



A new binary chaos-based metaheuristic algorithm for software defect prediction

Bahman Arasteh^{1,2} · Keyvan Arasteh¹ · Ali Ghaffari¹ · Reza Ghanbarzadeh³

Received: 13 December 2023 / Revised: 27 March 2024 / Accepted: 28 March 2024
© The Author(s) 2024

Abstract

Software defect prediction is a critical challenge within software engineering aimed at enhancing software quality by proactively identifying potential defects. This approach involves selecting defect-prone modules ahead of the testing phase, thereby reducing testing time and costs. Machine learning methods provide developers with valuable models for categorising faulty software modules. However, the challenge arises from the numerous elements present in the training dataset, which frequently reduce the accuracy and precision of classification. Addressing this, selecting effective features for classification from the dataset becomes an NP-hard problem, often tackled using metaheuristic algorithms. This study introduces a novel approach, the Binary Chaos-based Olympiad Optimisation Algorithm, specifically designed to select the most impactful features from the training dataset. By selecting these influential features for classification, the precision and accuracy of software module classifiers can be notably improved. The study's primary contributions involve devising a binary variant of the chaos-based Olympiad optimisation algorithm to meticulously select effective features and construct an efficient classification model for identifying faulty software modules. Five real-world and standard datasets were utilised across both the training and testing phases of the classifier to evaluate the proposed method's effectiveness. The findings highlight that among the 21 features within the training datasets, specific metrics such as basic complexity, the sum of operators and operands, lines of code, quantity of lines containing code and comments, and the sum of operands have the most significant influence on software defect prediction. This research underscores the combined effectiveness of the proposed method and machine learning algorithms, significantly boosting accuracy (91.13%), precision (92.74%), recall (97.61%), and F1 score (94.26%) in software defect prediction.

Keywords Software defect prediction · Binary olympiad optimisation algorithm · Feature selection · Machine learning · Module classification

1 Introduction

The presence of defects within a software system poses a considerable risk to its overall quality. Predicting software defects is a critical aspect of software engineering, aimed at improving software quality by identifying and addressing

these defects [1]. Detecting defects before the software is released is vital for enhancing its overall quality. The Pareto principle, which underscores that a majority of software defects arise within specific modules, is applicable here. Hence, forecasting and spotting defects in the early stages of software development significantly enhances the resulting software's quality. Software defect prediction entails identifying modules susceptible to defects before the testing phase, thus reducing testing time and costs. As software systems grow in size and complexity, testing every module comprehensively becomes unfeasible, underscoring the importance of predicting modules prone to defects to enhance software quality. In this pursuit, machine learning (ML) methods provide valuable models

✉ Bahman Arasteh
Bahman.arasteh@istinye.edu.tr

¹ Department of Software Engineering, Faculty of Engineering and Natural Science, Istinye University, Istanbul, Turkey

² Applied Science Research Center, Applied Science Private University, Amman, Jordan

³ Faculty of Science and Engineering, Southern Cross University, Gold Coast, Australia

that empower developers to classify faulty software modules effectively [2].

Recently, there has been a surge in the adoption of software defect prediction approaches to bolster software quality. This study delves into software defect prediction, concentrating on categorising software components (modules) into two categories: prone to defects and non-prone to defects [1, 2]. The classification technique hinges on extracting a model based on the history of defective modules, subsequently employed to enhance accuracy in predicting defects in new modules. Past research reveals a robust correlation between software module metrics and defect prediction [3]. Multiple algorithms exist for software module classification, including Decision Trees (DT), the K-Nearest Neighbour algorithm (KNN), the Naive Bayes (NB) algorithm, Support Vector Machines (SVM), and Artificial Neural Networks (ANN). However, a prevalent challenge in classification lies in handling a vast array of features, which compromises classification accuracy. Feature selection methods come into play to mitigate this challenge and decrease feature dimensions. The process of identifying effective features for classification is an NP-hard problem that can be addressed using evolutionary algorithms [4].

This paper introduces a novel approach to predicting software defects. Initially, a Binary variant of the Chaos-based Olympiad Optimisation Algorithm (BCOOA) was developed to select the most impactful features from the training dataset. Subsequently, various ML algorithms were employed to construct a classification model using this optimal training set. BCOOA draws inspiration from swarm intelligence and is designed to emulate the learning process of a group of students preparing for the Olympiad examination. The stages of teaching and learning amongst students produce population evolution. The primary objective here is to leverage BCOOA's capability to select crucial features for predicting software defects using ML algorithms. The aim is to employ these algorithms to detect and address software defects before software release. Diverse ML algorithms, such as KNN, DT, SVM, NB, and ANN, were utilised for faulty module classification. By selecting the most influential features in classification, there's potential to enhance precision and accuracy in the software module classifier. Ultimately, in the testing phase, the effectiveness of the new feature selection method was assessed using test data.

The following are the main objectives of the current study:

- Determining the most effective features of software defect prediction datasets.
- Increasing the accuracy, precision, and sensitivity of software defect predictors.

- Enhancing the performance and stability of software defect predictors.

The primary contributions of this study are:

- Proposing a novel binary and hybrid version of the Olympiad Optimisation Algorithm (OOA) to select the most effective features of the defect prediction dataset. To achieve greater population diversity, the operators of the Genetic Algorithm (GA) were embedded into the OOA.
- Developing and adapting the theory of chaos in the OOA to improve its convergence with regard to both exploration and exploitation. Chaos maps have been used for population initialisation. The developed binary and chaos-based OOA was adapted to address the challenge of software defect prediction.
- Developing an effective and efficient classification model to detect faulty software modules.
- Increasing the efficiency of software defect detection methods by selecting the smallest subset and the most effective features.

The remainder of the current paper is organised as follows: Sect. 2 reviews the related works on the problem of software defect prediction. In Sect. 3, the details of the proposed method are presented. This section includes two subsections; the first subsection suggests and utilises BCOOA to select the most effective features in the training dataset. The second subsection discusses the development of the desired classifier using the optimal train set and different machine learning algorithms. Section 4 presents all the relevant results from the tests conducted with the specified criteria on real-world datasets, and Sect. 5 concludes the article and recommends guidelines for future work.

2 Related works

During software development, the testing phase holds significant importance [5–7]. The reliability of software hinges on the presence of bugs¹(faults) within the software [8, 9]. This phase incurs substantial expenses in terms of both budget and time, underscoring the criticality of predicting software modules prone to defects. This prediction occurs before commencing the testing phase, aiding in identifying and rectifying modules susceptible to defects. Drawing from the historical data of problematic software modules in prior project implementations or similar projects, a model is derived to facilitate accurate defect prediction in newly developed modules. Research in defect

¹ In this paper, the terms “fault”, “defect”, “bug” and “mistake” have been used in the same meaning and can be used interchangeably.

prediction and estimation indicates that the underlying hypothesis used in constructing the model significantly influences the efficiency of the prediction model [10]. Various approaches exist for software defect prediction. This section scrutinises four ML methods: normalisation-based, unbalanced learning-based, feature selection-based, and blended learning-based methods.

2.1 Normalisation learning methods

Data normalisation involves pre-processing datasets prior to training and testing to standardise the values of independent features within the dataset. This process ensures consistency across the diverse range of feature values present in datasets, which is crucial as some ML algorithms' objective functions may not perform optimally without normalisation. This pre-processing technique plays a pivotal role in enhancing algorithm accuracy by remapping the measured values to a consistent range [11]. In the realm of software defect prediction research, three primary normalisation methods—logarithmic normalisation, min-max, and Z-standardisation—have been widely adopted.

Logarithmic normalisation transforms all sample feature values into their logarithmic equivalents, effectively managing skewed data distributions and reducing the impact of outliers. The min-max method normalises features within a specified range by identifying the minimum and maximum values for each feature and scaling accordingly. In contrast, Z-standardisation adjusts feature values based on their mean and standard deviation, remapping them to a range where the mean is 0 and the standard deviation is 1. These techniques have been applied in various studies, utilising logarithmic normalisation, min-max, and Z-standard methods, respectively [12]. It is important to note that in classification problems, the number of samples in each class may vary, presenting additional challenges in data pre-processing and model training. By employing these normalisation techniques, researchers and practitioners can mitigate some of these challenges, ensuring more robust and accurate software defect prediction models.

2.2 Unbalanced learning-based methods

In binary classification scenarios, the challenge of imbalance occurs when there is a significant disparity between the number of samples in one class compared to the other. This imbalance can lead to suboptimal performance of learning algorithms, as they generally assume an equal distribution of samples across classes. To mitigate the adverse effects of dataset imbalance on prediction accuracy, several strategies have been developed, broadly categorised into data-level, algorithm-level, and cost-sensitive

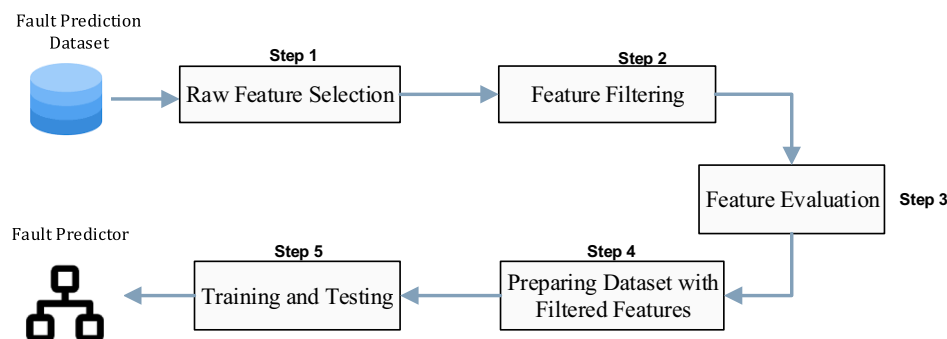
learning approaches. Data-level approaches focus on pre-processing techniques that aim to rebalance the dataset before the learning process begins, without directly altering the algorithm itself. These methods adjust the data distribution at the pre-processing stage to counteract the imbalance. They achieve this by either augmenting the number of samples in the underrepresented class (oversampling) or reducing the samples in the overrepresented class (undersampling), thus fostering a more balanced class distribution for the training process [13]. Oversampling seeks to enhance the representation of the minor class by duplicating existing samples or generating new ones, whereas undersampling reduces the imbalance by removing some samples from the major class.

On the other hand, algorithm-level approaches modify the learning algorithms to make them more sensitive to the minority class. These methods do not alter the distribution of the data but instead adjust the learning process to focus more on correctly classifying the underrepresented class. Techniques such as bagging and boosting are examples of algorithm-level approaches that can improve classification performance in the context of imbalanced datasets by enhancing the model's focus on the minority class. Together, these strategies provide a multifaceted toolkit for addressing the challenges posed by imbalanced datasets in binary classification scenarios, ensuring more accurate and equitable predictions across both classes.

2.3 Feature selection-based methods

Machine learning methods dealing with high-dimensional data, such as datasets with a large number of features, encounter several challenges. These include increased computational complexity, difficulty in extracting meaningful insights, and the need to manage model complexity to avoid overfitting during training [14]. To address these issues, dimensionality reduction techniques aim to simplify datasets by representing them in a lower-dimensional space while preserving essential characteristics of the original data. Dimension reduction strategies can be broadly categorised into feature extraction-based methods and feature subset selection-based methods. Feature extraction methods work by transforming the original high-dimensional data into a lower-dimensional space, effectively combining existing features to create a new, smaller set of features that capture the core information of the original dataset. A well-known technique in this category is Principal Component Analysis (PCA), which identifies the principal components that account for the most variance in the data. Feature subset selection is another critical approach in machine learning, especially relevant for tasks such as classification and regression. Despite the presence of numerous features in these tasks, not all contribute equally to the learning

Fig. 1 The procedure of the filter-based method for feature selection



process—some may be redundant or even detrimental to model accuracy. By removing these non-contributory features, computational efficiency is improved, and model accuracy is enhanced. The objective of feature selection is to find the smallest subset of features that is sufficient for the task at hand. Within the realm of defect prediction, feature selection methods are categorised into filtering and classification techniques [15].

The filtering approach operates independently of the machine learning algorithm, without incorporating a classification function. This method evaluates features based on specific criteria, assigning a score to each feature. Features are then ranked according to their scores, and those with the lowest rankings, typically below a certain threshold, are removed. The selected feature subset is then used for classification, as illustrated in Fig. 1, which outlines the steps involved in the filtering process. Several filter-based feature ranking techniques, such as Information Gain (IG), Information Gain Rate (GR), Symmetric Uncertainty (SU), Chi-Square Test (CS), and two variants of Relief, have been extensively studied [16]. The CS filter evaluates the distribution of classes and feature correlations. The IG filter measures the information a particular feature (feature Y) provides about the target class based on the value of another feature (feature X). However, IG has a tendency to favor features with a large number of values, which may not always be the most informative. The GR and SU methods overcome this bias by adjusting for the value count of features; GR penalizes features with many values, while SU calculates the combined entropies of features X and Y to provide a balanced measure. The ReliefF algorithm, an enhancement of the original Relief method, excels in handling noisy and multiclass datasets by effectively identifying relevant features. Figure 1 illustrates the procedure of the filter-based method for feature selection.

The set of techniques that use an evaluation function based on the error rate of the learning algorithm are referred to as classification or wrapper methods. This strategy involves generating new feature subsets through a

generator function, which are then evaluated using a ML technique. The effectiveness of each subset is determined by the number of errors in the test set or the error rate of the learning method. Typically, the classification (wrapper) method provides superior performance compared to the filter method, albeit at a higher computational cost. Two primary techniques for selecting the optimal set of features are forward selection and backward selection. Forward selection evaluates each feature for potential inclusion step by step, while backward selection starts with all features and gradually eliminates them based on a predefined stopping criterion, efficiently determining the essential features for software defect prediction [15]. Figure 2 demonstrates the workflow of classification methods, including both forward and backward selection. These selection strategies have been utilised in [17], with greedy forward selection—a specific form of forward selection—beginning with no features and adding them progressively to improve performance accuracy [18].

2.4 Blended learning-based methods

Blended learning, within the context of machine learning, emerges as a potent methodology [19]. This technique integrates the predictions from several classifiers to bolster overall learning precision. Blended learning is characterised by two principal applications. In the first, various classification algorithms are applied to tackle defect prediction challenges. The initial step often involves identifying the most effective classification algorithm. Yet, this approach does not leverage the potential insights available from employing multiple algorithms and struggles with the task of pinpointing the singularly best classifier. The second application addresses situations involving extensive and varied features, which makes it impractical to integrate all features within a single classifier. A crucial element in creating an effective blended classifier is the selection of underlying classification principles. The absence of appropriate classifiers diminishes the potential benefits of

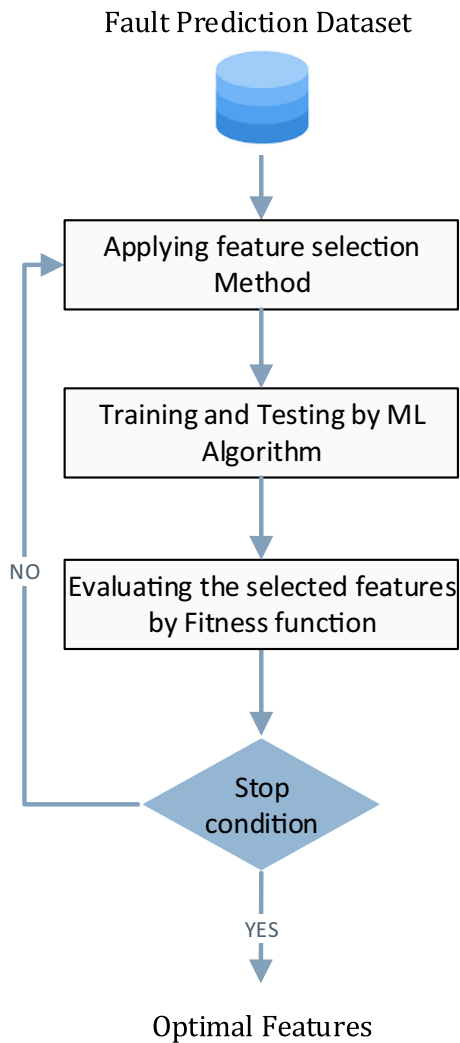


Fig. 2 The procedure for the classification methods for feature selection

diversity among classifiers, thereby limiting the effectiveness of the blended approach.

