and Nearest Neighbor based Class Imbalance Reduction (CNNCIR), generates new samples depending on the centroid calculated using all minority class samples. The general methodology of our approach is given below.

Let $D_i = s_1, s_2, s_3,..., s_n$, where s_i $(1 \le i \le n)$ is the i^{th} sample of the i^{th} module in the imbalanced dataset. Each s_i having 'm' features, where each feature represents one software metric anda class label attribute. The class label indicates whether or not the corresponding module is defective. A non-defective module has a class label attribute value of zero, whereas a defective module has a value greater than zero. We converted the values greater than zero to one

because as per binary classification algorithms, class label attribute should contain either one or zero. Figure 1 depicts the proposed framework.

3.1. Centroid and Nearest Neighbor based Class Imbalance Reduction Algorithm (CNNCIR)

The imbalanced dataset Di is partitioned into two groups depending on the value of class label. The minority class is the group with the fewest samples, as shown in Figure 2, whereas the majority class is the group with the more number of samples. Let the number of nearest neighbors be 'k' (5 in our study) and 'n' be the difference of samples between two classes. The centroid (C) is calculated from the minority class samples, and K-nearest neighbors of the centroid are identified by using Euclidean distance measure. 'n/k' is the number of synthetic samples to be generated between the centroid and each nearest neighbor. After that, the associated scalar differences between the centroid (C) and its nearest neighbors are computed. The nearest sample is added to the



scalar multiplication of a random number whose value is between 0 and 1 and its associated difference to obtain a new sample. This process is carried out for the remaining nearest neighbors also. To balance the minority and majority classes, the created synthetic samples are appended to the minority class.

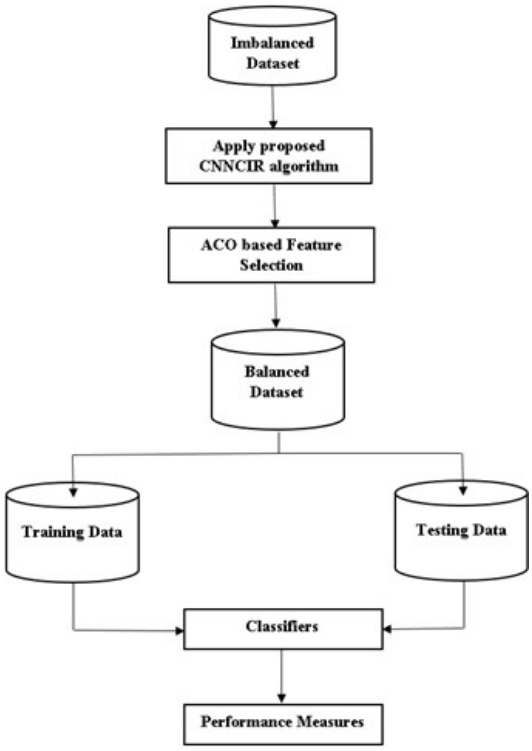


Fig. 1. CNNCIR Framework

Fig. 2. Demonstration of CNNCIR with minority and majority classes

Algorithm1: Centroid and Nearest Neighbor based Class Imbalance Reduction Algorithm (CNNCIR)

Input: D_i be the Imbalanced Dataset, $A_1, A_2, A_3, \ldots, A_m$ be the features (software metrics) with a classlabel and s_1 , s_2 , s_3, . . . , s_n are samples of dataset
Output: Balanced Dataset (BD)
Step-1: Divide the imbalanced dataset D_i into two partitions based on value of class label which representnon-defective and defective classes
Step-2: Minority class is represented as D_{mi}, which has less number of samples related to a class
Step-3: Majority class is represented as D_{mj}, which has more number of samples related to a class
Step-4: Centroid (C) is calculated for D_{mi} using C =average (A_1), average (A_2), average (A_3), . . . , average (A_m)
Step-5: For every sample s_i in D_{mi}
Step-5.1: Compute the distance between s_i and C using Euclidian distance
Step-6: Arrange the samples in non-decreasing order of their distance
Step-7: Select first 'k' samples ('k' nearest neighbors) denoted as $nn_0, nn_1, nn_2, \ldots, nn_{k-1}$
Step-8: Calculate the difference between those 'k'

samples and centroid(C) and let them be d_0, d_1, d_2,. . . , d_{k-1}

Step-9: Generate 'n/k' random numbers $rand_0$, $rand_1$, $rand_2$, . . . $rand_{(n/k)-1}$, between 0 and 1, where n/k = (|D_{mj}|-|D_{mi}|)/k such that defective and non-defective samples get balanced

