Egyptian Informatics Journal 24 (2023) 100386

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Full length article

Reliable prediction of software defects using Shapley interpretable machine learning models

Yazan Al-Smadi [a], Mohammed Eshtay [b], Ahmad Al-Qerem [a], Shadi Nashwan [c,*], Osama Ouda [c], A.A. Abd El-Aziz [d,e]

[a] Department of Computer Science, Faculty of Information Technology, Zarqa University, Zarqa 13110, Jordan
[b] Luminus Technical University College, Amman 11118, Jordan
[c] Computer Science Department, Computer and Information Sciences College, Jouf University, Sakaka 72388, Saudi Arabia
[d] Information System Department, Computer and Information Sciences College, Jouf University, Sakaka 72388, Saudi Arabia
[e] Information Systems & Technology Department, Faculty of Graduate Studies for Statistical Research, Cairo UNI, Giza, Egypt

ARTICLE INFO

Keywords:
Software Defect Prediction
Feature importance
Machine learning
Model interpretation
Shapley Additive Explanation

ABSTRACT

Predicting defect-prone software components can play a significant role in allocating relevant testing resources to fault-prone modules and hence increasing the business value of software projects. Most of the current software defect prediction studies utilize traditional supervised machine learning algorithms to predict defects in software applications. The software datasets utilized in such studies are imbalanced and therefore the reported results cannot be reliably used to judge their performance. Moreover, it is important to explain the output of machine learning models employed in fault-predication techniques to determine the contribution of each utilized feature to the model output. In this paper, we propose a new framework for predicting software defects utilizing eleven machine learning classifiers over twelve different datasets. For feature selection, we employ four different nature-inspired search algorithms, namely, particle swarm optimization, genetic algorithm, harmony algorithm, and ant colony optimization. Moreover, we make use of the synthetic minority oversampling technique (SMOTE) to address the problem of data imbalance. Furthermore, we utilize the Shapley additive explanation model for highlighting the highest determinative features. The obtained results demonstrate that gradient boosting, stochastic gradient boosting, decision trees, and categorical boosting outperform others tested model with over 90% accuracy and ROC-AUC. Additionally, we found that the ant colony optimization technique outperforms the other tested feature extraction techniques.

1. Introduction

Nowadays, the globe has seen tremendous growth and application of technology in a variety of essential industries. As a consequence, software quality is the most critical part of software system development. Software testing, on the other hand, is a well-known quality assurance activity that aims to evaluate the internal and external quality aspects of a given software module, subsystem, and component using predefined criteria under certain conditions [1]. The process of ensuring high software quality is inextricably linked to the presence of software flaws. However, a reliable software system is deemed to be good [2]. For many years, artificial intelligence (AI) technologies [3] were used to face several challenges in software quality aspects, including forecasting software flaws at early stages, software size and cost prediction, and software development efforts.

Software organizations are seeing significant expansion and an increase in the demand for software projects in numerous medical, military, and industrial domains. However, to our best knowledge, it is difficult to produce faultless software projects. Therefore, the process of detecting and controlling software defects is critical to the project's success or failure. A software defect, according to the International Software Testing Qualifications Board (ISTQB) [4], is a problem in a specific software component or system that causes the component or system to fail to execute its necessary functions. Most software defects are caused by fundamental development flaws such as insufficient coding and design experience, confusing software requirements and documentation, insufficient test cases, and poor team communication [5].

* Corresponding author.
E-mail address: shadi_naswan@ju.edu.sa (S. Nashwan).

https://doi.org/10.1016/j.eij.2023.05.011
Received 10 October 2022; Received in revised form 29 April 2023; Accepted 24 May 2023
Available online 31 July 2023

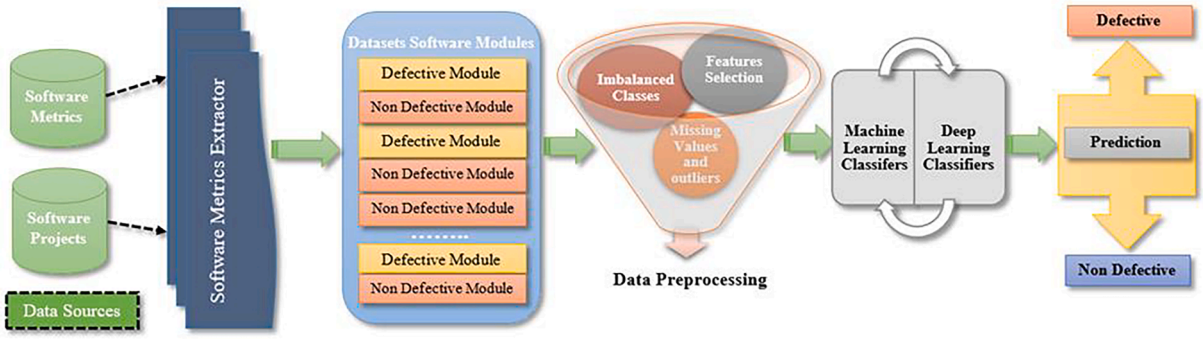

Fig. 1. General Software Defects Prediction Process.

Software companies rely heavily on the process of detecting software defects. However, for long years, companies manually spotted faults by classifying reported problems as defects and then utilizing test cases to determine if the problem was replicated, delayed, or not [6]. Nonetheless, because of the extended cycle, this strategy is costly, needs a lot of human labor and time, and relies heavily on developers' experience [7].

The practice of finding defects in specific software modules using machine learning theories and methodologies is known as software defect prediction [8]. Machine learning, on the other hand, is an artificial intelligence subfield that aims to offer computers the capacity to learn tasks by fitting mathematical functions to large amounts of data to generate various rules for data prediction and classification [9]. However, predicting software defects, in contrast to manual approaches, is an automated approach that, in its early stages, has the advantages of improving software design structure, improving the testing process, lowering development costs, providing a powerful approach for software planning and scheduling, and increasing software reliability [10].

In general, as shown in Fig. 1, the process of predicting software defects involves numerous consecutive steps [11]. The approach begins by strongly relying on the usage of open-source software projects [12] or software metrics [13] as employed by prior researchers. However, software metrics extraction technologies such as CodeMR may be employed with software projects [14]. Data preprocessing, on the other hand, is a critical stage in machine learning workflows; it is described as the process of repairing data from missing values (e.g., null values), cleaning noisy data, balancing data, feature selection, and scaling data towards more significant practices [15]. Finally, fitting machine learning and deep learning approaches may be used to make predictions.

Several researchers proposed alternative ways of predicting software defects over many years. Deep learning approaches [16], decision trees [17], random forest [18], and boosting ensemble-based methods [19] are among the approaches used. In this paper, we propose a new framework for predicting software defects. The proposed methodology evaluates the use of eleven machine learning classifiers, including logistic regression (LR), K-Nearest neighbors (KNN), decision trees (DT), random forests (RF), support vector machine (SVM), adaptive boosting (AdaBoost), gradient boosting (GB), stochastic gradient boosting (SGB), extreme gradient boosting (XGBoost), categorical boosting (CatBoost), and stacked-generalization over twelve datasets from the National Aeronautics and Space Administration (NASA). In addition, four nature-inspired algorithms are used for feature selection. Particle swarm optimization (PSO), genetic algorithm (GA), harmony algorithm (HA), and ant colony optimization (ACO). However, due to the NASA dataset distribution, we use the synthetic minority oversampling technique (SMOTE) to balance the data. Also, we apply the Shapley additive explanation (SHAP) library to explore how much a single data sample and a series of sequential observations contribute to the prediction process.

The key contributions are as follows: (1) Employing several old and modern statistical, bagging, boosting, and stacking classifiers to compare their performance; (2) Applying oversampling techniques to compensate for imbalanced classes; (3) Utilizing nature-inspired algorithms for feature selection; (4) Using Shapley additive explanations library to explain the outputs of predictive models and demonstrate how much a single observation may add to the prediction process.

The rest of the paper is structured as follows: section 2 showcases a variety of related studies conducted in the field. Section 3 illustrates the proposed software defects prediction framework, used datasets, feature selection approaches, and employed classifiers. Section 4 shows the used evaluation metrics and hyperparameters tuning process. Section 5 discusses the experimental results and shows classifiers output explanations using SHAP. Finally, the conclusion and future directions are shown in section 6.

2. Related works

Many experimental, investigational, empirical, and comparative research approaches have recently been proposed in the domain of software defect prediction. In [7] using seven datasets obtained from the National Aeronautics and Space Administration, six machine learning classifiers were tested: logistic regression, random forest, naive Bayes, gradient boosting, support vector machine, and neural networks. Moreover, the results showed that neural networks offer the greatest results with an accuracy of 93% while utilizing 10 folds cross-validation.

In [20] deep learning and bio-inspired feature selection methods are being used. Particle swarm optimization (PSO) was used to evaluate neural network performance by reducing strongly correlated features. In addition, four NASA datasets were gathered and preprocessed with min–max scaling. When considering the results, an area under the curve (AUC) of 0.92 was found to outperform others. However, the problem of imbalanced data distribution remains. In [21] firefly and wolf bio-inspired search-based algorithms were employed to select the most relevant features. Furthermore, utilizing the largest NASA datasets: JM1, PC5, and MC1, two classifiers were evaluated: support vector machine and random forest. However, using the Waikato Environment for Knowledge Analysis (WEKA), the results revealed that employing a support vector machine with a wolf algorithm yields the best results with an accuracy of 99.3% when compared to others. Also, a receiver operating characteristic (ROC) of 51% demonstrated that the classifiers continue to execute random classification. Besides that, imbalanced classes were not remedied. The study in [22] investigated the use of four feature selection filter techniques and fourteen search-based algorithms based on correlation and consistency feature subset selection to assess the performance of four classifiers: Naive Bayes, decision trees, logistic regression, and k-nearest neighbors. In addition, five NASA datasets were gathered and preprocessed. However, the results show that applying filter approaches outperforms others, particularly when leveraging information gain (IG). Also, when compared to other search methods, feature selection approaches based on exhaustive search have the highest effect and consequences. Nonetheless, the classifiers'

Table 1
Summarizes Related Works.

Reference	Datasets	Classifiers	Data Balancing	Meta-Heuristic	Results
[7]	7 NASA Datasets	LR, NB, GB,SVM,RF,ANN	NO	NO	ANN 93% and GB 89.5% validation accuracy.
[20]	4 NASA Datasets	ANN	NO	Binary PSO	Best AUC of 92.9% of overall datasets.
[21]	3 NASA Datasets	SVM, RF	NO	Firefly and gray wolf	SVM and gray wolf yield the best results with an accuracy of 99.3%.
[22]	5 NASA Datasets	NB,DT,LR,KNN	NO	14 search-based algorithms and 4 filter-based methods	Filter methods outperform others, particularly information gain.
[23]	KC1 NASA dataset	SVM,NB,KNN	NO	Firefly	SVM has the best accuracy improvement by 4.53%.
[24]	5 NASA Datasets	FNN, RNN,ANN, DNN	NO	GA and PSO	DNN has the best results compared to others with an accuracy of 98.4%.
[25]	9 NASA Datasets	NB, DT	NO	NO	The findings show that NB accuracy increased by 6.73% and DT increased by 1.87%.
[26]	4 NASA Datasets	NB, ANN, RBF, SVM, KNN,KStar, OneR,DT, RF, PART	NO	NO	The proposed method results in 91.86% accuracy, 75% F-measure, and 66% of MCC.
[27]	10 NASA Datasets	RF,LG,DT,SVM	SMOTE	NO	RF and RF with AdaBoost outperform others with an accuracy of 97%.
[28]	11 NASA Datasets	DT, KNN, ANN,SMO	NO	NO	SMO yields the best results when compared to others with an accuracy of 88.2%.
[29]	9 open-source software projects	LR, KNN, NB	NO	NO	LR has the best outcome with an AUC of 88%.
[30]	4 open-source software project	NB, DT, KNN, Bayesian belief network	NO	NO	Bayesian belief network had the highest accuracy of 80.53%.

performance was unstable and varied from one dataset to another.

Using a firefly search-based algorithm (FA) for feature selection [23]. Three machine learning classifiers were used to evaluate the utility of FA for selecting the best features and enhancing model performance: support vector machine, Naive Bayes, and k-nearest neighbors. When compared to the original findings, the support vector machine resulted in the best results and improved accuracy by 4.53%. The study in [24] proposed a new method comprising the use of several deep learning approaches for software defects prediction utilizing the genetic algorithm (GA) for features selection and particle swarm optimization (PSO) for data clustering. Moreover, five NASA datasets were collected and cleaned from missing values. The results revealed that deep neural networks outperform others with an accuracy of 98.47% using 10 folds cross-validation.