3 The proposed method

3.1 Feature selection

This section outlines the proposed method for software defect prediction, which leverages the Binary Chaos-based Olympiad Optimisation Algorithm (BCOOA) in conjunction with ML algorithms. BCOOA is utilised to identify the most impactful features within defect prediction datasets for subsequent analysis using ML algorithms. The suite of ML algorithms applied in this study comprises Artificial Neural Networks (ANN), Decision Trees (DT), K-Nearest Neighbors (KNN), Naive Bayes (NB), and Support Vector

Machines (SVM). Figure 3 depicts the workflow of the proposed method. Standard datasets were used to train the ML algorithms and evaluate the performance of the resulting software defect predictors. Consequently, BCOOA plays a crucial role in selecting features that notably improve the accuracy and precision of the developed defect predictor, thereby enhancing its classification capabilities.

3.2 Training datasets

The datasets employed in this study were sourced from the NASA repository [20], publicly accessible since 2005. The software metrics extracted from this dataset include McCabe’s complexity metrics, Halstead’s metrics, branch count, and five distinct metrics related to lines of code. Table 1 presents the datasets used in this research, while Table 2 details the 21 features within these datasets. The final feature (the 22nd feature) serves as the dependent variable, indicating the presence of defects in the program code.

In BCOOA, each participant (referred to as a “student”) is modelled as a binary vector of length 21, matching the count of features in the training dataset, labeled from F1 to F21. Each element in this binary vector is directly associated with a corresponding feature in the training dataset, as illustrated in Fig. 4. A bit value of zero means the related feature is excluded from the training set, whereas a bit value of one signifies the inclusion of that feature in the model training phase. After executing a cycle of BCOOA with a selected set of features, a fitness score is calculated for each binary vector. This score evaluates the combination of the volume of training errors and the quantity of features chosen, with the objective to reduce both the error and the number of features simultaneously. Consequently, through iterative refinement, the algorithm seeks to achieve an optimal set of features that minimizes training errors.

3.3 Structure of olympiad optimization algorithm

Following the assembly of the training dataset, the next step introduces a binary adaptation of the Olympiad Optimisation Algorithm (BCOOA) for feature selection. This adaptation is pivotal for pinpointing the most advantageous features within the training dataset prior to developing the desired classification model using machine learning algorithms. BCOOA, drawing inspiration from swarm intelligence, leverages both local and global search strategies. This method, characterised by its population-based and collaborative metaheuristic approach, tackles optimisation challenges through a divide-and-conquer strategy. Within this framework, each entity in the BCOOA population

mimics the learning process of a student preparing for an Olympiad competition, with each “student” represented by a 21-element vector corresponding to the number of features in the dataset. Initially, every element of the vector is set to 1, indicating that all features are considered for training. This vector undergoes evolution within a population matrix that consists of nPOP individuals across 21 features. The population progresses through cycles of interactive teaching and learning, continually refining the feature selection process. As depicted in Algorithm 1, BCOOA’s methodology is outlined, showcasing its approach to optimisation by effectively combining local and global search methods. In this structured process, the population is periodically divided into smaller, equally sized groups, facilitating a focused and efficient search for the optimal feature set. The time complexity of the proposed algorithms is $O(\text{MaxIter} \times \text{TeamNum} \times \text{TeamSize})$. The variable TeamNum indicates the number of teams and the TeamSize indicates the number of individuals in each team. All teams include same number of students (TeamSize).

Algorithm 1 Pseudocode of the proposed BCOOA

```

BCOOA Algorithm {
1      Initialize the first population by Chaos;
2      itr=0;
3      for i=1 to maxitr do
4          {
5              Partition population into n team
6              i=2;
7              while (i<=n)
8                  {
9                      Imitation of team(i) from team (i-1);
10                     i = i + 1;
11                 }
12             for i=1 to n do in parallel
13                 {
14                     imitation of team(i) members from the local best member;
15                 }
16             for i=1 to n do in parallel
17                 {
18                     mutation of team(i)’s best student;
19                 }
20             mutation of global best student;
21             merging teams into a single total population;
22             Evaluating the fitness of students by Eq. (4);
23         }
24     return the global best as the best selected features;
25 }
26

```

the initial populations are generated using chaotic maps, specifically logistic maps, which can produce a solution space with a favourable distribution. Chaotic sequences promote diversity within the initial population, expedite convergence, and enhance global search capabilities. Two sets of experiments were conducted to explore the use of chaotic-based initial populations in the defect prediction problem. In the first experiment, the initial population was randomly generated, while in the second, chaos theory was utilised. A primary chaotic map used in the optimisation problem is the logistic map, applied as a second-degree polynomial mapping to generate the initial population in the second set of experiments. In Eq. (1), which represents the logistic map equation, X is the population members at time t , and r is a positive constant indicating the growth rate. When r is set to a very low value, the population diminishes, whereas stable outcomes are achieved with higher growth rates (r). For the benchmarking of the replica placement, r is experimentally calibrated to 4. In Eq. (1), the value of X_n falls within the range of 0 to 1.

3.4 Chaos-based population

To ensure an even distribution of the population across the search spaces, two key characteristics to consider are the initial population’s ergodicity and diversity. In this study,

$$X_{n+1} = r \times X_n \times (1 - X_n) \quad (1)$$

$$r = 4$$

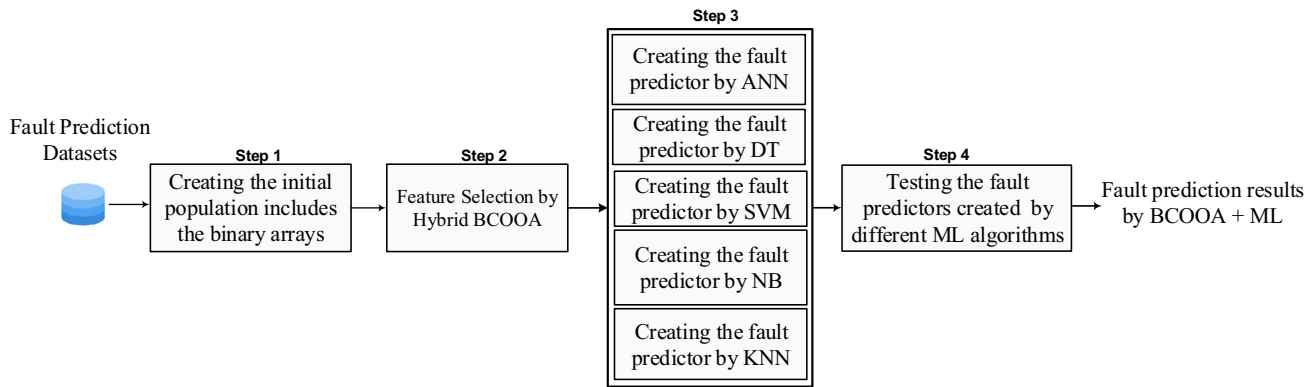


Fig. 3 The workflow of the proposed defect prediction

Table 1 Specification of NASA datasets utilised in this study

Number	Project name	Programming language	Number of models	Percentage of defected models (%)	Project description
1	CM1	C	498 (349 Train, 149 Test)	9.7	NASA spacecraft instrument
2	JM1	C	10,885 (7619 Train, 3266 Test)	19	A real-time predictive ground system that utilises simulations to produce forecasts
3	KC1	C++	2109 (1476 Train, 633 Test)	15.4	System executing storage management to receive and process ground data
4	KC2	C ++	522 (365 Train, 157 Test)	20	Science data processing
5	PC1	C	1109 (776 Train, 333 Test)	6.8	Flight software for Earth-orbiting satellite

3.5 Learning operation in OOA

Algorithm 2 presents the pseudo-code for the core segment of the Olympiad Optimisation Algorithm (OOA), with line 9 detailing the MATLAB implementation for dividing the population into subgroups. These subgroups utilise unique imitation strategies to probe different areas of the extensive solution space. A competitive learning atmosphere is cultivated among the individuals, each possessing a memory to record their progress. Within OOA, every participant, referred to as a student, is modelled and executed as a numerical array, together constituting the student population that represents potential solutions. As depicted in Algorithm 1, the process begins by dividing the student population into n teams after sorting. Each team consists of m students, with the foremost team acknowledged as the global best and the final team as the global worst. Teams embark on exploring their designated local solution spaces, with the lead student of each team acting as its local best.

Students, organised into teams, strive to assimilate knowledge from either the local best student of a neighbouring team or from the global best team. The learning

operator introduced here integrates both local and global search approaches within OOA, identified as the key mechanism for identifying the optimal solution. This operator's goal is to augment the collective knowledge (fitness) of the population, classifying individuals by their knowledge (fitness function). Through the deployment of this learning operator, OOA seeks to enhance the overall knowledge of the population, encompassing four critical phases. Initially, students in each team gain insights from their peers in the neighbouring team, leading to the foremost team's knowledge flowing to others, similar to a bubble effect. This phase involves sequential knowledge transfer from one team to another. Should the lesser-performing students within a team fail to advance their knowledge, the next step involves mutating the best student within the team to introduce variety. This mutation acts as a local search targeting the optimal student of the less proficient team. In the absence of improvement from this intervention, the third phase is initiated, where all students in the teams with lesser proficiency teams receive instruction from their counterparts in the globally leading team, the first team.

In situations where the top-performing students are unable to transfer knowledge to those in the least performing team, a mutation strategy is applied to the student with the highest global performance. This mutation, targeting the globally best student, is designed to prevent convergence to local optima by introducing a degree of variability. As a result, agents from various teams come together to form a refreshed population. This process of renewing the population is iteratively managed by the

(student) in the population of this method contains a serial number. Since the problem is about selecting or not selecting a feature, the new binary answer must contain 0 and 1. A feature is chosen for the new dataset if its value is 1, and it is not selected if its value is 0. The Sigmoid function [21] was used to shift the values of the individual array of the OOA into the binary space. In the proposed method, BCOOA is utilised to select effective features for ML algorithms. OOA repositions entities within the state

Algorithm 2 The pseudo-code of the OOA's learning operator

```

1  Function studentType Olampiyad_Learn (BestStd, WorstStd)
2  {
3      nVar= length(Student_array);
4      LearnCount=ceil((30 * nVar)/100);    %imitation count
5      count=0;
6      i = 0;
7      NewStd=WorstStd;
8      while (count<=LearnCount and i < nVar)
9          {
10             aL=randi(nVar);
11             if (WorstStd(aL) !=BestStd(aL))
12                 {
13                     NewStd(aL)=BestStd(aL);
14                     count=count+1;
15                 }
16             i = i + 1;
17         }
18     Return (NewStd);
19 }

```

application of the learning operator, as detailed in Algorithm 2, which outlines the pseudo-code for each iteration within the OOA. The learning process, also referred to as the imitation mechanism, is implemented through a crossover technique. This process entails the substitution of specific bits (elements) in the least performing individual with those from the highest performing individual. The parameter LearnCount specifies the quantity of bits to be exchanged, with the optimal proportion identified through empirical research to be 30% of array's length.

3.6 Binary OOA

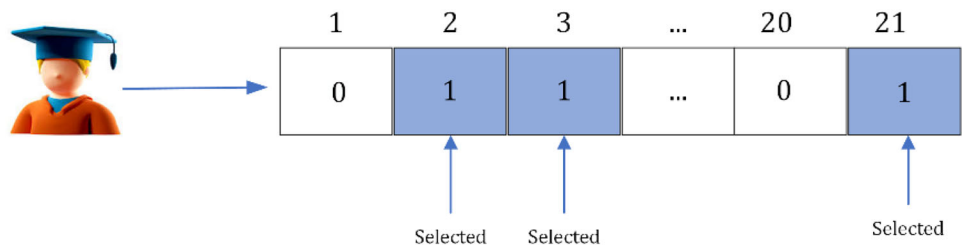
In this section, a novel binary strategy based on the Sigmoid function for OOA is presented. Every solution

space. In specific problems like feature selection, solutions are constrained to binary values of 0 and 1, creating a binary variant. In BCOOA, the learning algorithm (Algorithm 2) updates the student array. It attracts each student towards the first solution (student). The vector $\vec{X}_i^d(t)$ shows the position of i^{th} student in dimension d and iteration t . Therefore, in this proposed model, the Sigmoid function is applied in a binary form to change the solution's location in the OOA method, as shown in Eq. (2). The output of the Sigmoid transfer function is a value between zero and one. Therefore, to convert it into a binary (discrete) value, a threshold needs to be provided. The Sigmoid function incorporates the random threshold given in Eq. (3) to choose the feature by converting it to a binary solution. The *rand* is a number that has a uniform distribution and ranges from 0 to 1. The available solutions for the BCOOA

Table 2 Description of features used in NASA datasets

Number	Feature	Description
1	LOC	'Line count of code (McCabe)'
2	v(g)	'Cyclomatic complexity (McCabe)'
3	eV(g)	'Essential complexity (McCabe)'
4	iv(g)	'Design complexity (McCabe)'
5	N	'Operators + Operands (McCabe)'
6	v	'Volume (Halstead)'
7	L	'Program length (Halstead)'
8	D	'Difficulty (Halstead)'
9	I	'Intelligence (Halstead)'
10	B	'Effort: Halstead effort to write a program'
11	E	'Number of Delivered Bugs (Halstead)'
12	T	'Time to write program (Halstead)'
13	LOCcode	'Line count (Halstead)'
14	LOCcomment	'Count of lines of comments (Halstead)'
15	LOBlank	'Count of blank lines (Halstead)'
16	LOCcodeAndComment	'Count of lines which contain both code and comments (Halstead)'
17	uniq_Op	'unique operators (Halstead)'
18	uniq_Opnd	'unique operands (Halstead)'
19	total_Op	'Total operators (Halstead)'
20	total_Opnd	'Total operands (Halstead)'
21	branchCount	'Number of branches in flow graph (Halstead)'
22	defects	'Number of defects'

Fig. 4 The structure of a student in the BCOOA, as a binary vector to specify features



population are enforced to move inside a binary (discrete) search space by using Eqs. (2) and (3). These equations are then precisely entered into BCOOA.

$$SG(X_i^d(t)) = \frac{1}{1 + e^{-X_i^d(t)}} \tag{2}$$

$$X_i^d(t + 1) = \begin{cases} 0 & \text{if } r \text{ and } < SG(X_i^d(t)) \\ 1 & \text{if } r \text{ and } \geq SG(X_i^d(t)) \end{cases} \tag{3}$$