Step-9.1: For every random number $rand_j$

Step-9.1.1: A new sample nn_p + $rand_j$ * d_r is generated

Step-9.1.2: Add newly generated sample to D_{mi}

3.2 ACO based Feature Selection Algorithm

The ant colony optimization algorithm is modeled after how ants look for food. Different ants take different paths from their colony to the food source and then return to their nest in the process of lookingfor food by leaving a chemical (pheromone) on the path. In comparison to the ants which travelled the longest path, the ants which walked the shortest path may be able to reach the nest more quickly. As timepasses, the chemical that has been deposited on the path may evaporate, resulting in more pheromone being deposited on the shortest path than on the longest path. In this way the behavior of ants can be used to find the optimal or shortest path. In this paper, we simulated ant behavior for the selection of relevant subset of features i.e., the ones which contributed the most to improve the model's performance. The ACO based feature selection algorithm is given below.

Dataset	***Defective Class***	***Non-Defective Class***	***Minority Class%***
CM1	42	285	12.84
KC2	107	415	20.49
KC3	36	158	18.55
MC2	44	81	35.2
PC1	61	644	8.65
PC3	134	943	12.44

1. Algorithm 2 Feature Selection using ACO

Input: D be the Dataset, $A_1, A_2, A_3, \ldots, A_m$ be the features (software metrics)
ph: Array[n] initialized to zero, which contains pheromone values
ρ: Denotes evaporation ratio of Pheromone. (Considered the value as 0.25)
Output: *m* features which are relevant and lies between $\frac{n}{2}$ and *n*

Step-1: Randomly 'm' number of features are selected from D consisting 'n' features. Store these features in another dataset DS.
Step-2: Split DS into training and test set.
Step-3: Calculate the accuracy (AC_p) using logistic regression classifier on DS.
Step-4: Pheromone values of ph related to 'm' features are incremented by 1.
Step-5: Perform the step-5 for *MaxIteration* (User input) times

Step-5.1: Once again, randomly 'm' features are selected from D and Store these features in datasetDS.
Step-5.2: Split DS into training and test set.
Step-5.3: Calculate the accuracy (AC_c) using logistic regression classifier on DS. (AC_c represents current accuracy and AC_p represents previous accuracy)
Step-5.4: if ($AC_c \geq AC_p$) then Pheromone value of *ph* related to 'm' features are incremented by 1 and remaining n-m features decremented by ρ else Pheromone values of *ph* related to 'm' features are decremented by ρ

Step-6: *ph* array is sorted in descending order.
Step-7: Select first 'm' pheromone values from *ph* and retrieve corresponding features from dataset D.

4. Experimentation and Results

In our proposed method, we chose six open-source software defect datasets from PROMISE repositorythat were related to defect prediction. Table 1 shows the list of datasets as well as the percentages of class imbalance. All of the data sets have higher numbered features and include a variety of features.

Table 1. Experimentation datasets

4.1 Performance Measures

There are various performance evaluation metrics used for classification models are given in eq. 1 to 6.

1. Accuracy: It calculates the percentage of true findings among all cases.

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

2. Precision: It calculates percentage of expected positives compared to all positives. The standard deviation of a set of variables is used to calculate it.

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

3. Sensitivity or Recall: It's a metric for determining how many positives are accurately categorized. Itrefers to how close a measurement is to a given value.

$$Sencitivity = Recall = \frac{TP}{TP + FN} \quad (3)$$

4. F-Measure: It is a harmonic mean or can also be defined as average of precision and recall.

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (4)$$

5. Specificity: It is a measure to correctly identify true negatives.

$$Specificity = \frac{TN}{TN + FP} \quad (5)$$

6. Geometric mean: It measures the performance of classifier against minority and majority classes.

$$Geometric\ Mean = \sqrt{Precision * Recall} \quad (6)$$

7. AUC (Area under ROC Curve): An evaluation metric for binary classification problems is the "area under the receiver operator characteristic (ROC) curve". This AUC curve summarizes the ROC curve and is used to define a classifier's ability to distinguish between positive and negative classifications. The AUC curve indicates how well a classifier can distinguish between classes. The higher the AUC curve, the better the classifier's ability.

Comparison of seven performance metrics between proposed CNNCIR technique and SMOTE technique using nine classifiers with Mean ±SD is listed in Table 2.