Several research recommends combining numerous feature approaches into a single list rather than using individuals. In [25] Introduced a novel method to handle the problem of high dimensionality data and filter method ranking in software defect prediction by combining various feature filter methods: Chi-square, information gain, and relief. The suggested method compares the outcomes by assessing the effectiveness of decision trees (DT) and Naive Bayes (NB). In addition, nine NASA datasets were used and sanitized. The findings show that the suggested method outperforms the employment of individual methods, with NB accuracy increasing by 6.73% and DT increasing by 1.87%.

In [26] ten machine learning classifiers hyperparameters were fine-tuned and optimized based on several variants to boost the discriminative power. The proposed method employs the use of three directions Best First features subset selection to four NASA datasets. Datasets were preprocessed by resampling and normalizing. However, results were evaluated using three evaluation metrics: F-measure, accuracy, and Matthews's correlation coefficient (MCC). The proposed method results in 91.86% accuracy, 75% F-measure, and 66% of MCC. Using ensemble learning. In [27] Four classifiers: Random forest, decision trees, support vector machine, and logistic regression were set up as base learners using bagging and boosting ensemble-based methods. Moreover, ten NASA datasets were preprocessed and balanced using the synthetic minority oversampling technique. However, no feature selection techniques were performed. The experiment results revealed that RF base learner, RF with AdaBoost, and RF with bagging outperform others with an accuracy of 97%. In [28] boosting, bagging, stacking, and voting ensemble-based methods were applied to four classifiers: DT, KNN, NN, and sequential minimal optimization (SMO). Eleven NASA datasets were also obtained and preprocessed. However, for minimizing highly correlated features, a greedy search-based algorithm was used. The findings of the ensemble learning techniques indicated that boosted SMO yields the best results when compared to others, with an accuracy of 88.2%, and voting produces an accuracy of 87.9%.

Other studies have used open-source software project datasets to predict software defects. In [29] an experimental study was carried out to investigate the usage of three machine learning classifiers: LR, KNN, and NB across nine class-level open-source software projects with 33 distinct versions. Furthermore, a text matching method was used to remove duplication from the datasets. However, utilizing the process metrics, the findings indicated that LR had the highest AUC of 88%. Soumi Ghosh et al., [30] proposed a new method for predicting software defects based on non-linear manifold methods. Apache, eclipse, safe, and zxing open-source software project datasets were collected. Furthermore, four classifiers were used in the experiment: Bayesian belief network, NB, DT, and KNN. When compared to others employing the stochastic proximity embedding non-linear approach, the Bayesian belief network had the highest accuracy of 80.53%. Several studies used various methodologies to predict software defects. Table 1 summarizes the related works. In this paper, we propose a new framework for predicting software defects based on *meta*-heuristics methods and eleven machine learning classifiers. In addition, we employ the SHAP library for model explanations and SMOTE as a resampling technique for data balancing problems.

3. Proposed methodology

In the sections that follow, we will outline the proposed framework for software defect prediction and emphasize the key process and its goals. The designed framework for predicting software defects attempts to improve prediction accuracy and performance by eliminating highly correlated features and addressing the imbalanced data problem. Additionally, the proposed framework's main purpose is to explain the outcomes of several machine learning models. It is useful to understand how much a single observation adds to the prediction process. As evaluation metrics in this work, classification measures such as testing accuracy and ROC-AUC were used. The proposed framework is shown in Fig. 2. To assure data authenticity, the essential processes of dataset preparation were used. The synthetic minority oversampling technique was used for data balancing since it does not replicate data records. Nonetheless, for feature selection, this study employs four

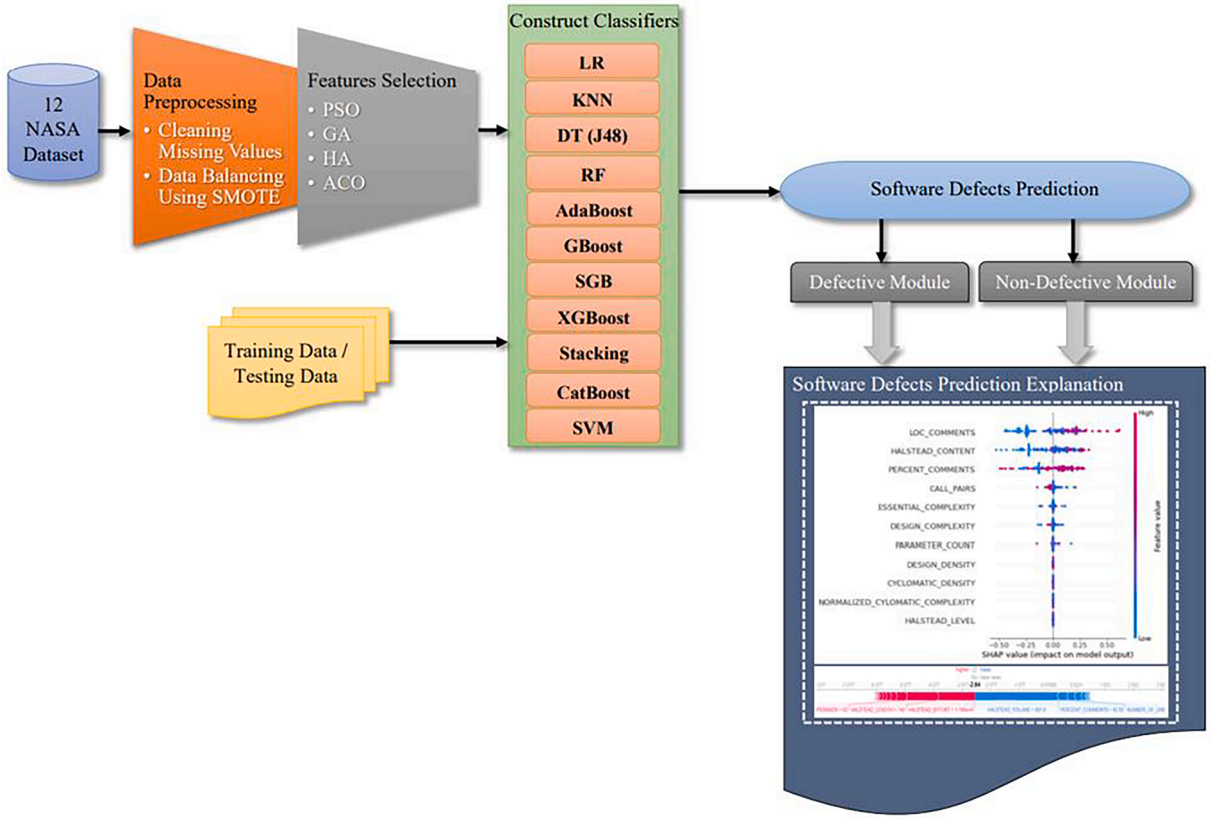

Fig. 2. Overview of the Proposed Software Defects Prediction Framework.

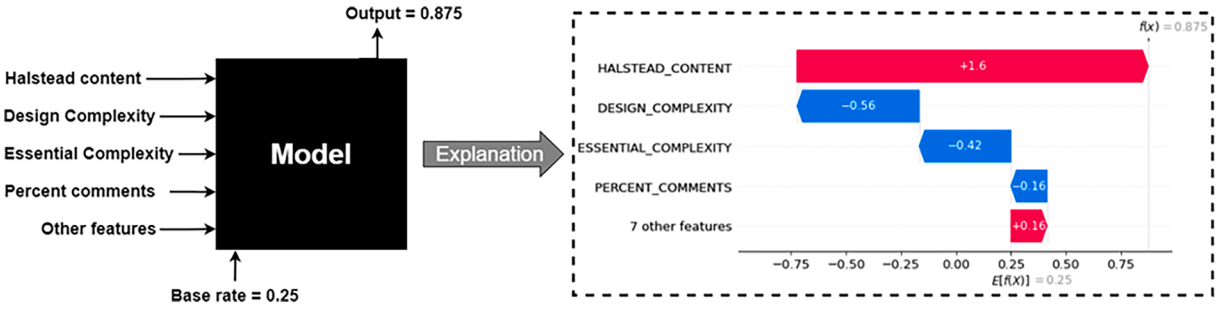

Fig. 3. Shapley additive explanations Model.

metaheuristics search-based algorithms: particle swarm optimization, genetic algorithm, harmony algorithm, and ant colony optimization. On the other hand, eleven machine learning classifiers were constructed and evaluated using testing sets of 20% over twelve NASA datasets. Finally, we employ Shapley additive explanations, a well-designed tool that aids in the explanation of prediction models by highlighting the most valuable features of the predictive models [31]. Also, it is a good visualization tool to show how much a single observation or a set of sequential data samples can affect the prediction process by calculating SHAP values. The SHAP model explanation is shown in Fig. 3.

3.1. Data collection and analysis

The National Aeronautics and Space Administration dataset is made up of thirteen datasets that describe various software components. In this study, we employ twelve NASA MDP datasets to predict software defects. The KC4 dataset, on the other hand, was omitted owing to missing attributes and values. The data were obtained from an online repository, which can be found at https://figshare.com/collections/NASA_MDP_Software_Defects_Data_Sets/4054940/1. The major predictors for defect prediction are software metrics such as McCabe, Halsted, miscellaneous, and lines of code count. Table 2 shows a description of the datasets. Furthermore, the datasets were preprocessed to clean missing values and balance the data. The datasets, however, were devoid of missing values, noisy data, and missing attributes.

3.2. Data balancing

Imbalanced data, also known as imbalanced target classes, is a data distribution problem that causes machine learning classifiers to be biased toward one category over others, negatively affecting prediction [32]. NASA datasets, on the other hand, exhibit a wide distribution gap across target classes. As a result, this study is based on the employment of SMOTE for data balance. SMOTE is an oversampling technique that generates synthetic data samples to increase minority class data samples by locating the nearest K - neighbors, computing the distance between them then multiplying a random value between 0 and 1 for the new sample [33].

Table 2
NASA MDP D' Version Datasets Description.

		NASA MDP D' version datasets											
category	**Metrics**	**CM1**	**JM1**	**KC1**	**KC3**	**MC1**	**MC2**	**MW1**	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**
McCabe	CYCLOMATIC COMPLEXITY	×	×	×	×	×	×	×	×	×	×	×	×
	CYCLOMATIC DENSITY	×			×	×	×	×	×	×	×	×	×
	DESIGN COMPLEXITY	×	×	×	×	×	×	×	×	×	×	×	×
	ESSENTIAL COMPLEXITY	×	×	×	×	×	×	×	×	×	×	×	×
Halsted	CONTENT	×	×	×	×	×	×	×	×	×	×	×	×
	DIFFICULTY	×	×	×	×	×	×	×	×	×	×	×	×
	EFFORT	×	×	×	×	×	×	×	×	×	×	×	×
	ERROR_EST	×	×	×	×	×	×	×	×	×	×	×	×
	LENGTH	×	×	×	×	×	×	×	×	×	×	×	×
	LEVEL	×	×	×	×	×	×	×	×	×	×	×	×
	PROG TIME	×	×	×	×	×	×	×	×	×	×	×	×
	VOLUME	×	×	×	×	×	×	×	×	×	×	×	×
	NUM OPERANDS	×	×	×	×	×	×	×	×	×	×	×	×
	NUM OPERATORS	×	×	×	×	×	×	×	×	×	×	×	×
	NUM UNIQUE OPERANDS	×	×	×	×	×	×	×	×	×	×	×	×
	NUM UNIQUE OPERATORS	×	×	×	×	×	×	×	×	×	×	×	×
Misc. metric	BRANCH COUNT	×	×	×	×	×	×	×	×	×	×	×	×
	CALL PAIRS	×			×	×	×	×	×	×	×	×	×
	CONDITION COUNT	×			×	×	×	×	×	×	×	×	×
	DECISION COUNT	×			×	×	×	×	×	×	×	×	×
	DECISION DENSITY	×			×		×	×	×	×	×	×	
	DESIGN DENSITY	×			×	×	×	×	×	×	×	×	×
	EDGE COUNT	×			×	×	×	×	×	×	×	×	×
	ESSENTIAL DENSITY	×			×	×	×	×	×	×	×	×	×
	PARAMETER COUNT	×			×	×	×	×	×	×	×	×	×
	MAINTENANCE SEVERITY	×			×	×	×	×	×	×	×	×	×
	MODIFIED CONDITION COUNT	×			×	×	×	×	×	×	×	×	×
	MULTIPLE CONDITION COUNT	×			×	×	×	×	×	×	×	×	×
	NODE COUNT	×			×	×	×	×	×	×	×	×	×
	NORMALIZED CYCLOMATIC COMPLEXITY	×			×	×	×	×	×	×	×	×	×
	PERCENT COMMENTS	×			×	×	×	×	×	×	×	×	×
	GLOBAL DATA COMPLEXITY				×	×	×						×
	GLOBAL DATA DENSITY				×	×	×						×
LOC counts	LOC BLANK	×	×	×	×	×	×	×	×		×	×	×
	LOC CODE AND COMMENT	×	×	×	×	×	×	×	×	×	×	×	×
	LOC COMMENTS	×	×	×	×	×	×	×	×	×	×	×	×
	LOC EXECUTABLE	×	×	×	×	×	×	×	×	×	×	×	×
	NUMBER OF LINES	×			×	×	×	×	×	×	×	×	×
	LOC TOTAL	×	×	×	×	×	×	×	×	×	×	×	×
Total number of features	Total number of instances	37	344	42	302	C17	219	593	175	212	096	325	39	200	361	64	Java	8	389	277	689	209	C++	66	391	274	483	C++	6	372	642	723	7C8	377	596	169	8C26	361	585	161	569	C25	371	125	140	985	C36	371	399	178	1221	C30	3817,00150316, 498C++162
	Defective modules						97	834	C457																																											
	Non-defective modules										1771	C++43																																										
	Programming language																																																				
	Lines of code (Thousands)																																																			

3.3. Features selection

The process of selecting the most informative predictors of existing features by removing highly correlated features is known as feature selection. In this paper, we apply four bio-inspired search-based feature selection algorithms.

