Applied Intelligence (2023) 53:18715–18757
https://doi.org/10.1007/s10489-022-04427-x

Classification framework for faulty-software using enhanced exploratory whale optimizer-based feature selection scheme and random forest ensemble learning

Majdi Mafarja[1] · Thaer Thaher[2,3] · Mohammed Azmi Al-Betar[4] · Jingwei Too[5] · Mohammed A. Awadallah[6,7] · Iyad Abu Doush[8,9] · Hamza Turabieh[10]

Accepted: 23 December 2022 / Published online: 9 February 2023

Abstract

Software Fault Prediction (SFP) is an important process to detect the faulty components of the software to detect faulty classes or faulty modules early in the software development life cycle. In this paper, a machine learning framework is proposed for SFP. Initially, pre-processing and re-sampling techniques are applied to make the SFP datasets ready to be used by ML techniques. Thereafter seven classifiers are compared, namely K-Nearest Neighbors (KNN), Naive Bayes (NB), Linear Discriminant Analysis (LDA), Linear Regression (LR), Decision Tree (DT), Support Vector Machine (SVM), and Random Forest (RF). The RF classifier outperforms all other classifiers in terms of eliminating irrelevant/redundant features. The performance of RF is improved further using a dimensionality reduction method called binary whale optimization algorithm (BWOA) to eliminate the irrelevant/redundant features. Finally, the performance of BWOA is enhanced by hybridizing the exploration strategies of the grey wolf optimizer (GWO) and harris hawks optimization (HHO) algorithms. The proposed method is called SBEWOA. The SFP datasets utilized are selected from the PROMISE repository using sixteen datasets for software projects with different sizes and complexity. The comparative evaluation against nine well-established feature selection methods proves that the proposed SBEWOA is able to significantly produce competitively superior results for several instances of the evaluated dataset. The algorithms' performance is compared in terms of accuracy, the number of features, and fitness function. This is also proved by the 2-tailed P-values of the Wilcoxon signed ranks statistical test used. In conclusion, the proposed method is an efficient alternative ML method for SFP that can be used for similar problems in the software engineering domain.

Keywords Software fault prediction · Machine learning · SMOTE · Dimension reduction · Meta-heuristics · Imbalanced data

Abbreviations

AAE	Average absolute error
ABC	Artificial bee colony
ACO	Ant colony optimization
ADASYN	Adaptive synthetic sampling method
ALO	Ant lion optimizer
ANN	Artificial neural networks
ARE	Average Relative Error
ASD	Agile software development model
AUC	Area under the curve
BBAT	Binary bat algorithm
BCS	Binary cuckoo search
BFFA	Binary firefly algorithm
BGWO	Binary grey wolf optimization
BHHO	Binary harris hawk optimization
BJAYA	Binary jaya algorithm
BMFO	Binary moth flame optimization
BN	Bayesian networks
BQSA	Binary queuing search algorithm
BWOA	Binary whale optimization algorithm
CBL	Case-based learning

✉ Mohammed Azmi Al-Betar
m.albetar@ajman.ac.ae

Extended author information available on the last page of the article.

COA	Coyote optimization algorithm
CS	Chi-square
CSA	Crow search algorithm
DA	Dragonfly algorithm
DE	Differential evolution
DT	Decision tree
EA	Evolutionary algorithm
FFA	Firefly algorithm
FIS	Fuzzy inference system
FN	False negative
FP	False positive
FS	Feature selection
GA	genetic algorithm
GBRCR	Gradient boosting regression-based combination rule
GOA	Grasshopper optimization algorithm
GP	Genetic programming
GWO	Grey wolf optimizer
HHO	Harris hawks optimization
IG	information gain
KNN	K-nearest neighbors
LDA	Linear discriminant analysis
LR	Linear regression
LRCR	linear regression-based combination rule
ML	Machine learning
MLP	Multi-layer perceptron
MLR	Multi-nomial logistic regression
MVO	multiverse optimizer
NB	Naive Bayes
OO	Object-Oriented
PCA	Principle component analysis
PCC	Pearson correlation coefficient
PSO	Particle swarm optimization
QMOOD	Quality metrics for object-oriented design
RF	Random forest
ROC	Receiver operating characteristic
SBWOA	Binary whale optimization algorithm with S-shaped transfer function
SBEWOA	Enhanced SBWOA
SC	Soft computing
SDLC	Software sevelopment life cycle
SDP	Software defect prediction
SFP	Software fault prediction
SMOTE	Synthetic minority oversampling technique
SSA	Salp swarm algorithm
SVM	Support vector machine
TF	Transfer function
TN	True negative
TNR	True negative rate
TP	True positive
TPR	True positive rate
VBWOA	Binary whale optimization algorithm with V-shaped transfer function
WOA	Whale optimization algorithm

