**Educational Administration: Theory and Practice**
2024, 30(4), 9863-9872
ISSN: 2148-2403
https://kuey.net/

**Research Article**

# Optimized Ensemble Techniques For Precise Software Error Detection

Raghvendra Omprakash Singh[1], Dr. Sunil Gupta[2]

[1*]Research Scholar, Department Of Computer And Systems Sciences, Jaipur National University, Jaipur, India.
Email Id: Raghvendrasingh@Live.In
[2*]Head Of The Department And Guide, Department Of Computer And Systems Sciences, Jaipur National University, Jaipur, India. Email Id: Scss_Jnu@Jnujaipur.Ac.In

**Citation:** Raghvendra Omprakash Singh, Dr. Sunil Gupta (2024), Optimized Ensemble Techniques For Precise Software Error Detection, Educational Administration: Theory and Practice, 30(4), 9863-9872
Doi: 10.53555/kuey.v30i4.5891

**ARTICLE INFO**

**ABSTRACT**

Recent advancements in software engineering technology have led to an increase in data volume. A multitude of software quality evaluations are being created to evaluate built software in order to handle the exponential expansion of data. One of the software engineering models' best features is the evaluation of software flaws. Effective software defect management starts with correctly classifying software faults. A unique approach to forecast software fault classes is presented in this study. The prediction systems are performed in this case using an ensemble learning technique. First, information about defects is gathered from the open-source database. A more basic exploratory data analysis is carried out to determine the number of software flaws that are present and absent. The gathered dataset is preprocessed using the SMOTE algorithm. The minority classes' involvement in the training procedure has decreased due to the existence of processed data. The oversampling data is in line with the generation of synthetic data in order to take advantage of the minority classes as well as the existence of data uncertainty concerns. The synthetic instances that are developed in accordance with the real-time data exhibit characteristics of feature space rather than data space. The closest data points line segments merge with each class minority. After accurately defining the majority and minority classes, the oversampled data are sorted. The ensemble of classifiers, which includes Bagging, Adaboost, and K-Nearest Neighbors (k-NN), is then given the scaled features. To categorize the software flaws, these 3 classifiers use feature-scaled data as input. The effectiveness of the suggested ensemble classifiers in terms of sensitivity, accuracy, precision, and specificity has been demonstrated by simulation of the proposed framework. the comparison of the analysis conducted before and after SMOTE use. The obtained findings make it abundantly evident that using feature-scaled data in the ensemble classifiers produced superior results.

**Keywords:** Software defects; Software engineering; feature scaling process; SMOTE technique; Oversampling data; Ensemble classifiers.

## 1. Introduction:

One of the leading disciplines that works with the structure of created and evolving software is software engineering (SE). Software quality assurance, or QA, plays a vital role in the software development sector. As per the Wang et al. (2016), QA is a type of behavioral activity carried out during the software project's execution. Software Testing (ST) is a dynamic field of study that is closely related to software product quality. In general, testing the test cases requires more time and effort. Even if there are many types of software testing accessible, certain defects and mistakes still need to be fixed because of a lack of effort as well as time (Yang et al, 2015). As a result, the software testing industry needs to adapt in order to save businesses' time and money. Early software bug analysis and prediction is a skill that the developer (or) tester should possess. In nature, software faults are unavoidable. Complex software applications have emerged as a result of recent

breakthroughs in the software industry. It is required that there be no flaws in the produced program (Huo X et al, 2016). Software bugs are imperfections in programming functionality. It significantly lowers the overall software applications' performance. Every program has an ongoing Bug Tracking System (BTS) that collects, arranges, and keeps track of problems and their reports. Software entities such as users, developers, and testers utilize the BTS to create test reports. A problem is fixed through a number of stages after it is discovered. In this research work, a lot of attention is being paid to the analysis and prediction of the flaws utilizing ensemble classifiers. A thorough examination of the root causes of software defects enables the creation of predictive classifiers (CJ Clemente, et al, 2018). To forecast the number of defects along with the steps involved in fixing them, various prediction models based on the bug indicators have been created. Prediction models help developers by giving them access to software testers. To raise the caliber of the software products, a number of actions are done to assist the bug prediction models. It stops subsequent bugs from being as severe (Li et al., 2017). A significant area of research that lowers expenses associated with software administration and quality determination is called SDP (Software Defect Prediction). Software faults are defined as the software's persistent failures over time. The stakeholders and software developers provide the bugs in the software. A SDP system's job is to anticipate errors by raising the quality of software while cutting expenses. Using efficient learning techniques, a number of academics are attempting to enhance the static features that determine the software quality measures. It forecasts the software quality systems and is dependent on program properties like Line of Code (LoC). Different versions of prediction techniques are available for various software module contexts and testing scenarios in the context of software quality systems.