3.3.1. Particle swarm optimization (PSO)

PSO is a nature-inspired optimization algorithm influenced by birds' behavior. It's a population-based algorithm in which each bird is referred to as a particle and a group of birds is referred to as a swarm (i.e., population). The core notion of PSO is that each particle represents the best solution to the problem. However, following the random initialization, the location of each particle is modified, resulting in new positions: best previous position P_i^K and best global positionP_g^K as shown in Fig. 4. Where *r1* and *r2* represent randomly generated values between [0 and 1]. Also, *c1* and *c2* represent constant values that update the weight of P_i (previous position of *ith* particle) and P_g (previous position of all particles) [34].

3.3.2. Genetic algorithm (GA)

A genetic algorithm is a *meta*-heuristic that is population-based and is used to address optimization problems. It is a chromosome-inspired search-based algorithm. GA passes through the following stages: Initialize the population, use the fitness function, selection, and reproduction [35]. To begin, random individuals are initiated, each of which has a parameter known as a gene, which creates chromosomes. The core idea behind GA is that each individual offers a solution to a problem. However, each individual is evaluated by a fitness score using the fitness function to determine whether or not the individual is suitable for reproduction. Following the selection process, the crossover function can be used to generate a new individual [11]. Fig. 5 shows the GA process.

3.3.3. Harmony search algorithm (HA)

Harmony search is a population-based *meta*-heuristic algorithm. It was inspired by the musical practice of finding the best harmonies. The entire point of HA is to emulate the musicians' approach to discover the optimal solution. However, HA passes through multiple phases, which are as follows: Initialization of parameters, initialization of harmony memory, improvisation of new harmony, and updating harmony memory [36]. Following the initialization of the basic parameters: harmony memory (HM) size, harmony memory consideration rate (HMCR), and pitch adjustment rate (PAR). The harmony memory matrix is designed and filled at random with the values of the possible variables. Each row in the matrix indicates a potential solution. However, relying on HMCR and PAR, additional vectors are generated. Finally, if the new harmony vectors outperform the old ones, they will be preserved as the best solution in the HM [37]. The HA pseudocode is shown in Fig. 6.

3.3.4. Ant colony optimization (ACO)

In several disciplines, ACO is one of the most widely utilized *meta*-heuristic search-based algorithms. It is an improved version of Marco Dorigo's original algorithm, the ant system, which was introduced in 1992 [38]. ACO was inspired by the foraging activity of ant colonies. The goal of ACO is to locate the shortest path between the ant nest and the supplies. Each route shows a possible solution. However, after the pathways have been initialized, and with the assistance of ant pheromones, which are organic chemical substances, ants travel the shortest path to the nest, which represents the ideal solution. Many more models have been used and enhanced with ACO. To solve the challenge of autonomous surface vehicles, Dimitrios [39] suggested an updated version of ACO based on fuzzy logic. The ACO pseudocode is shown in Fig. 7.

3.4. Classification models

In this paper, we propose a new software defect prediction framework comprised of the following eleven machine learning models: Statistical classifiers (logistic regression), K-nearest neighbors approach, support vector machine-based classifiers (Support vector machine), Decision trees approach (J48), and numerous ensemble methods (Stacking, random forest, adaptive boosting, gradient boosting, stochastic gradient boosting, extreme gradient boosting, and categorical boosting).

3.4.1. Logistic regression

A logistic regression model, like a linear regression model, predicts a dependent variable by examining the relationship between independent data variables. However, it may be utilized for true or false binary predictions rather than continuous predictions [40]. Furthermore, it fits an s-shape line using the sigmoid function where$h_\theta(x) = 1/(1 + e^{-[03B8]^T}.x)$, causing the predicted value to range between 0 and 1, giving us the possibility that y = 1, which is true based on the × value. Furthermore, a high × value indicates a high probability that y = 1, but an intermediate × value indicates a 50% likelihood that y = 1 when the anticipated Y equals 0.5.

3.4.2. K-nearest neighbors

KNN [41] is a distance-based method that determines the distance between training and test samples. KNN classifier is a sluggish algorithm since it learns during the evaluation phase and saves data samples during the learning stage. Without standing, the KNN passes through many phases. Begin by determining the value of K (number of nearest samples), then calculate the distance between the training samples and the new data sample using a distance function. Furthermore, the distance values will be sorted and the nearest samples identified based on the value of K, and the prediction value will be determined based on the majority of the class values.

3.4.3. Decision trees

DT [42] is a tree decision-based classifier that divides the qualities of the training data into parts. In DT, predictions are created by making judgments based on a series of questions that are closely relevant to the prediction's aim. It is a non-parametric approach that does not necessitate the use of a mapping function between predictors and target data. The process of picking the optimum decision values at each depth level is referred to as impurity in DT. The Gini-index function, on the other hand, may be used to measure the impurity of the tree.

3.4.4. Random forests

RF [43] is a bagging ensemble-based classifier. As the basic learners, it divides the training data into several decision trees. RF is also a non-parametric model. However, a restricted random number of rows is chosen from the data population, and a set of decision trees is built for each segment to generate classification output. An individual decision tree might be bigger than others. A majority vote in RF indicates a *meta*-classifier, which is a classification decision made based on the judgments of the basic learners. It supports numeric and categorical data types, as well as complicated predictor functions.

3.4.5. Support vector machine

A support vector machine [44] is a classifier that uses hyperplanes to find the best decision boundary in dimensional space based on the number of features. However, because the hyperplane has numerous options, the goal of the support vector machine is to pick the ideal hyperplane with the largest margins among data samples to improve classification accuracy. Additionally, linear and non-linear classification can be done. Generally, a support vector machine performs better and is less prone to overfitting when the class distribution is known. However,

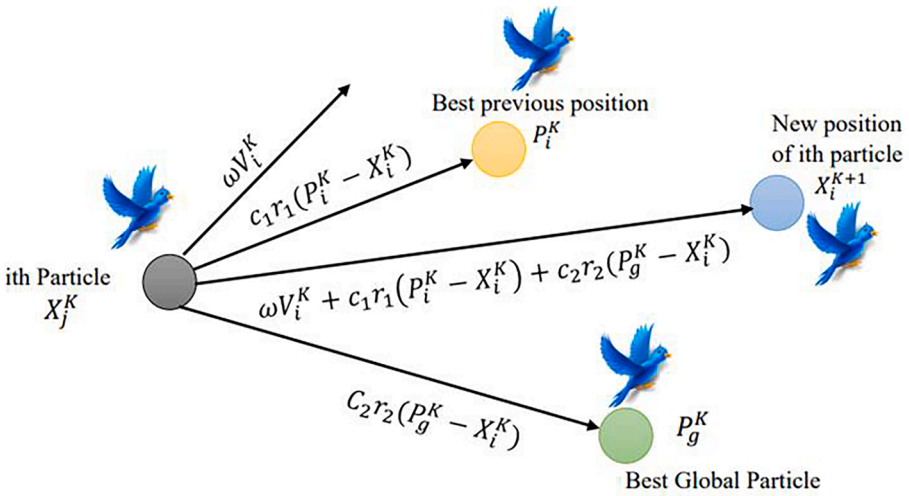

Fig. 4. Particle Swarm Optimization Architecture Illustration.

Genetic Algorithm (GA) Pseudocode

Parameters: S – set of blocks
Output: superstring of set S
Initialization:
$t \leftarrow 0$
Initialize P_t **to random individuals from S**
Evaluate – Fitness – GA (S, P_t**)**
While termination condition is not met Do:
  Select individuals from P_t **(fitness proportionate)**
  Recombine individuals
  Mutate individuals
  Evaluate – Fitness – GA (S, modified individuals)
  $P_{t+1} \leftarrow$ **newly created individuals**
  $t \leftarrow t + 1$
Return (superstring derived from best individuals in P_t**)**
Procedure Evaluate – Fitness – GA (S, P)
  S – set of blocks
  P – population of individuals
For each individual $i \in P$ **Do:**
  Generate derived string $s(i)$
  $m \leftarrow$ **all blocks from S that are not covered by** $s(i)$
  $\acute{s}(i) \leftarrow$ **concatenation of** $s(i)$ **and m**
  $fitness\ (i) \leftarrow \frac{1}{||\acute{s}(i)||^2}$

Fig. 5. Genetic Algorithm Pseudocode.

when maximum margins are employed, new data may be appropriately fit and categorized.

3.4.6. Adaptive boosting

AdaBoost [45] is an ensemble-based boosting classifier. In contrast to decision trees and random forests. Several decision trees are merged independently in AdaBoost. Each of these contains a single node and two leaves, forming what is known as a stump. A stumps forest is a collection of stumps. However, stumps are poor classification learners. AdaBoost's entire concept is to combine numerous weak learners for classification. The sequence of stump creation is critical in AdaBoost since each produced tree influences the output of the following one. Each data sample is weighted (w) to determine the order of stumps where w = 1/Total number of samples. However, sample weight is updated frequently by $sw = w_{old} * e^{amountofsay(\alpha)}$. Also, the Gini-Index may be used to rank stumps, with the lower the Gini value, the more essential the stump. Fig. 8 shows AdaBoost pseudocode.

3.4.7. Gradient boosting

GB [46] is a boosting ensemble-based algorithm for classification and regression that employs the idea of pseudo residuals (PR). Unlike AdaBoost, gradient boosting begins with a single leaf that reflects the average values of anticipated classes rather than a stump. GB, on the other hand, creates a fixed-sized tree, similar to AdaBoost, where each tree can be larger than a stump. Decision trees are used as base learners in GB. The difference between the actual and anticipated values is represented by PR, which may be determined by fitting a linear regression line as a loss function. In GB, prediction errors may be reduced by frequently updating PR values from tree to tree, making it a powerful prediction approach. Fig. 9 shows the GB pseudocode.

Harmony Search Algorithm (HA) Pseudocode

Begin:
Define objective function $f(x) = (x_1, x_2, \ldots, x_d)^T$
Define harmony memory considering rate (HMCR)
Define pitch adjusting rate (PAR) and other parameters
Generate harmony memory with random harmonies
While ($t <$ max $number\ of\ iterations$)
While ($i \leq number\ of\ variables$)
If ($rand < HMCR$)
Choose a value from HM for the variable i
If ($rand < PAR$)
Adjust the value by adding a certain amount
END If
Else
Choose a random value
END If
END while
Accept the new harmony solution if better
END while
Find the current best solution
END

Fig. 6. Harmony Search Algorithm Pseudocode.

Ant Colony Optimization (ACO) Pseudocode

Begin:
Initialize
While stopping criterion not satisfied **do:**
Position each ant in a starting node
Repeat
For each ant **do:**
Choose the next node by applying the state transition rule
Apply step by step pheromone update
END For
Until every ant has built a solution
Update the best solution
Apply offline pheromone update
END While
END

Fig. 7. Ant Colony Optimization pseudocode.

Adaptive Boosting (AdaBoost) Pseudocode

Initially set uniform example weights
For each base learner **do:**
Train base learner with a weighted sample
Test base learner on all data
Set learner weight with a weighted error
Set example weights based on ensemble predictions
END For

Fig. 8. Adaptive Boosting pseudocode.

Gradient Boosting (GB) Pseudocode

Initialize the model with a constant value:

$$f_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma) \qquad L: loss\ function$$

For $m = 1$ **to** M **Do:**

Compute pseudo residuals $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$, $for\ i = 1, \dots, n$

Train regression tree with features x against r and create a terminal node $R_{jm}\ for\ j = 1, \dots, J_m$

Compute $\gamma_{jm} = argmin_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$, $for\ j = 1, \dots, J_m$

Update the model: $f_m(x) = f_{m-1}(x) + v \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$

END For

Fig. 9. Gradient Boosting Pseudocode.

