Automatic Detection of Software Defects based on Machine Learning

Nawal Elshamy[1], Amal AbouElenen[2], Samir Elmougy[3]
Computer Science Department-Faculty of Computer and Information Science, Mansoura University, Mansoura, Egypt[1]
Computer Science Department-Faculty of Computer and Information Science, Mansoura University, Mansoura, Egypt[2, 3]

***Abstract*—Defects in software are one of the critical problems in software engineering community because they provide inaccurate results and negatively affect the quality and reliability of the software. These defects must be detected in the early stages of software development. Researchers had used Software Defect Detection (SDD) techniques to allow predicting module fault-proneness. By implementing the hyperparameter optimization techniques and exploiting data imbalances in predicting defects, this paper proposes and develops an SDD model with high performance and generalization capability. To classify defects in software modules, machine learning algorithms and ensemble learning techniques are used on the balanced datasets. The balanced datasets are obtained through using a hybrid of synthetic minority oversample (SMOTE) and Support Vector Machine (SVM). To obtain the optimal hyperparameters needed for the used classifiers and for the dataset balanced algorithms, Non-dominated Sorting Genetic Algorithm II (NDSGA-II) is used. To reduce the time and save other used resources, Hyperband technique, which is a multi-fidelity optimization, is used in NDSGA-II. A 10-fold Cross Validation (CV) is applied to overcome the overfitting and underfitting problems. The accuracy, recall, F-measure, and ROC AUC metrics are used to evaluate the SDD model. The results show that the proposed model predicts defects more accurately than the compared studies.**

Keywords*—Software defect detection; NDSGA-II; hyperband; imbalance dataset

I. Introduction

One of the critical topics in the software engineering community is the development of high software quality and reliability while making effective use of limited resources. The Software Development Lifecycle (SDLC) is a structured method developed to ensure the production of stable, high-quality software. To ensure a timely and effective software system, it is essential to follow the SDLC's stages, which include requirement collecting, requirement analysis, system design, system development, and maintenance. A software fault may be a human error or a system-related error, failure, or crash. Defects have a significant effect on software quality and even the economics of software. So, fixing defects is an important part of software maintenance, but it also wastes time and resources. Detecting software faults before software deployment is crucial, as the correct detection of faulty software modules or components allows good use of resources and time [1] [2].

"Defect detection technology" is the ability to find bugs in every code change that developers send. SDD used ML approaches to software defect datasets characterized by software metrics (as features) to identify software module or component problems. Researchers had developed and implemented ML approaches for SDD.

One of the most important steps in creating a reliable ML model is tuning the model's hyperparameters. It's important to note that the tuning process differs for categorical, discrete, and continuous hyper-parameters. Manual testing is a common method for altering hyper-parameters, but it requires a comprehensive understanding of ML algorithms and their hyper-parameter settings. Due to the high number of hyper-parameters, the complexity of the models, the length of time required to evaluate the models, and the non-linear interactions between the hyper-parameters, manual tuning is often ineffective [3]. These considerations have motivated more studies in Hyper-parameter Optimization (HPO) approaches especially when working with huge datasets or when using complex ML algorithms with a large number of hyper-parameters. The main goal of HPO is to automate this process to improve the performance of the ML model, find the best ML model for a specific problem, and reduce the amount of human effort needed. To find ideal hyper-parameters, it is critical to use the optimal optimization technique. Because many HPO issues are often non-convex or non-differentiable optimization issues, traditional optimization techniques may be inadequate for them, resulting in a local rather than a global optimum [4]. A Non-Dominated Sorted Genetic Algorithm (NDSGA-II) is used in this paper for hyperparameter search hybrid with hyperband speed configuration.

SDD models were developed using Machine Learning (ML) classifiers. Nevertheless, SDD datasets contain more nondetectable than detectable occurrences; this is known as the "class imbalance problem." In addition, multiple studies on defect detection models revealed that minority classes contain more instances of faults than do majority classes that are defect-free. Hence, applying ML algorithms to such unbalanced data yields biased outcomes for minority-class occurrences. To successfully manage an imbalance in datasets, oversampling techniques are utilized [5].

This paper implements the SMOTE-SVM algorithm to balance the imbalanced data with tuned parameters based on the NDSGA-II hybrid with the Hyperband algorithm. Decision Tree (DT), Random Forest (RF), and ensemble learning with Adaboost (AB) and Bagging (BG) classifiers are the ML classifiers used in this work. The proposed model is evaluated on nine defect detection NASA datasets. This work is structured as follows: The second and third sections provide

"Background" and "literature review," respectively. The methodology and requirements for the experiment are discussed in Section IV. Experiments and their results are presented in Section V. The final section of the paper discusses the conclusion and what comes next.

II. BACKGROUND

Several facets of SDD are discussed in this section. We describe optimization strategies for hyperparameter tuning and data-imbalance-resolving algorithms.

A. Class Imbalanced Problem

Problems with class imbalance arise in datasets where the values for different classes are distributed unevenly. Imagine a dataset in which 95% of the class values are from one class and only 5% are from another; this dataset is unbalanced. Minority classes and values are rarely reported, compared to hundreds, thousands, or even millions of majority examples. When ML classifiers are applied to such datasets, models lose detection capacity and provide off-target results. Biased outcomes are produced when a predictive model is used for classification on an uneven dataset. ML procedures function best with an evenly distributed dataset. In this way, ML models fail to accurately predict class values in unbalanced datasets. For minority class norms, this phenomenon is especially prevalent. Effectively addressing class imbalance issues is critical because minority class values are regarded as more significant than majority class values [6].

By simulating or synthesizing instances from underrepresented groups, defect detection experts have found that sampling strategies can produce more representative datasets. This research reconstructed instances of under-represented groups by using oversampling techniques. As a result, the oversampling techniques significantly improved the ML classifiers' detection performance.

1) SMOTE-SVM: SMOTE is a popular oversampling technique for achieving more equitable class distribution by simulating the emergence of new instances of the minority class along roads connecting existing instances of the minority class to their nearest neighbors. To aid in the establishment of class boundaries, SVM-SMOTE generates new instances of the minority class along boundary lines. The synthetic sampling technique with data generation (Synthetic Minority Oversampling Technique, SMOTE) [7] is one of the efficient special algorithms for re-establishing class parity after oversampling by increasing the number of objects in the minority class. Using k-nearest Neighbor (KNN) approach, SMOTE algorithm generates minority-class synthetic items from similarities in the feature space between existent objects. With this method, we can manufacture an arbitrary number of artificial objects that are "similar" to those in the minority class but are otherwise unique [8].

2) Support Vector Machine (SVM): It is a supervised learning algorithm that works by mapping low-dimensional data points into a high-dimensional feature space to make them linearly separable and then using an optimal separating hyperplane as the classification boundary to partition the data by increasing the difference between the two classes. With the assumption of n data points, SVM's objective function is [4][9][10]:

$$arg\ min_m \left\{\frac{1}{n}\sum_{i=1}^{n}\max\{0,1-\ y_i f(x_i)\} + C\boldsymbol{w}^T\boldsymbol{w}\right\} \quad (1)$$

where w is a normalization vector, and C is the penalty parameter of the error term, which is an important hyper-parameter of all SVM models.