Table 2. Comparison of seven performance metrics between SMOTE and proposed CNNCIR techniques using nine classifiers with Mean±SD

Classifier/ Measures	***ACCURACY***	
	SMOTE	***CNNCIR***
AdaBoost	0.85±0.04	**0.86±0.10**
Decision Tree	0.82±0.10	**0.86±0.08**
Extra Tree	0.81±0.10	**0.86±0.06**
Gradient Boost	0.53±0.07	0.50±0.02
KNN	0.75±0.08	**0.85±0.10**
Logistic Regression	0.78±0.03	**0.87±0.05**
Naïve Bayes	0.63±0.08	**0.81±0.11**
Random Forest	0.88±0.10	**0.90±0.05**
SVM	0.65±0.10	**0.85±0.06**

Classifier/ Measures	***PRECISION***	
	SMOTE	***CNNCIR***
AdaBoost	0.82±0.05	**0.87±0.13**
Decision Tree	0.80±0.12	**0.87±0.07**
Extra Tree	0.77±0.12	**0.87±0.06**
Gradient Boost	0.53±0.06	0.50±0.02
KNN	0.71±0.09	**0.85±0.16**
Logistic Regression	0.81±0.08	**0.85±0.07**
Naïve Bayes	0.73±0.15	**0.81±0.13**
Random Forest	0.86±0.11	**0.92±0.07**
SVM	0.73±0.11	**0.85±0.08**

Classifier/ Measures	RECALL	
	SMOTE	CNNCIR
AdaBoost	0.87±0.08	0.86±0.05
Decision Tree	0.84±0.10	**0.85±0.12**
Extra Tree	0.89±0.05	0.85±0.07
Gradient Boost	0.56±0.09	0.51±0.06
KNN	0.81±0.14	**0.85±0.06**
Logistic Regression	0.75±0.07	**0.89±0.04**
Naïve Bayes	0.51±0.22	**0.85±0.14**
Random Forest	0.90±0.09	0.88±0.05
SVM	0.44±0.25	**0.85±0.09**

Classifier/ Measures	F-Measure	
	SMOTE	CNNCIR
AdaBoost	0.84±0.05	**0.86±0.09**
Decision Tree	0.82±0.12	**0.86±0.09**
Extra Tree	0.82±0.09	**0.86±0.06**
Gradient Boost	0.54±0.07	0.50±0.04
KNN	0.75±0.10	**0.84±0.11**
Logistic Regression	0.77±0.04	**0.82±0.12**
Naïve Bayes	0.56±0.06	**0.81±0.09**
Random Forest	0.88±0.10	**0.90±0.05**
SVM	0.53±0.18	**0.85±0.07**

Classifier/ Measures	Specificity	
	SMOTE	CNNCIR
AdaBoost	**0.82±0.03**	**0.86±0.16**
Decision Tree	**0.80±0.10**	**0.88±0.05**
Extra Tree	**0.74±0.15**	**0.87±0.06**
Gradient Boost	**0.51±0.07**	**0.50±0.05**
KNN	**0.69±0.11**	**0.86±0.14**
Logistic Regression	**0.81±0.08**	**0.86±0.07**
Naïve Bayes	**0.74±0.36**	**0.77±0.22**
Random Forest	**0.87±0.10**	**0.92±0.06**
SVM	0.85±0.09	0.84±0.08

Classifier/ Measures	Geometric Mean	
	SMOTE	CNNCIR
AdaBoost	**0.85±0.04**	**0.86±0.11**
Decision Tree	**0.82±0.10**	**0.86±0.08**
Extra Tree	**0.81±0.10**	**0.86±0.06**
Gradient Boost	**0.53±0.07**	**0.50±0.02**
KNN	**0.74±0.08**	**0.85±0.10**
Logistic Regression	**0.81±0.09**	**0.87±0.04**
Naïve Bayes	**0.52±0.21**	**0.80±0.12**
Random Forest	**0.88±0.09**	**0.90±0.05**
SVM	0.59±0.14	**0.85±0.06**

Classifier/ Measures	AUC	
	SMOTE	CNNCIR
AdaBoost	**0.85±0.05**	**0.86±0.10**
Decision Tree	**0.82±0.10**	**0.86±0.08**
Extra Tree	**0.81±0.09**	**0.86±0.06**
Gradient Boost	**0.54±0.07**	0.50±0.02
KNN	**0.75±0.08**	**0.85±0.10**
Logistic Regression	**0.78±0.04**	**0.87±0.05**
Naïve Bayes	**0.63±0.08**	**0.81±0.11**
Random Forest	**0.88±0.09**	**0.90±0.05**
SVM	0.65±0.10	**0.85±0.06**