### 1.1 Contributions of the Study

This paper's salient features are:

a) To address the minority classes as well as data ambiguity concerns, synthetic and historical data are used.

b) By generating false information, the SMOTE approach has made use of minority classes to determine the closest data points.

c) To manage oversampled data, the data space is handled as feature space.

d) By employing feature-scaled data, an ensemble of classifiers, including K-NN, Bagging, and Adaboost approaches, has increased the prediction quality.

e) Compared to traditional classifiers, the prediction accuracy is better.

### 1.2 Organization of the Paper

The structure of the paper has been mentioned below:
The literature study which examines the current methods is presented in Section II.
The suggested framework, which outlines the stages, is presented in Section III.
The experimental outcomes as well as the discussion, which details the performance attained with the suggested strategies, are presented in Section IV.
The study's conclusions are represented in Section V as the Conclusion.

## 2. Related Work:

A review of previous research is conducted with consideration given to goals, methods, benefits, and drawbacks in this area. The deep learning application in bug prediction systems has been investigated by (Rudolf Ferenc et al., 2020). When tested on Java class flaws, it produced an F-measure of 55.27% higher than that of traditional prediction algorithms. Replication problems arise when the bugs' dimensions are not taken into account for analytical reasons. In 2020, Cheng Zhou and colleagues presented the named entity recognition prediction model that primarily emphasized the information extraction procedure. The fine-grained factual information acquired led to an improvement in bug prediction accessibility under neural networks. The prediction error of 91.1% has been reduced thanks to the idea of conditional random fields. (Sushant Kumar Pandey et al., 2019) has provided ensemble learning methods and deep features. It was created to improve the NASA dataset testing on the Promise repository. The auto-encoders which improve the efficiency of the bug predictions by using the greatest correlation coefficients thoroughly describe and evaluate the bug attributes. Bug prediction was investigated by (Sushant Kumar Pandey et al, 2020) on the subsequent count of the software systems. Each bug's information has been included in the deep feature training layers, preventing overfitting and class imbalance problems. Seven datasets are examined, resulting in a decrease in the MSE value from 0.71 - 4.715 and the MAE from 0.22 -1.679.
(Yan Xiao et al., 2018) used augmented convolutional neural networks to increase the bugs' location. Here, word embedding models are used to determine the frequency of the faults and then correct them. The technology has slashed the amount of time needed to generate test cases and address bugs. The basic idea is to categorize bugs by the developer using a set of preset bug types and comparative expertise. A Word2vec has been submitted by (Guo et al., 2020) to the implementation of CNN for the bug summary procedure. To improve the forecast accuracy of the bug assignment procedure, several researchers have looked at the parts, priority, severity, and products of the issue. To compute the bug reports similarity, (Zhao et al., 2019) suggest using a topic model-based LDA (“Latent Dirichlet Allocation”) approach. The many attribute data are used to

address some of the discrepancies found in bug reports. The LDA that determined the primary subjects oversees the data (Xia et al, 2017). In any case, tagging bug reports with disparate label information may result in the loss of context information. Topic modeling systems require the biggest data amount because of information retrieval techniques. To reliably provide the bug report data, a "semi-supervised text categorization model (Xuan et al., 2017) has been examined. In addition to the Bayesian method notion, the report data creation was superior. By giving developers priority, social networking approaches (Xuan et al., 2012) have provided bug classes. The developers" have given priority to a number of important criteria, including temporal variation, product features, and noise tolerance.

Methods such as path tossing were used to address the bug reports. The model for classifying bugs using transfer graphs has been given by (Jeong et al., 2009). The correctness of the bug report assignment was also recorded. It does not, however, guarantee the classification bugs' feasibility rate. (Sajedi Badashian et al., 2020) presented the model of transfer graphs with various properties. According to the bug-throwing issues in bug triaging, 93% of bug reports are discarded. According to the statement, the problems were not adequately guaranteed for various situations. (Shokripour et al., 2015) has produced the bug parameters' class score. The bug reports are categorized into the appropriate classifications with the use of fuzzy logic algorithms. A strong model which illustrates the relationship among software metrics along with the incidence of problems has been presented by (Couto et al., 2012). The corpus was used to assist in determining the prediction of bugs. On the other hand, there is now a higher false positive rate for bug categorization.

HyGRAR is a hybrid classification model that (Miholca et al., 2018a) combines association rule mining with the hidden layer features of ANN. Software entities that are faulty and those that are not were categorized based on the relationship between the software metrics. The NASA PC software databases were used to investigate it. A significant rise in hidden units led to class imbalance problems. To preserve the development of the flaws, unsupervised learning under coupling metrics was investigated (Miholca et al., 2017). In order to measure the connection of application situations, an object-oriented system was implemented. The performance of textual data has improved with the representation of large-dimensional information. The structural class rates have increased with the Doc2Vec conversion. Contextual information rate was raised by the participation of semantics along with structural characteristics (Miholca et al., 2019) throughout the learning procedure. The files were transformed into token vectors by coding them using Abstract Syntax Trees (AST). The token vectors were worked out using CNN classifiers. The classes have a 99.5% accuracy rate based on the semantic characteristics. Moreover, a hybrid method was investigated by employing artificial neural networks and association rules (Miholca et al, 2018b). In order to improve the classifier's efficiency, ten open-source data sets were used; still, significant computational steps were noted.