Several kernels are available to be used in SVM models to determine the similarity between two data points, x_i and x_j. Hence, the type of kernel would be a crucial hyperparameter to adjust. The most common kernels used in SVM include linear kernels, Radial Basis Function (RBF) kernels, polynomial kernels, and sigmoid kernels. The different kernel functions can be denoted as follows [11]:

- Linear kernel

$$f(x) = x_i^T x_j \quad (2)$$

- Polynomial kernel

$$f(x) = (\gamma\, x_i^T x_j + r)^d \quad (3)$$

- RBF kernel

$$f(x) = \exp\left(-\gamma\, ||x - x'||^2\right) \quad (4)$$

- Sigmoid kernel:

$$f(x) = \tanh\left(-\gamma\, x_i^T x_j + r\right) \quad (5)$$

After deciding on a kernel type, further hyper-parameters must be adjusted, as demonstrated by the corresponding kernel function equations. When the "kernel type" hyper-parameter is set to polynomial, RBF, or sigmoid, the coefficient is the conditional hyper-parameter; r is the conditional hyper-parameter of polynomial and sigmoid kernels. Extra conditioned hyper-parameter, d, defines the "degree" of the polynomial kernel function.

As part of the training process for an SVM classifier, the kernel function type k (x_i, x_j) values and the value of the regularization parameter C are calculated so that a trade-off may be made between increasing the distance between classes and reducing the total error [12].

B. Hyperparameter Optimization (HPO)

The training of a ML model is subject to number of hyperparameters. These hyperparameters determine the model's strategy for learning a given connection between input and detections. Using hyperparameter optimization (HPO), a model, that has been fine-tuned using the most effective hyperparameters, could be obtained. This model ought to be able to produce minimum-loss solutions. The challenge with optimization problems is that the search space is typically infinite, while the resources available to carry out the search are finite (maximum x amount of time or iterations). Consequently, for an algorithm to search for the global minimum effectively, it must incorporate strategies to make the most of the available funds. HPO's slow execution time is a

serious drawback, especially when dealing with a wide variety of hyper-parameter configurations or massive datasets [4].

1) Non-dominated sorting GA-II: As non-dominated sorting is used to generate a preliminary coarse ranking of the population, NSGA-II (Pareto dominance-based) continues to be one of the most well-liked algorithms. The non-dominated solutions can be found using this technique and then moved to the next class before being discarded. The solutions in each obtained class are sorted from best to worst according to the objective's crowding distance, the total of the differences between an individual's left and right neighbors. It's best to have a lot of space between individuals. Non-dominated sorting becomes less discriminative and the left and right neighbors in each goal are generally different solutions when there are more than two objectives, hence this strategy fails for problems with more than two objectives [13].

2) Multi-fidelity optimization technique: In order to work around the problem of having insufficient time or resources, multi-fidelity optimization techniques are frequently used. The original dataset or the features used can be reduced to a subset to save time [14]. Low-fidelity and high-fidelity evaluations are combined in multi-fidelity for use in the real world [15]. Low-fidelity evaluations are those that only test a small fraction of the data and so are very inexpensive, but have poor generalization performance. Better generalization performance is achieved at the expense of increased cost in high-fidelity assessments, where a larger subset is assessed. Poor performing configurations are eliminated from multi-fidelity optimization methods at each iteration of hyper-parameter evaluation on newly generated subsets, leaving only the best performing configurations to be evaluated on the full training set. Bandit-based algorithms are a kind of multi-fidelity optimization algorithms, and they make use of strategies like sequential halving [16] and Hyperband approach [17].

- Successive Halving: By testing each possible combination of hyper-parameters, successive halving can determine which one works best. However, in real-world applications, numerous considerations must be taken into account, such as time and resource constraints. The term "budget" is used to describe these considerations (B). The following is the primary procedure for employing successive halving algorithms for HPO. The first assumption is that there are n sets of hyper-parameter combinations to test and that these sets are tested using equally distributed resources ($b = {}^{B}/_{n}$). Finally, at the end of every cycle, half of the underperforming hyper-parameter configurations are thrown out, while the other half are passed on to the next cycle with double budgets ($b_{i+1} = 2 * b_i$). The preceding steps are continued until the best possible set of hyperparameters is found. The trade-off between the feasible hyper-parameter configurations and available budgets affects the cost-benefit analysis of succeeding halves [18]. With this in mind, the main issue with consecutive halving is deciding how to divide the budget, specifically between testing fewer configurations with a larger budget for each and testing more configurations with a smaller budget for each.
- Hyperband approach [17] is considered a solution to the issue of successive halving methods by dynamically picking an appropriate number of configurations. It seeks to strike a balance between the total budget (B) and the number of hyper-parameter configurations (n) by apportioning a portion of the total budget to each configuration ($b = {}^{B}/_{n}$). Each batch of random configurations is routinely halved in order to get rid of the inefficient hyperparameter setups and boost performance. The total quantity of data points, the minimal number of instances required to train a meaningful model, and the available budgets all contribute to the restrictions b_{min} and b_{max} . After that, we use b_{min} and b_{max} to get the total number of configurations n and the budget size for each (max). Based on n and b, a random sample of configurations is generated and fed into the shown successive halving model. Each iteration of the consecutive halving method takes the top half of the configurations and discards the bottom half, keeping only the best-performing ones. This is done again until the best possible set of hyperparameters is found.

C. Ensemble Learning (EL)

1) Random Forest (RF): RF is a bagging approach that generates many independent, tiny decision trees from the dataset at random. Selecting a small number of attributes at each node in order to find the best branching technique allows for a deeper tree structure. Dataset and feature randomization makes overfitting less likely. In a classification problem, the majority vote is used to determine the final class label [19].

2) Bagging: The goal of "bagging" [20] is to increase the reliability of ML algorithms, especially decision trees. It helps prevent data from being overfit and lowers the model's variance. It takes the original training dataset and randomly selects n subsets of features and data samples, with replacement, to create n new datasets. Parallel models for making detections, using each of the n sub datasets as a single input, are built. The bagging classifier selects as its output the label class predicted by the majority of the base models.

*3) Adaptive Boosting (AdaBoost)**:*** It is a technique used to increase the accuracy and performance of numerous weak classifiers by combining them into a single strong classifier. Adaboost's weak classifiers frequently use decision stumps, a variant of the decision tree that consists of a single node (the root) and two branches. Various classes of faulty classifiers are taught. When a less accurate classifier misclassifies a sample, the sample is assigned more weight in the next classifier. Further, the aggregate weight is based on how well each poor-quality classifier performed. Adaboost classifier's ultimate output is just a weighted sum of the results obtained by the individual weak classifiers [21].