1 Introduction

Software Development Life Cycle (SDLC) represents the phases that software passes through while it is being developed. Starting with requirements elicitation, then the analysis and design of the collected requirements. After that, the programmers start developing the proposed software based on the analysis and design phases. A vital phase in SDLC is software testing. This phase follows the development phase and consists of a set of activities that assure the team is developing the right software with high-quality levels [1]. Numerous testing types are available to test various aspects of a software product. These tests include but are not limited to unit testing, component testing, integration testing, regression testing, and user acceptance testing. Many software development methodologies are available to be used by the development team. The most popular SDLC models are waterfall, agile and spiral models.

The testing stage plays an essential role in the development process. It is usually performed as a traditional linear model (e.g., waterfall) or a cyclic model (e.g., agile model). Testing process concerns with enhancing the software quality and reducing the total cost [2, 3]. However, many factors affect the results of the testing process, such as the limited resources (e.g., time or software testers). Therefore, early-stage procedures such as Software Fault Prediction (SFP) are utilized to facilitate the testing process in an optimal way [4]. In SFP, the faulty components of the software are detected prior to system deployment in the early stages of the SDLC. This is achieved by utilizing software faults datasets collected from previous projects or predefined software metrics. It is worth mentioning that the SFP process became more straightforward since the adoption of the agile software development (ASD) model in 2001 [5] as a replacement for the waterfall model which was introduced in 1970 [6]. Adopting the ASD methodology has many benefits since the software is developed incrementally. Moreover, ASD opens the door to adopting volatile requirements, optimizing resources (time and cost), bridging the gap between the development team and business owners [7], and facilitating performance software engineering tasks regularly such as review, maintenance, and testing [2].

The early prediction of faults in software components such as modules, classes, and so on has a significant impact in reducing the needed time and effort for the project outcomes to be delivered to the end-user. SFP is one of the approaches that help in optimizing the

development process by reducing the number of potential faults in the early stages of the SDLC process [4]. Various SFP approaches were recorded in the literature. The main approaches include but are not limited to Soft Computing (SC) and Machine Learning (ML) [8]. These methods need data to be able to predict software faults. Design features (metrics) gathered during the design stage or historical fault datasets accumulated during the implementation of previous versions of similar projects are two essential resources to be used with SFP approaches for benchmarking [9].

Various types of metrics such as method-level and class-level have been proposed for the SFP problem [10]. Method-level metrics can be collected from structured programming or object-oriented programming-based source codes. Halstead [11], and McCabe [12] metrics are the most common method-level measures used by many researchers. Class-level metrics are only appropriate when developing SFP models for object-oriented programming-based projects. Examples of class-level suite of metrics for object-oriented design are CK (Chidamber–Kemerer) [13], L&K (Lorenz-Kidd) [14], and quality metrics for object-oriented design (QMOOD) [15]. However, in comparison with other suites, CK metric is mostly applied when class-level metrics are chosen [10].

Automated systems become available for almost all fields in real life. With the advancement of software development, and the availability of large-scale projects, analyzing the collected software metrics becomes complicated and forms a significant challenge. Thus, ML techniques have been proposed as SFP solutions and shown a good performance [16]. The main purpose behind these techniques is to predict the faulty components in software based on the supplied datasets. Examples of ML techniques that have been used as SFP approaches are K-Nearest Neighbors (KNN), Naive Bayes (NB), Linear Discriminant Analysis (LDA), Linear Regression (LR), Decision Tree (DT), Support Vector Machine (SVM), and Random Forest (RF) [4, 17, 18].