A thorough study of software defect prediction quality measures was conducted (Radjenovic D. Heri, 2013). Contextual characteristics investigated the accessibility of choosing the quality metrics since the evaluation indicated that the ability to forecast the location of faults is very beneficial in influencing the quality metrics. Deep Belief Networks investigated an intelligent SDP (Wang et al., 2016). Semantic characteristics were collected and examined for both WPDP (within-project defect prediction) and CPDP (cross-project defect prediction) using syntax trees. When compared to the TCA+ algorithm, both prediction models had a 10% lower false positive rate. The author of (Jing et al., 2017) focused on using an enhanced Subclass Discriminant Analysis (SDA) to address the class imbalance issues. SDA has distributed the source and target data uniformly and consistently in order to purposefully resolve the feature learning modules. Based on logical constraints, the prediction model has reduced the class separation from the cross-project domain perspective. According to Liu et al. (2014), a two-stage cost-sensitive learning module was developed in order to take advantage of the cost-sensitivity study that was conducted on advanced learning algorithms. Accordingly, for feature selection algorithms like CSCS ("Cost-Sensitive Constraint Score"), CSVS ("Cost-Sensitive Variance Score"), and CSLS ("Cost-Sensitive Laplacian Score"), a cost-sensitive technique was also created. It reduced the computational time and made use of cost measures according to the quality of the software; nevertheless, the effectiveness of contextual information determination has not been examined. A ranking technique was used in similar research to determine the most effective utilization of testing resources (Yang et al., 2015). The module was developed to handle both noisy data and feature learning modules, depending on the number of flaws. Under optimization approaches, the classifier's characterization of the association between the square errors was corrected.

Latent semantic indexing has been investigated by writers in the computational linguistics domains (Czibula et al, 2014). Latent semantic analytics was used to investigate the genetic ontology's processing capabilities. The experimental data was processed by extracting the themes' similarities. Not all of the structured data under the heterogeneous classes was extracted. Haghighi et al. (2012) developed Relational Association Rules (RAR) as a means of resolving defect prediction under various association rules. It has been investigated using NASA records that took a long time to clean up. Although it was processed on many datasets, a semi-supervised approach is not the primary focus of the generated class. Cost reduction is one of the software's attributes, and data mining has been utilized to study it (Kirbas et al, 2017a). To find the errors, the Naive Bayes classifier has been utilized in addition to the output of 37 other classifiers. According to the authors, the performance of the classifiers was improved by the lower expenses. Although the technology has improved classification accuracy, bagging methods do not make it possible to identify class borders. flaws in software have caused distortions in the link between evolutionary coupling and flaws in industrial sectors (Kirbas et al., 2017b). It conducted trials

on both the contemporary telecoms system and the antiquated finance system. To determine the association between the software metrics, regression analysis was used. Various evolutionary coupling effects were used to identify the defects according to their sizes, kinds, and process metrics. In a similar vein, the infrastructure problems are not examined for various testing cycles. A conceptual coupling-based metric for SDP systems was proposed by the author in the article that was published in 2009 by Cataldo et al. One of the software quality measures used to describe the prediction modules' performance is coupling metrics. It was determined by calculating how similar the bugs were to one another. It is unable to manage the problems of class imbalance in data-packed environments, nevertheless. Empirical research was used to expand on a similar study (Chen et al, 2013). The software systems' structural, semantic, and dynamic measurements are included in the coupling class. Java classes like jEdit, JHotDraw, and ArgoUML were examined here. More research was done on semantic coupling than on other coupling metrics. Call methods were used to update the encapsulation between the classes (Sushan et al, 2020). There is no study done on the logical restrictions on class call methods.

## 3. Proposed Methodology:

The ELMA+, which uses ensemble classifiers to classify software problems, is explained in this section.

### 3.1 Data Collection & Preprocessing:

Promise datasets, a public repository, is where the database is gathered. It includes many software bug class instances. Many noises, such as missing and unbounded data, are present in the dataset. To correct for these imbalances, random sampling techniques are used. Data preparation is the term for this stage. To eliminate uncertainty and extraneous information from the data, preprocessing procedures must be used for the raw datasets that were obtained. Sampling techniques are typically used to preprocess the data. The training data's computational steps are lowered by the data sampling. The sampling techniques used here are divided into two categories: undersampling and oversampling. This work investigates automatic suitable sampling techniques to minimize the amount of unnecessary data in the computing space. Smaller random samples are created from the bigger dataset. T tuples of data make up each sample. The tuples probability under the dataset D, or 1/T, is probably what will be used to pick a sample from the tuples T, ensuring that each tuple is sampled equally. Prior to the other tuples being added, certain tuples are recorded. This allows all tuples to be sampled under the specified functions, which may help to reduce the amount of redundant and noisy data.