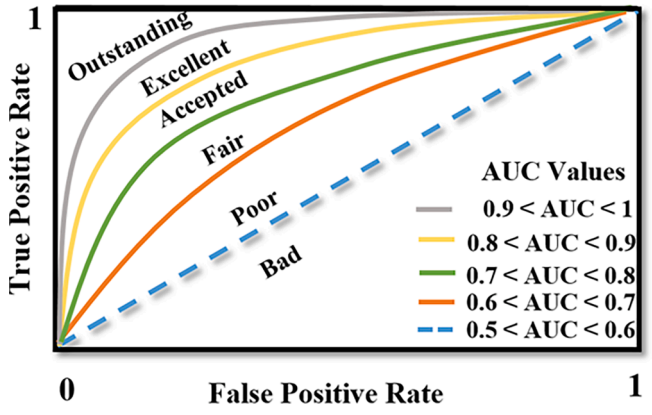

Fig. 10. ROC-AUC ranges and values.

Table 3
Boosting Hyperparameters Grid-Search Space.

Parameters	Grid-Search Space
Number of estimators	50, 70, 90, 100, 150, 120, 180, 200
Learning rate	0.001, 0.01, 0.1, 1, 10
Loss function	Deviance, exponential
Algorithms	SAMME, SAMME.R

3.4.8. Stochastic gradient boosting

Friedman [47] introduced the stochastic gradient boosting approach, which is an ensemble-based method. The hybrid procedure of homogenous (boosting and bagging) approaches inspired it. By weighting the base learners, it may be utilized for regression and classification. Decision trees, on the other hand, are the most widely used SGB base learners. Each tree, on the other hand, is trained by selecting a random number of data sample records. SGB, as compared to bagging methods, is an effective instrument for limiting the chance of overfitting by randomly eliminating a portion of input data [48]. SGB has been utilized in a variety of areas for many years. Sérgio Godinho et al., [49] employed SGB for multi-image classification in their inquiry study.

3.4.9. Extreme gradient boosting

XGBoost [50] is an ensemble method based on boosting. It is a more advanced version of gradient boosting. It was also created for large and intricate datasets. Unlike adaptive and gradient boosting, the XGBoost employs unique regression trees. Forming what is known as born-again trees. XGBoost tree formation, on the other hand, begins with a single leaf, much like gradient boosting. Furthermore, gradient boosting and regularization are critical steps in XGBoost. For many years, XGBoost has been one of the most effective boosting methods. In their experimental investigation, Ismail and Faisal [51] demonstrated that XGBoost outperforms other classifiers such as RF, SVM, radial basis function neural network, and naive Bayes. It is simple to use and provides a high level of discriminative prediction accuracy. In addition to its ability to

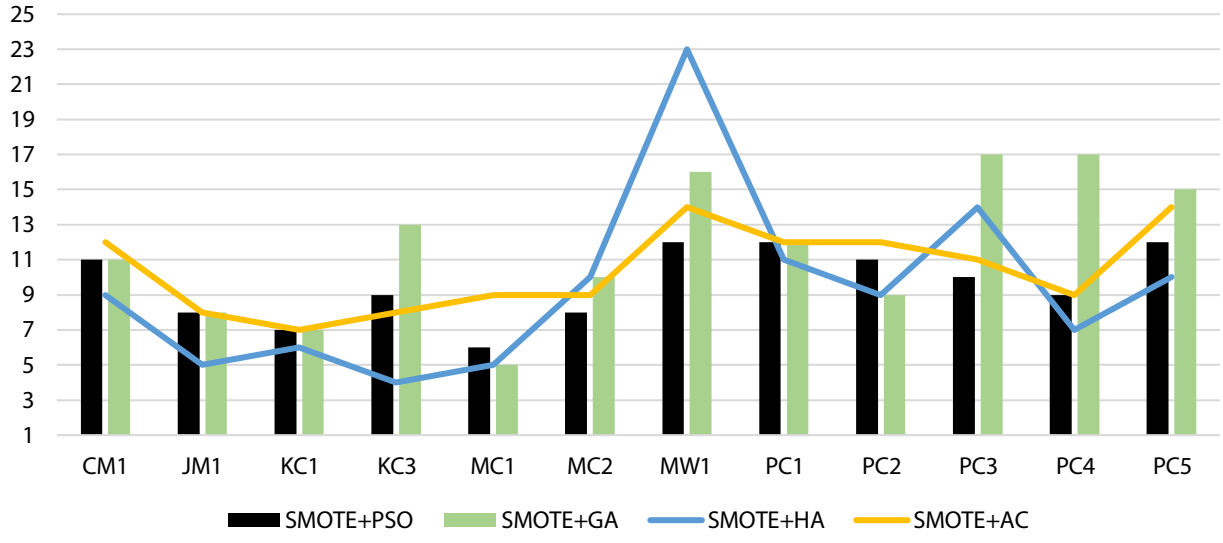

Fig. 11. NASA MDP Selected Features Using Meta-Heuristic Search-Based.

Table 4
Classifiers Testing Accuracy Comparison over CM1 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.826087	0.867188	0.859375	0.789062	0.882812
KNN	0.884058	0.898438	0.875000	0.906250	0.890625
Decision trees	0.869565	0.906250	0.882812	0.914062	0.859375
Random Forest	0.884058	0.906250	0.906250	0.906250	0.906250
Ada Boost	0.884058	0.906250	0.906250	0.906250	0.906250
Gradient Boosting	0.913043	0.937500	0.921875	0.929688	0.937500
SGB	0.942029	0.921875	0.906250	0.937500	0.929688
XGBoost	0.884058	0.820312	0.882812	0.851562	0.632812
Stacking	0.857143	0.875000	0.890625	0.890625	0.890625
Cat Boost	0.884058	0.906250	0.898438	0.898438	0.906250
SVM	0.884058	0.914062	0.890625	0.914062	0.914062
Accuracy (AVG)	0.882	0.896	0.892	0.894	0.877

handle large data distribution with imbalanced target classes.

3.4.10. Categorical boosting

CatBoost is a more advanced kind of gradient boosting. It was proposed in 2018 by Anna Veronika Dorogush and the Yandex corporate team [52]. It was created to be the optimum option for heterogeneous data, as opposed to bagging and stacking. Based on a boosting approach, it was designed to handle categorical and numerical features. CatBoost is an open-source package that allows for extremely rapid graphics processing unit (GPU) and central processing unit (CPU) calculations. Very effective for small datasets and simple to utilize. It is also a decision tree-based method, which has the benefit of decreasing overfitting. CatBoost has long been a rival to other gradient boosting methods. Jengei Hong [53] found that CatBoost outperformed XGBoost and LightGBM implementations in his experimental investigation on predicting house pricing.

3.4.11. Ensemble stacking

Stacking [54] is a heterogeneous ensemble-based method. Stacking, as opposed to bagging and boosting, divides a machine learning model into two layers. Stacking proceeds in numerous stages: Dataset gathering and features extractor, base learner construction, and Meta learner creation. The base layer has N distinct classifiers. To begin, training data is divided into K portions. Each base learner produces a prediction based on K. A Meta dataset, on the other hand, integrates the prediction outcomes of base learners. A Meta classifier is the second level of prediction. It generates prediction output based on previous predictions. In this work, we employ the use of RF and SVM as base learners. Also, we use LG as a Meta classifier.

4. Evaluation metrics and hyperparameters tuning

Several evaluation metrics may be used to evaluate machine learning classifiers, including accuracy, precision, recall, receiver operating characteristics (ROC), F-measure, and area under the curve (AUC). A

Table 5
Classifiers Testing Accuracy Comparison over JM1 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.870766	0.690316	0.700403	0.794889	0.701076
KNN	0.936425	0.972091	0.972764	0.970746	0.972428
Decision trees	0.930693	0.956288	0.954943	0.954270	0.949227
Random Forest	0.915060	0.924681	0.920309	0.912239	0.914257
Ada Boost	0.915060	0.924681	0.920309	0.912239	0.914257
Gradient Boosting	0.955706	0.969401	0.970746	0.970746	0.971083
SGB	0.959354	0.975118	0.975118	0.975454	0.975118
XGBoost	0.818134	0.790182	0.459314	0.828850	0.823134
Stacking	0.925000	0.967720	0.968393	0.967720	0.967720
Cat Boost	0.957791	0.974781	0.974781	0.973436	0.973100
SVM	0.819177	0.968393	0.969738	0.969065	0.968729
Accuracy (AVG)	0.851	0.919	0.889	0.929	0.92

Table 6
Classifiers Testing Accuracy Comparison over KC1 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.850000	0.865041	0.865041	0.860163	0.871545
KNN	0.828571	0.889431	0.889431	0.887805	0.887805
Decision trees	0.809524	0.856911	0.866667	0.879675	0.876423
Random Forest	0.852381	0.882927	0.886179	0.892683	0.891057
Ada Boost	0.852381	0.882927	0.886179	0.892683	0.891057
Gradient Boosting	0.866667	0.884553	0.884553	0.889431	0.884553
SGB	0.869048	0.894309	0.891057	0.899187	0.895935
XGBoost	0.847619	0.461789	0.577236	0.856911	0.866667
Stacking	0.890476	0.889251	0.889251	0.889251	0.889251
Cat Boost	0.876190	0.897561	0.897561	0.895935	0.895935
SVM	0.850000	0.886179	0.886179	0.884553	0.886179
Accuracy (AVG)	0.853	0.844	0.856	0.883	0.884

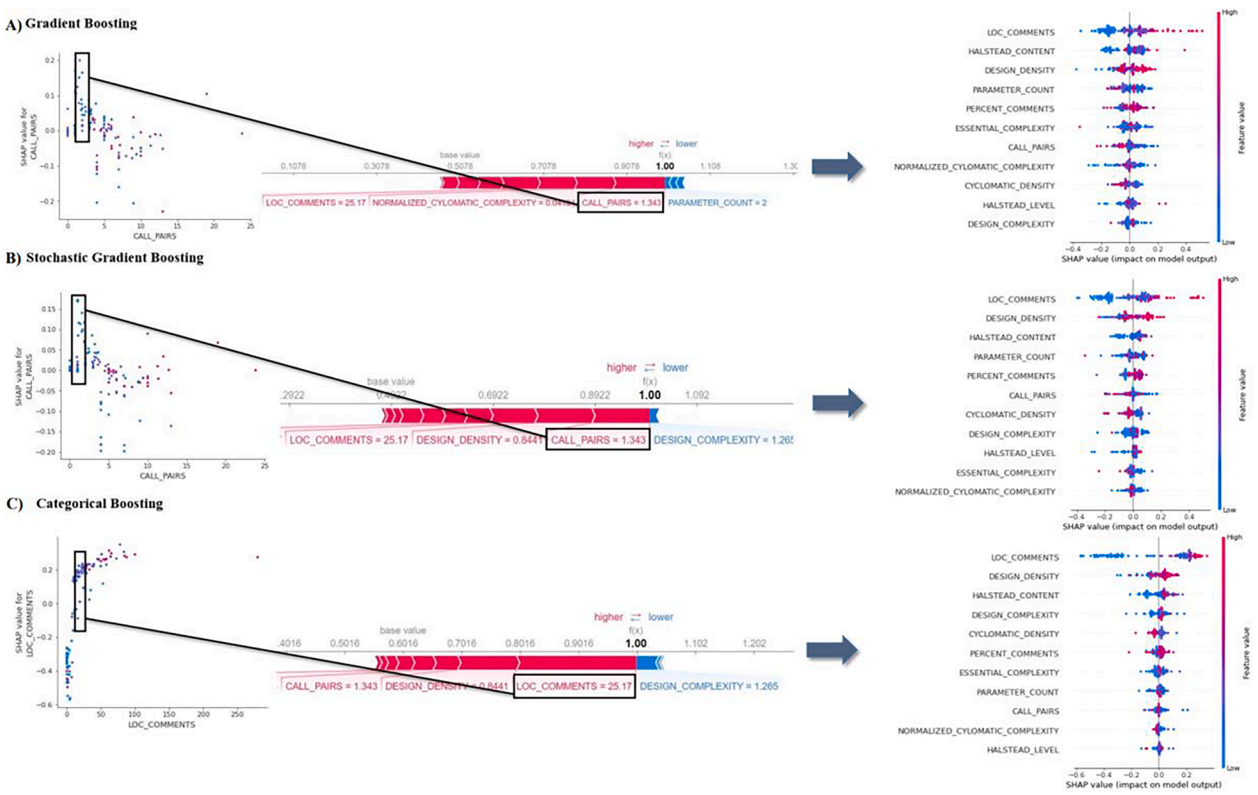

Fig. 12. The 10th Observation Impact on Predictive Models and Highest Discriminative Power Features over CM1 dataset. A) Shows call pairs as the most impact feature in GB with a 1.343 value compared to a 0.5 base value. B) Shows call pairs with a 1.343 value compared to a 0.49 base value in SGB. C) Shows lines of code comments with a 25.17 value compared to a 0.6 base value in Cat Boost.