D. Classification Algorithm

1) Decision Tree (DT): DT is a predictive modeling algorithm that can partition features in a dataset in multiple ways based on different conditions, resulting in a tree-like structure. The tree is made up of a decision node, which seeks the best feature split, and leaves (termina l nodes), which are used to create a final detection. Different criteria, such as gini and entropy, are used to divide the features. One significant consideration while constructing terminal nodes for the decision tree is deciding when to stop developing trees and when to create more terminal nodes. This can be done using two criteria: maximum tree depth (the number of nodes in the tree after the root node) and minimum node records (the number of training patterns represented by a given node). Once a node is built, the same method can be used on all created data by dividing the dataset into subsets to produce child nodes. The detection process entails selecting the appropriate node in a decision tree and then proceeding to walk down it with the relevant row of data.

III. Literature Review

Several SDD studies had recently been conducted to accurately detect early-stage developmental defects. Researchers had attempted to improve the model's performance using various methods, algorithms, and measurements. Some of the most current papers relevant to this work are discussed here.

Benala and Tantati [22] examined the effects of five oversampling methods to solve the imbalanced data, including random oversampling, SMOTE, adaptive synthetic sampling (ADASYN), Safe-Level-SMOTE (SL-SMOTE), and SVM-SOMTE, to for SDD using nine imbalanced NASA datasets. They used decision tree (j48-classifier), the RF, Naive Bayes (NB), and EL classifiers. They concluded that SVM-SMOTE is the best oversampling technique due to its ability to produce minority-class instances in a region bordered by support vectors. Kassaymeh et al. [23] suggested a combination of Salp Swarm Algorithm (SSA) and Backpropagation Neural Network (BPNN) to solve the SDD problem. They used BPNN to find the best BPNN parameters. Different performance measures are used to evaluate the results, in which the results of their experiments showed that the combined work is often the best way to solve SDD problems. Goyal [24] proposed a filtering method, called FILTER to accurately predict defects. They used SVM-based classifiers (linear, polynomial, and radial basis functions) and a suggested filtering technique. They declared that their work improves the accuracy, AUC, and F-measure by 16.73%, 16.80%, and 7.65%, respectively, based on five datasets. Azzeh et al. [25] studied the effect and consistency of four kernel functions with feature selection on the performance of SVM for SDD. Four kernel functions, 10 feature subset selection thresholds based on the information gain technique, thirty-eight publicly available datasets, and a single evaluation measure were used in this comprehensive study. Since then, 1520 experiments had been conducted. Since the performance of other kernel functions is constrained, the results showed that SVM with an RBF kernel is the best option for defective datasets. Sharma et al. [26] analyzed the application of the ensemble method of ML technique in the field of SDD. They focused on the global state during the period 2018–2021, which had been examined from a multidimensional perspective, including the selection of a specific ML algorithms, and the research gap that may lead to the future scope of the work that can be accomplished. Ye et al. [27] developed a multi-objective immunity optimization method based on a thorough fitness evaluation mechanism, which allows it to efficiently tackle the used model. Two objectives are optimized: defect detection rate and false alarm rate for defects. Their proposed algorithm is based on comprehensive fitness evaluation, which has a better selection ability to attain the predicted effect of population evolution software and further assists decision makers in selecting a better scheme that meets their needs. In addition, to validate the efficacy of their proposed algorithm, they compared it against eight distinct public data sets, in which the results showed that their suggested work handles the multi-objective undersampling SDD problem more effectively. Shafiq et al. [28] developed an approach for SDD using ML to enhance software quality. They used PC1 data set as input data. Ant Colony Optimization (ACO) is used to determine which characteristics are the most crucial. The chosen characteristics are fed into SVM. The author declared that their results showed that ACO-based SVM performs better than SVM, NB, and KNN classifiers in solving SDD.

IV. Proposed Approach

The proposed approach in this work classifies the defects that exist in the software system and discovers the defect modules during the software development process, so the model that suffers from defects have high priority during quality assurance checks. The framework of the proposed approach in this work is shown in Fig. 1, which is logically divided into the following four phases. The preprocessing, which is Phase I, includes the standardization features and label encoding target features. Phase-II includes oversampling the dataset to balance the class distribution by applying SMOTE-SVM and hyperparameter tuning optimization using NDSGA-II and the Hyperband approach. Phase-III is focusing on training the SDD models using the balanced dataset and making the detections. Phase IV is concerned with evaluating the performance of SDD models and conducting comparative analyses.

This work aims to develop a high-quality approach to detect bugs in the early stages of the software development life cycle, by balancing the modules affected by defects in the dataset using the high-performance balanced algorithm. The SMOTE oversampling approach uses KNN similarity measures between items to generate synthetic instances in the defect class. This leads to the creation of an unknown number of artificial items that are "similar" to those in the current defect class but do not duplicate them. This algorithm has some disadvantages, such as sample overlap, noise interference, and blindness of neighbor selection. To address these problems, SMOTE is hybridized with SVM, so SVM-SMOTE generates new instances of defect classes near borderlines with SVM to help establish boundaries between classes.

However, the behavior of these algorithms is controlled by a set of parameters that remain static during the training process. The quality of the detection can be improved by fine-tuning these parameters to achieve the best possible results. Here, we apply an optimization algorithm called NDSGA-II to search for and fine-tune the hyperparameters of these algorithms and then pick the best possible parameters. Since NDSGA-II is resource-intensive and time-consuming algorithm, it has been integrated with Hyperband approach, to speed up the configuration evaluation by getting rid of ineffective parameters that don't have a global minimum. Testing the model's intermediate scores for a given set of hyperparameters is how it functions. For instance, after a fixed number of rounds, one could examine all the intermediate scores and eliminate the least effective parameters.

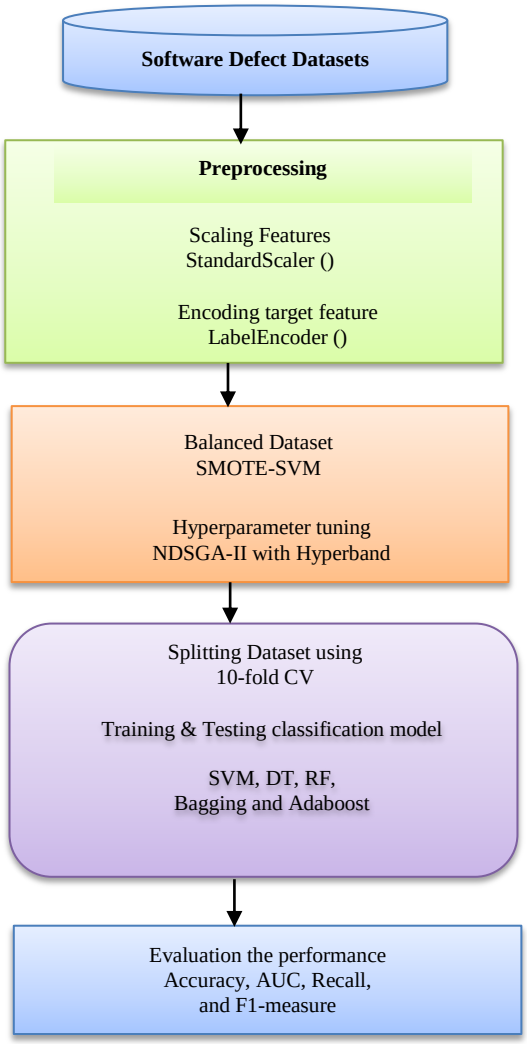

Fig. 1. The proposed approach.