To determine the fitness of feature vectors, Eqs. (4) and (5) are proposed. These equations assess the fitness of a solution (the selected features) as the sum of the training error and the normalised count of features, functioning as a minimisation objective. In Eq. (4), the training error, which ranges from zero to one, is calculated by subtracting the model’s accuracy from 100 and then normalising this value within the [0, 1] interval. Additionally, the term “number of features” refers to the count of selected features for

training, which is also normalised to fall within the [0, 1] range.

$$Fitness(i) = Error\ Percentage\ Test(i) + \frac{Used\ Features}{22} \tag{4}$$

$$Error\ Percentage\ Test(i) = \frac{(100 - Accuracy(i))}{100} \tag{5}$$

4 Experiments and results

4.1 Experiments platform and datasets

This section evaluates the performance of the proposed method for predicting software defects. Initially, results were gathered from machine learning algorithms (ANN, DT, K-NN, NB, and SVM) using all features available through a conventional approach. Subsequently, we applied

Table 3 The BCOOA parameters' value to select the effective features

Algorithms	Parameter name	Value
Olympiad Optimisation Algorithm (OOA)	Quantity of students	40
	Quantity of teams	10
	Size of teams	4
	Rate of Learning	Random values between [0.2–0.8]
	Imitation count	1
Ant Colony Optimisation algorithm (ACO)	Number of iterations	100
	Quantity of nests	30
	Lavy distribution parameter	1.5
	Step length	0.01
	Quantity of iterations	100

Table 4 Features of the used training and test data

Dataset	Total quantity of samples	Quantity of test samples	Quantity of test samples
CM1	496	347	149
JM1	10,885	7619	3266
KC1	2109	1476	633
KC2	522	365	157
PC1	1109	776	333

Table 5 Confusion matrix

Model prediction		
Yes	No	
TP	FN	Real Yes (A)
FP	TN	Real No (B)

the proposed method using BCOOA, which is specifically tailored for effective feature selection. A comparative analysis of the outcomes from both methodologies is presented. The implementation of the proposed method, integrated with the Ant Colony Optimiser (ACO), was executed on MATLAB version 2022 on a Windows 10 PC, equipped with a Core i7 Intel processor operating at 2 GHz and 4 GB RAM. Table 3 details the calibration parameters used in both the BCOOA and ACO methods for effective feature selection in software defect prediction. The binary version of ACO was introduced as a feature selection tool in [22].

The training and testing phases were conducted using five datasets as detailed in Table 1. Initially, 70% of the data were randomly allocated for training the ANN, DT, K-NN, NB, and SVM models. The testing phase then utilised the remaining 30% of the data, which was not involved in the training phase, to evaluate the performance

of the learning models. Table 4 presents the characteristics of both the training and testing data. The Confusion Matrix, provided in Table 5, offers the essential metrics required to calculate sensitivity, specificity, and accuracy.

4.2 Evaluation criteria

In this study, 'accuracy', 'precision', 'recall', and 'F1' are used as evaluation criteria in software modules classification problems; Eqs. (6), (7), (8) and (9) were used to calculate these criteria.

$$Accuracy = \frac{True\ Positive + True\ Negative}{All\ test\ samples} \quad (6)$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (7)$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (8)$$

$$F_1 = \frac{2 * (Precision * Recall)}{Precision + Recall} \quad (9)$$

Generally, the confusion matrix serves as a tool to assess the accuracy and effectiveness of classification models. Its analysis in classification and prediction involves delineating four outcomes: True Positives (TP), where the model correctly predicts the positive class; False Positives (FP), where the model incorrectly predicts the positive class; True Negatives (TN), where the model correctly predicts

the negative class; and False Negatives (FN), where the model incorrectly predicts the negative class. The confusion matrix provides three critical metrics—precision, accuracy, and recall—used to assess the performance of software module classification models. A comprehensive series of experiments was conducted on classifiers developed using various ML algorithms, including ANN, DT, K-NN, NB, and SVM. These experiments aimed to answer specific Research Questions (RQs) related to each dataset.

- RQ1: Does the proposed method increase the accuracy, precision, recall, and F1 of ML algorithms in the software defect prediction problem?
- RQ2: Does the proposed method identify and eliminate the ineffective features of the training datasets?
- RQ3: Is the convergence speed and success rate of the proposed BCOOA higher than the other methods?

4.3 Discussion of the results from the first experiment (CM1 dataset)

The initial experiment utilised the CM1 dataset for both training and testing to address RQ1. Table 6 shows the results obtained from applying the ML algorithm to the CM1 dataset, evaluating the performance metrics for ANN, DT, K-NN, NB, and SVM without using BCOOA for feature selection. This table compares the classifiers developed using the aforementioned ML algorithms in terms of accuracy, precision, recall, and F1 score. These experiments were first conducted on the raw training set, which included all available features.

Redundant and irrelevant features within the training dataset can lead to overfitting in machine learning (ML) algorithms. The primary objective of feature selection is to identify a concise set of features that are sufficient for accurate prediction of the target label. Redundant features have the potential to mislead ML algorithms, while feature selection methods aid in reducing costs by eliminating unnecessary information. Subsequent experiments were conducted on each dataset using BCOOA in conjunction with ML algorithms. BCOOA was employed to identify the optimal features within the training sets. Table 7 demonstrates the outcomes of applying ML algorithms to the CM1 dataset, utilising BCOOA for feature selection. A comparative analysis of the results presented in Tables 6 and 7 reveals that the efficiency of predicting software defects, based on the features selected by BCOOA, surpasses that of the previous method. Specifically, the accuracy and precision of the classifier created by ANN without using BCOOA were 88.15% and 88.46%, respectively; these figures increased to 92.03% and 98.02% when BCOOA was applied for feature selection. The experimental results show that the integration of BCOOA

Table 6 Performance criteria of ML algorithms on the CM1 dataset without feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	88.15	88.46	99.07	93.47
DT	83.89	93.23	89.20	91.17
KNN	89.93	95.58	93.52	94.54
NB	88.59	95.52	92.08	93.77
SVM	84.56	90.37	92.42	91.38

Table 7 Performance of ML algorithms on the CM1 dataset using the selected effective features by BCOOA

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	92.03	98.02	98.14	97.26
DT	94.18	91.38	99.00	96.8
KNN	98.20	99.00	97.8.0	94.1
NB	93.45	96.14	99.87	98.00
SVM	90.91	92.39	99.83	97.08

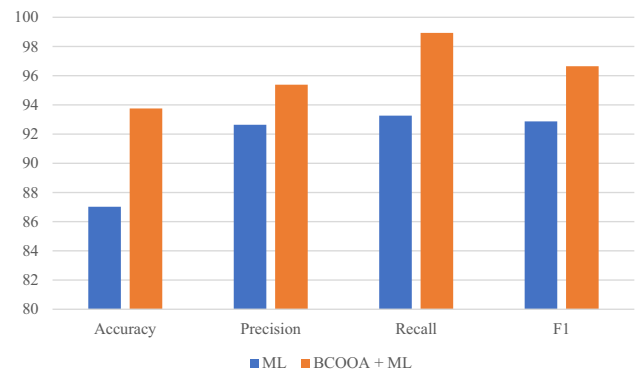


Fig. 5 The average values of the performance criteria of ML algorithms (ANN, DT, KNN, SVM and NB) with and without BCOOA as the feature selection method in the CM1 dataset

enhances the performance of ML algorithms in predicting software defects.

Figure 5 displays the average performance of classifiers generated by ML algorithms, both with and without feature selection. The average accuracy of classifiers created without the feature selection method is 87.02%, which significantly improves to 93.75% when applying the proposed feature selection method across the same ML algorithms (ANN, DT, K-NN, NB, and SVM). Similar improvements are observed in other performance metrics. For example, the average precision for classifiers developed without and with the proposed feature selection algorithm is approximately 92.63% and 95.38%, respectively. Moreover, the average recall metric for classifiers developed using the proposed method markedly exceeds that of classifiers generated solely by ML algorithms. Thus,

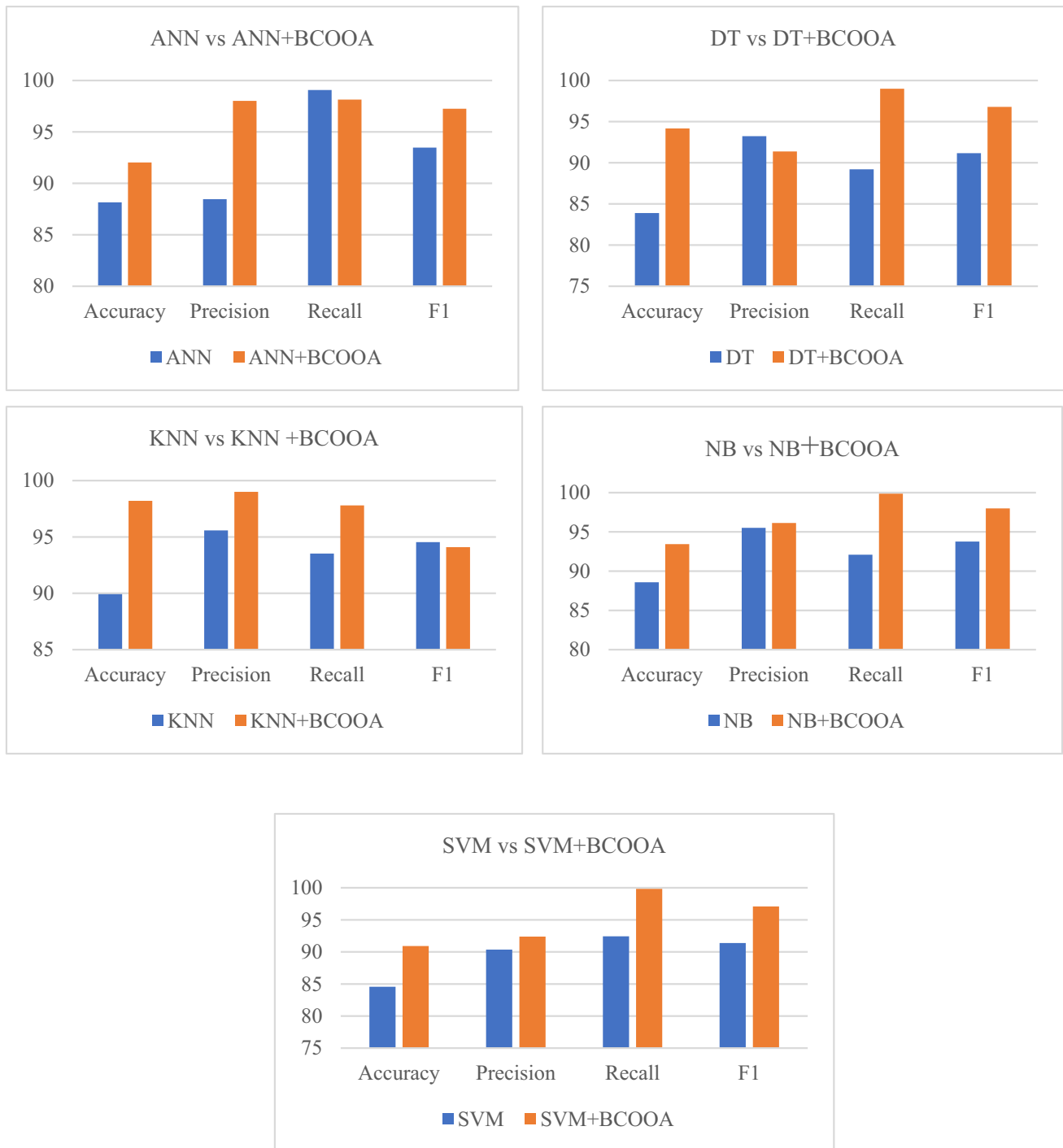


Fig. 6 The impact of BCOOA on the performance of ML algorithms in the CM1 dataset

the proposed method exhibits greater effectiveness in comparison to the use of ML algorithms without feature selection.

Figure 6 illustrates a comparison between the performance metrics of traditional ML algorithms and those optimised by BCOOA (a feature selection method) on the CM1 dataset. The simulation results reveal that ML

algorithms, when applied to a training dataset with selectively chosen features, outperform those using a conventional ML approach. The incorporation of selected effective features through BCOOA leads to the ML + BCOOA method, showcasing superior performance compared to standalone ML algorithms. Overall, the proposed BCOOA method for selecting effective features

demonstrates notable performance improvements when applied to ML algorithms using the CM1 dataset.

Table 8 The average quantity of selected features by BCOOA in different ML algorithms in CM1 dataset in ten times executions

	ANN	DT	KNN	NB	SVM
Brute force ML	21	21	21	21	21
BCOOA + ML	9	6	9	8	9
ACO + ML	8	7	15	4	11

Fig. 7 The selection rate of the 21 features by the BCOOA in the CM1 dataset during ten executions

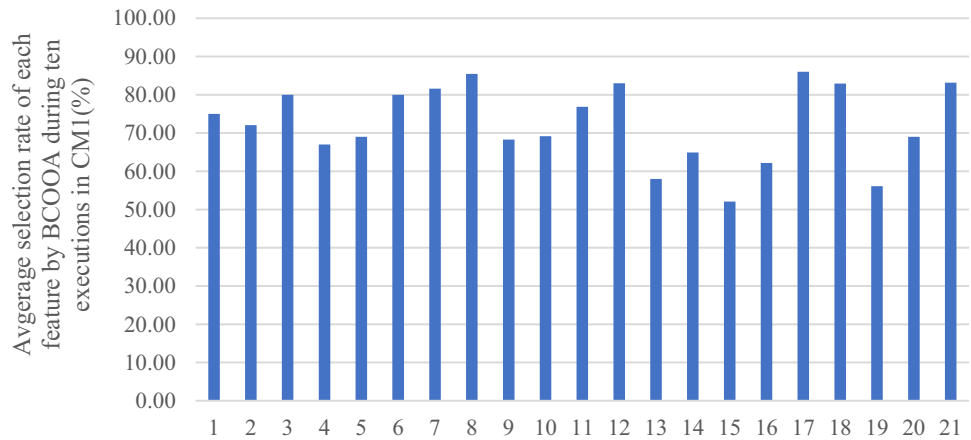


Table 9 The selected features by BCOOA and ACO in different ML algorithms in CM1 in the best result from 10 executions

	SVM		NB		KNN		DT		ANN		
	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	
0	1	1	0	1	0	0	1	1	1	1	1
1	1	0	0	0	0	0	0	1	1	0	2
1	1	0	0	1	1	0	0	0	1	1	3
1	0	1	1	0	1	1	1	0	1	0	4
0	0	0	0	1	1	0	0	0	0	1	5
0	1	0	0	1	1	0	0	1	1	1	6
0	1	1	0	1	1	1	1	0	0	0	7
1	1	0	1	1	1	0	1	0	0	1	8
0	0	0	0	0	0	1	0	0	0	1	9
1	1	1	0	1	0	0	0	0	0	1	10
0	0	0	1	0	1	0	1	0	0	1	11
0	1	0	1	0	1	1	1	1	1	1	12
1	1	0	0	1	0	1	0	1	0	0	13
1	1	0	0	0	1	0	0	0	0	0	14
1	1	0	0	1	0	0	0	0	0	0	15
1	1	0	1	1	0	1	0	0	0	0	16
1	1	0	1	1	1	1	1	0	0	1	17
1	1	0	0	1	1	0	1	0	0	0	18
0	0	0	1	0	0	0	0	0	0	0	19
1	1	0	1	1	0	1	1	1	0	0	20
0	1	1	1	0	1	0	1	0	1	1	21

Another critical aspect of feature selection-based defect prediction methods is the number of features selected by the BCOOA algorithm. Ineffective features can undermine a model’s ability to generalise, potentially diminishing overall classifier accuracy. Moreover, as the quantity of variables increases, so does the model’s complexity. The proposed feature selection algorithm identifies effective features, thereby improving defect prediction accuracy. Table 8 displays the number of effective features identified by BCOOA across various ML algorithms. For the CM1 dataset—a real-world defect prediction dataset consisting

of 21 independent and one dependent feature—BCOOA selects a concise subset of the most effective features.