Table 7
Classifiers Testing Accuracy Comparison over KC3 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.825	0.860163	0.860163	0.919355	0.967742
KNN	0.775	0.887805	0.887805	0.903226	0.838710
Decision trees	0.925	0.873171	0.876423	0.903226	0.919355
Random Forest	0.800	0.889431	0.889431	0.935484	0.935484
Ada Boost	0.800	0.889431	0.889431	0.935484	0.935484
Gradient Boosting	0.875	0.889431	0.889431	0.919355	0.903226
SGB	0.875	0.897561	0.895935	0.935484	0.919355
XGBoost	0.825	0.855285	0.858537	0.919355	0.870968
Stacking	0.700	0.889251	0.889251	0.903226	0.903226
Cat Boost	0.900	0.895935	0.895935	0.935484	0.935484
SVM	0.825	0.884553	0.884553	0.919355	0.919355
Accuracy (AVG)	0.829	0.882	0.882	0.92	0.913

Table 8
Classifiers Testing Accuracy Comparison over MC1 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.990841	0.950879	0.950879	0.950879	0.933296
KNN	0.994073	0.998046	0.998046	0.998046	0.997767
Decision trees	0.991379	0.998046	0.998325	0.998046	0.996930
Random Forest	0.994612	0.996651	0.996651	0.996651	0.996651
Ada Boost	0.994612	0.996651	0.996651	0.996651	0.996651
Gradient Boosting	0.993534	0.996930	0.996930	0.996930	0.997488
SGB	0.993534	0.996372	0.997767	0.996651	0.919355
XGBoost	0.992457	0.485906	0.991627	0.975440	0.891711
Stacking	0.995690	0.996094	0.996094	0.996094	0.996094
Cat Boost	0.994612	0.996093	0.996093	0.996093	0.996093
SVM	0.992457	0.995814	0.995814	0.995814	0.996372
Accuracy (AVG)	0.992	0.945	0.992	0.99	0.973

Table 9
Classifiers Testing Accuracy Comparison over MC2 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.730769	0.950879	0.542857	0.542857	0.714286
KNN	0.769231	0.998046	0.542857	0.542857	0.771429
Decision trees	0.576923	0.998046	0.657143	0.657143	0.771429
Random Forest	0.769231	0.996651	0.800000	0.857143	0.828571
Ada Boost	0.769231	0.996651	0.800000	0.857143	0.828571
Gradient Boosting	0.769231	0.996930	0.828571	0.828571	0.800000
SGB	0.692308	0.996930	0.800000	0.857143	0.857143
XGBoost	0.653846	0.973207	0.657143	0.685714	0.657143
Stacking	0.846154	0.996094	0.705882	0.705882	0.647059
Cat Boost	0.692308	0.996093	0.800000	0.800000	0.800000
SVM	0.692308	0.995814	0.685714	0.685714	0.685714
Accuracy (AVG)	0.723	0.99	0.71	0.728	0.759

confusion matrix, on the other hand, is a table that contains a collection of characteristics that may be used to describe the performance of a classifier [55]. It may be used to visualize and compare the statistical performance of various algorithms.

In short, a confusion matrix has four key properties that may be utilized to derive different evaluation metrics, which are as follows:

- **True positive (TP):** Presents the number of defective modules that have been classified correctly as defective modules.
- **True negative (TN):** Presents the number of non-defective modules that have been classified correctly as non-defective modules.
- **False positive (FP):** Presents the number of misclassified non-defective modules that are defective.
- **False negative (FN):** Presents the number of misclassified defective modules that are non-defective modules.

In this study, we evaluate the proposed classifiers using testing accuracy and AUC. AUC, on the other hand, is a performance metric that indicates how well a certain classifier performs classification by distinguishing between target classes [56]. True positive rate (sensitivity) and false positive rate (1-specificity) are two threshold values that have an inverse relationship, with sensitivity increasing as specificity decreases and vice versa. The higher the AUC, however, the better, indicating that the classifier is not doing random classification. Fig. 10 shows the AUC ranges and values. 20% of the data were maintained as testing sets to evaluate classification accuracy. The confusion matrix may be used to extract all of the statistic measures listed above as follows:

- $\boldsymbol{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$
- $\boldsymbol{Turepositiverate}(\boldsymbol{Sensitivity}) = \frac{TP}{TP+FN}$
- $\boldsymbol{Falsepositiverate} = 1 - \boldsymbol{specificity} = 1 - \frac{FP}{TN+FP}$

In this work, we follow the use of the Grid-Search method for tuning and optimizing classifiers' hyperparameters. Grid-Search, on the other hand, is a search strategy for choosing the optimum parameters by merging all of the possible value entries in the search space [57]. Boosting ensemble methods parameters including the number of estimators, learning rate, loss function, and algorithms were tuned based on a set of values. Also, 5

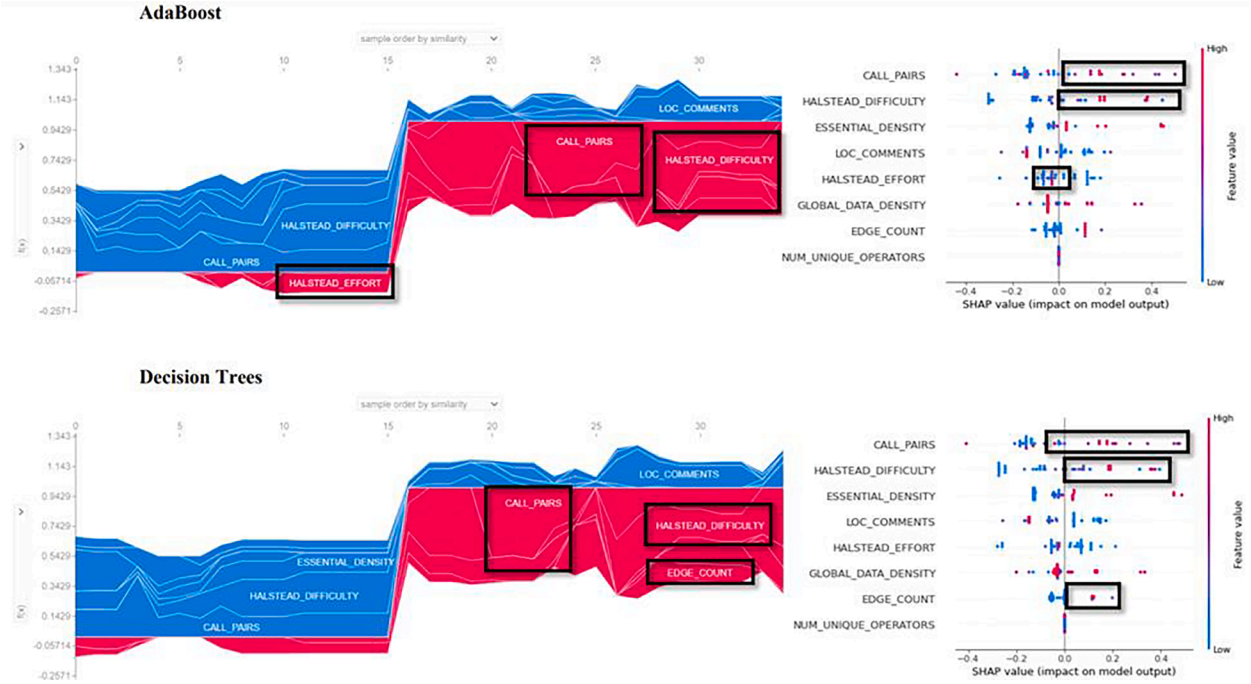

Fig. 13. Shows 35 data samples contribution using AdaBoost and DT over the MC2 dataset utilizing SMOTE and PSO. A) Shows Halsted effort, call pairs and Halsted difficulty metrics as the highest discriminative features using AdaBoost. B) Shows call pairs, edge count, and Halsted difficulty metrics as the most affected features over 35 data samples using DT.

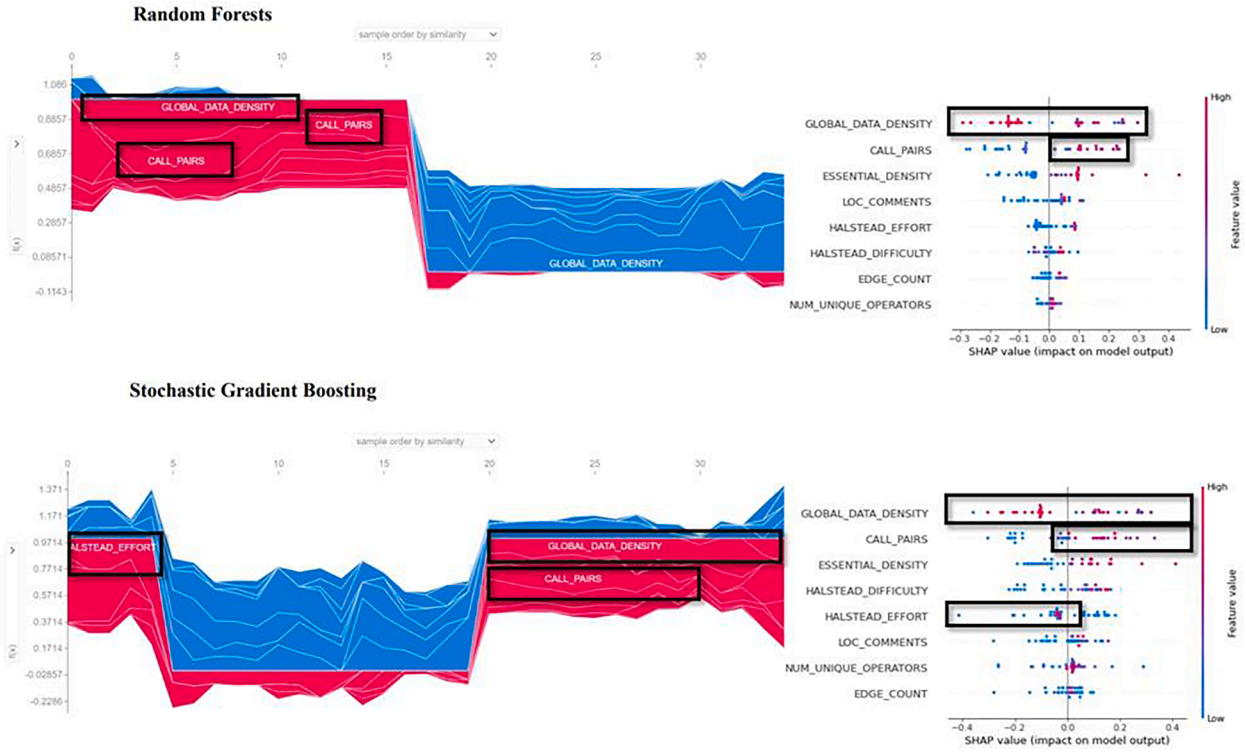

Fig. 14. Shows 35 data samples contribution using RF and SGB over the MC2 dataset utilizing SMOTE and PSO. A) Shows global data density and call pairs metrics effects using RF. B) Shows Halsted effort, global data density, and call pairs metrics effects using SGB.