In Fig. 2, the steps of the search optimization technique used to obtain the optimal parameters for the SMOTE-SVM algorithm are illustrated. This process is described in stages as follows:

- Stage 1: Initialize the hyperparameters search space that represents the population (P), which consists of a combination of input variables (V).
- Stage 2: Generate random parent population.
- Stage 3: Apply the SMOTE-SVM-based model described in Algorithm 1 for each variable.
- Stage 4: Each V's accuracy is calculated using 10-fold CV to identify the fitness vectors, and each vector was given a fitness rank proportional to its non-domination frequency.
- Stage 5: Assign ranks to all V in a P by first selecting all of the non-dominated solutions from P and placing them in rank 1, then selecting all of the remaining solutions and placing them in rank 2, and so on.
- Stage 6: Sorts each V according to a dominance rule that said:

A variable ($\ddot{X}$) is said to dominate another variable ($\dot{X}$), if

  - ✓ There is no objective of $\ddot{X}$ worse than that objective of $\dot{X}$.
  - ✓ There is at least one objective of $\ddot{X}$ better than that objective of $\dot{X}$.

- Stage 7: Offspring resulting from recombination between two unrelated parents enter the progeny P. Throughout the process of mutation, the child's values shift. That process is continued until the P is twice as large as it was at the outset.
- Stage 8: Nondenominational criteria are used to re-classify P. In this way, a new generation will be selected according to established hierarchy.
- Stage 9: In the next iteration, crowding-sort will be used to determine the density of solutions if the partially included case holds. Less dense trials are selected for the next generation until the population count is back to its starting point.
- Stage 10: Iteratively producing and checking poorly configured parameters, then discarding their offspring, is repeated until the maximum number of generations is reached, at which point the optimal hyperparameter is returned.

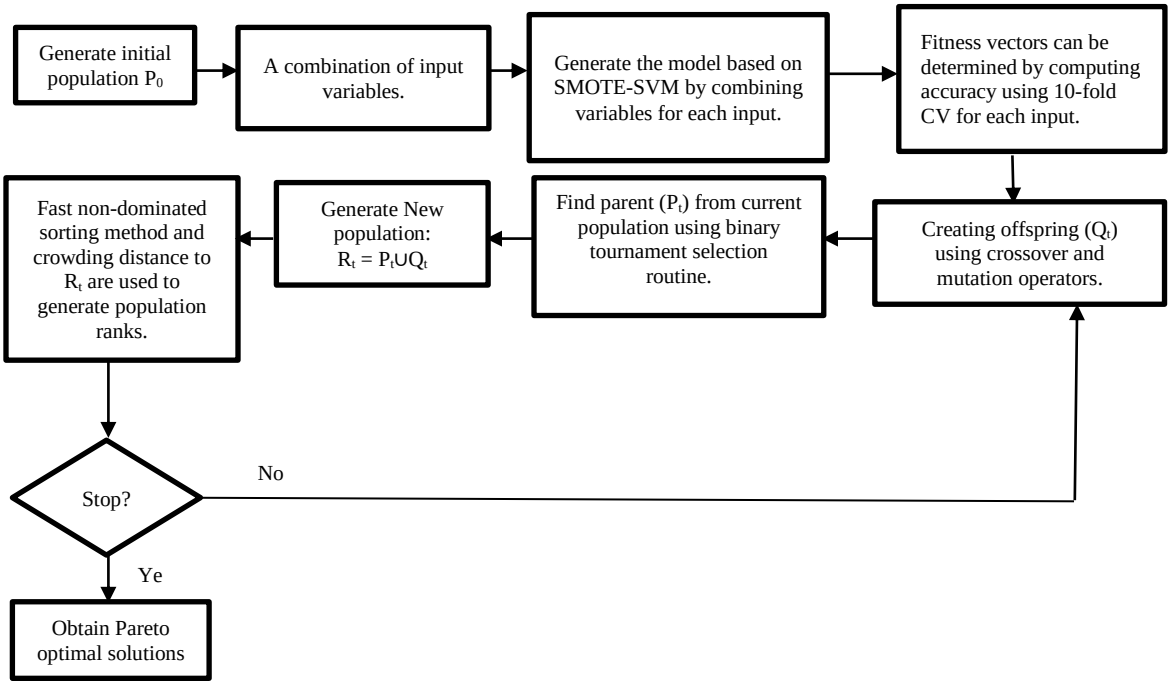

Fig. 2. Main steps of obtaining the optimal parameters of SMOTE-SVM algorithm.

Algorithm 1: *The proposed balancing datasets (SMOTE-SVM-based model)*

Input:
Software defects dataset (Ð)**:**
where:
D = ***Defects*** ∪ ***non_defects***;
Defects *represents instances in the dataset with defect class*
Non_defects *represent instances in the dataset from non_defects class.*
Output: The balanced dataset

1 ***for*** *D'* ∈ Ð ***do***
begin
2 ***X*** = *instances*
y = *target class*
// Encoding target features that contain categorical data
3 *y_Encode = LabelEncoder* (*y*);
// Appling standardization technique to be in the same range
4 *X_Scale = StandardScaler* (*X*);
5 ***for each*** *data point* $<x_i, y_i>$
begin
// <xi,, yi>: the defect instances which denotes the minority class in Ð
6 *Defect support vectors = SVM algorithm* (*defect subsets*, i.e., *minority subsets*);
7 ***m*** = *kNN* (*defect support vector*);
8 ***If*** *number of majority neighbors* $< \frac{m}{2}$
9 $X_{new} = X_i + (\hat{X}i\ ,\ X_i) * \Re$, ***where*** $\Re \in [0,1], Xi \in Smax,$
$\hat{X}i \in$ *KNN of* X_i *, and Smax, is a majority instance*
End if
10 ***Else***
11 $X_{new} = X_i + (\hat{X}i.\ ,\ X_i) * \Re$, ***where*** $\Re \in [0,1], X_i \in S_{min},$
$\hat{X}i \in$ *one of KNN of* X_i, *which* S_{min}, *is minority instance*
12 $\check{D} = X_{new} \cup Ð$
End for
13 ***return*** $\check{D}$
End for

The following paragraphs discuss the SDD model that is presented in Algorithm 1, where the used defective datasets are imbalanced and the output is the balanced dataset. The necessary preprocessing steps are executed for each dataset. Encoding the categorical data in the chosen dataset is represented by yEncode, and xScale represents the features after standardization. The SVM model is applied in the minority subset, and neighbors of the defect support vector are obtained. If the number of non-defects class neighbors is less than half of the nearest defect support vector, a new object of the majority class is generated; otherwise, the new object will be of the minority class.