The proposed method's average performance, calculated over ten runs, showed consistent effectiveness in detection. Table 8 compares the average number of features selected by BCOOA and binary ACO across ten runs. For the ANN classification, both BCOOA and ACO selected 9 and 8 features respectively. However, for the DT classification algorithm, BCOOA chose seven features, while ACO picked eight. Similarly, for the KNN classification algorithm, BCOOA identified nine features, in contrast to ACO's 15. The number of features selected by BCOOA consistently remained lower than that of other algorithms, highlighting BCOOA's superior efficiency in pinpointing the most effective features compared to ACO. Figure 7 shows how often various features in the CM1 dataset were selected by BCOOA across ten runs.

Table 9 lists the features selected by BCOOA for different ML algorithms in the CM1 dataset, showcasing the optimal results from 10 runs. Each ML algorithm, alongside the feature selectors, was executed ten times, and the features chosen during the most successful runs (based on the highest evaluation criteria) were deemed the most effective. This table shows the unique features identified by different algorithms, revealing variations in feature selection. The proposed method's application on the CM1 dataset was repeated ten times to verify its reliability, and the findings were documented. These results demonstrate significant improvements in accuracy and precision. With a total of 21 features, the algorithm leverages these effective features for both training and testing, thus shortening the execution time of the learning algorithm and boosting its efficiency. Some features were consistently selected during the training by BCOOA, indicating their higher effectiveness. Figure 7 illustrates the comparative importance of each feature in the CM1 dataset predictions. Table 9 specifies the features BCOOA selected in the most successful run on the CM1 dataset.

4.4 Discussion of the results from the second experiment (KC1 dataset)

The second series of experiments was conducted using the KC1 dataset to address specific research questions. The experiment employed the KC1 dataset for both training and testing purposes, representing features from a real-world software application. KC1 comprises 2107 records and 21 features, with 1475 records allocated for training and 633 records for testing. Table 10 shows the performance metrics of classifiers generated by various ML algorithms without BCOOA. The defect predictors (classifiers) created by ANN, DT, KNN, NB, and SVM, utilising the raw training set encompassing 21 features, are depicted. These

Table 10 Performance criteria of ML algorithms on the KC1 dataset without feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	84.67	85.01	97.05	91.49
DT	83.57	88.47	91.65	90.56
KNN	89.63	89.06	90.19	81.83
NB	80.41	88.16	91.01	89.91
SVM	84.67	84.67	100.00	91.49

Table 11 Performance criteria of ML algorithms on the KC1 dataset with BCOOA feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	89.41	91.20	99.70	92.20
DT	87.20	89.90	96.00	92.51
KNN	96.08	95.88	97.89	93.28
NB	87.08	90.18	97.00	93.00
SVM	88.40	89.21	99.80	94.93

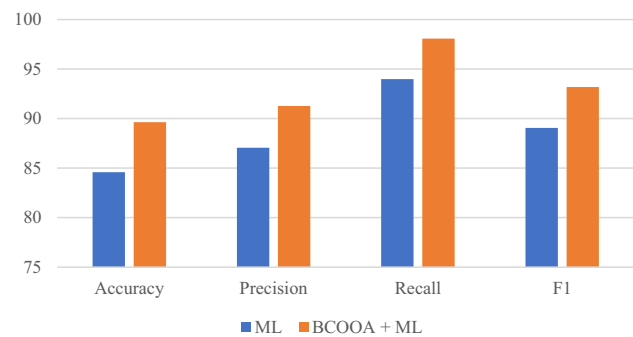


Fig. 8 The average values of the performance criteria of ML (ANN, DT, KNN, NB, and SVM) algorithms with and without BCOOA as the feature selection in the KC1 dataset

classifiers have been assessed based on accuracy, precision, recall, and F1 metrics. In the initial trials using the raw training set with all features considered, KNN's classifier, trained on the filtered dataset by BCOOA, demonstrated the highest accuracy and precision. Conversely, the defect predictor generated by ANN outperformed other models concerning recall and F1 metrics.

The presence of unnecessary features in the training dataset can lead to overfitting in ML algorithms, potentially misleading the models. To evaluate the impact on ML algorithm performance, we utilised the KC1 dataset as another benchmark. In the second round of tests conducted on the KC1 dataset, we employed BCOOA alongside ML algorithms. Table 11 presents the outcomes derived from applying the ML method to the KC1 dataset to ascertain the most effective features. For instance, the classifier built using ANN without BCOOA exhibited an accuracy and

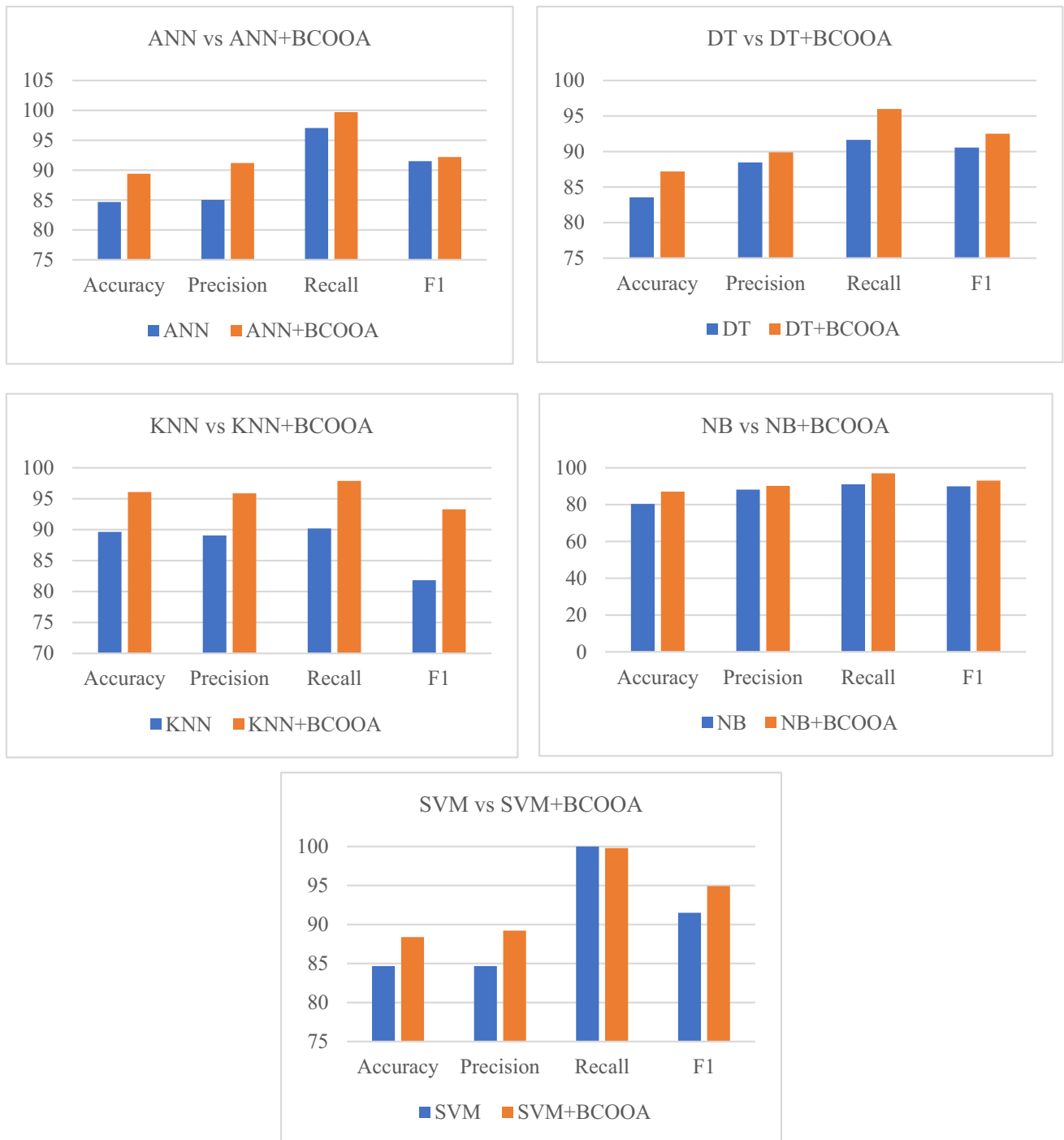


Fig. 9 The impact of BCOOA on the performance of the ML algorithms in the KC1 dataset

precision of 84.67% and 85.01%, respectively. However, upon employing BCOOA as a feature selection method, these metrics notably increased to 89.41% and 91.20%. These results indicate that the utilisation of BCOOA significantly enhances the performance of ML algorithms in the software defect prediction domain when applied to the KC1 dataset. Comparing Tables 10 and 11, it is evident that

the efficiency criteria for predicting software defects via the selection of effective features using BCOOA surpass the prior method.

Figure 8 illustrates the average performance of classifiers generated by ML algorithms, both with and without feature selection, applied to the KC1 dataset. Upon employing the suggested feature selection strategy with

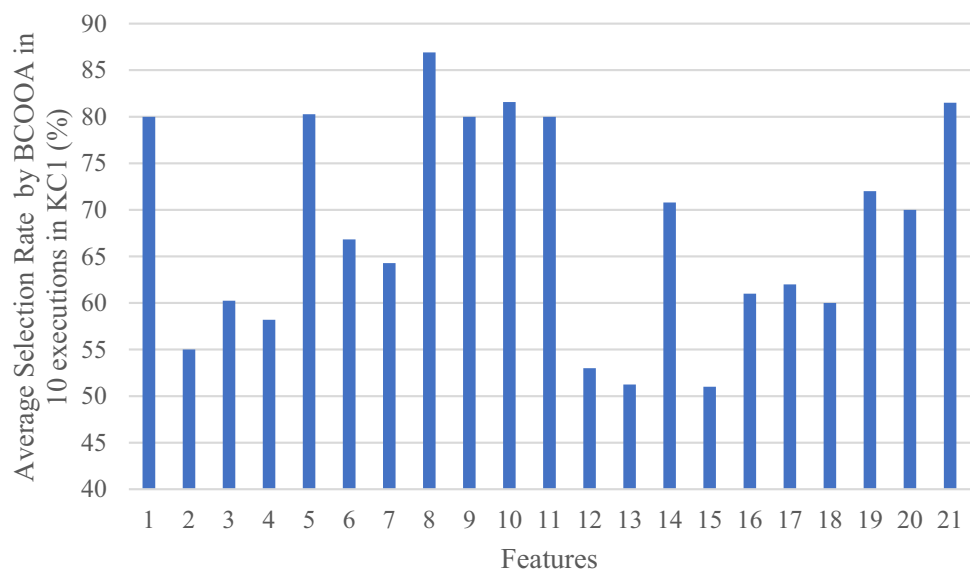
ML algorithms (ANN, DT, KNN, NB, and SVM), the average accuracy of classifiers created by ML and standalone ML + BCOOA algorithms stands at 84.59% and 89.63%, respectively. Comparable trends are observed across other performance criteria. Specifically, classifiers generated by ML algorithms without (ML) and with the suggested feature selector (ML + BCOOA) exhibit an average precision of approximately 87.05% and 91.27%, respectively. Notably, the classifiers developed using the proposed technique show significantly higher average recall metrics compared to standalone ML algorithms. In summary, classifiers derived through the suggested technique outperform those produced by ML algorithms lacking feature selection. BCOOA enhances defect prediction model accuracy by identifying and eliminating ineffective features. Following ten executions, the suggested approach's average performance on the KC1 dataset shows commendable detection capabilities.

Figure 9 presents a comparison between the performance metrics of enhanced ML algorithms, utilising BCOOA as a feature selection technique, and traditional ML methods on the KC1 dataset. Notably, the ML algorithms operating on the training dataset with selected features outperform the traditional ML approach. This figure provides a comprehensive comparison of

Table 12 The average quantity of selected features by BCOOA and ACO in different ML algorithms in the KC1 dataset in ten times executions

	ANN	DT	KNN	NB	SVM
Brute force ML	21	21	21	21	21
BCOOA + ML	9	9	11	8	9
ACO + ML	16	12	9	3	9

Fig. 10 The selection rate of the 21 features by the BCOOA in the KC1 dataset during ten executions



performance measures between the conventional and optimised ML models applied to the KC1 dataset. The ML + BCOOA approach exhibits superior performance, capitalising on the chosen effective features. Using BCOOA for feature selection demonstrates satisfactory performance when applied to ML algorithms using the KC1 dataset. Notably, the accuracy, precision, recall, and F1 metrics of the optimised ML algorithms employing the BCOOA method surpass all other models. In summary, BCOOA for selecting effective features results in commendable performance on the KC1 dataset.

Table 12 illustrates the average quantity of features selected by binary ACO and BCOOA across ten executions within the KC1 dataset. Observing Table 12, when ANN was employed as the classification algorithm, BCOOA selected 9 features while ACO selected 16. In the case of the DT classification algorithm, BCOOA and ACO selected 9 and 12 features, respectively. For the KNN classification algorithm, BCOOA and ACO selected 11 and 9 features, respectively. Moreover, in the NB classification algorithm, BCOOA selected 8 features, whereas ACO selected 3. In summary, the average quantity of features selected by the proposed BCOOA consistently remains lower compared to brute force ML algorithms.

Figure 10 illustrates the selection frequency of features in the KC1 dataset across ten executions by BCOOA. Table 13 presents the features selected by BCOOA across various ML algorithms, yielding the most optimal outcomes after ten executions. Each ML method underwent ten iterations, and the most effective features were those selected in the best-performing implementations, marked by the highest evaluation criteria. This table shows the most efficient features obtained from diverse techniques in their most successful executions.