Among the various ML models, ensemble learning has proven excellent performance in dealing with various complex classification problems [19]. Ensemble learning combines a number of ML models to create an ensemble learner to improve the model performance by proving a more general robust model. The RF is recognized as a well-regarded ensemble technique that was originally introduced by Breiman, Leo [20]. In RF, a number of DT classifiers are fit on various sub-samples of the dataset and combine the output of all the trees. The RF has several merits that make it superior when compared to other traditional ML models. It controls the over-fitting problem of DT, reduces the variance within the forest, and thus enhances the predictive accuracy [21, 22].

The performance of the ML-based SFP approaches depends mainly on a set of factors which is the applied ML technique and the quality of the utilized dataset (in terms of noise, irrelevant features, and imbalanced representation of data) [23]. Therefore, dimensionality reduction (e.g., feature selection) and data resampling (e.g., Synthetic Minority Oversampling Technique (SMOTE)) techniques are needed before applying the ML technique. These features can be defined in the context of the feature selection problem which can be tackled by feature selection techniques.

In feature selection (FS) the problems with high-dimension feature space increase the hardness of the search process. In common, various search strategies, including complete, random, and heuristic, are available for searching the feature space to obtain the optimal subset of features [24]. The complete search requires generating and evaluating all possible subsets of features. In this way, for a set of m features, 2^m features subsets will be formed. For example, if the given problem has four features, sixteen subsets of features will be produced. In case of random search, the next candidate solution (subset of features) is generated randomly while heuristic search strategies conduct the search in adaptive way, and generate possible solutions (feature subsets) for the problem [24–27].

Recently, metaheuristics is widely used by the research community as a successful FS method. The metaheuristics are conventionally categorized based on the initial solutions into population-based, and trajectory-based [28]. A trajectory-based metaheuristic is initiated with a single solution. The search follows a trajectory in the search space based on the local modification of the current solution until a local optimum is obtained. These methods like tabu search [29], β-hill climbing [30], stochastic local search [31], and variable neighborhood search [31]. In contrast, the population-based algorithm is initiated with a population of individuals. Iteratively, the population inherits its strong elements to come up with an optimal solution. Normally, population-based algorithms are classified into evolutionary algorithms (EAs) and swarm intelligence approaches. Genetic Algorithm (GA) [32], Genetic Programming (GP) [33], and Differential Evolution (DE) [34] are the base EAs for feature selection. A swarm-based algorithm is normally built based on the idea of a group of solutions where the group members are divided into leaders and followers. Particle Swarm Optimization (PSO) algorithm and Ant Colony Optimization (ACO) are the base swarm intelligence methods. Quite recently, several swarm intelligence methods have been proposed for FS such as PSO [35], Salp Swarm Algorithm (SSA) [36, 37], Dragonfly Algorithm (DA) [38], Rate Swarm Optimizer [39], Ant Lion Optimizer (ALO) [40], Harmony Search [41], Coronavirus herd immunity optimizer [42], ant colony optimization (ACO) [43], β-hill climbing optimizer [44], Crow Search Algorithm (CSA) [45], JAYA algorithm [46], Firefly algorithm (FFA) [47], Artificial Bee Colony (ABC) algorithm [48],

Coyote Optimization Algorithm (COA) [49], and Grasshopper Optimization Algorithm (GOA) [50]. Furthermore, the hybrid between them such as Genetic-Whale-Ant colony algorithms [51], Grey Wolf optimizer and Random Forest [52], hybrid Salp Swarm Algorithm [53], genetic and coral reefs [54], etc. There are also real opportunities to adapt newly-established optimization algorithms for FS problems like starling murmuration optimizer [55], Quantum-based avian navigation optimizer [56], Farmland fertility algorithm [57], African vultures optimization algorithm [58], and artificial gorilla troops optimizer [59].