V. Experimentations and Results

The methodology behind the experiments is discussed here, as well as the infrastructure used and the datasets that were examined.

A. Environment

The proposed approach was tested on a laptop running Microsoft Windows 10 Pro 64-bit and an Intel(R) Core (TM) i7-8565U Processor at 1.80 GHz (92 MHz). Oversampling methods are taken from the imblearn library, and ML classifiers are imported into Sklearn 1.0.2 using an x64 processor running Jupyter notebook 6.1.4 with Python 3.8.5 and OPTUNA (HPO framework).

B. Description to Dataset and Software Metrics used

This work used nine NASA datasets and was investigated by Shepperd et al. [29]. Halstead metrics, McCabe metrics, size metrics, and other properties are included in these datasets to help establish the quality of a software model. If a dataset has bad class values, it probably is bad. The model is fine if the class value is "0" or "no," but it's broken if it's "1" or "yes." Minority class samples range from being extremely under-represented to being evenly distributed across the datasets. The nine NASA datasets are illustrated in Table I.

TABLE I. NASA DATASET DESCRIPTION

Database	# Of Features	# Of Instances	# Of Defective	# Of Non-Defective	Defective %
KC1	22	1183	314	869	15.45
KC2	22	522	104	415	20.5
KC3	40	194	36	158	18.55
CM1	22	344	49	295	12.21
PC1	22	705	77	644	8.03
PC2	37	1585	16	1569	1
MC1	39	1988	46	1942	2.31
MW1	38	253	27	226	10.67
JM1	22	7782	326	1783	21.48

C. Parameter Setting for SDD Models

Table II details the various classifier parameter settings.

TABLE II. PARAMETER SETTINGS OF THE USED ALGORITHMS

Parameters	Values
NDSGA-II	
iterations	100
Population size	Hyperparameter space
Crossover rate	1
Mutation rate	$\frac{1}{generated\ groups}$
SVM	
Kernel type	Rbf
gamma	0.61719
C	16.1657
degree	1
RandomState	104
KNN	
K	8
m	10
DT, RF	
criterion	entropy

D. Evaluation Criteria

The proposed classifiers' efficacy is measured using standard metrics including the confusion matrix, ROC, AUC, accuracy mean, and F-measure.

- Accuracy: It is a ratio of the number of correct detections to the total number of observations, as illustrated in Eq. (6).

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (6)$$

- Precision is the ratio of correct positive detections out of the total number of positive detections as given in Eq. (7)

$$precision = \frac{TP}{TP+FP} \quad (7)$$

- Recall is the ratio of correct positive observations to all positive observations made in class, as given in Eq. (8).

$$recall = \frac{TP}{TP+FN} \quad (8)$$

- F1 score is a weighted average of precision and recall. F1 is usually preferable to accuracy, particularly if the class distribution is uneven, as calculated from Eq. (9).

$$F1\ score = \frac{2*Recall*Precision}{Recall+Precision} \quad (9)$$

- AUC-ROC It shows TPR versus FPR at different thresholds to differentiate "signal" from "noise." It separates classes and summarizes the ROC curve. High AUC means the model separates positive and negative groups well.

E. Results

1) Results of applying the decision tree algorithm: the results of DT for defect detection using the proposed approach are shown in Table III. The best results obtained on the PC2, MC1, and CM1 datasets, respectively, are as follows: 1) the accuracy achieved 98.18%, 95.59%, and 94.28%. 2) The AUC achieved 9.9798, 0.9546, and 0.9466. 3) F-measure achieved 0.9736, 0.936, and 0.9259.

2) Results of applying the Random Forest algorithm: The results of RF for defect detection using the proposed approach are shown in Table IV. The best results obtained on the KC3, MC1, and PC2 datasets, respectively, are as follows: 1) the accuracy achieved 100 %, 99.66 %, and 99.06 %. 2) AUC achieved 1.0, 0.9974, and 0.9868. 3) F-measure achieved 1.0, 0.9950, and 0.9866.

3) Results of applying the Adaboost algorithm: The results of Adaboost for defect detection using the proposed approach are shown in Table V. The best results obtained on the KC3, PC2, and MC1 datasets, respectively, are as follows: 1) The achieved accuracy is 100%, 99.09%, and 97.28%. 2) The achieved AUC 1.0, 0.9868, and 0.9746. 3) The achieved F-measure achieved 1.0, 0.9866, and 0.9611.

TABLE III. RESULTS OF APPLYING DT ALGORITHM

DATASETS	ACCURACY (%)	AUC	RECALL	F1-MEASURE
KC1	88.20	0.8820	0.8707	0.8806
KC2	89.15	0.8922	0.8333	0.8860
KC3	93.548	0.9354	0.93333	0.9333
CM1	**94.285**	**0.9466**	**0.9615**	**0.9259**
PC1	90.77	0.9077	0.9126	0.9082
PC2	**98.18**	**0.9798**	**0.9736**	**0.9736**
MC1	**95.59**	**0.9546**	**0.9504**	**0.9365**
MW1	86.66	0.8656	0.9130	0.8749
JM1	83.47	0.8347	0.85535	0.8381

TABLE IV. RESULTS OF APPLYING RF ALGORITHM

DATASETS	ACCURACY (%)	AUC	RECALL	F1-MEASURE
KC1	92.415	0.9241	0.9325	0.9247
KC2	95.180	0.9520	0.92857	0.95121
KC3	**100**	**1.0**	**1.0**	**1.0**
CM1	92.857	0.9431	1.0	0.9122
PC1	95.14	0.9514	0.9417	0.9509
PC2	**99.09**	**0.9868**	**0.9736**	**0.9866**
MC1	**99.66**	**0.9974**	**1.0**	**0.9950**
MW1	93.33	0.9328	0.9565	0.9361
JM1	89.80	0.8980	0.89977	0.8982

TABLE V. RESULTS OF APPLYING ADABOOST ALGORITHM

DATASETS	ACCURACY (%)	AUC	RECALL	F1-MEASURE
KC1	82.022	0.82022	0.7640	0.8095
KC2	91.566	0.9155	0.92857	0.91764
KC3	**100**	**1.0**	**1.0**	**1.0**
CM1	88.571	0.90122	0.96153	0.8620
PC1	89.80	0.8980	0.9029	0.8985
PC2	**99.09**	**0.9868**	**0.9736**	**0.9866**
MC1	**97.28**	**0.9746**	**0.9801**	**0.9611**
MW1	88.88	0.8873	0.9565	0.8979
JM1	75.27	0.7527	0.71070	0.74197

4) Results of applying the bagging algorithm: The results of bagging for defect detection using the proposed approach are shown in Table VI. The best results obtained on the MC1, PC2, and KC3 datasets, respectively, are as follows: 1) the accuracy achieved 98.64%, 98.18%, and 96.77%. 2) The achieved AUC is 0.9825, 0.9736, and 0.9666. 3) The achieved F-measure 0.98, 0.9729, and 0.9655.