The results reveal that the proposed CNNCIR technique outperforms the SMOTE algorithm for all seven performance measures listed in Table 3. CNNCIR outperforms the SMOTE algorithm when employed with the nine classification techniques mentioned above. Table 3 shows a comparison of the improvement of the CNNCIR algorithm vs the SMOTE algorithm utilizing 9 classifiers and 7 performance indicators. In Table 3 each row represent the number of data sets in CNNCIR that outperforms SMOTE in terms of performance. As seen in the first row, CNNCIR outperforms SMOTE in 5 out of 6 datasets when using the AdaBoost (AB) classifier. For the Decision Tree, 5 out of 6 datasets indicate performance improvement with CNNCIR over SMOTE, while 1 dataset shows equivalent performance with CNNCIR over SMOTE. In five out of six datasets, CNNCIR outperforms the SMOTE method employing ExtraTree. For the GradientBoost, 2 out of 6 datasets show performance improvement with CNNCIR over SMOTE method, while 1 dataset shows the same performance with CNNCIR over SMOTE algorithm. For the K-nearest neighbor, CNNCIR outperforms the SMOTE method in six out of six datasets. For

the Logistic Regression, CNNCIR improves performance in 6 out of 6 datasets. For the Nave Bayes, 5 out of 6 datasets indicate performance improvement with CNNCIR over SMOTE, while 1 dataset shows the same performance with CNNCIR over SMOTE. For the Random Forest, two out of two datasets show performance improvement with CNNCIR over SMOTE algorithm, whereas two datasets show the same performance with CNNCIR over SMOTE algorithm. In terms of SVM accuracy, CNNCIR outperforms the SMOTE method in six out of six datasets. In the same way, the second row compares precision, the third row compares recall, the fourth row compares F-Measure, the fifth row compares specificity, the sixth row compares geometric mean, and the seventh row compares AUC.

Table 4 shows how the CNNCIR method outperforms the SMOTE algorithm when utilizing various classifiers. Table 4 shows the output enhancement of CNNCIR over the SMOTE method for each classifier. The accuracy of KNN, LR, and SVM output is higher. The precision of DT and SVM output is higher. In recall, LR and NB output is higher. In F-Measure, DT, NB, and SVM produce higher results. The specificity of DT, ET, and KNN is high. In the Geometric mean, KNN, NB, and SVM produced greater results, while in the AUC, KNN, LR, NB, and SVM produced higher results.

Table 3. Comparison of SMOTE with CNNCIR considering nine classifiers and seven performance metrics for six datasets. (X represents number of datasets exhibiting enhanced performance against SMOTE, Y represents number of datasets exhibiting same performance against SMOTE)

Classifier/ Measures	AB		DT		ET		GB		KNN		LR		NB		RF		SVM	
	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
ACCURACY	5	0	5	1	5	0	2	1	6	0	6	0	5	1	2	2	6	0
PRECISION	5	0	6	0	5	0	2	0	5	0	3	1	5	1	5	1	6	0
RECALL	2	0	3	1	1	0	1	0	5	0	6	0	6	0	1	0	5	1
F-MEASURE	5	0	6	0	4	0	1	0	5	0	5	0	6	0	2	1	6	0
SPECIFICITY	5	0	6	0	6	0	2	1	6	0	5	0	2	0	5	1	3	0
GM	5	0	5	1	5	0	2	1	6	0	5	0	6	0	2	2	6	0
AUC	5	0	5	1	5	0	2	1	6	0	6	0	6	0	2	3	6	0

Table 4. Performance improvement (in percentage) with CNNCIR vs SMOTE using nine different classifier

Classifier/ Measures	*ACCURACY*	*PRECISION*	*RECALL*	*F-MEASURE*	*SPECIFICITY*	*GM*	*AUC*
AB	83.3	83.3	33.3	83.3	83.3	83.3	83.3
DT	83.3	**100**	50	**100**	**100**	83.3	83.3
ET	83.3	83.3	16.6	66.6	**100**	83.3	83.3
GB	33.3	33.3	16.6	16.6	33.3	33.3	33.3
KNN	**100**	83.3	83.3	83.3	**100**	**100**	**100**
LR	**100**	50	**100**	83.3	83.3	83.3	**100**
NB	83.3	83.3	**100**	**100**	33.3	**100**	**100**
RF	33.3	83.3	16.6	33.3	83.3	33.3	33.3
SVM	**100**	**100**	83.3	**100**	50	**100**	**100**