Table 13 The selected features by BCOOA and ACO in different ML algorithms in KC1 in the best result from 10 executions

	SVM		NB		KNN		DT		ANN		#
	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	
	1	0	0	1	1	1	1	1	1	1	1
	1	1	0	0	1	0	1	1	1	0	2
	0	0	1	1	0	1	1	0	1	1	3
	0	1	1	0	1	0	0	1	1	1	4
	0	0	0	1	0	1	1	1	1	1	5
	1	0	0	1	1	1	0	0	1	1	6
	1	1	1	0	1	0	0	1	1	1	7
	1	1	0	1	1	1	1	1	0	1	8
	0	1	0	1	0	1	0	0	0	1	9
	0	1	1	1	0	0	1	1	1	1	10
	0	1	1	1	0	1	1	1	1	0	11
	1	1	0	0	0	1	1	0	1	0	12
	0	0	0	0	0	0	0	1	0	0	13
	0	0	0	1	1	1	0	0	0	1	14
	0	0	0	1	0	0	1	0	1	0	15
	1	1	1	1	1	0	1	0	1	0	16
	0	0	0	0	0	1	1	0	1	1	17
	1	0	0	0	1	1	0	0	0	1	18
	1	1	0	1	1	0	1	1	0	0	19
	1	1	1	1	1	1	0	0	0	0	20
	0	1	0	1	0	1	0	0	0	1	21

Table 14 Performance criteria of ML algorithms on the JM1 dataset without feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	75.44	81.97	85.82	84.89
DT	75.44	81.67	85.82	84.89
KNN	86.16	87.70	90.01	81.19
NB	80.94	84.52	93.94	88.98
SVM	79.87	79.87	100.00	88.81

Table 15 Performance criteria of ML algorithms on the JM1 dataset with the BCOOA feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	83.01	84.17	98.92	91.07
DT	80.73	83.27	96.35	89.49
KNN	93.00	93.43	93.81	90.00
NB	82.00	83.76	96.80	90.18
SVM	92.08	91.00	93.08	90.08

The proposed approach was executed ten times on the KC1 dataset to establish its stability, exhibiting noticeable enhancements in precision and accuracy metrics. This method considers 21 features and selectively utilises the most effective features to train the classifier. During BCOOA's training, a consistent selection of specific features occurred repeatedly, highlighting their significant impact on the accuracy and precision of the generated classifier. Table 13 indicates the selected features by BCOOA in the best-case scenario in the KC1 dataset.

4.5 Discussion of the results from the third experiment (JM1 dataset)

The third series of experiments, targeting specific research inquiries, employed the JM1 dataset for both training and testing purposes. The JM1 dataset consists of 21 features and 10,885 records, with the training set comprising 7619 records and the test dataset containing 3266 records. Table 14 shows the performance metrics of classifiers developed using various ML algorithms without BCOOA. These classifiers were built using the raw training set, which includes 21 features and 7619 records, utilising algorithms such as ANN, DT, KNN, NB, and SVM. All 21 features were considered in this raw training set. Notably, the KNN classifier emerged as the most accurate and

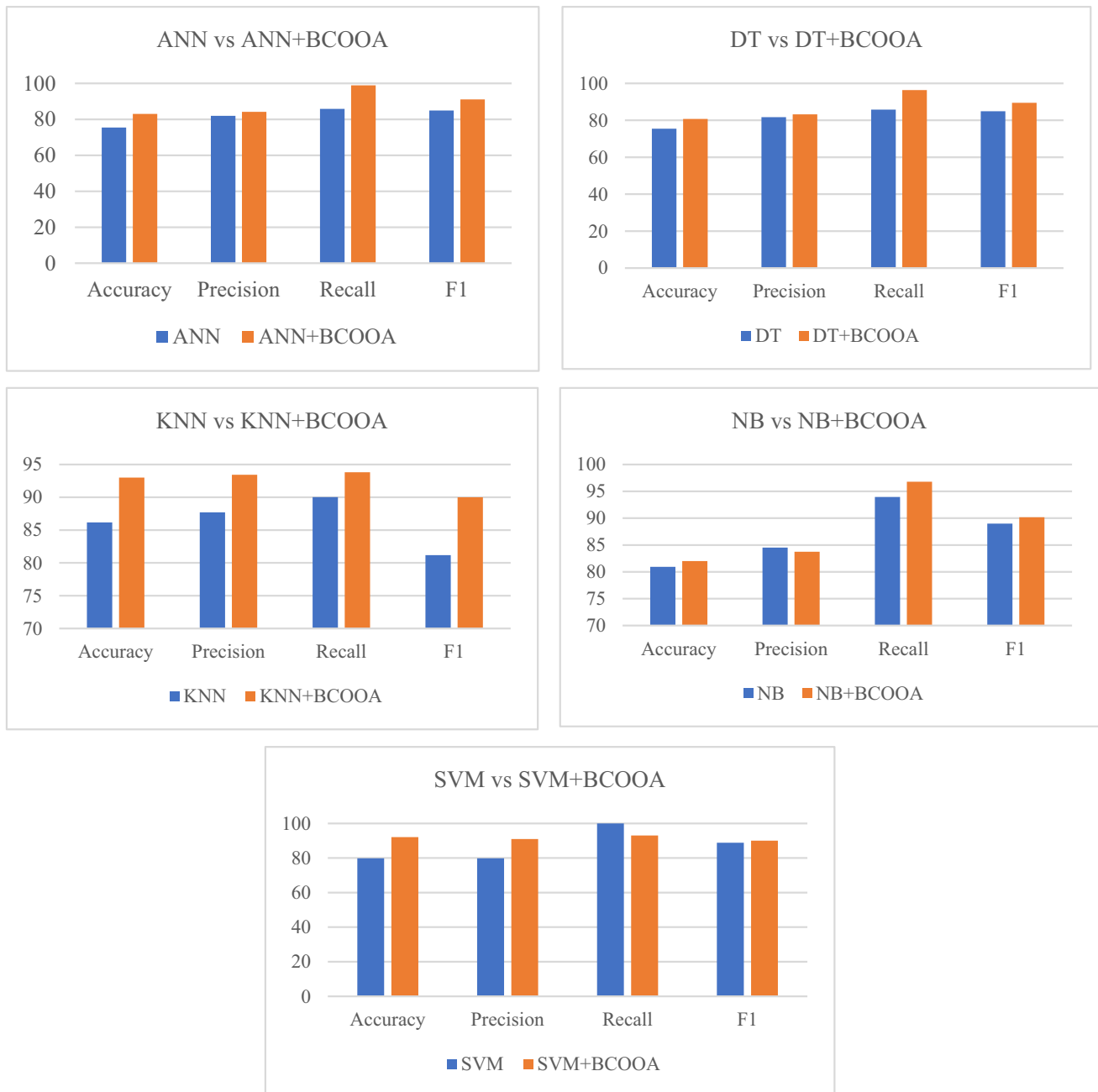


Fig. 11 The impact of BCOOA on the ML algorithms' performance in the JM1 dataset

precise among the prediction models, outperforming others in terms of accuracy and precision.

Ineffective features within the training dataset can often lead to overfitting in ML algorithms. In the second phase of the third experiment, the JM1 dataset underwent evaluation using BCOOA in conjunction with ML algorithms. Table 15 presents the test results of classifiers developed by ML algorithms leveraging BCOOA to determine the most effective features for the JM1 dataset. Classifiers developed using KNN and SVM, integrated with BCOOA, achieved accuracies of 93.00% and 92.08%, respectively. Notably,

employing BCOOA in ML algorithms enhanced the accuracy and precision of the generated classifiers.

For example, the recall and F1 metrics of the defect predictor built by ANN escalated from 85.82% and 84.89%, respectively, to 98.92% and 91.07% when BCOOA was employed as the feature selection method. Consistent with other datasets, these experiments underscore that ML algorithms perform significantly better when BCOOA is utilised as a feature selector within the JM1 dataset. Comparing Tables 14 and 15, it is evident that the efficiency of software defect predictors leveraging BCOOA

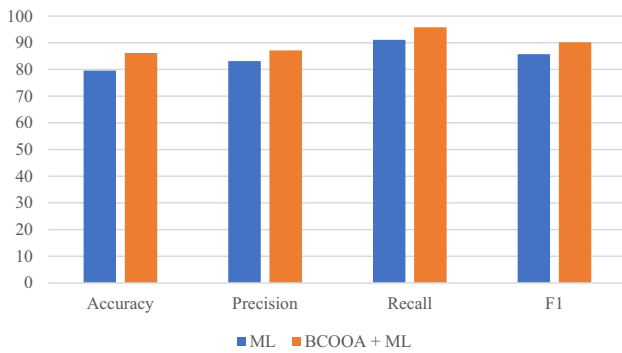


Fig. 12 The average values of the performance criteria of ML (ANN, DT, KNN, NB and SVM) algorithms with and without BCOOA as the feature selection in the JM1 dataset

Table 16 The average quantity of the selected features by ACO and BCOOA in different ML algorithms in the JM1 dataset

	ANN	DT	KNN	NB	SVM
Brute force ML	21	21	21	21	21
BCOOA + ML	12	8	9	8	8
ACO + ML	12	9	8	6	8

as a feature selector surpasses the previous approach. The proposed feature selector significantly enhances the efficacy of defect predictors developed by various ML algorithms. Figure 11 provides a comparison of performance metrics between ML techniques with and without the feature selector algorithm on the JM1 dataset.

The ML algorithms utilising selected features on the JM1 dataset outperform typical ML techniques across accuracy, precision, recall, and F1 metrics. Specifically, these algorithms demonstrate superior performance when equipped with selected features. Additionally, Fig. 11 contrasts the performance metrics of defect predictors generated by optimised (ML + BCOOA) and standard ML algorithms on the KC1 dataset. Notably, ML + BCOOA achieves superior results by leveraging the chosen effective features. The BCOOA method for selecting valuable features demonstrates effective performance across ML algorithms employed within the JM1 dataset. The defect predictors developed by the optimised ML (ML + BCOOA) exhibit higher accuracy, precision, recall, and F1 metrics compared to any other model. In summary, the outcomes from the experiments conducted on the JM1 dataset highlight the efficacy of BCOOA for efficient feature selection, showing its significant impact on enhancing the performance of ML algorithms.

Figure 12 presents the average performance of defect predictors generated by ML algorithms on the JM1 dataset, both with and without feature selection. The average

accuracy of defect predictors produced by standalone ML and ML + BCOOA algorithms stands at 79.57% and 86.16%, respectively. Comparable trends were observed across other performance criteria. Specifically, the average precision of classifiers generated by ML algorithms with and without the suggested feature selector (ML + BCOOA) hovers around 83.14% and 87.12%, respectively. Notably, classifiers developed using the recommended method (ML + BCOOA) exhibit a markedly higher average recall metric in comparison with standalone ML techniques. Ultimately, classifiers generated by the proposed method surpass those produced by ML algorithms that do not involve feature selection. BCOOA efficiently identifies and eliminates ineffective features, enhancing the accuracy of the generated defect predictor. Moreover, Table 16 shows the average quantity of effective features identified by BCOOA across different ML techniques, selectively choosing the most impactful features from the 21 available in the JM1 dataset.

The average performance of the new approach was assessed over ten executions on the JM1 dataset, demonstrating commendable detection performance. Table 16 illustrates the average quantity of features selected by binary ACO and BCOOA across ten runs in the JM1 dataset. When using the ANN as the ML algorithm, BCOOA and ACO each selected 12 features. For the DT ML algorithm, BCOOA and ACO opted for 8 and 9 features, respectively. In the KNN classification process, BCOOA and ACO selected 9 and 8 features, respectively. Similarly, for the NB classification, BCOOA and ACO selected 8 and 6 features, respectively. Overall, when compared to brute force ML, BCOOA notably selects a significantly smaller average quantity of features.

Figure 13 illustrates the feature selection rate of BCOOA across ten executions in the JM1 dataset, showcasing the method's consistency in feature selection. Table 17 outlines the features from the JM1 dataset that were selected and refined by BCOOA using various ML algorithms, resulting in optimal outcomes after ten iterations. Each ML approach underwent ten runs, with features meeting the highest assessment criteria being identified as the most effective.

To assess the reliability of the proposed technique, it underwent ten iterations on the JM1 dataset. The outcomes revealed enhancements in metrics including accuracy, precision, recall, and F1 scores. Employing all 21 features, the method trained the ML algorithm using selectively identified features by BCOOA. A distinct subset of features consistently surfaced in BCOOA's output, shaping the defect predictor and significantly influencing the resulting classifier's accuracy and precision. The most effective features selected by BCOOA across different ML algorithms are presented in Table 17.

Fig. 13 The selection rate of the 21 features by the BCOOA in the JM1 dataset during ten times executions

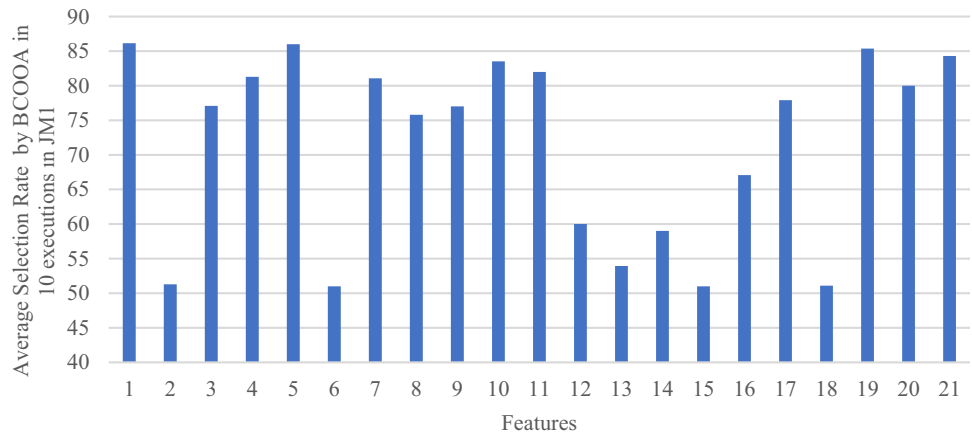


Table 17 The selected features by BCOOA and ACO in different ML algorithms in JM1 in the best result from 10 executions

	SVM		NB		KNN		DT		ANN		#
	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	
0	1	1	1	1	0	1	1	1	1	1	1
1	0	1	0	1	1	0	0	1	0	0	2
1	0	0	1	1	1	0	1	1	0	1	3
0	1	1	1	1	0	1	1	0	1	1	4
0	1	1	1	1	1	1	1	1	0	1	5
0	0	0	0	0	1	1	0	0	1	1	6
1	1	0	1	1	0	0	0	1	1	1	7
1	1	0	1	1	1	0	0	0	0	1	8
1	0	0	1	1	0	0	1	1	1	1	9
0	1	1	1	1	0	1	1	1	0	1	10
0	1	0	1	1	0	1	1	1	1	1	11
0	0	1	0	1	1	1	1	0	1	1	12
1	0	0	1	1	1	0	0	0	1	0	13
0	1	1	0	1	1	0	0	0	0	1	14
1	0	0	1	1	1	0	1	0	1	0	15
0	0	0	1	1	0	0	1	0	1	1	16
1	1	1	1	1	0	0	0	1	1	1	17
1	0	0	0	1	1	0	1	0	0	1	18
1	1	0	1	1	1	1	1	1	1	1	19
1	1	0	1	1	0	1	0	0	0	1	20
1	1	0	1	1	1	1	0	1	0	1	21

Table 18 Performance criteria of ML algorithms on the KC2 dataset without feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	77.70	82.70	90.16	86.27
DT	84.07	92.00	88.46	90.19
KNN	90.90	93.49	88.46	85.35
NB	80.25	80.85	96.61	88.03
SVM	80.89	91.66	82.50	86.84

Table 19 Performance criteria of ML algorithms on the KC2 dataset with BCOOA feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	88.00	89.49	98.91	94.28
DT	86.84	92.18	93.00	92.07
KNN	94.20	93.91	96.80	91.15
NB	90.80	91.94	98.88	96.02
SVM	87.51	93.49	92.80	93.00