### 3.2 Feature Extraction:

The SMOTE is a unique approach used in the proposed work for an oversampling procedure to address the class imbalance issue as well as ensure that "the training dataset has sufficient samples of both the majority along the minority classes. The replacement of a large amount of oversampled data improves minority class detection. Consideration is given to the implications of oversampling under various feature space regions. Lowering the feature space reduces the minority class regardless. The oversampling data is substituted for the minority classes' benefit by producing 'synthetic' cases. On the actual data, more data operators are applied here. Rotation and skew operators are applied to the training set. The synthetic instances that are developed in accordance with the real-time data exhibit characteristics of feature space rather than data space". The line segments of the closest data points merge with the minority of each class. After accurately defining the majority and minority classes, the oversampled data are sorted. Next, the ensemble of classifiers receives the scaled features. To scale the features, Gaussian distributions and k-NN are combined.

### a) K- Nearest Neighbors (k-NN):

It is constructed in this stage. Regardless of distance, k-NN takes into account the impartial weighting of occurrences in the decision rule while classifying. The traditional k-NN classifier's stages are as follows:

- Locating the k-training examples
- Finding the data that is most frequently encountered on these k-instances
- Calculating an estimate of the distance values that exist between those k samples.

Rather than using a majority vote, the KNN approach bases its decision rule on item strength. The class with the item strength greatest summation is designated as the label for each test point that has to be examined. The training data is divided into more manageable subgroups here. Each subset has a classifier model created for it, and the testing data is then classified using a distance method. The performance of the classifiers will be determined on the basis of assessed distance on the testing data. The Euclidean distance formula may be utilized to simply determine the distance estimation among the data because it works with continuous qualities. Let the first subset data be displayed as, $(x_1, x_2 \ldots . x_p)$and the data in the $2^{nd}$ subset has been displayed as, $(y_1, y_2 \ldots . y_q)$. After that, the Euclidean distance between two subsets is found by using the formula:,

$$\text{Distance} = \sqrt{\quad}$$

In order to analyze each of the data in the subsets, the equation shown above is utilized. According to the findings, the values that are the smallest cause problems for the values that are the highest.

### b) Gaussian Distribution:

Determining the variance of cases pertaining to software flaws has been made easier with the aid of the extracted characteristics. Next, in order to distinguish between the various bug types, a multivariate Gaussian distribution was developed in the testing cases. The application of multivariate Gaussian models has led to a significant evolution of the data interaction while reducing complexity. Assuming x to be the number of bug classes, let € be the best-fit parameter. The training set's highest likelihood value is ascertained. Gaussian Distribution p(x) is used to find the mean & SD value of the features utilized for training. It is stated as

$$P(x) = \prod_{k=1}^{n} P(x_k : \mu_k \sigma_j$$

| | |
|---|---|
| k | Count of features taken from the sample data |
| σ | standard deviation |
| μ | gaussian distribution function mean value |

In order to define the anomalous classes, the following assumptions were formulated:
If "P(x)>= optimized fit value €, then the value belongs to the normal class
If P(x) < optimized fit value €, then the value is said to be bug class"

### 3.3 Ensemble Classifier:

To determine the defect categorization, ensemble classifiers are designed. Three popular classifiers, including bagging, adaboost, and random forest, are developed here.

### a) Classifiers for random forests:

The following are the random forest classifiers' suggested steps:

- A tree's diversity of records guarantees linked properties between its nodes and leaves.
- The trees are constructed using randomly chosen attributes.
- The root node is not changed for any reason, including parameter fitness, stopping criteria, and so on.
- Only when the information acquired changes are root nodes modified.
- guaranteed effective features for the classifier models' training.

Imagine a potential input data collection of N records. Single aspects C is the label that is assigned to the random trees that fall under the single-objective category, and each record is represented as a d-dimensional space vector. It is said in the following manner:

$$S_0 = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$$

Where,
The record labels are X and Y.
The following is a two-dimensional splitting criterion for a random tree:

$$h(\mathbf{x}; \theta_1, \theta_2) = \begin{cases} 1 & \text{if } x_{\theta_1} > x_{\theta_2} \\ 0 & \text{otherwise.} \end{cases}$$

The splitting criteria combined with mutual information creates a random tree. More specifically, the splitting functions help build a tree's left and right children. The restrictions of a tree's left and right children are shown in the equation below.