Whale Optimization Algorithm (WOA) is a recent swarm intelligence imitates the behavior of humpback whales in hunting fish in the oceans [60]. It has impressive characteristics over other optimization methods such as it has few control parameters, easy to implement, simple structure, and it has maneuver behavior to find a suitable balance between local exploitation and global exploration. Due to its successful attributes, WOA has been widely utilized to deal with feature selection problems [25, 61–65]. The original version of WOA was designed to handle continuous search space problems. In this paper, to match the binary search space of the FS problem, WOA was boosted with eight fuzzy transfer functions from S-shaped and V-shaped families. Due to the No Free Lunch [66] argument which points out that there is no superb optimization algorithm that can excel all others for all optimization problems, therefore, the opportunity is still possible to investigate modifying efficient methods to handle the SFP to improve the algorithm efficiency.

In this paper, a systematic SFP approach that considered several ML techniques with different pre-processing methods was proposed. The major contributions are summarized as follows:

- Several pre-processing and re-sampling techniques are applied to prepare SFP datasets to be suitable to the ML techniques.
- Various classification techniques, namely KNN, LDA, SVM, LR, DT, and NB, RF, are applied. Their performance is compared in the same environment to adopt one technique for further experiments. As a result, the RF classifier is adopted in this stage.
- A dimensionality reduction method based on the Binary version of WOA was utilized to eliminate the irrelevant/redundant features to enhance the performance of the RF classifier. The newly proposed method is called BWOA, which utilizes eight transfer functions where a transfer function that yields good results is chosen.
- An enhanced WOA version (EWOA) is introduced, where the exploration strategies from the grey wolf optimizer (GWO) and harris hawks optimization (HHO) algorithms are used to enhance the diversity of the WOA. By means of this enhancement mechanism, the performance of WOA is improved to deal more efficiently with the search space of the FS problem. This yields a superior optimization framework for the faulty-software prediction problem.

The newly proposed EWOA reveals very successful outcomes in terms of choosing the most informative features in the area of SFP. The findings prove that the classification performance can be significantly improved by removing useless features. The performance is compared with nine state-of-the-art methods and it shows the viability of the proposed method in terms of the accuracy, number of features, and fitness function.

The remainder of the paper is structured as follows: a review of the related works is presented in Section 2. In Section 3, a theoretical background of the related aspects to this paper is introduced. Section 4 presents the proposed methodology. The experimental design and the obtained results are discussed in Section 5. Finally, Section 6 includes a conclusion about the main findings of this paper in addition to some future work directions.

2 Related works

Recently, different ML approaches were considered to solve the SFP problem with remarkable success [4]. Accordingly, different datasets (e.g., PROMISE repository, NASA datasets, and Qualitas corpus) became publicly available to the researchers [4, 67]. This section presents the most relevant related work in the field of SFP. A general overview of the SFP techniques is provided, and the related ML approaches are investigated, followed by the related work of the enhanced ML approaches by applying some preprocessing approaches like feature selection.

2.1 ML based SFP

Different supervised and unsupervised ML techniques were applied as prediction models in SFP. Examples of the ML that were used with SFP are: SVM [68], DT [69], Bayesian Networks (BN) [70], NB [71], KNN [72], Multi-layer Perceptron (MLP) [73], Artificial Neural Networks (ANN) [2, 74], LR [75], Multi-nomial Logistic Regression (MLR) [73], RF [76] and ensemble MLP [77].

Singh and Malhotra [68] conducted an empirical study to evaluate the performance of an SVM classifier in determining the relationship between some software Object-Oriented (OO) design matrices and fault proneness. A dataset from the NASA repository (KC1) and Receiver Operating Characteristic (ROC) were used to evaluate the proposed model. The study shows that the SVM classifier

was feasible and helpful in predicting faulty classes in OO-based systems.