Table 10
Classifiers Testing Accuracy Comparison over MW1 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.924528	0.542857	0.542857	0.542857	0.824176
KNN	0.905660	0.542857	0.542857	0.542857	0.956044
Decision trees	0.886792	0.657143	0.657143	0.657143	0.879121
Random Forest	0.924528	0.828571	0.857143	0.828571	0.956044
Ada Boost	0.924528	0.828571	0.857143	0.828571	0.956044
Gradient Boosting	0.886792	0.800000	0.828571	0.800000	0.901099
SGB	0.849057	0.800000	0.800000	0.800000	0.923077
XGBoost	0.905660	0.685714	0.800000	0.742857	0.912088
Stacking	0.925926	0.705882	0.705882	0.705882	0.869565
Cat Boost	0.924528	0.800000	0.800000	0.800000	0.956044
SVM	0.905660	0.685714	0.685714	0.685714	0.934066
Accuracy (AVG)	0.905	0.715	0.733	0.72	0.915

Table 11
Classifiers Testing Accuracy Comparison over PC1 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.914474	0.862816	0.862816	0.862816	0.894958
KNN	0.901316	0.945848	0.945848	0.945848	0.962185
Decision trees	0.914474	0.909747	0.916968	0.924188	0.941176
Random Forest	0.921053	0.938628	0.924188	0.938628	0.957983
Ada Boost	0.921053	0.938628	0.924188	0.938628	0.957983
Gradient Boosting	0.907895	0.942238	0.942238	0.935018	0.962185
SGB	0.901316	0.942238	0.945848	0.942238	0.957983
XGBoost	0.940789	0.837545	0.859206	0.902527	0.907563
Stacking	0.934211	0.927536	0.927536	0.927536	0.974790
Cat Boost	0.921053	0.942238	0.942238	0.942238	0.962185
SVM	0.921053	0.942238	0.942238	0.942238	0.949580
Accuracy (AVG)	0.917	0.92	0.927	0.933	0.953

Table 12
Classifiers Testing Accuracy Comparison over PC2 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.990536	0.862816	0.862816	0.862816	0.955684
KNN	0.990536	0.945848	0.945848	0.945848	0.996146
Decision trees	0.987382	0.909747	0.916968	0.909747	0.996146
Random Forest	0.990536	0.938628	0.935018	0.938628	0.996146
Ada Boost	0.990536	0.938628	0.935018	0.938628	0.996146
Gradient Boosting	0.990536	0.938628	0.935018	0.945848	0.996146
SGB	0.993691	0.945848	0.942238	0.945848	0.998073
XGBoost	0.990536	0.898917	0.675090	0.862816	0.445087
Stacking	0.987421	0.927536	0.927536	0.927536	0.992308
Cat Boost	0.990536	0.942238	0.942238	0.942238	0.996146
SVM	0.990536	0.942238	0.942238	0.942238	0.994220
Accuracy (AVG)	0.9902	0.932	0.909	0.929	0.9406

folds cross-validation was employed. The grid-search parameter space for boosting methods is shown in Table 3.

5. Experimental results and models explanations

The findings of the proposed classifiers for predicting software defects are discussed in this section. SMOTE was utilized for data balance after data preprocessing. Four Meta heuristics methods, on the other hand, were tuned and used for feature selection. We employ the logistic map function as a chaotic type and 20 initial particles with 20 iterations in ACO. We utilized a population size of 20 with 20 iterations in PSO. In GA, we employed a crossover probability of 6% with a maximum generation of 20. In addition, we employed the logistic map function in HA with 20 iterations and population size.

Fig. 11 shows the number of features selected for each dataset. In comparison to the others, GA has the most selected amount of features whereas ACO comes next.

Tables 4-6 compare the accuracy of classifiers in the CM1, JM1, and

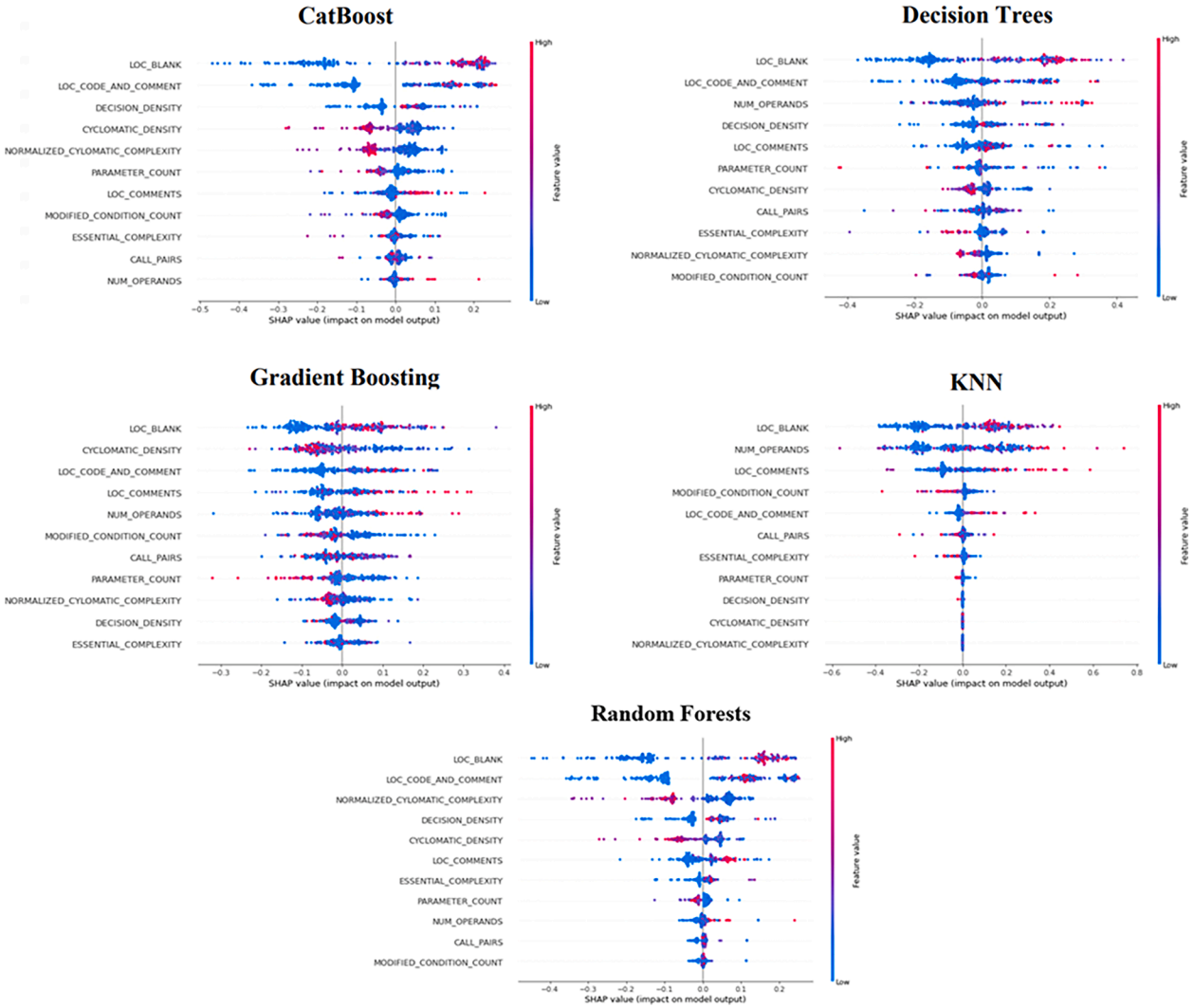

Fig. 15. Shows the most effective features over the PC1 dataset using ACO and SMOTE.

Table 13
Classifiers Testing Accuracy Comparison over PC3 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.871111	0.862816	0.862816	0.862816	0.805479
KNN	0.871111	0.945848	0.945848	0.945848	0.928767
Decision trees	0.857778	0.916968	0.916968	0.898917	0.893151
Random Forest	0.875556	0.935018	0.942238	0.931408	0.920548
Ada Boost	0.875556	0.935018	0.942238	0.931408	0.920548
Gradient Boosting	0.906667	0.938628	0.938628	0.942238	0.906849
SGB	0.880000	0.935018	0.945848	0.942238	0.931507
XGBoost	0.235556	0.884477	0.873646	0.884477	0.857534
Stacking	0.902655	0.927536	0.927536	0.927536	0.939891
Cat Boost	0.875556	0.942238	0.942238	0.942238	0.920548
SVM	0.875556	0.942238	0.942238	0.942238	0.926027
Accuracy (AVG)	0.815	0.930	0.931	0.928	0.914

Table 14
Classifiers Testing Accuracy Comparison over PC4 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.925000	0.862816	0.862816	0.862816	0.860262
KNN	0.875000	0.945848	0.945848	0.945848	0.938865
Decision trees	0.964286	0.924188	0.924188	0.924188	0.978166
Random Forest	0.928571	0.935018	0.931408	0.938628	0.958515
Ada Boost	0.928571	0.935018	0.931408	0.938628	0.958515
Gradient Boosting	0.975000	0.938628	0.935018	0.942238	0.973799
SGB	0.971429	0.942238	0.938628	0.945848	0.931507
XGBoost	0.860714	0.469314	0.758123	0.866426	0.871179
Stacking	0.921429	0.927536	0.927536	0.927536	0.890830
Cat Boost	0.971429	0.942238	0.942238	0.942238	0.982533
SVM	0.871429	0.942238	0.942238	0.942238	0.901747
Accuracy (AVG)	0.926	0.89	0.917	0.931	0.938

Table 15
Classifiers Testing Accuracy Comparison over PC5 Dataset.

Classifiers	Pure	PSO/SMOTE	GA/SMOTE	HA/SMOTE	AC/SMOTE
LR	0.963834	0.862816	0.862816	0.913023	0.955054
KNN	0.975889	0.945848	0.945848	0.991103	0.988802
Decision trees	0.973243	0.898917	0.909747	0.987728	0.989109
Random Forest	0.971479	0.942238	0.931408	0.978831	0.979291
Ada Boost	0.971479	0.942238	0.931408	0.978831	0.979291
Gradient Boosting	0.978536	0.938628	0.942238	0.990950	0.991256
SGB	0.976183	0.938628	0.938628	0.989262	0.931507
XGBoost	0.971479	0.862816	0.862816	0.969781	0.961190
Stacking	0.970588	0.927536	0.927536	0.987423	0.985276
Cat Boost	0.974125	0.942238	0.942238	0.986041	0.987421
SVM	0.970891	0.942238	0.942238	0.984814	0.982973
Accuracy (AVG)	0.973	0.928	0.927	0.984	0.977

KC1 datasets before and after using *meta*-heuristic feature selection algorithms. The testing accuracy average, on the other hand, was computed to highlight the comparison of outcomes. Overall, the results showed an improvement in testing accuracy. GB, SGB, and CatBoost, on the other hand, outperform the others. This is not surprising given that boosting methods enhance variance while decreasing bias. In addition, by expanding minority class samples, SMOTE increases the number of data samples.

However, XGBoost performs the worst when compared to others utilizing PSO and GA in the JM1 and KC1 datasets, with testing accuracy ranging from 45 to 57 percent. To be clear, the reason is most likely overfitting. In light of this, GridSeachCV was used to adjust XGBoost hyperparameters. To the best of our understanding, boosting methods in general are prone to overfitting. Particularly, deep trees. SMOTE with the selected features utilizing PSO yields the best results in the CM1 dataset, with an average accuracy of 89.6%. The results of the JM1 dataset demonstrate that SMOTE and HA outperform others with an average testing accuracy of 92.2%. Also, in the KC1 dataset, SMOTE and ACO outperform others with an average accuracy of 88.4%.

Fig. 12 shows SHAP plots to explain the GB, SGB, and Cat Boost results. The bee swarm plot was used to describe important features based on SHAP values, and the local force plot was used to demonstrate how a single observation contributed to the prediction models. We chose the tenth observation at random over the CM1 dataset. The call pairs metric pushes the prediction to the right in GB and SGB, with a value of 1.343 compared to the base value of 0.5.

However, the call pair's impact on the prediction model decreases as parameter count and design complexity metrics increase. Lines of code comments have the strongest impact in Cat Boost at the 10th observation, at 25.17, relative to the base value of 0.6, however, the design complexity metric has a negative impact. Overall, it can be notated that lines of code comments, design density, and Halsted content metrics have the highest discriminative power to the predictive models.