$$\begin{aligned} \mathcal{S}_i^L(\boldsymbol{\theta}) &= \{\mathbf{x} \in \mathcal{S}_i \mid h(\mathbf{x}; \theta_1, \theta_2) = 1\} \\ \mathcal{S}_i^R(\boldsymbol{\theta}) &= \{\mathbf{x} \in \mathcal{S}_i \mid h(\mathbf{x}; \theta_1, \theta_2) = 0\} \end{aligned}$$

The data set quality Si is represented by the creation of a child in a tree. Each record's information gain is calculated using equation (20):

$$I_i(\boldsymbol{\theta}) = H(\mathcal{S}_i) - \sum_{j \in \{L,R\}} \frac{|\mathcal{S}_i^j(\boldsymbol{\theta})|}{|\mathcal{S}_i(\boldsymbol{\theta})|} H(\mathcal{S}_i^j(\boldsymbol{\theta})),$$

Where,
H(s) represents the entropy value of set S.
Every tree indicates the class that has gained knowledge from the datasets. Si. From now on, the continuous records sampling. As a result, the two equations above create the trees by a continuous sampling of the data.

In particular, a tree's interior nodes have been assigned based on the greatest information gain. The following equations provide the overall split of a tree for tree depth as T= d(d-1).

$$\boldsymbol{\theta}_i^* = \arg\max_t I_i(\boldsymbol{\theta}_t).$$

Overfitting of the data may happen when the trees are being constructed, i.e., when the node is placed as the left (or right) tree branch. When building the random forest, the diversity problem significantly alters the classifiers. The forest's tree collection is shown as F= (T1... Tf). Before creating forests, a tree's left and right branches are regularized. Every tree is trained on its own.

### b) Adaboost classifier:

Boosting is a method that combines weak and comparatively incorrect rules to produce prediction rules. It proceeds and finds the class that way. The AdaBoost algorithm's pseudocode is,

"Pseudocode:
Given data: $(x_1, y_1 \ldots\ldots (x_m, y_m)$ where x and y are the set of samples
Initialize: $D_1(a) = 1/n$ fora $= 1, \ldots, n$
For q=1, ..... P
Train the weak learner using the distribution $D_q$.
Get the weak hypothesis, $h_t$: $\aleph \rightarrow \{-1, +1\}$
Select the $h_t$ with the low weighted error,

$$\epsilon_t = P_{ra\ D_q}[h_t(x_a \neq y_m$$

Select $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
Update for, a= 1,...., n:

$$D_{t+1}(a) = \frac{D_t(a)\exp(-\alpha_t y_m h_t(x_m))}{Z_t}$$

Where, $Z_t$ is the normalization factor.
The final hypothesis output: $H(a) = \text{signh}_t(a))$"

### c) Bagging classifier:

Using a voting method, the bag classifier unifies the decisions made by the several learners into a single prediction model. In any case, there are three simple ways to complete the bagging process: using the path that leads to the experts. It is deemed right in the event that one class receives more votes than the others. Voting modules are used to classify the classes. In the training sets, the weight is similarly estimated at random.

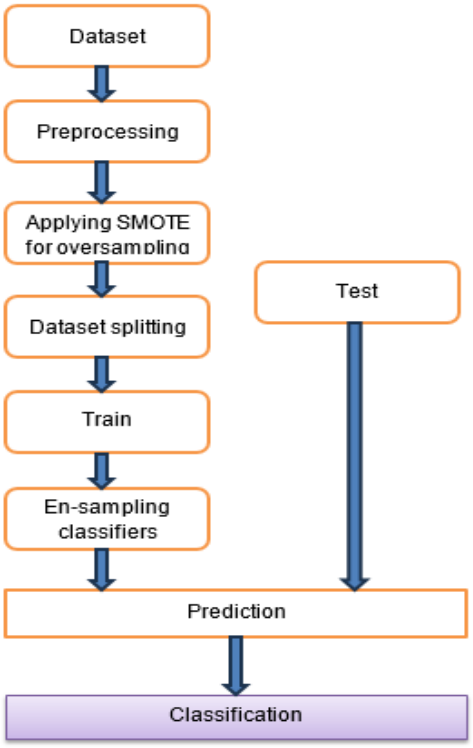


**Fig.1. Proposed workflow**

## 4. Results and Discussion:

The suggested framework is tested using certain performance indicators. The public repository promising SDP provided the dataset which has been utilized in this investigation. The high-level programming language Python is used for the data analysis. The most current programming language used in data analytics research is Python. It offers an excellent selection of libraries for artificial intelligence (AI) and machine learning (ML) that investigate the effectiveness of real-time analytic systems. The confusion matrix in our suggested study is provided "as,
False positive (FP)- No. Of samples incorrectly identified as true.
True positive (TP)- No. Of samples correctly identified as true
False Negative (FN)- No. Of samples incorrectly identified as false
True Negative (TN)- No. Of samples correctly identified as false"