Moreover, Cahill et al. [78] introduced an approach named Rank Sum for data representation to improve the performance of fault porousness prediction modules. The proposed approach is evaluated by applying the well-known ML classifiers SVM and NB over various datasets from the NASA repository. It was found that NB is better compared to the SVM classifier. Erturk and Sezer [79] introduced an SFP model that combines Fuzzy Inference System (FIS) and Artificial Neural Network (ANN) classifiers. FIS was applied at the beginning of the project to make predictions depending on expert opinion because it does not need historical data for prediction, and then ANN was employed in the later iterations when some data about the software project are obtainable. The proposed iterative system was tested using a set of datasets including various versions of many projects from the PROMISE repository. The selected datasets consist of common OO metrics such as coupling between objects, response for a class, and weighted methods per class. The evaluation of the results according to the receiver operating characteristics (ROC) with the area under the curve (AUC) method shows that the iterative module is capable of locating fault-prone modules in the software.

An approach named multi-strategy classifier (RB2CBL) was introduced by Khoshgoftaar et al. [80] for the SFP problem, where Rule-Based (RB) classifier was hybridized with two variants of the Case-Based Learning (CBL) model. Moreover, an embedded GA was utilized to optimize the parameters of CBL models. The experimental results reveal that the proposed RB2CBL classifier is superior compared to the RB model alone. Carrozza et al. [81] proposed a new set of software matrices for detecting mandelbugs in complex software systems. In addition, considering the newly proposed matrices and the conventional software matrices, several algorithms, including DT, SVM, BN, NB, and MLR, were applied to various datasets from the NASA repository. The authors reported that MLR and SVM are the best among all examined algorithms in finding Mandelbug-prone modules.

A model based on the principle of ensemble learning methods was employed by Rathore and Kumar [82] to predict software faults in which linear regression-based combination rule (LRCR) and gradient boosting regression-based combination rule (GBRCR) approaches were used to ensemble the output of Genetic Programming (GP), MLP, LR algorithms. Moreover, eleven datasets belonging to six software projects were accumulated from the PROMISE data repository to assess the performance of the proposed ensemble models. Results of different performance evaluation measures, including Average Absolute Error (AAE) and Average Relative Error (ARE), provided evidence that ensemble techniques can produce better results for predicting software faults compared to individual fault prediction techniques. Choudhary et al. [83] defined a set of change matrices in addition to the existing ones to enhance the performance of SFP modules.

Various ML classifiers were applied along with code matrices and change matrices. Experimental results on different releases of Eclipse projects demonstrate that the newly introduced change matrices can improve the performance of fault prediction modules. In [84], Shatnawi used the ROC analysis to examine the relationship between software matrices (features) and faults where threshold values of matrices were identified accordingly. A threshold value is defined for each metric to be used for deciding whether a software module is faulty or not. Moreover, the results of ROC were also considered for selecting the most correlated matrices with faults. Only selected matrices were applied to train and test a set of ML classifiers (LR, NB, KNN, and decision trees C4.5).

From the previously investigated related work, researchers confirmed that having a considerable number of features in a dataset affects the performance of the ML technique. Therefore, many researchers considered dimensionality reduction methods to eliminate the irrelevant/redundant features from the datasets. The most popular dimensionality reduction technique is FS.

2.2 Preprocessing ML methods

FS is a well-known preprocessing step in the data mining process that aims to eliminate noisy, irrelevant, and redundant features to reduce data dimensionality, and hence, improve the performance of the employed ML technique [24, 85]. In many works in the field of SFP, different filter, and wrapper FS approaches were investigated. Catal, C. and Diri, B. [86] employed a correlation-based feature selection approach to select the highly relevant matrices with varying techniques of ML (i.e., RF, DT, NB, and AIRS). They found that FS positively affected the performance of the employed ML approaches and that the RF classifier outperformed other classification techniques. In [87], eighteen filter FS methods were employed on five datasets from the NASA repository with various classification techniques. The obtained results revealed that using FS enhanced the performance of the prediction models.

As presented in [88], a set of filter FS methods, including Chi-square (CS), information gain (IG), and Pearson Correlation Coefficient (PCC), was used to develop a hybrid feature selection method to improve the performance of Software Defect Prediction (SDP). In the hybrid FS method, the features were ranked and selected according to their values using these filter ranking methods. In addition, for

comparison purposes, each of the three filter methods was applied separately. Using five NASA datasets for validating the FS method and a RF classifier for building the prediction model, the results of AUC show that the hybrid FS approach is superior compared to other filter FS methods.