The comparative findings using *meta*-heuristics and SMOTE across the KC3, MC1, and MC2 datasets are shown in Tables 7-9. Aside from general performance enhancement. SGB, Cat Boost, LR, and RF surpass others using the HA algorithm in the KC3 dataset, with an average accuracy of 92% compared to the original findings of 82%. The MC1

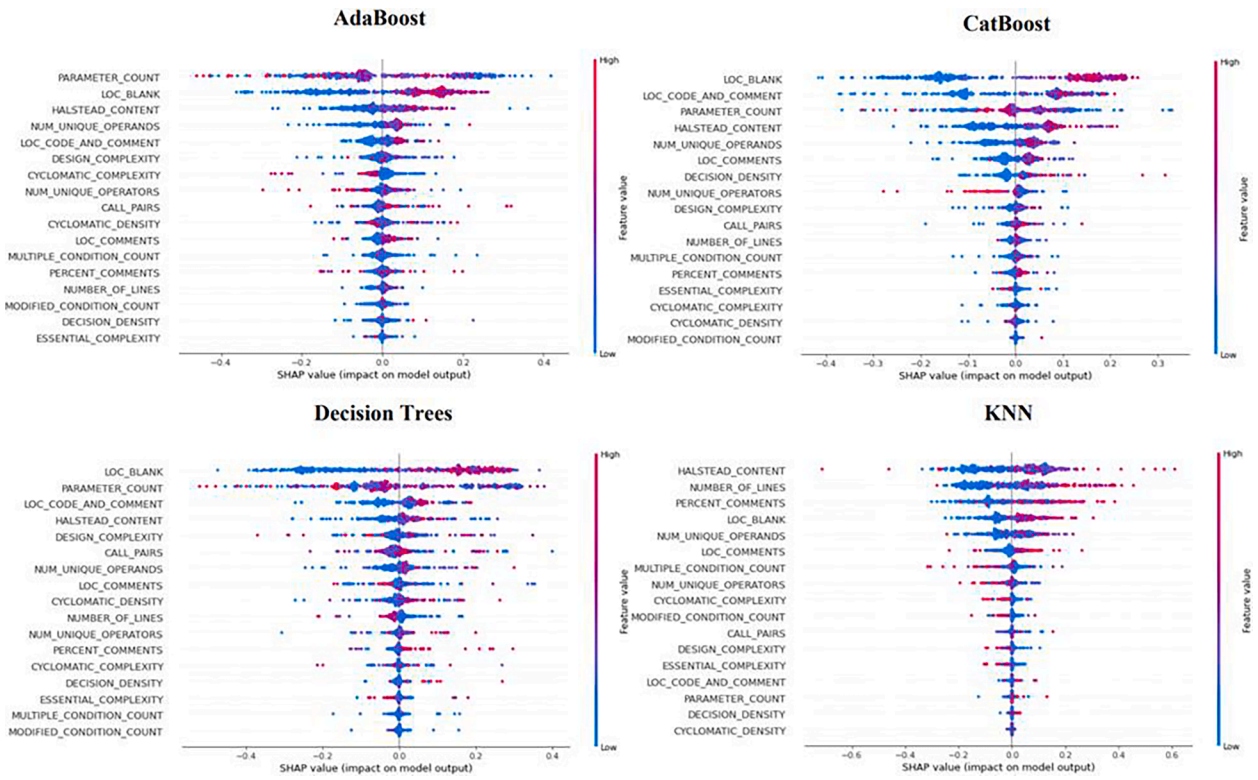

Fig. 16. Shows the Most Effective Features Utilizing KNN, AdaBoost, CatBoost, and DT with GA over the PC3 Dataset.

dataset, on the other hand, revealed that DT, KNN, and GB outperform others employing GA with an average accuracy of 92%.

However, the MC1 dataset is regarded to be one of the largest NASA datasets, with a total of 9277 occurrences, and we were able to retain the same accuracy by applying the GA algorithm due to the broad data dispersion over training and testing sets. The findings of the MC2 dataset revealed that six classifiers, including DT, KNN, GB, SGB, AdaBoost, and RF, out of eleven, had the best accuracy by comparison. Additionally, SMOTE and PSO yield the best outcomes, with an average accuracy of 99%.

Figs. 13 and 14 show the SHAP explanation of the DT, AdaBoost, RF, and SGB models on the MC2 dataset using SMOTE and PSO. These classifiers have the best outcome on the KC3, MC1, and MC2 datasets. Explaining how a single observation adds to the prediction model, on the other hand, is useful but insufficient. As a result, we employ the SHAP global force plot to demonstrate how a series of consecutive data samples contribute to the prediction model.

The plots show the contributions of 35 data samples. The x-axis in global force reflects the number of data samples, while the y-axis shows SHAP predicted values. In addition, we employ a bee swarm plot to emphasize the features with the highest discriminative power. For clarity, the most significant features of overall classifiers are call pairs, global data density, and essential density metrics.

The results of classifiers using *meta*-heuristic methods on the MW1, PC1, and PC2 datasets are shown in Tables 10-12. Overall, ACO has the best performance improvement when compared to others. The findings on the MW1 dataset demonstrate that KNN, RF, AdaBoost, and Cat Boost surpass others using ACO with an average accuracy of 91.5%. With only 264 total instances, the MW1 dataset is considered one of the smallest NASA datasets. We demonstrate overall results improvement using all search methods on the PC1 dataset. However, we found that ACO has the best results when compared to others, with an average accuracy of 95.3%. It is also notable that KNN, GB, Cat Boost, and Stacking outperform the others. ACO outperforms the others in the PC2 dataset, with an average testing accuracy of 94%. However, unlike in PC1 and MW1, XGBoost performs the lowest in PC2, which is most likely due to overfitting. Nonetheless, SGB surpasses others. Fig. 15 shows the most effective features over the PC1 dataset utilizing KNN, DT, GB, RF, and Cat boost with ACO and SMOTE.

Tables 13-15 show the findings before and after using metaheuristics and SMOTE on the PC3, PC4, and PC5 datasets, respectively. The findings of the PC3 dataset revealed an overall improvement. Particularly by applying GA, which has an average testing accuracy of 93.1%. However, it is worth noting that KNN, SGB, and stacking exceed others in terms of accuracy. PSO also improved its accuracy, with an average accuracy of 93%. In comparison to the original, XGBoost accuracy improved significantly utilizing PSO and HA, with an accuracy of 88.4%. ACO has the best results in the PC4 dataset, with an average testing accuracy of 93.8%, followed by HA, which has an accuracy of 93.1%. Also, when comparing metaheuristics, KNN, DT, SGB, and Cat Boost outperform the others. XGBoost and LR, on the other hand, perform poorly. Even though the fact that the PC5 dataset is the largest NASA dataset, with 17,001 total instances, we found that using HA and SMOTE performs the best, with an average accuracy of 98.4% when compared to the original accuracy of 97.3%. ACO also showed a slight improvement of 0.004%. Surprisingly, the KNN outperforms others by utilizing PSO, GA, and HA. However, in ACO, the KNN, DT, and GB performed the best.

Fig. 16 shows the most effective features (software metrics) using KNN, AdaBoost, CatBoost, and DT with GA on the PC3 dataset. These classifiers have the best overall performance. The SHAP explanation plot, on the other hand, reveals that lines of code blank, parameters count, and Halsted content metrics have the highest discriminative power.

Fig. 17 shows the ROC-AUC performance of classifiers across NASA datasets. Notably, when compared to others, GB, SGB, AdaBoost, RF, DT, and Cat Boost have the best performance. This is not unexpected given that these classifiers had the greatest overall accuracy scores. XGBoost and SVM, on the other hand, performed admirably. However, with an AUC of higher than 90%, all classifiers performed best on the PC5, PC2, MC1, MW1, and PC4 datasets. In comparison to the others, SGB and Cat

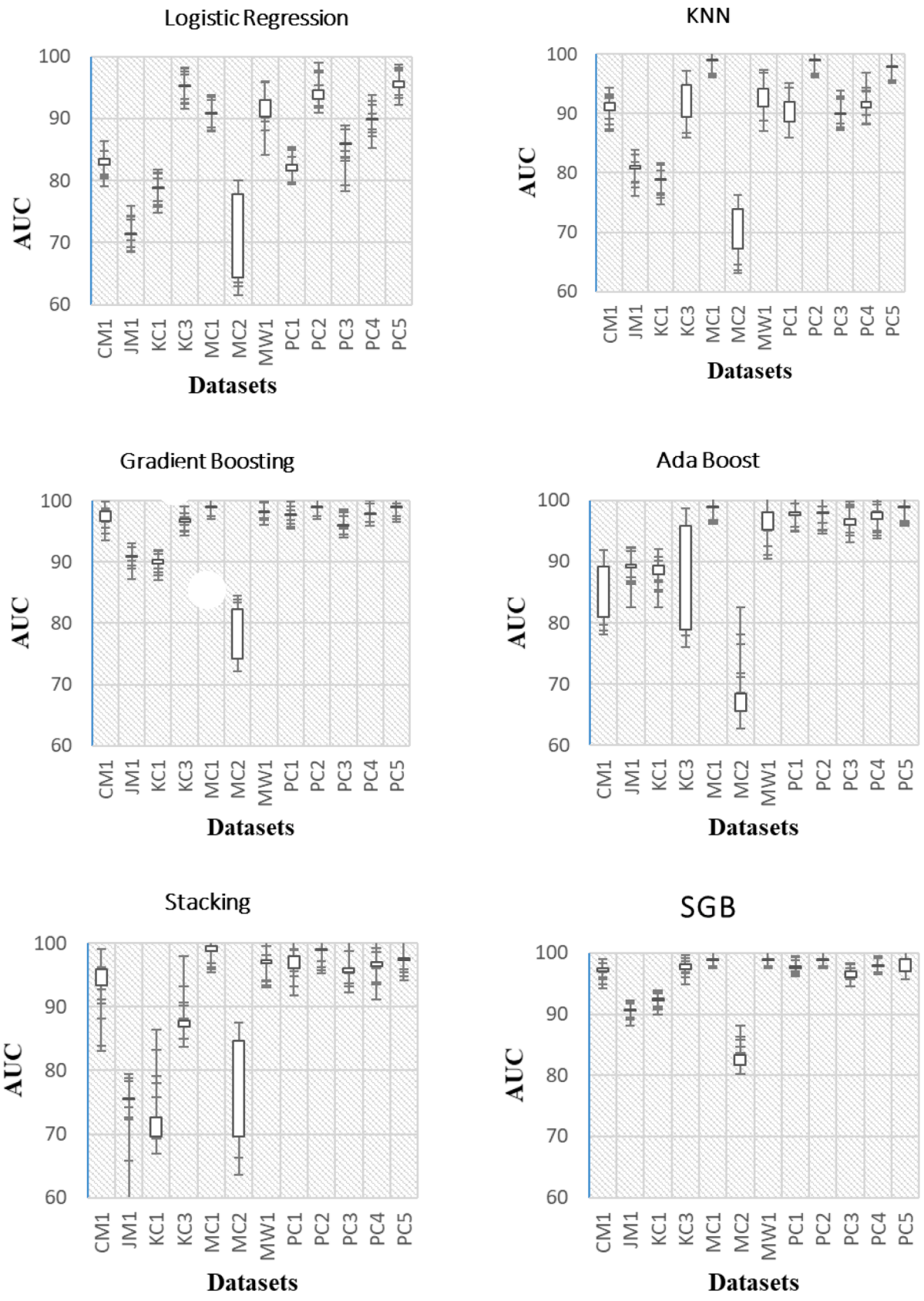

Fig. 17. Shows Classifiers ROC-AUC Comparison.

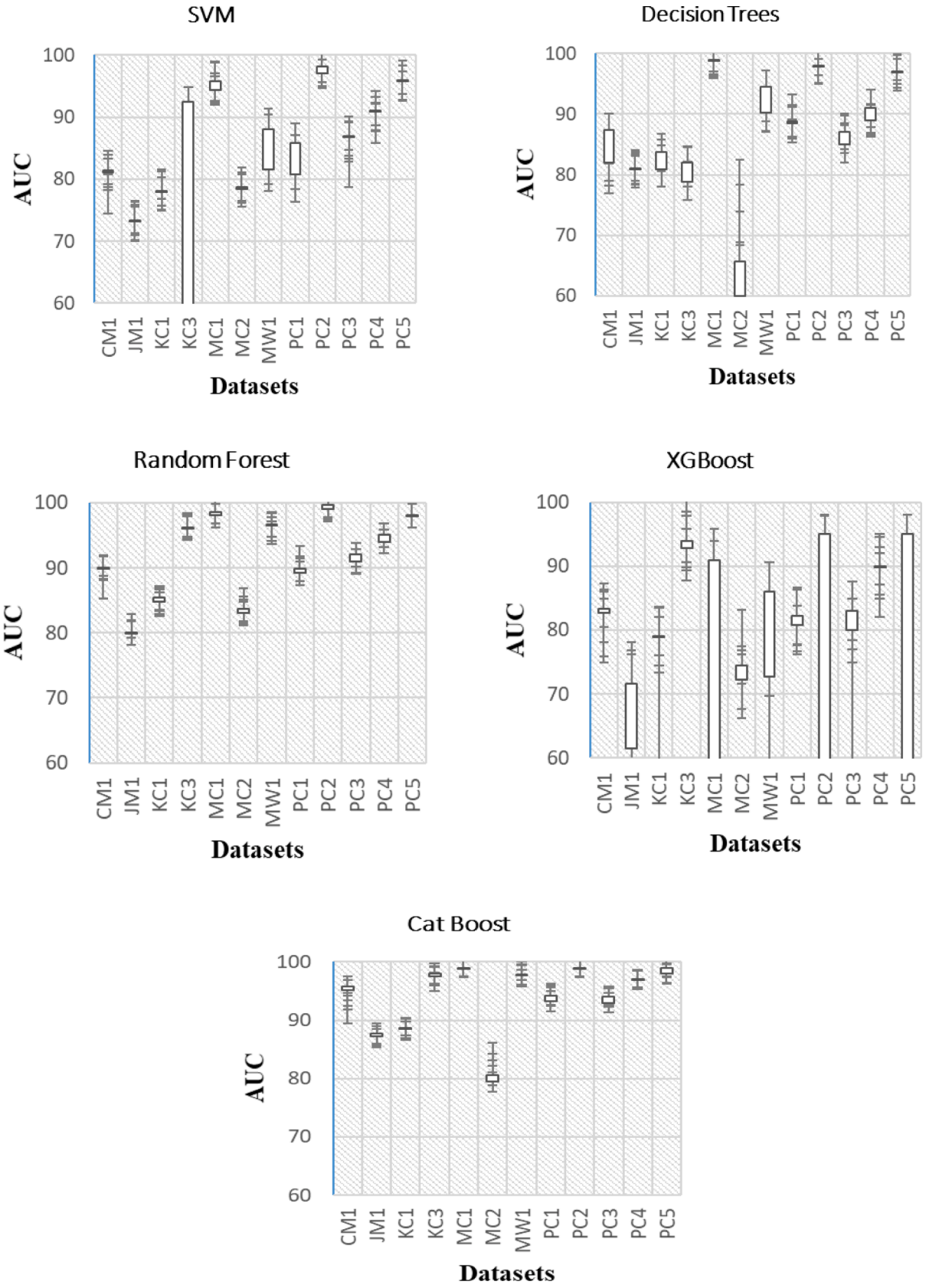

Fig. 17. (*continued*).