**Table 1. Class Distribution**

| Class | Label values |
|---|---|
| True | 0 |
| False | 1 |

**Premises:**
Labels: Both "true (software defects are present) and false (software defects are absent)

**a) Recall:** It is possible to express the ratio of true cases to anticipated cases as follows:
Recall= (TP)/(TP+FN)

**b) Precision:** Its definition is the percentage of true values which have been accurately anticipated to be true. It is said as follows: Precision= (TN)/(TN+FP)

**c) Prediction Accuracy:** Measuring the accurately anticipated observation against the total observations is the most intuitive performance. It describes the capacity to discriminate between real and fake unusual instances. It is said as follows:
Accuracy= (TP+ TN)/ (TP+TN+FP+FN)"

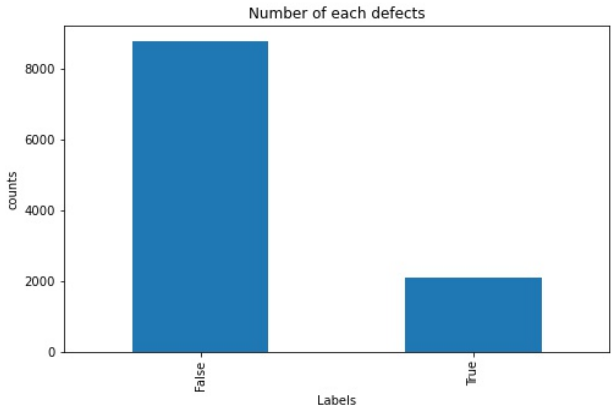


**Fig.2. Data analysis before the SMOTE application**

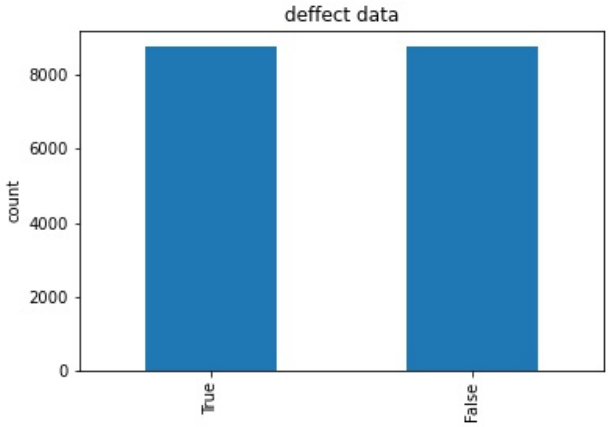

**Fig.3. Data analysis after the SMOTE application**

The SMOTE application's data analysis is shown in Figures 2 and 3. In this case, the examined dataset is examined in terms of false & true, that is, whether bugs in the software are there (True) or not (False). Examining the distribution of majority and minority classes is beneficial. To take advantage of minority data classes, oversampling processing using the SMOTE approach is required.

**Table 2: Comparison between existing and proposed classifiers**

| Parameters | Proposed | Existing |
|---|---|---|
| Specificity | 82 | 17 |
| Sensitivity | 96 | 96 |
| Accuracy | 89 | 82 |
| Precision | 96 | 96 |

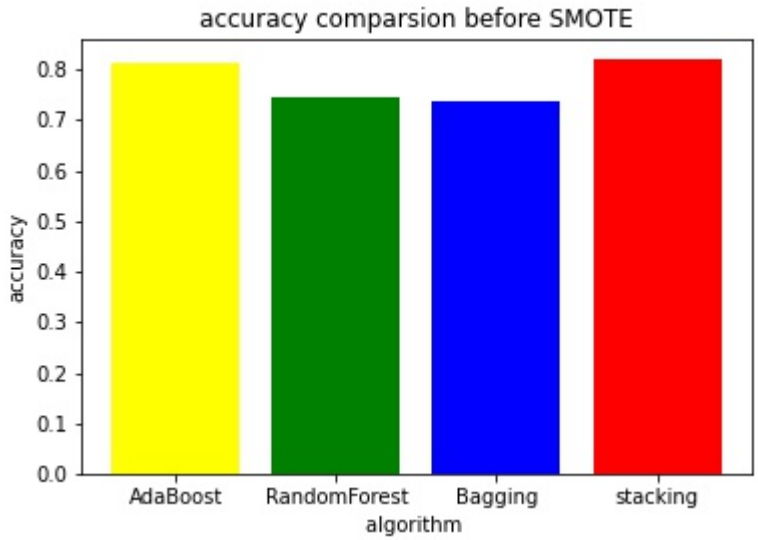


**Fig. 3. Existing - Accuracy performance of ensemble classifiers**

Figure 3 shows the classifier ensemble's accuracy performance prior to using the SMOTE approach. The findings show that the classifiers have produced improved accuracy; nevertheless, the software defect classification's sensitivity and specificity are deemed to be higher values. It demonstrates the classifiers' incapacity.