TABLE VI. RESULTS OF APPLYING BAGGING ALGORITHM

DATASETS	ACCURACY (%)	AUC	RECALL	F1-MEASURE
KC1	91.85	0.9185	0.8876	0.9159
KC2	91.566	0.9160	0.8809	0.9135
KC3	**96.77**	**0.9666**	**0.93333**	**0.9655**
CM1	91.428	0.9239	0.96153	0.89285
PC1	94.17	0.9417	0.9320	0.9411
PC2	**98.18**	**0.9736**	**0.9473**	**0.9729**
MC1	**98.64**	**0.9825**	**0.9702**	**0.98**
MW1	95.555	0.9555	0.9565	0.9565
JM1	88.54	0.8854	0.8747	0.8842

5) Results of applying the svm algorithm: The results of SVM for defect detection using the proposed approach are shown in Table VII. The best results obtained on the PC2, KC2, and MC1 datasets, respectively, are as follows: 1) the accuracy achieved 99.09 %, 98.79 %, and 98.64 %. 2) The AUC achieved 0.9868, 0.9878, and 0.973. 3) F-measure achieved 0.9866, 0.9882, and 0.9803.

TABLE VII. RESULTS OF APPLYING SVM ALGORITHM

DATASETS	ACCURACY (%)	AUC	RECALL	F1-MEASURE
KC1	87.92	0.8792	0.8707	0.8781
KC2	**98.79**	**0.9878**	**1.0**	**0.9882**
KC3	93.54	0.9354	0.9333	0.9333
CM1	91.42	0.9160	0.92307	0.8888
PC1	97.57	0.9757	0.9805	0.9758
PC2	**99.09**	**0.9868**	**0.9736**	**0.9866**
MC1	**98.64**	**0.9873**	**0.9900**	**0.9803**
MW1	95.55	0.9565	0.9130	0.9545
JM1	82.16	0.8216	0.8018	0.8181

VI. RESULT ANALYSIS AND DISCUSSION

Different experiments were conducted to find out how the oversampling and HPO techniques affected the performance of SDD models. We first investigated how well SDD models with NDSGA-II hybrid using Hyperband approach for HPO and SMOTE-SVM balanced datasets performed. Tables III to VII display the results for various performance measures for the available NASA software defect dataset. Therefore, it can be asserted that we have successfully implemented all the classification algorithms with high performance. From these tables, it is clear that RF is the most accurate method to classify defects, while Adaboost is the least accurate of all evaluation measures. With KC3 dataset, Adaboost and RF classifier were able to achieve 100% accuracy. Table VIII presents the accuracy results of the proposed work with and without parameter tuning on different datasets, to illustrate the influence of tuning these parameters on the model's performance. Also, Fig. 3 and Fig. 4 show the comparison between these methods. These results show that tuning the algorithm's parameters helped make the models more accurate than the other methods which don't tune the parameters.

TABLE VIII. ACCURACY MEASURE FOR ALL CLASSIFIERS WITH AND WITHOUT HYPERPARAMETER TUNING

DATASETS	ACCURACY WITHOUT PARAMETER TUNING (%)		ACCURACY WITH PARAMETER TUNING (%)	
KC1	SVM	77.52	SVM	87.92
	DT	85.95	DT	88.20
	RF	91.57	RF	92.415
	ADABOOST	80.33	ADABOOST	82.022
	BAGGING	91.01	BAGGING	91.85

KC2	SVM	85.54	SVM	98.79
	DT	89.15	DT	89.15
	RF	96.38	RF	95.180
	ADABOOST	91.566	ADABOOST	91.566
	BAGGING	90.36	BAGGING	91.566
KC3	SVM	90.32	SVM	93.54
	DT	90.32	DT	93.548
	RF	100	RF	100
	ADABOOST	90.32	ADABOOST	100
	BAGGING	96.77	BAGGING	96.77
CM1	SVM	79.72	SVM	91.42
	DT	85.13	DT	94.285
	RF	89.18	RF	92.857
	ADABOOST	81.08	ADABOOST	88.571
	BAGGING	93.24	BAGGING	91.428
PC1	SVM	79.12	SVM	97.57
	DT	91.74	DT	90.77
	RF	95.14	RF	95.14
	ADABOOST	91.26	ADABOOST	89.80
	BAGGING	94.17	BAGGING	94.17
PC2	SVM	93.27	SVM	99.09
	DT	97.47	DT	98.18
	RF	98.31	RF	99.09
	ADABOOST	96.63	ADABOOST	99.09
	BAGGING	96.63	BAGGING	98.18

MC1	SVM	89.9	SVM	98.64
	DT	96.84	DT	95.59
	RF	99.05	RF	99.66
	ADABOOST	96.21	ADABOOST	97.28
	BAGGING	98.73	BAGGING	98.64
MW1	SVM	77.77	SVM	95.55
	DT	88.88	DT	86.66
	RF	86.66	RF	93.33
	ADABOOST	80	ADABOOST	88.88
	BAGGING	86.66	BAGGING	95.555
JM1	SVM	69.23	SVM	82.16
	DT	82.1	DT	83.47
	RF	89.51	RF	89.80
	ADABOOST	75.78	ADABOOST	75.27
	BAGGING	87.92	BAGGING	88.54

These results also demonstrated that the proposed method outperformed the method proposed in [22] that also used SMOTE-SVM for balanced defect datasets by an average of 10.59% with DT, 8.0246% with RF, 2.25% with Adaboost, and 14.8276% with bagging. This is due to the ability of the proposed algorithm with using the optimal parameters, shown in Table II, obtained from applying NDSGA-II algorithm. Hyperband approach [17] is used in NDSGA-II to reduce the time and save other used resources.

Table IX compares the method described in this paper to several other methods for detecting software defects.