5. Conclusion

In this research, a novel approach is proposed for handling class imbalance in software defect prediction called Centroid and Nearest Neighbor based Class Imbalance Reduction (CNNCIR), which takes into account the dataset's distribution aspects. To choose relevant characteristics from the datasets, ant colony optimization based feature selection is utilized. To produce synthetic data, the suggested method employs a centroid and nearest neighbor methodology. Several experiments using the suggested approach are carried out on six openly accessible public datasets. The obtained results using the CNNCIR approach compared with SMOTE results. In terms of seven standard prediction measures, our experiment findings show that the suggested approach CNNCIR outperforms SMOTE. When used nine machine algorithms, CNNCIR outperforms SMOTE. The presented approach can be expanded to provide defect prediction across several projects (CPDP).

Acknowledgments
The authors would like to thank Deanship of Scientific Research at Majmaah University for supporting this work under Project Number No. xxxx. The author is also thankful to the anonymous reviewers for their useful comments.

2. References

[1] RamanaRao, GNV., Balaram, VVSSS. & Vishnuvardhan, B. (2018) Software defect prediction: past present and future. International Journal of Computer Engineering & Technology (IJCET), 9(5):116–131.

[2] Somya, G. (2021) Handling class-imbalance with KNN (Neighborhood) under-sampling for software defect prediction. Artificial Intelligence Review: 1-42

[3] Shang, Zheng., Jinjing, Gai., Hualong, Yu., Haitao Zou. & Shang, Gao. (2021) Training data selection for imbalanced cross-project defect prediction. Computers & Electrical Engineering, 94

[4] Sushant Kumar, Pandey. & Anil Kumar, Tripathi. (2021) An empirical study toward dealing with noise and class imbalance issues in software defect prediction. Soft Computing, 25: 13465–13492

[5] Mohammad AmimulIhsan, Aquil. & Wan Hussain, Wan Ishak. (2020) Predicting software defects using machine learning techniques. International Journal of Advanced Trends in Computer Science and Engineering, 9(4): 6609 – 6616

[6] Asad, Ali. & Gravino, Carmine. (2021) Software fault prediction using bio-inspired algorithms to select the features to be employed: an empirical study. 29thInternational Conference on Information Systems Development

[7] Harzevili., Shiri, Nima. & Alizadeh, Sasan H. (2021) Analysis and modeling conditional mutual de- pendency of metrics in software defect prediction using latent variables. Neuro Computing 460:309-330

[8] SrinivasaKumar, C., RangaSwamy, Sirisati. & Srinivasulu, Thonukunuri. (2021) Software defect prediction using optimized cuckoo search based nature-inspired technique. Smart Computing Techniques and Applications. Springer: 183-192.

[9] Abdullateef, Balogun., Fatimah B Lafenwa, Balogun., Hammed, Mojeed. & Fatima Enehezei Hamza, Usman. (2020) Data sampling-based feature selection framework for software defect pre- diction. The International Conference on Emerging Applications and Technologies for Industry 4.0. Springer

[10] Haitao, Xu., Ruifeng, Duan., Shengsong, Yang. & Lei, Guo. (2021) An empirical study on data sampling for just-in-time defect prediction. International Conference on Artificial Intelligence and Security. Springer.

[11] Faseeha, Matloob., Taher, M, Ghazal., Nasser, Taleb., Shabib, Aftab., Munir, Ahmad. & Muham- mad, Adnan Khan. (2021) Software defect prediction using ensemble learning: a systematic literature review. IEEE Access, 9: 98754-98771

[12] Shubhra, Goyal Jindal. & Arvinder, Kaur. (2019) Bug severity prediction using class imbalance problem. International Journal of Recent Technology and Engineering (IJRTE), 8(4): 2687-2695

[13] Kalaivani, N. & Beena, R. (2020) Boosted relief feature subset selection and heterogeneous cross project defect prediction using firefly particle swarm optimization. International Journal of Recent Technology and Engineering (IJRTE), 8(5): 2605-2613

[14] Jayalath, Ekanayake. (2021) Bug severity prediction using keywords in imbalanced learning environment. International Journal of Information Technology and Computer Science, 3:53-60

[15] Faseeha, Matloob., Shabib, Aftab., Munir, Ahmad., Adnan Khan, Muhammad., Fatima, Areej., Iqbal, Muhammad., Alruwaili, Wesam Mohsen. & Elmitwally, NouhSabri. (2021) Software defect prediction using supervised machine learning techniques: a systematic literature review. Intelligent Automation & Soft Computing, 29(2): 403-421