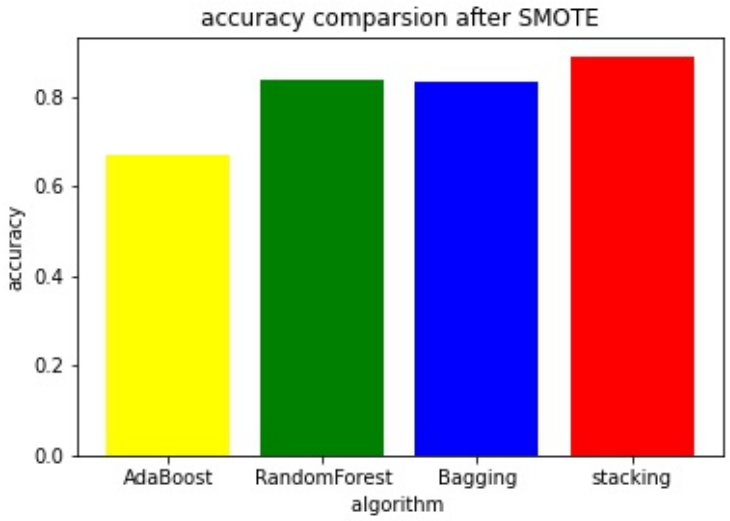


**Fig.4. Proposed - Accuracy performance of an ensemble of classifiers**

Figure 4 shows the classifier ensemble accuracy performance following the use of the SMOTE approach. In

this case, the set of classifiers is represented by the Y-axis and the set of ensemble classifiers by the X-axis. It is evident that the class imbalance problems with the SMOTE approach have been resolved by the suggested ensemble classifiers.

## 5. Conclusion:

The goal of this research is to reliably anticipate the kinds of software defects by designing a unique ELMA+ approach. This paper examines a group of popular classifiers, including Bagging, Adaboost, and k-nearest neighbors (k-NN). The defect data is gathered by the public repository to start the investigation. A more basic exploratory data analysis is carried out to determine the number of software flaws that are present and absent. The gathered dataset is preprocessed using the SMOTE algorithm. The minority classes' involvement in the training procedure has decreased because of the existence of oversampled data. The oversampling data is in line with the generation of synthetic data in order to take advantage of "the minority classes along with the existence of data ambiguity concerns. The synthetic instances that are developed in accordance with the real-time data exhibit characteristics of feature space rather than data space". The closest data points line segments merge with the minority of each class. The ensemble of classifiers, which includes Bagging, Adaboost, and K-Nearest Neighbors (k-NN), is then given the scaled features. To categorize the software problems, these 3 classifiers use the feature-scaled data as input. To be more precise, minority classes are settled before being examined to address data ambiguity concerns. The suggested ensemble classifiers have demonstrated their efficiency based on improved sensitivity, accuracy, specificity, and precision, as demonstrated by the experimental findings. the comparison of the analysis conducted before and after SMOTE use. The obtained findings make it abundantly evident that using feature-scaled data in the ensemble classifiers has improved the accuracy of the predictions.

## References


1. Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction. In: Software engineering (ICSE), 2016 IEEE/ACM 38th international conference on. IEEE; 2016. p. 297–308.
2. Yang X, Lo D, Xia X, Zhang Y, Sun J. Deep learning for just-in-time defect prediction. QRS; 2015. p. 17–26.
3. Huo X, Li M, Zhou Z-H. Learning unified features from natural and programming languages for locating buggy source code. IJCAI; 2016. p. 1606–12.
4. Clemente CJ, Jaafar F, Malik Y. Is predicting software security bugs using deep learning better than the traditional machine learning algorithms?. In: 2018 IEEE international conference on software quality, reliability and security (QRS). IEEE; 2018. p. 95–102.
5. Li J, He P, Zhu J, Lyu MR. Software defect prediction via convolutional neural network. In: 2017 IEEE international conference on software quality, reliability and security (QRS). IEEE; 2017. p. 318–28.
6. Rudolf Ferenc et al.,. Deep learning in static, metric-based bug prediction. Array (Elsevier), 6, 2020.
7. Cheng Zhou et al.,. Improving software bug-specific named entity recognition with deep neural networks. The Journal of Systems and Software, 165, 2020.
8. Sushan Kumar Pandey et al.,. BPDET: An Effective Software Bug Prediction Model using Deep Representation and Ensemble Learning Techniques. Expert Systems With Applications. 2019.
9. Sushan Kumar Pandey et al.,. BCV-Predictor: A bug count vector predictor of a successive version of the software system . Knowledge based systems. 2020.
10. Yan Xiao et al.,.Improving Bug Localization with Word Embedding and Enhanced Convolutional Neural Networks. Information and Software Technology.2018.
11. Guo, Shikai, Zhang, Xinyi, Yang, Xi, Chen, Rong, Guo, Chen, Li, Hui, Li, Tingting, 2020. Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network. Neural Processing Letters 51 (3), 2589–2606.
12. Zhao, Huimin, Zheng, Jianjie, Xu, Junjie, Deng, Wu, 2019. Fault Diagnosis Method Based on Principal Component Analysis and Broad Learning System. IEEE Access 7, 99263–99272.
13. X. Xie, W. Zhang, Y. Yang, and Q. Wang, "DRETOM: Developer recommendation based on topic models for bug resolution," 2012, doi: 10.1145/ 2365324.2365329.
14. Xuan, J., Jiang, H., Ren, Z., Yan, J., Luo, Z., 2017. Automatic bug triage using semi- supervised text classification. arXiv.
15. Xuan, Jifeng, Jiang, He, Hu, Yan, Ren, Zhilei, Zou, Weiqin, Luo, Zhongxuan, Wu, Xindong, 2015. Towards effective bug triage with software data reduction techniques. IEEE Trans. Knowl. Data Eng. 27 (1), 264–280.
16. G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," 2009, doi: 10.1145/1595696.1595715.
17. Sajedi-Badashian, Ali, Stroulia, Eleni, 2020. Guidelines for evaluating bug-assignment research. Journal of Software: Evolution and Process. 32 (9).
18. Shokripour, Ramin, Anvik, John, Kasirun, Zarinah M., Zamani, Sima, 2015. A time- based approach to automatic bug report assignment. Journal of Systems and Software 102, 109–122.