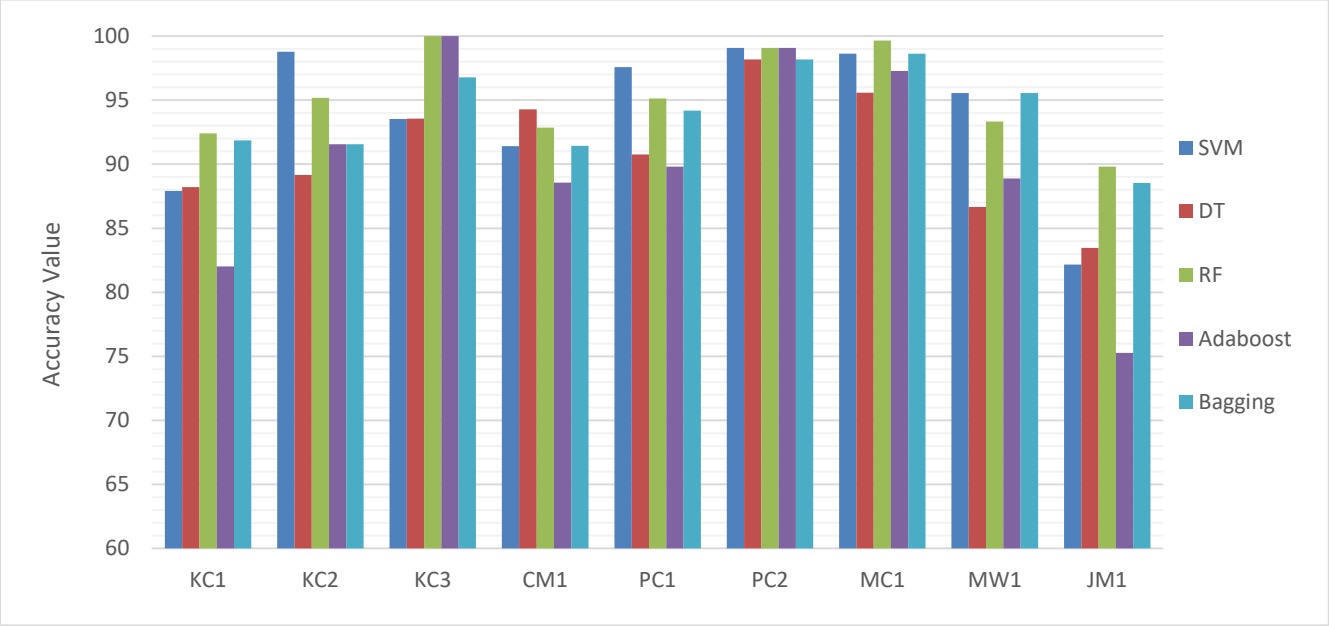

Fig. 3. Comparison accuracy values of five classifiers based on the proposed approach.

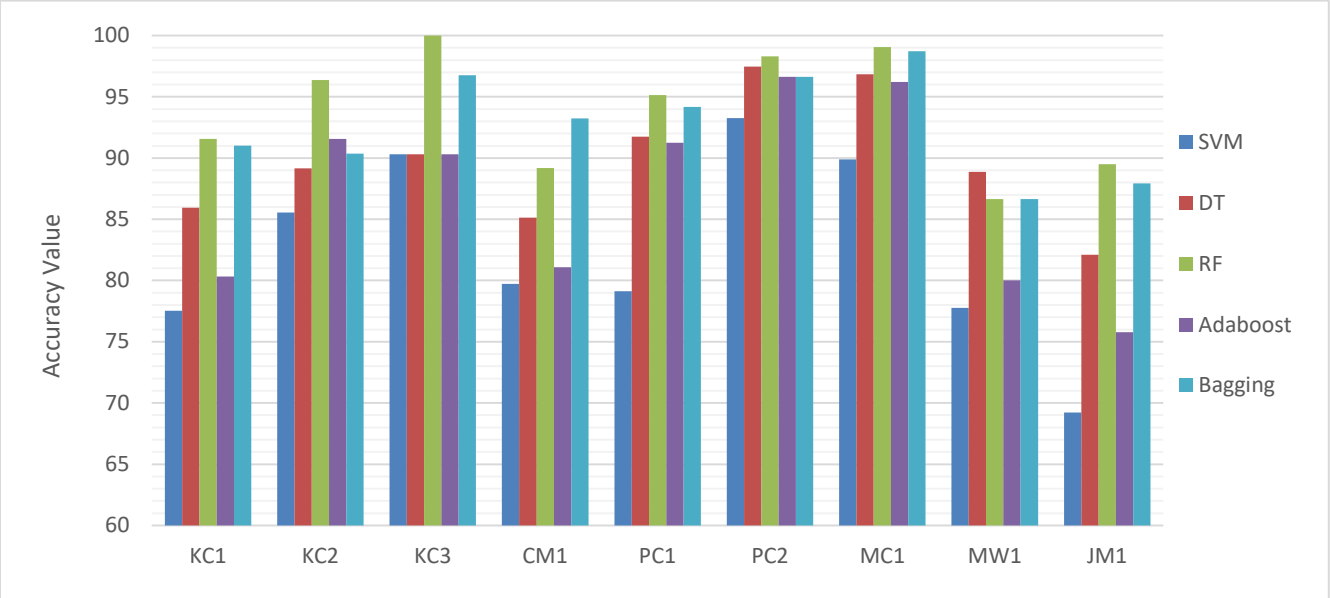

Fig. 4. Comparison accuracy values of five classifiers based on the proposed approach without tuning algorithm parameters.

TABLE IX. PERFORMANCE COMPARISON OF MULTIPLE METHODS FOR SDD ON NASA DATASET

Ref.	METHOD	CLASSIFIER	DATASET	ACCURACY	AUC	F-MEASURE
[22]	SMOTE-SVM	DT	KC2	80		
			KC3	87		
			CM1	84		
			PC1	73		
			PC2	79		
			MC1	85		
			MW1	88		
			JM1	68		
		RF	JM1	80		
			KC2	82		
			KC3	79		
			CM1	89		
			PC1	95		
			PC2	88		
			MC1	97		
			MW1	88		
		Adaboost	JM1	80		
			KC2	82		
			KC3	79		
			CM1	86		
			PC1	95		
			PC2	97		
			MC1	97		
			MW1	88		
		Bagging	KC1	68		
			KC2	78		
			KC3	64		
			CM1	87		
			PC1	91		

			PC2	79		
			MC1	85		
			MW1	82		
[23]	SSA-BPNN		KC1	88.92	0.79	
			KC2	88.54	0.85	
			KC3	93.48	0.92	
			CM1	88	0.85	
			PC1	90.69	0.79	
			PC2	99.64	0.93	
			JM1	82.03	0.70	
			MW1	94.21	0.93	
	Proposed model	DT	KC1	88.20	0.8820	0.8806
			KC2	89.15	0.8922	0.8860
			KC3	93.548	0.9354	0.9333
			CM1	94.285	0.9466	0.9259
			PC1	90.77	0.9077	0.9082
			PC2	98.18	0.9798	0.9736
			MC1	95.59	0.9546	0.9365
			MW1	86.66	0.8656	0.8749
			JM1	83.47	0.8347	0.8381
		RF	KC1	92.415	0.9241	0.9247
			KC2	95.180	0.9520	0.95121
			KC3	100	1.0	1.0
			CM1	92.857	0.9431	0.9122
			PC1	95.14	0.9514	0.9509
			PC2	99.09	0.9868	0.9866
			MC1	99.66	0.9974	0.9950
			MW1	93.33	0.9328	0.9361
			JM1	89.80	0.8980	0.8982
		Adaboost	KC1	82.022	0.9185	0.9159
			KC2	91.566	0.9160	0.9135
			KC3	100	0.9666	0.9655
			CM1	88.571	0.9239	0.89285
			PC1	89.80	0.9417	0.9411
			PC2	99.09	0.9736	0.9729
			MC1	97.28	0.9825	0.98
			MW1	88.88	0.95553	0.9565
			JM1	75.27	0.8854	0.8842
		Bagging	KC1	91.85	0.9185	0.9159
			KC2	91.566	0.9160	0.9135
			KC3	96.77	0.9666	0.9655
			CM1	91.428	0.9239	0.89285
			PC1	94.17	0.9417	0.9411
			PC2	98.18	0.9736	0.9729
			MC1	98.64	0.9825	0.98
			MW1	95.555	0.95553	0.9565
			JM1	88.54	0.8854	0.8842