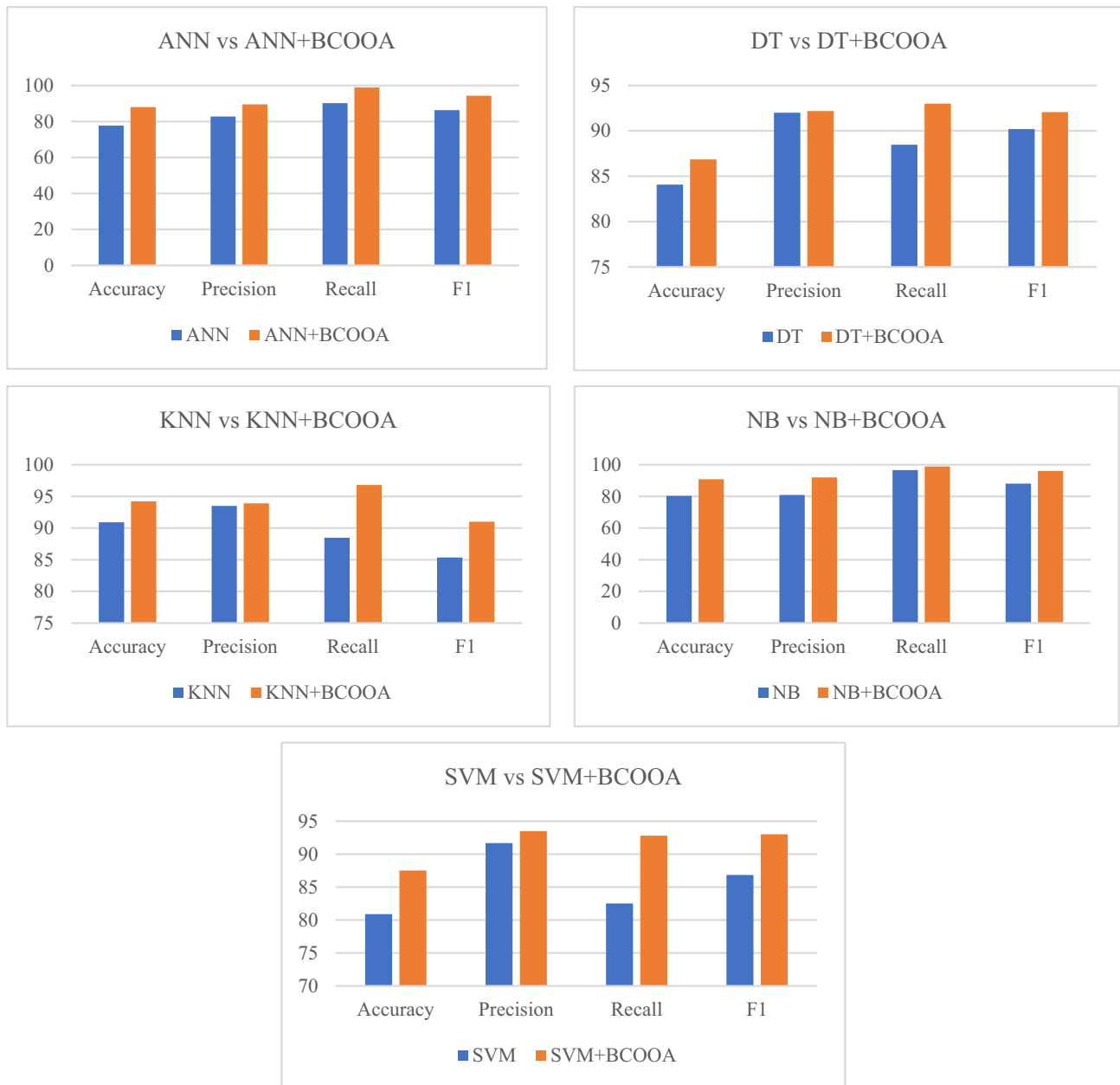


Fig. 14 The impact of BCOOA on the ML algorithms' performance in the KC2 dataset

4.6 Discussion of the results from the fourth experiment (KC2 dataset)

The fourth series of experiments focused on the KC2 dataset to address specific research inquiries. This dataset was utilised for both training and test sets, featuring 522 records and 21 distinct features. Within the KC2 dataset, the training set consists of 365 records, while the test set comprises 157 records. Table 18 presents the performance metrics of defect predictors (classifiers) generated by ANN, DT, KNN, NB, and SVM using the raw training set, which includes all 21 features and 522 records. This

table provides insights into the defect predictors' performance derived from various ML algorithms without the involvement of BCOOA. Notably, the KNN-based classifier exhibits the highest accuracy and precision among the prediction models.

As discussed earlier, the ineffective features within the KC2 training dataset may contribute to overfitting in ML algorithms. To address this concern, the KC2 dataset underwent a second phase of experimentation, utilising BCOOA in conjunction with ML algorithms. Table 19 presents the test results of defect predictors developed by ML algorithms using BCOOA to identify the most

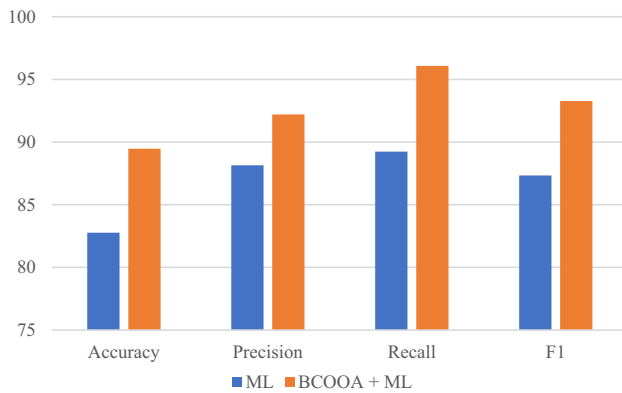


Fig. 15 The average values of the performance criteria of ML algorithms (ANN, DT, KNN, NB and SVM) with and without BCOOA as the feature selector in the KC2 dataset in ten times executions

beneficial features within the KC2 dataset. With the implementation of BCOOA, classifiers constructed through KNN and SVM exhibit accuracies of 94.20% and 87.51%, respectively. Incorporating BCOOA into ML algorithms significantly enhances the precision and accuracy of the resulting classifiers. For example, the recall and F1 scores of the defect predictor generated by ANN without BCOOA stand at 90.16% and 86.27%, respectively. However, with the utilisation of BCOOA for feature selection, these metrics notably increase to 98.91% and 94.28%. Consistent with other datasets, the studies demonstrate improved performance in the KC2 dataset when employing BCOOA as the feature selector within ML algorithms. A comparison between Tables 15 and 16 suggests that software defect predictors leveraging BCOOA as the feature selector display superior efficiency compared to the previous method. Overall, the introduced feature selection method significantly enhances the effectiveness of defect predictors produced by various ML algorithms.

Figure 14 provides a comparative analysis of the performance metrics of ML algorithms with and without feature selection on the KC2 dataset. The ML + BCOOA methods outperform traditional ML algorithms across accuracy, precision, recall, and F1 measures, owing to the selected features from the KC2 dataset. This figure illustrates the contrast between defect predictors generated by ML + BCOOA and conventional ML techniques. ML + BCOOA demonstrates superior performance by leveraging the selected effective features. The BCOOA method, aimed at optimal feature selection, demonstrates effectiveness when applied to ML algorithms using the KC2 dataset. The defect predictors derived from the enhanced ML (ML + BCOOA) exhibit the highest F1, recall, accuracy, and precision in comparison with previous models. Overall, the results obtained from experiments on the KC2 dataset

Table 20 The average quantity of selected features by ACO and BCOOA in different ML algorithms in the KC2 dataset in ten times executions

	ANN	DT	KNN	NB	SVM
Brute force ML	21	21	21	21	21
BCOOA + ML	10	9	10	8	8
ACO + ML	7	5	9	8	7

underscore the effectiveness of BCOOA for efficient feature selection.

Figure 15 presents the average performance of defect predictors generated by ML methods on the KC2 dataset, both with and without feature selection. The defect predictors generated by standalone ML and ML + BCOOA exhibit average accuracies of 82.76% and 89.47%, respectively. Comparable outcomes were achieved for the remaining performance criteria. The classifiers produced by ML techniques, both without and with the suggested feature selection (ML + BCOOA), had average precisions of around 88.14% and 92.20%, respectively. In terms of outcomes, the classifiers developed using the proposed approach (ML + BCOOA) demonstrate an average recall metric value that is notably higher than those generated by standalone ML methods. Ultimately, the defect predictor produced by the suggested technique (ML + BCOOA) outperforms those generated by ML algorithms without feature selection. By identifying and removing ineffective features, BCOOA enhances the defect predictor's accuracy. The quantity of useful features discovered by BCOOA using various ML methods is displayed in Table 20. Out of the 21 features in the KC2 dataset, BCOOA effectively selects the most valuable ones.

After ten executions, the average performance of the suggested method was evaluated on the KC2 dataset. Figure 20 illustrates the average number of features chosen during ten runs in the KC2 dataset by the binary ACO and BCOOA. When ANN was employed as the ML method, BCOOA and ACO selected 10 and 7 features, respectively. For the KNN method, BCOOA and ACO selected 10 and 9 features, respectively, while for the SVM classification algorithm, these numbers were 8 and 7, respectively. Overall, the average number of features selected by BCOOA is substantially fewer than that of the brute force ML algorithms. The feature selection rate of BCOOA is depicted in Fig. 16 after ten iterations of the proposed method on the KC2 dataset. Table 21 enumerates the features of the KC2 dataset selected and enhanced by BCOOA using various ML algorithms, yielding the best results after ten iterations. Ten runs were conducted for each machine learning strategy, aiming to identify the most beneficial aspects that met the highest evaluation standards.

Fig. 16 The selection rate of the 21 features by the BCOOA in the KC2 dataset during ten times executions

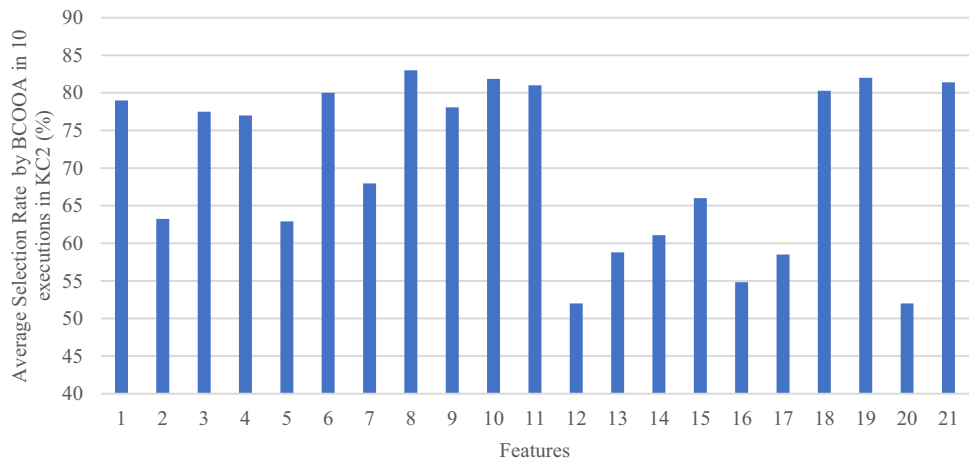


Table 21 The selected features by BCOOA and ACO in different ML algorithms in KC2 in the best result from ten executions

	SVM		NB		KNN		DT		ANN		#
	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	
0	1	0	1	0	1	0	0	1	1	1	1
0	1	1	0	0	1	0	1	0	1	1	2
1	1	1	1	1	0	0	1	1	1	0	3
0	1	1	1	1	0	1	0	0	0	0	4
0	0	0	1	1	1	1	0	0	0	0	5
0	0	0	1	1	1	1	0	1	1	1	6
0	1	1	0	0	1	0	1	1	1	1	7
1	1	1	0	0	1	1	1	1	0	1	8
0	1	0	0	0	1	0	1	1	0	1	9
0	1	1	1	1	0	0	0	1	1	1	10
0	1	0	1	1	0	0	0	1	1	1	11
1	0	1	0	0	1	1	1	0	1	0	12
0	0	0	0	0	0	0	1	1	0	1	13
1	0	1	1	1	0	0	0	1	0	0	14
1	1	0	0	0	0	0	0	1	1	1	15
0	0	0	0	0	1	1	1	0	0	1	16
1	1	1	1	1	0	0	1	0	1	0	17
0	1	0	1	1	0	0	0	1	1	1	18
0	0	0	1	1	0	1	0	1	0	1	19
1	0	1	0	0	1	0	1	1	1	0	20
0	0	1	1	1	1	1	0	1	0	1	21

Table 22 Performance criteria of ML algorithms on the PC1 dataset without feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	91.59	91.74	99.66	95.54
DT	90.99	92.79	97.68	95.17
KNN	96.02	98.05	94.08	92.49
NB	80.25	80.85	96.61	88.03
SVM	85.28	94.49	87.57	91.81

The suggested strategy was applied ten times on the KC2 dataset to demonstrate its stability. The outcomes reveal improved accuracy, precision, recall, and F1 measures. It considers 21 features and uses BCOOA’s selected features to train the ML algorithm. A subset of features chosen from BCOOA’s output has been selected to generate the defect predictor. These selected features predominantly influence the accuracy and precision of the resulting defect predictor. The relative contribution of each

feature to the defect predictor in the KC2 dataset is depicted in Fig. 16.