19. Miholca, D., 2018. An improved approach to software defect prediction using a hybrid machine learning model, in: 2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 443–448. doi:10.1109/SYNASC.2018. 00074.
20. Miholca, D., Czibula, G., Zsuzsanna, M., Czibula, I.G., 2017. An unsupervised learning based conceptual coupling measure, in: 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017, Timisoara, Romania, September 21-24, 2017, pp. 247–254.
21. Miholca, D.L., Czibula, G., 2019. Software defect prediction using a hybrid model based on semantic features learned from the source code, in: Douligeris, C., Karagiannis, D., Apostolou, D. (Eds.), Knowledge Science, Engineering and Management -LNCS, volume 11775, Springer International Publishing, Cham. pp. 262–274.
22. Miholca, D.L., Czibula, G., Czibula, I.G., 2018. A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. Information Sciences 441, 152 – 170.
23. Radjenovic, D., Heri ´ cko, M., Torkar, R., ˇ Zivkovi ˇ c, A., 2013. Software fault prediction metrics: A systematic literature review. In- ˇ formation and Software Technology 55, 1397 – 1418.
24. Wang, S., Liu, T., Tan, L., 2016. Automatically learning semantic features for defect prediction, in: Proc. of the 38th Int. Conf. on Softw. Engineering, ACM, New York, NY, USA. pp. 297–308
25. Jing XY, Wu F, Dong XW, Xu BW. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans Softw Eng 2017;43:321–39
26. Liu M, Miao L, Zhang D. Two-stage cost-sensitive learning for software defect prediction. IEEE Trans Reliab 2014;63:676–86.
27. Yang XX, Tang K, Yao X. A learning-to-rank approach to software defect prediction. IEEE Trans Reliab 2015;64:234–46.
28. Czibula, G., Marian, Z., Czibula, I.G., 2014. Software defect prediction using relational association rule mining. Inf. Sci. 264, 260–278
29. Haghighi, A.S., Dezfuli, M.A., Fakhrahmad, S., 2012. Applying mining schemes to software fault prediction: A proposed approach aimed at test cost reduction, in: Proc/ of the World Congress on Engineering, IEEE Computer Society, Washington, DC, USA. pp. 1–5.
30. Kirbas, S., Caglayan, B., Hall, T., Counsell, S., Bowes, D., Sen, A., Bener, A., 2017. The relationship between evolutionary coupling and defects in large industrial software. J. Softw. Evol. Process 29, 1–19.
31. Kirbas, S., Caglayan, B., Hall, T., Counsell, S., Bowes, D., Sen, A., Bener, A., 2017. The relationship between evolutionary coupling and defects in large industrial software. J. Softw. Evol. Process 29, 1–19
32. Cataldo, M., Mockus, A., Roberts, J.A., Herbsleb, J.D., 2009. Software dependencies, work dependencies, and their impact on failures. IEEE Transactions on Software Engineering 35, 864–878.
33. Chen, H., Martin, B., Daimon, C., Maudsley, S., 2013. Effective use of latent semantic indexing and computational linguistics in biological and biomedical applications. Frontiers in Physiology 4, 8
34. Sushan Kumar Pandey et al.,. BCV-Predictor: A bug count vector predictor of a successive version of the software system . Knowledge based systems. 2020.