VII. Conclusion and Future Work

In this work, a new software defect detection (SDD) approach is proposed and developed as an efficient and smart way to find software defects. This approach uses SMOTE-SVM algorithm, to address the issue of imbalanced behavior in NASA datasets. The proposed method used NDSGA-II algorithm with Hyperband approach for hyperparameter optimization of SMOTE-SVM algorithm, followed by using standard ML methods and ensemble techniques for training. The experimental results were assessed using NASA datasets, in which the results showed that our proposed work outperforms the conventional techniques and methods in predicting software faults based on accuracy, AUC, recall, and F-measures. Also, the results showed that RF performed the best with 95.2746% average accuracy, while Adaboost performed the lowest with 90.2754% average accuracy. As future work, we plan to investigate the impact of using deep learning on the improvement of SDD when imbalanced data is used. Also, we plan to use other techniques for HPO, and other assessment measures such as G-measure, balance, and Matthews' Correlation Coefficient (MCC).

References

[1] G¨orkem Giray, Kwabena Ebo Bennin, ¨Omer K¨oksal, ¨Onder Babur, and Bedir Tekinerdogan. On the use of deep learning in software defect prediction. Journal of Systems and Software, 195:111537, 2023.

[2] Wei Zheng, Tianren Shen, Xiang Chen, and Peiran Deng. Interpretability application of the just-in-time software defect prediction model.Journal of Systems and Software, 188:111245, 2022.

[3] Amal Alazba and Hamoud Aljamaan. Software defect prediction using stacking generalization of optimized tree-based ensembles. Applied Sciences, 12(9):4577, 2022.

[4] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing,415:295–316, 2020.

[5] Tirimula Rao Benala and Karunya Tantati. Efficiency of oversampling methods for enhancing software defect prediction by using imbalanced data. Innovations in Systems and Software Engineering, pages 1–17,2022.

[6] Ruchika Malhotra and Shine Kamal. An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data. Neurocomputing, 343:120–140, 2019.

[7] Shujuan Wang, Yuntao Dai, Jihong Shen, and Jingxue Xuan. Research on expansion and classification of imbalanced data based on smote algorithm. Scientific reports, 11(1):1–11, 2021.

[8] Hien M Nguyen, Eric W Cooper, and Katsuari Kamei. Borderline over-sampling for imbalanced data classification. International Journal of Knowledge Engineering and Soft Data Paradigms, 3(1):4–21, 2011.

[9] Jian Zhang, Rong Jin, Yiming Yang, and Alexander Hauptmann. Modified logistic regression: An approximation to svm and its applications in large-scale text categorization. 2003.

[10] Li Yang. Comprehensive visibility indicator algorithm for adaptable speed limit control in intelligent transportation systems. PhD thesis,University of Guelph, 2018.

[11] Omar S Soliman and Amira S Mahmoud. A classification system for remote sensing satellite images using support vector machine with non-linear kernel functions. In 2012 8th International Conference on Informatics and Systems (INFOS), pages BIO–181. IEEE, 2012.

[12] Ga¨el Varoquaux, Lars Buitinck, Gilles Louppe, Olivier Grisel, Fabian Pedregosa, and Andreas Mueller. Scikit-learn: Machine learning without learning the machinery. GetMobile: Mobile Computing and Communications, 19(1):29–33, 2015.

[13] Ankita Golchha and Shahana Gajala Qureshi. Non-dominated sortinggenetic algorithm-ii–a succinct survey. International Journal of Computer Science and Information Technologies, 6(1):252–255, 2015.

[14] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. arXiv preprint arXiv:1502.02127, 2015.

[15] Si Zhang, Jie Xu, Edward Huang, and Chun-Hung Chen. A new optimal sampling rule for multi-fidelity optimization via ordinal transformation.In 2016 IEEE International Conference on Automation Science and Engineering (CASE), pages 670–674. IEEE, 2016.

[16] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In International Conference on Machine Learning, pages 1238–1246. PMLR, 2013.

[17] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh,and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research, 18(1):6765–6816, 2017.

[18] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Automated machine learning: methods, systems, challenges. Springer Nature, 2019.

[19] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. Machine learning, 63:3–42, 2006.

[20] Leo Breiman. Bagging predictors. Machine learning, 24:123–140,1996.

[21] Yoav Freund. Boosting a weak learning algorithm by majority. Information and computation, 121(2):256–285, 1995.

[22] Tirimula Rao Benala and Karunya Tantati. Efficiency of oversampling methods for enhancing software defect prediction by using imbalanced data. Innovations in Systems and Software Engineering, pages 1–17, 2022.

[23] Sofian Kassaymeh, Salwani Abdullah, Mohammed Azmi Al-Betar, and Mohammed Alweshah. Salp swarm optimizer for modeling the software fault prediction problem. Journal of King Saud University-Computer and Information Sciences, 34(6):3365–3378, 2022.

[24] Somya Goyal. Effective software defect prediction using support vector machines (svms). International Journal of System Assurance Engineering and Management, 13(2):681–696, 2022.

[25] Mohammad Azzeh, Yousef Elsheikh, Ali Bou Nassif, and Lefteris Angelis. Examining the performance of kernel methods for software defect prediction based on support vector machine. Science of Computer Programming, 226:102916, 2023.

[26] Tarunim Sharma, Aman Jatain, Shalini Bhaskar, and Kavita Pabreja. Ensemble machine learning paradigms in software defect prediction. Procedia Computer Science, 218:199–209, 2023.

[27] Ye, T., Li, W., Zhang, J. and Cui, Z., 2023. A novel multi-objective immune optimization algorithm for under sampling software defect prediction problem. Concurrency and Computation: Practice and Experience, 35(4), p.e7525.

[28] Muhammad Shafiq, Fatemah H Alghamedy, Nasir Jamal, Tahir Kamal, Yousef Ibrahim Daradkeh, and Mohammad Shabaz. Scientific programming using optimized machine learning techniques for software fault prediction to improve software quality. IET Software, 2023.

[29] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect datasets. IEEE Transactions on Software Engineering, 39(9):1208–1215, 2013.