Table 23 Performance criteria of ML algorithms on the PC1 dataset with BCOOA feature selection

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
ANN	96.00	97.07	100.00	98.19
DT	95.17	97.16	98.26	99.00
KNN	98.80	99.08	98.00	97.00
NB	95.90	97.01	100.00	97.26
SVM	97.53	98.27	99.81	98.90

4.7 Discussion of the results from the fifth experiment (PC1 dataset)

Using the PC1 dataset, the final series of experiments has been conducted in response to the study's research question. The training and test datasets for the final experiment were created using the PC1 dataset, which comprises 21 features and 1109 entries. The test dataset contains 333 entries, while the training set consists of 776 entries. Table 22 displays the performance of defect predictors generated by various ML algorithms in the absence of BCOOA. In terms of accuracy and precision, the defect

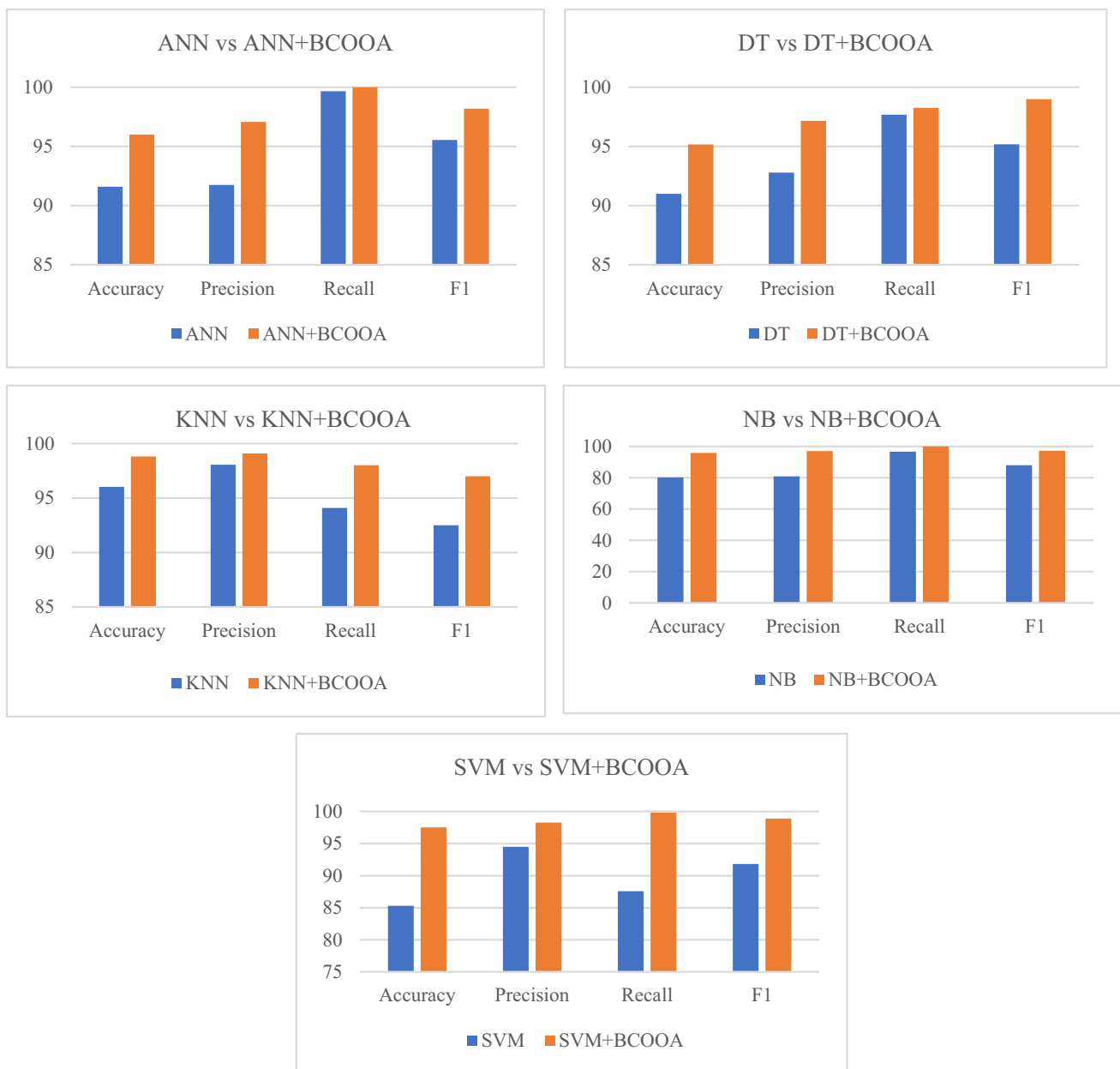


Fig. 17 The impact of BCOOA on the ML algorithms' performance in the PC1 dataset

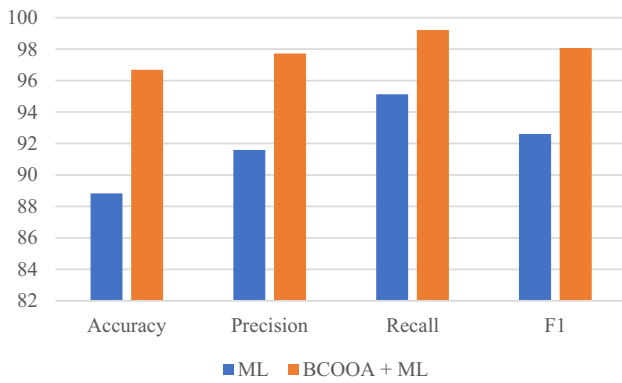


Fig. 18 The average values of the performance criteria of ML algorithms (ANN, DT, KNN, NB and SVM) with and without BCOOA as the feature selection in the PC1 dataset

predictor produced by KNN outperforms other prediction models.

The second phase of the fifth (final) experiment was conducted on the PC1 dataset, employing BCOOA in conjunction with ML algorithms. The defect predictors developed using ML algorithms with BCOOA were tested, and the outcomes are depicted in Table 23. When BCOOA was integrated, the classifiers produced by ANN and KNN showed accuracy rates of 96% and 98.80%, respectively. For instance, the defect predictor generated solely by ANN exhibited a recall and F1 score of 99.66% and 95.54%, respectively; however, with BCOOA as the feature selector, these metrics escalated to 100.00% and 98.19%. The findings highlight BCOOA's efficacy as a superior feature selector for ML algorithms within the PC1 dataset, mirroring its performance in other datasets. A comparison between Tables 22 and 23 underscores that software defect predictors leveraging BCOOA are markedly more effective than the previous methodology.

Figure 17 presents a comparison of performance metrics between ML algorithms with and without feature selectors. Leveraging features selected from the PC1 dataset, the ML + BCOOA approaches surpass conventional ML algorithms across accuracy, precision, recall, and F1 measures. Additionally, Fig. 17 illustrates the contrast between defect predictors generated by ML + BCOOA and standard ML methods. ML + BCOOA's superiority stems from adeptly utilising selected effective features, leading to enhanced outcomes. BCOOA, when applied to ML algorithms for optimal feature selection, demonstrates commendable performance on the PC1 dataset. Notably, the defect predictors produced by the improved ML (ML + BCOOA) exhibit the highest F1, recall, accuracy, and precision compared to previous models. BCOOA significantly elevates defect predictor performance by discerning and eliminating ineffective features.

Table 24 The quantity of selected features by ACO and BCOOA in different ML algorithms in the PC1 dataset

	ANN	DT	KNN	NB	SVM
Brute force ML	21	21	21	21	21
BCOOA + ML	9	8	10	10	11
ACO + ML	9	11	10	8	12

Figure 18 illustrates the average performance of defect predictors generated by ML techniques on the PC1 dataset, both with and without feature selection. The defect predictors produced by ML + BCOOA exhibit an average accuracy of 96.68%, while those from standalone ML methods show 88.82%. Comparable trends were observed across other performance metrics. The results highlight a substantially higher average recall metric for defect predictors developed with the suggested technique compared to those from standalone ML approaches. In summary, the defect predictor derived from the proposed method (ML + BCOOA) outperforms those from ML methods without feature selection.

The average performance of the new method on the PC1 dataset was assessed after ten executions. Table 24 illustrates the average number of features selected over ten runs in the PC1 dataset by binary ACO and BCOOA. In the case of using ANN as the ML technique, BCOOA and ACO selected 9 and 9 features, respectively. For DT + BCOOA, 8 features were selected, while DT + ACO selected 11 features; similarly, in the KNN classification algorithm, 10 features were selected for both BCOOA and ACO. NB + BCOOA selected 10 optimal features, while NB + ACO selected 8 features. Overall, the average quantity of features selected by BCOOA was lower than that by brute force ML techniques.

Figure 19 illustrates BCOOA's feature selection rate in PC1. The features from the PC1 dataset selected and refined by BCOOA across ten rounds using various ML algorithms are detailed in Table 25. Each ML technique underwent ten iterations, and the most beneficial features meeting stringent evaluation criteria were identified. To demonstrate the stability of the new technique, it was applied ten times to the PC1 dataset. The results consistently showcased improved F1 scores, recall, accuracy, and precision. Employing BCOOA, the method selects and incorporates features to train the machine learning algorithm while considering all 21 features. The defect predictor is then constructed using a selected subset of features derived from BCOOA's output, greatly influencing the accuracy and precision of the defect predictor.

Figure 20a demonstrates the convergence pattern of the proposed BCOOA in identifying the most effective features

Fig. 19 The selection rate of the 21 features by BCOOA in the PC1 dataset during ten times executions

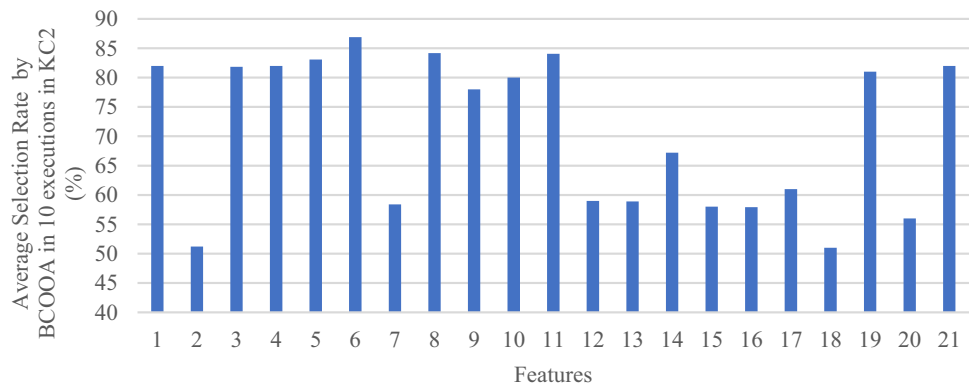


Table 25 The selected features by BCOOA and ACO in different ML algorithms in PC1 in the best result from 10 executions

	SVM		NB		KNN		DT		ANN		#
	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	ACO	BCOOA	
1	1	0	1	1	1	0	1	1	1	1	1
0	0	1	1	1	0	0	0	0	0	0	2
1	1	1	0	0	1	1	0	1	1	1	3
1	1	1	0	1	1	1	1	1	1	1	4
1	0	1	1	1	1	1	0	1	0	1	5
1	1	0	1	1	1	1	1	1	1	1	6
0	1	1	0	0	0	1	1	0	0	0	7
0	1	1	1	1	1	1	0	1	0	1	8
1	1	0	1	0	0	1	0	0	1	1	9
0	1	1	1	1	1	1	0	0	0	1	10
1	1	0	1	0	0	1	0	1	0	1	11
0	0	1	0	0	0	0	1	1	1	1	12
1	0	0	0	0	0	1	1	1	1	0	13
1	0	0	1	1	1	0	1	1	0	1	14
1	1	1	0	0	1	1	1	0	1	0	15
0	1	0	1	0	0	0	0	0	1	0	16
1	0	1	1	0	0	0	1	1	0	1	17
0	0	1	0	0	1	0	1	1	1	0	18
1	0	0	1	0	0	1	1	0	1	1	19
1	0	0	0	0	1	1	0	1	0	0	20
1	1	0	1	1	1	1	0	0	1	1	21

in the CM1 dataset. BCOOA exhibits varying performances across different ML algorithms; in the CM1 dataset, BCOOA + ANN achieves the maximum fitness value and demonstrates superior performance compared to other algorithms in software defect prediction. In summary, BCOOA, functioning as a feature selector, significantly enhances the performance of ML algorithms in addressing software defect prediction problems. Figure 20b depicts the convergence pattern of the suggested BCOOA, illustrating its progression towards optimal results in the KC1 dataset. BCOOA demonstrates varied performance across diverse ML techniques, achieving optimal performance

characterised by minimal fitness values and maximum convergence speed, particularly evident in Fig. 20b. Notably, ANN and KNN exhibit superior performance within software defect prediction techniques compared to other algorithms. Additionally, Fig. 20c illustrates how BCOOA converges toward optimal feature selection in the JM1 dataset. BCOOA's performance varies across different ML algorithms, demonstrating optimal performance with NB + BCOOA and SVM + BCOOA setups. Notably, NB and SVM outperform other algorithms regarding defect prediction accuracy.

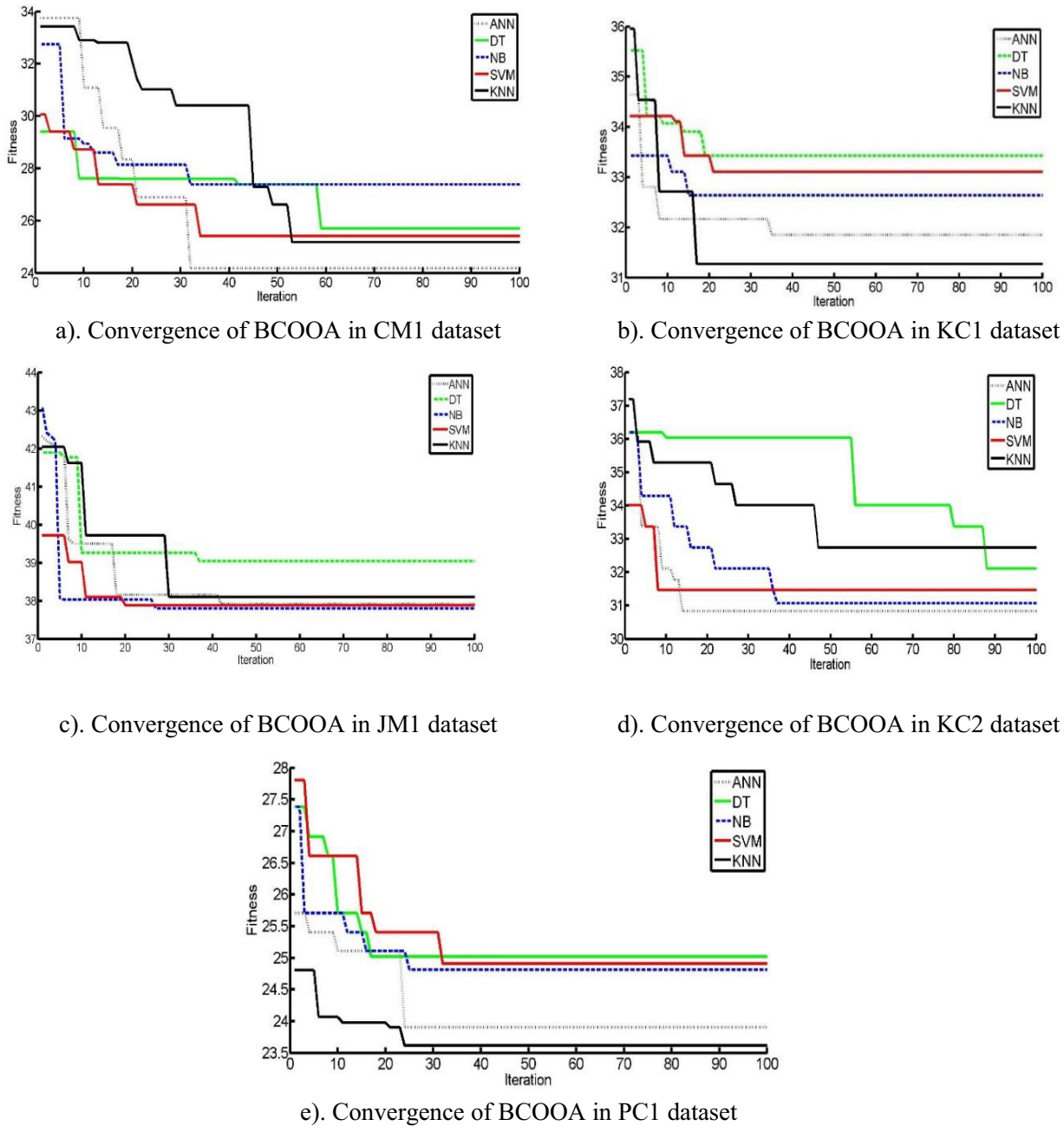


Fig. 20 Convergence of the BCOOA for finding effective features with different ML algorithms in different datasets

Figure 20d depicts the convergence of BCOOA to the optimal features in the KC2 dataset. BCOOA’s performance varies with different ML techniques. The results show that the ANN + BCOOA and NB + BCOOA setups operate at their best in terms of convergence speed. Compared to other algorithms, ANN, NB, and SVM perform better in accurate defect prediction. Finally, Fig. 20.e illustrates how BCOOA converges towards the optimal features in the PC1 dataset. BCOOA’s performance varies

with different ML methods, demonstrating optimal convergence speed with KNN + BCOOA and ANN + BCOOA compared to other methods. Notably, in precise defect prediction, ANN and KNN outperform other methods. Overall, BCOOA, serving as the feature selector, significantly enhances the performance of ML algorithms in software defect prediction tasks.