[16] Shuo, Feng., Jacky, Keung., Xiao, Yu., Yan, Xiao. & Miao, Zhang. (2021) Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Information and Software Technology, 139

[17] Kun, Zhu., Shi, Ying., Nana, Zhang. & DandanZhun. (2021) Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network,

Journal of Systems and Software, 180

[18] Ha, Th Minh Phuong., Le Thi My Hanh. & Nguyen Thanh, Binh. (2021) A Comparative Analysis of Filter-Based Feature Selection Methods, Journal of Research and Development on Information and Communication Technology, 1(6):1-7

[19] Guo, Shikai., Dong, Jian., Li, Hui. & Wang, Jiahui. (2021) Software defect prediction with imbalanced distribution by radius-synthetic minority over-sampling technique. Journal of Software: Evolution and Process 33(1)

[20] Ramesh, Ponnala. & Reddy, CRK. (2021) Software defect prediction using machine learning algorithms: current state of the art. Solid State Technology. 64(2)

[21] Rahul, Yedida. & Menzies, Tim. (2021) On the value of oversampling for deep learning in software defect prediction. IEEE Transactions on Software Engineering. 2021:1-11

[22] Inderpreet, Kaur. & Arvinder, Kaur. (2021) Comparative analysis of software fault prediction using various categories of classifiers. International Journal of System Assurance Engineering and Management 12(1):520-535

[23] Xu, Xiaolong., Chen, Wen. & Wang, Xinheng. (2021) RFC: A feature selection algorithm for software defect prediction. Journal of Systems Engineering and Electronics. 32(2): 389-398

[24] Zheng, Jianming., Wang, Xingqi., Wei, Dan., Chen, Bin. & Shao, Yanli. (2021) A novel imbalanced ensemble learning in software defect predication. IEEE Access 9:86855-86868

[25] Amit,Singh., Ranjeet Kumar, Ranjan. & Abhishek, Tiwari. (2021) Credit card fraud detection under extreme imbalanced data: a comparative study of data-level algorithms. Journal of Experimental & Theoretical Artificial Intelligence. 1-28

[26] Ebiaredoh-Mienye, Sarah., Esenogho, Ebenezer. & Swart, Theo. (2021) Improved machine learning methods for classification of imbalanced data

[27] Satya Srinivas, Maddipati. & Srinivas, Malladi. (2021) Machine learning approach for classification from imbalanced software defect data using PCA & CSANFIS. Materials Today: Proceedings

[28] ZYuqing, Zhang., Xuefeng, Yan. & Arif Ali, Khan. (2020) A kernel density estimation-based variation sampling for class imbalance in defect prediction. IEEE International Conference on Big Data and Cloud Computing

[29] Jiang, Z., Pan, T., Zhang, C. & Yang, J. (2021) A new oversampling method based on the classification contribution degree. Symmetry 13(2):1-13

[30] Mahesh Kumar, Thota., Francis H, Shajin. & Rajesh, P. (2020) Survey on software defect prediction techniques. International Journal of Applied Science and Engineering 17(4): 331-344

B. Kiran Kumar received the Ph. D in Computer Science and Engineering from JNTU, Hyderabad in 2021. Working as an associate professor in the Dept. of Information Technology, Kakatiya Institute of Technology & Science, Warangal. His research interest includes Data Mining, Machine learning, Software engineering.

JAYADEV GYANI is working as Assistant Professor in the Department of Computer Science at CCIS, Majmaah University, Kingdom of Saudi Arabia. He received his PhD in Computer Science from University of Hyderabad, India in 2009. His teaching experience is 25 years. His research interests include software engineering, big data analytics, distributed computing, machine learning algorithms, and their applications. He is a member of ACM and senior member of IEEE.

Y. Bhavani received the Ph. D in Computer Science and Engineering from JNTU, Hyderabad in 2020. Working as an associate professor in the Dept. of Information Technology, Kakatiya Institute of Technology & Science, Warangal. Her research interests includes Network Security and Cryptography, Machine Learning.

Ganesh Reddy P received his Bachelor degree from Dept. of Information Technology, Kakatiya Institute of Technology & Science, Warangal. His research interests includes Data Mining and Machine Learning.

Nagasai Anjani Kumar received his Bachelor degree from Dept. of Information Technology, Kakatiya Institute of Technology & Science, Warangal. His research interests includes Data Mining, Ne twork Security and Cryptography and Machine Learning.