Boost performed the best. SGB, on the other hand, outperforms Cat Boost by a little percentage.

6. Conclusion

Software defects decrease software quality in terms of efficiency, testability, and maintainability. Also, it can lead to project success or failure. Therefore, the process of software defects prediction at the early stages of development enhance software quality, decrease development and maintenance cost, and keeps the project on track. In this paper, we proposed a new framework for predicting software defects. Twelve NASA datasets were subjected to the use of eleven machine learning classifiers from the statistical, boosting, bagging, and stacking families. Also, we investigated the use of four search-based algorithms (Particle swarm optimization, Genetic algorithm, Harmony algorithm, and Ant colony optimization) for feature selection. However, NASA datasets have a large distribution gap between minority and majority classes. Therefore, we employed the use of SMOTE as a resampling technique for data balancing. On the other hand, we used the SHAP library to explain the results of the models and highlight the most effective features. Testing accuracy and ROC-AUC were employed for model evaluation. The results revealed that GB, SGB, DT, and Cat Boost classifiers have the best accuracy and performance when compared to others. Also, we found that PC5 and MC1 datasets have the best results. However, when comparing the *meta*-heuristics algorithms, we found that ant colony optimization has the best results improvement followed by the harmony algorithm. Finally, we showed that call pairs, Halsted content, parameters count, and lines of code comments metrics had the highest discriminative power by SHAP employment. As a future direction, we intend to compare the *meta*-heuristic algorithms with filter methods for feature selection and compare the performance of SMOTE with other resampling techniques.

CRediT authorship contribution statement

Yazan Al-Smadi: Conceptualization, Methodology, Software, Data curation, Writing – original draft, Investigation, Validation, Writing – review & editing. **Mohammed Eshtay:** Conceptualization, Methodology, Software, Data curation, Writing – original draft, Investigation, Validation, Writing – review & editing. **Ahmad Al-Qerem:** Conceptualization, Methodology, Software, Data curation, Writing – original draft, Investigation, Validation, Writing – review & editing. **Shadi Nashwan:** Conceptualization, Methodology, Software, Data curation, Writing – original draft, Investigation, Validation, Writing – review & editing. **Osama Ouda:** Conceptualization, Methodology, Software, Data curation, Writing – original draft, Investigation, Validation, Writing – review & editing. **A.A. Abd El-Aziz:** Conceptualization, Methodology, Software, Data curation, Writing – original draft, Investigation, Validation, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Jöckel L, et al. Towards a Common Testing Terminology for Software Engineering and Data Science Experts. International Conference on Product-Focused Software Process Improvement. Springer; 2021.
[2] Quyoum A, Dar M-U-D, Quadri SMK. Improving software reliability using software engineering approach- a review. Int J Computer Applications 2010;10(5):41–7.
[3] Zhang D, Tsai JJJSQJ. Machine learning and software engineering. Available at SSRN 4141236. 2003;11(2):87–119.
[4] Graham, D., R. Black, and E. Van Veenendaal, Foundations of software testing ISTQB Certification. 2021: Cengage Learning.
[5] Asghar B, Awan I, Bhatti AM. Process Improvement through Reduction in Software Defects using Six Sigma Methods. IEEE; 2018.
[6] Kessentini, M., et al. Search-based design defects detection by example. in International Conference on Fundamental Approaches to Software Engineering. 2011. Springer.
[7] Yadav V, Singh R, Yadav V. Estimation Model for enhanced predictive object point metric in OO software size estimation using deep learning. IAJIT 2023;20(3).
[8] Helm JM, et al. Machine learning and artificial intelligence: definitions, applications, and future directions. Current Reviews in Musculoskeletal Medicine 2020;13(1):69–76.
[9] Özakıncı, R., A.J.J.o.S. Tarhan, and Software, Early software defect prediction: A systematic map and review. 2018. Journal of Systems and Software, 144: p. 216-239.
[10] Pachouly, J., et al., A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. 2022. Engineering Applications of Artificial Intelligence, 111: p. 104773.
[11] Chen X, et al. Software defect number prediction: unsupervised vs supervised methods. Information and Software Technology 2019;106:161–81.
[12] Xu Z, et al. Software defect prediction based on kernel PCA and weighted extreme learning machine. Information and Software Technology 2019;106:182–200.
[13] Moshin Reza S, et al. Performance analysis of machine learning approaches in software complexity prediction. Springer; 2021.
[14] Huang J, et al. An empirical analysis of data preprocessing for machine learning-based software cost estimation. Information and Software Technology 2015;67:108–27.
[15] Liang H, et al. Seml: a semantic LSTM model for software defect prediction. IEEE Access 2019;7:83812–24.
[16] Wu Y, Limcr,, et al. Less-informative majorities cleaning rule based on Naïve Bayes for imbalance learning in software defect prediction. Applied Sciences 2020;10(23):8324.
[17] Catolino G, Di Nucci D, Ferrucci F. Cross-project just-in-time bug prediction for mobile apps: An empirical assessment. IEEE; 2019.
[18] Gao K. The use of under-and oversampling within ensemble feature selection and classification for software quality prediction. Int J Reliability Quality and Safety Engineering 2014;21(01):1450004.
[19] Malhotra R, Shakya A, Ranjan R, Banshi R. Software defect prediction using binary particle swarm optimization with binary cross entropy as the fitness function. J. Phys.: Conf. Ser. 2021;1767(1):012003.
[20] Alauthman M, Aldweesh A, Al-qerem A, Aburub F, Al-Smadi Y, Abaker AM, et al. Tabular data generation to improve classification of liver disease diagnosis. Appl Sci 2023;13(4):2678.
[21] Balogun AO, et al. Performance analysis of feature selection methods in software defect prediction: a search method approach. Appl Sci 2019;9(13):2764.
[22] Anbu, M. and G.J.C.C. Anandha Mala, Feature selection using firefly algorithm in software defect prediction. 2019. Cluster Computing, 22(5): p. 10925-10934.
[23] Ayon SI. Neural network based software defect prediction using genetic algorithm and particle swarm optimization. IEEE; 2019.
[24] Balogun AO, et al. Empirical analysis of rank aggregation-based multi-filter feature selection methods in software defect prediction. Electronics 2021;10(2):179.
[25] Ali U, Aftab S, Iqbal A, Nawaz Z, Salman Bashir M, Anwaar Saeed M. Software defect prediction using variant based ensemble learning and feature selection techniques. Int J Modern Education & Computer Science 2020;12(5):29–40.
[26] Alsaeedi, A., M.Z.J.J.o.S.E. Khan, and Applications, Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. 2019. Journal of Software Engineering and Applications, 12(5): p. 85-100.
[27] Balogun AO, et al. Software defect prediction using ensemble learning: an ANP based evaluation method. FUOYE J Eng Tech 2018;3(2):50–5.
[28] Yu Q, et al. Process metrics for software defect prediction in object-oriented programs. IET Software 2020;14(3):283–92.
[29] Ghosh S, Rana A, Kansal VJPcs. A nonlinear manifold detection based model for software defect prediction. Procedia Computer Science 2018;132:581–94.
[30] Ghosh S, et al. A benchmarking framework using nonlinear manifold detection techniques for software defect prediction. Int J Comput Sci Eng 2020;21(4):593–614.
[31] Lundberg SM, et al. From local explanations to global understanding with explainable AI for trees. Nature machine intelligence 2020;2(1):56–67.
[32] Kaur, H., H.S. Pannu, and A.K.J.A.C.S. Malhi, A systematic review on imbalanced data challenges in machine learning: Applications and solutions. 2019. ACM Computing Surveys (CSUR), 52(4): p. 1-36.
[33] Fernández A, et al. SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. J Artificial Intelligence Res 2018;61:863–905.
[34] Dai H-P, Chen D-D, Zheng Z-S-J-A. Effects of random values for particle swarm optimization algorithm. Algorithms 2018;11(2):23.
[35] Katoch S, et al. A review on genetic algorithm: past, present, and future. Multimedia Tools and Applications 2021;80(5):8091–126.
[36] Ala'a, A., et al., Comprehensive review of the development of the harmony search algorithm and its applications. 2019. IEEE Access, 7: p. 14233-14245.
[37] Abualigah L, Diabat A, Geem ZWJAS. A comprehensive survey of the harmony search algorithm in clustering applications. Appl Sci 2020;10(11):3827.
[38] Dorigo M, Birattari M, Stutzle TJIcim. Ant colony optimization. IEEE Comput Intelligence Magazine 2006;1(4):28–39.
[39] Lyridis DVJOE. An improved ant colony optimization algorithm for unmanned surface vehicle local path planning with multi-modality constraints. Ocean Eng 2021;241:109890.

[40] Wang Q, et al. Overview of logistic regression model analysis and application. Chin J Prev Med 2019;53(9):955–60.
[41] Abu Alfeilat HA, et al. Effects of distance measure choice on k-nearest neighbor classifier performance: a review. Big Data 2019;7(4):221–48.
[42] Patel, H.H., P.J.I.J.o.C.S. Prajapati, and Engineering, Study and analysis of decision tree based classification algorithms. 2018. International Journal of Computer Sciences and Engineering, 6(10): p. 74-78.
[43] Buskirk TDJSP. Surveying the forests and sampling the trees: an overview of classification and regression trees and random forests with applications in survey research. Survey Practice 2018;11(1):1–13.
[44] Pisner DA, Schnyer DM. Support vector machine. In: Machine learning. Academic Press Elsevier; 2020. p. 101–21. https://doi.org/10.1016/B978-0-12-815739-8.00006-7.
[45] Ferreira, A.J. and M.A.J.E.m.l. Figueiredo, Boosting algorithms: A review of methods, theory, and applications. 2012. Ensemble machine learning: Methods and applications,p. 35-85.
[46] Natekin, A. and A.J.F.i.n. Knoll, Gradient boosting machines, a tutorial. 2013. Frontiers in neurorobotics, 7: p. 21.
[47] Friedman, J.H.J.C.s. and d. analysis, Stochastic gradient boosting. 2002. Computational statistics & data analysis, 38(4): p. 367-378.
[48] Shin, Y.J.A.i.C.E., Application of stochastic gradient boosting approach to early prediction of safety accidents at construction site. 2019. Advances in Civil Engineering, 2019.
[49] Godinho S, Guiomar N, Gil A. Estimating tree canopy cover percentage in a mediterranean silvopastoral systems using Sentinel-2A imagery and the stochastic gradient boosting algorithm. Int J Remote Sensing 2018;39(14):4640–62.
[50] Chen, T. and C. Guestrin. Xgboost: A scalable tree boosting system. in Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016.
[51] Babajide Mustapha I, Saeed FJM. Bioactive molecule prediction using extreme gradient boosting. Molecules 2016;21(8):983.
[52] Dorogush, A.V., V. Ershov, and A.J.a.p.a. Gulin, CatBoost: gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363. 2018. https://doi.org/10.48550/arXiv.1810.11363.
[53] Hong JJHFR. An application of XGBoost, LightGBM, CatBoost algorithms on house price appraisal system. Housing Finance Research 2020;4:33–64. https://doi.org/10.52344/hfr.2020.4.0.33.
[54] Alauthman M, Al-qerem A, Sowan B, Alsarhan A, Eshtay M, Aldweesh A, et al. Enhancing small medical dataset classification performance using GAN. Informatics 2023;10(1):28.
[55] Handelman GS, et al. Peering into the black box of artificial intelligence: evaluation metrics of machine learning methods. Am J Roentgenol 2019;212(1):38–43.
[56] Al-qerem A, Al-Naymat G, Alhasan M, Al-Debei M. Default prediction model: the significant role of data engineering in the quality of outcomes. Int Arab J Inf Technol 2020;17(4A):635–44.
[57] Alibrahim, H. and S.A. Ludwig. Hyperparameter optimization: comparing genetic algorithm against grid search and bayesian optimization. in 2021 IEEE Congress on Evolutionary Computation (CEC). 2021. IEEE.