Figure 21 displays the accuracy of defect predictors created by various ML algorithms using different datasets.

Fig. 21 Comparison of defect predictors created by different methods in terms of accuracy

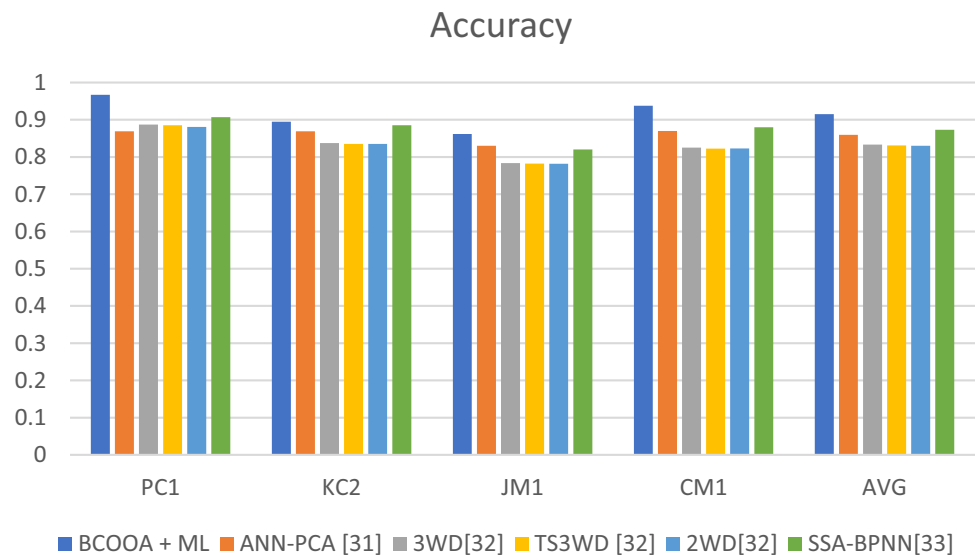


Table 26 The training time of different algorithm in different datasets

	PC1	KC2	JM1	KC1	CM1
BCOOA + SVM	140.808	76.798	1014.802	307.288	72.466
SVM	0.144	0.051	5.038	0.143	0.043
BCOOA + DT	30.628	24.086	381.260	119.031	44.738
DT	0.170	0.0278	0.354	0.056	0.024
BCOOA + NB	9.812	7.484	17.186	9.747	8.514
NB	0.028	0.006	0.0233	0.008	0.038
BCOOA + KNN	87.031	71.208	490.618	283.624	91.025
KNN	0.803	1.201	5.801	1.108	0.920
BCOOA + ANN	1082.873	941.028	3730.135	1327.065	1108.035
ANN	1.940	3.254	10.864	2.683	1.621

The defect predictor generated by the proposed method in the PC1 dataset achieves 96% accuracy, surpassing the accuracy of defect predictors created by other methods. In the KC2 dataset, the defect predictor by the proposed method achieves 89.50% accuracy, while in the JM1 and CM1 datasets, these figures are about 94% and 91.50%, respectively. Overall, the proposed method demonstrates superiority over other methods in terms of accuracy and performance. This improvement is attributed to the developed BCOOA, which identifies the most effective features of the training dataset. The elimination of ineffective attributes enhances the performance and accuracy of the created software defect predictor.

Table 26 illustrates the training time of the proposed method. The training time includes the time required for features selection by BCOOA and the creating fault predictor by the ML algorithm. In the proposed method, the ML algorithms train by the optimal dataset with the optimal features.

5 Conclusion

Developing a software defect predictor significantly impacts software development and quality. A notable drawback in machine learning-based defect predictors lies in their inability to discern the impact of each independent variable on the dependent one, often resulting in incorrect training. This paper introduces a binary variant of the Olympiad optimisation algorithm (BCOOA), specifically designed for selecting the most impactful features from defect prediction datasets. By utilising this algorithm, an effective and efficient classifier was created using a refined training set containing the most optimal features. This method enhanced the efficiency of the defect predictor by identifying the smallest yet most effective feature subset from standard defect prediction datasets, exemplified here using the PROMISE Software Engineering Repository dataset [20]. The approach selects features using BCOOA and employs them in training the machine learning algorithm, accounting for all 21 features.

Experimental results demonstrate the effectiveness of BCOOA as a feature selector for creating machine learning-based defect predictors. Key features identified by BCOOA include McCabe design complexity, total number of operators + operands in the code, Halstead program length and difficulty, number of branches in the source code, lines of code and comments, and number of unique operators and operands. It is essential to note that the removal of a feature by BCOOA does not discount its potential impact on prediction; rather, it signifies the discovery of more effective features that learning algorithms can utilise with greater generalisation power. In this research, the defect predictor, formed by combining BCOOA and ML algorithms, achieved an average accuracy of 91.13%, precision of 92.74%, recall of 97.61%, and F1 of 94.26%.

Acknowledging the limitations of the proposed BCOOA, one notable constraint is its application solely on a single dataset. Future studies will focus on assessing its performance using various real-world defect-prediction datasets. Moreover, this study utilised a dataset with homogeneous features; future research could explore the effectiveness of the proposed feature selector on raw datasets. Additionally, the study did not delve into the performance of the proposed feature selector with deep learning algorithms, which could be a subject for future investigation. Furthermore, the study did not evaluate the performance of the proposed defect predictor alongside test generation and test prioritisation methods, as outlined in references [23, 24]. Incorporating the proposed BCOOA into other learning problems prior to the training stage is another avenue for future studies. This may include applying the method to different software datasets and components. Moreover, there is potential in combining novel binary metaheuristic algorithms with ML techniques for software defect prediction. Additionally, integrating reinforcement learning methods and chaotic maps holds promise for enhancing ML-based defect predictors. The binary versions of recent metaheuristic algorithms, such as those referenced in [25–28], could serve as feature selectors in software defect prediction methods. Lastly, leveraging hybrid ML algorithms, as discussed in [17, 29] and hybrid heuristic algorithms [30], to enhance the accuracy of software predictors represents another potential avenue for future research.

Author contributions Research problem, method design, implementation, and experiments have been performed by B. Arasteh and K. Arasteh. Data analysis has been performed by B. Arasteh, A. Ghaffari and R. Ghanbarzadeh. B. Arasteh, and R. Ghanbarzadeh proofread and approved the final manuscript.

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK). The authors declare that no funds were received during the preparation of this manuscript.

Data availability The datasets used in this study can be accessed freely via the following link: https://drive.google.com/drive/folders/1zi66Fy978-GtFNJLXGUmZrgkieLI_ZhB?usp=drive_link

Declarations

Conflict of interest All authors state that there is no conflict of interest.

Ethical approval The authors confirm that this study represents their unique and unpublished work. It hasn't been submitted for consideration or published elsewhere. The paper authentically presents the authors' research and analysis truthfully and completely.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Arasteh, B.: Software fault-prediction using combination of neural network and Naive Bayes algorithm. *J. Netw. Technol.* **9**(3), 94–101 (2018). <https://doi.org/10.6025/jnt/2018/9/3/94-101>
2. Khanna, M., Toofani, A., Bansal, S., Asif, M.: Performance comparison of various algorithms during software fault prediction. *Int. J. Grid High-Perform. Comput.* (2021). <https://doi.org/10.4018/IJGHPC.2021040105>
3. Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J.: A general software defect-proneness prediction framework. *IEEE Trans. Softw. Eng.* **37**(3), 356–370 (2011)
4. Papa, P.J., Rosa, G.H., André, N., Afonso, C.S.L.: Feature selection through binary brain storm optimization. *Comput. Electr. Eng.* **72**, 468–481 (2018). <https://doi.org/10.1016/j.comp Eleceng.2018.10.013>
5. Ghaemi, A., Arasteh, B.: SFLA-based heuristic method to generate software structural test data. *J. Softw. Evol. Proc.* **32**, e2228 (2020). <https://doi.org/10.1002/smr.2228>
6. Shomali, N., Arasteh, B.: Mutation reduction in software mutation testing using firefly optimisation algorithm. *Data Technol. Appl.* **54**(4), 461–480 (2020). <https://doi.org/10.1108/DTA-08-2019-0140>
7. Hosseini, M.J., Arasteh, B., Isazadeh, A., Mohsenzadeh, M., Mirzarezaee, M.: An error-propagation aware method to reduce the software mutation cost using genetic algorithm. *Data Technol. Appl.* **55**(1), 118–148 (2021). <https://doi.org/10.1108/DTA-03-2020-0073>
8. Arasteh, B., Najafi, J.: Programming guidelines for improving software resiliency against soft-errors without performance

- overhead. *Computing* **100**, 971–1003 (2018). <https://doi.org/10.1007/s00607-018-0592-y>
9. Arasteh, B., Miremadi, S.G., Rahmani, A.M.: Developing inherently resilient software against soft-errors based on algorithm level inherent features. *J. Electron. Test.* **30**, 193–212 (2014). <https://doi.org/10.1007/s10836-014-5438-8>
 10. Batool, B.I., Khan, A.K.T.A.: Software fault prediction using data mining, machine learning and deep learning techniques: a systematic literature review. *Comput. Electr. Eng.* **100**, 107886 (2022). <https://doi.org/10.1016/j.compeleceng.2022.107886>
 11. Jiang, Y., Li, M., Zhou, Z., Member, S.: Software defect detection with ROCUS. *J. Comput. Sci. Technol.* **26**(2), 328–342 (2011)
 12. Wang, S.S., Yao, X.X.: Using class imbalance learning for software defect prediction. *IEEE Trans. Reliab.* **62**(2), 434–443 (2013)
 13. Galar, M., Fern, A., Barrenechea, E., Bustince, H.: A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Trans. Syst. Man Cybern.* **42**(4), 1–22 (2011)
 14. Anbu, M., Anandha, G.S.: Feature selection using firefly algorithm in software defect prediction. *Cluster Comput.* **22**, 10925–10934 (2019). <https://doi.org/10.1007/s10586-017-1235-3>
 15. Mafarja, M., Thaher, T., Al-Betar, M.A., et al.: Classification framework for faulty-software using enhanced exploratory whale optimiser-based feature selection scheme and random forest ensemble learning. *Appl. Intell.* **53**, 18715–18757 (2023). <https://doi.org/10.1007/s10489-022-04427-x>
 16. Yousef, A.H.: Extracting software static defect models using data mining. *Ain Shams Eng. J.* **6**(1), 133–144 (2014)
 17. Jayanthi, R., Florence, L.: Software defect prediction techniques using metrics based on neural network classifier. *Cluster Comput.* **22**(Suppl 1), 77–88 (2019). <https://doi.org/10.1007/s10586-018-1730-1>
 18. Laradji, I.H., Alshayeb, M., Ghouti, L.: Software defect prediction using ensemble learning on selected features. *Inf. Softw. Technol.* **58**, 388–402 (2015)
 19. Yucalar, F., Ozcift, A., Borandag, E., Kilinc, D.: Multiple-classifiers in software quality engineering: combining predictors to improve software fault prediction ability. *Int. J. Eng. Sci. Technol.* **23**(4), 938–950 (2020). <https://doi.org/10.1016/j.jestch.2019.10.005>
 20. Promise software engineering repository. <http://promise.site.uot.tawa.ca/SERepository/datasets-page.html>
 21. Shao, X., Wang, H.: Nonlinear tracking differentiator based on improved sigmoid function. *Control Theory Appl.* **31**, 1116–1122 (2014)
 22. Kanan, H.R., Faez, K., Taheri, S.M.: Feature selection using ant colony optimization (ACO): a new method and comparative study in the application of face recognition system. In: *Advances in Data Mining. Theoretical Aspects and Applications: 7th Industrial Conference, ICDM 2007, Leipzig, Germany, July 14–18, 2007. Proceedings 7*, vol. 4597, pp. 63–76. Springer, Berlin (2007)
 23. Sikandar, A., Ali, S., Bin, M.H., et al.: Multi objective test case prioritization using test case effectiveness: multicriteria scoring method. *Sci. Program.* (2021). <https://doi.org/10.1155/2021/9988987>
 24. Khanna, M., Chauhan, N., Sharma, D., et al.: Search for prioritized test cases in multi-objective environment during web application testing. *Arab. J. Sci. Eng.* **43**, 4179–4201 (2018). <https://doi.org/10.1007/s13369-017-2830-6>
 25. Arasteh, B., Sadegi, R., Arasteh, K.: Bölen: software module clustering method using the combination of shuffled frog leaping and genetic algorithm. *Data Technol. Appl.* **55**(2), 251–279 (2021). <https://doi.org/10.1108/DTA-08-2019-0138>
 26. Gharehchopogh, F.S., Abdollahzadeh, B., Arasteh, B.: An improved farmland fertility algorithm with hyper-heuristic approach for solving travelling salesman problem. *Comput. Model. Eng. Sci.* **135**(3), 1981–2006 (2023). <https://doi.org/10.32604/cmesci.2023.024172>
 27. Arasteh, B., Abdi, M., Bouyer, A.: Program source code comprehension by module clustering using combination of discretized gray wolf and genetic algorithms. *Adv. Eng. Softw.* **173**, 103252 (2022). <https://doi.org/10.1016/j.advengsoft.2022.103252>
 28. Soleimani, F., Abdollahzadeh, B., Barshandeh, S., Arasteh, B.: A multi-objective mutation-based dynamic Harris Hawks optimization for botnet detection in IoT. *Internet Things* **24**, 100952 (2023). <https://doi.org/10.1016/j.iot.2023.100952>
 29. Singh, L.K., Khanna, M., Singh, R.: A novel enhanced hybrid clinical decision support system for accurate breast cancer prediction. *Measurement* **221**, 113525 (2023). <https://doi.org/10.1016/j.measurement.2023.113525>
 30. Arasteh, B.: Clustered design-model generation from a program source code using chaos-based metaheuristic algorithms. *Neural Comput. Appl.* **35**, 3283–3305 (2023). <https://doi.org/10.1007/s00521-022-07781-6>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Bahman Arasteh was born in Tabriz. He received the master degree in software engineering from IAU, and the Ph.D degree in software engineering from IAU, Tehran Science and Research Branch, respectively. Currently, he is an associate professor at Istinye University, Istanbul, Turkiye. He has published more than 60 papers in refereed international journals and conferences. He is the coordinating editor in the springer journal of electronic test and Journal of Assurance Engineering and Management. His research interests include AI-based software engineering, software testing, software security, artificial intelligence, and software fault tolerance.



Keyvan Arasteh is a researcher at Istinye University in Turkey. His research interests include software testing, evolutionary algorithms and software security.



Reza Ghanbarzadeh is an associate professor in Southern Cross University in Turkey. His research interests include heuristic algorithms and learning systems.



Ali Ghaffari is an associate professor in Istinye University in Turkey. His research interests include evolutionary algorithms and network security.