Moreover, many wrapper FS methods were applied in the SFP field. A GA-based FS approach with a bagging technique was proposed in [89]. In this approach, two preprocessing techniques were used; FS (i.e., GA) and resampling (i.e., bagging). A similar approach was proposed in [90]. In this approach, two metaheuristics algorithms (i.e., GA and PSO) were applied as selection mechanisms in the FS process, in addition to considering a bagging technique to rebalance the used nine datasets.

Another wrapper FS was recently proposed by Turabieh, H., Mafarja, M. and Li, X. [2]. In this approach, the authors used several FS approaches to improve the efficiency of a Layered Recurrent Neural Network (L-RNN) classifier in predicting faulty software components. Three metaheuristics algorithms (i.e., GA, PSO, and ACO) were considered in this paper as FS approaches. A set of extensive experiments were conducted in this paper, and the performance of the proposed approach was compared with several ML classifiers (i.e., NB, LR, ANN, C4.5 DT, and KNN), and the area under the curve (AUC) was considered as an evaluation measurement. AUC confirmed that the proposed wrapper approach is better compared to other approaches. In this approach, a Binary Queuing Search Algorithm (BQSA) was proposed for the first time in literature in the SFP field. Moreover, the SMOTE technique was applied to rebalance the datasets that were obtained from the PROMISE repository. The presented results in the paper revealed the positive effects of dimensionality reduction and resampling techniques on the obtained datasets.

In 2020, Tumar, Iyad, et al. [3] proposed a modified version of binary Moth Flame Optimization named Enhanced MFO (EBMFO) as a wrapper FS approach in SFP, along with the Adaptive synthetic sampling method (ADASYN) as a resampling technique. Three ML classifiers were used in this paper (i.e., LDA, KNN, and DT), and the results confirmed that the performance of these classifiers was improved with the use of the preprocessing techniques. Recently, a Harris Hawk Optimization algorithm (called EBHHO) based FS approach in the SFP field was proposed in [91]. Again, the obtained results proved the positive influence of the employed preprocessing techniques on the used ML classifiers.

From the previously mentioned approaches, it is clear that preparing the datasets by employing preprocessing techniques (e.g., FS and resampling) greatly influences the performance of the prediction model in the SFP problem. It can be concluded that SFP becomes possible for large-scale projects when proper preprocessing techniques are used. These observations, besides the Non-Free-Lunch (NFL) theorem for optimization [66] which states that no best classifier to handle all possible classification problems [92], motivated our attempts to propose an advanced SFP approach that considers the RF classifier as an ML technique, improved by SMOTE as a resampling technique, along with an advanced wrapper FS approach with a novel WOA algorithm as a searching strategy.

3 Preliminaries

This section briefly describes the main theoretical concepts utilized in this research which are: the RF classifier, the oversampling technique (i.e., SMOTE), feature subset selection, and WOA to tackle the FS problem. In the SFP problem, the aim is to predict fault-prone software modules in the early stages of the SDLC based on the designed metrics of the software project. SFP is considered a binary classification problem since each software component has two options in the target class: faulty or non-faulty. Several supervised classification paradigms can be used to tackle this problem. After conducting deep experimental studies, we considered the RF classifier to be adopted in this research.

3.1 Random Forest classification paradigm

Random Forest (RF) is a classification algorithm that was initially proposed in [20]. RF considers a combination of decision trees in a model called ensemble learning, where each tree in the forest depends on an independently sampled vector of random values. The main advantages of the RF classifier are the few parameters to be tuned, its capability of high generalization, and it requires less training time than other classifiers regardless of the size of the dataset [93]. Ensemble learners combine different classification algorithms to get a generalized model that enhances the prediction performance where the number of wrongly classified instances comes at a very low rate. Ensemble methods can be distinguished into three families, namely, bagging, boosting, and stacking methods.

RF classifiers use the bagging method. As demonstrated in Fig. 1, a random sample of the original training dataset is provided by applying a bootstrap re-sampling technique for each model. After applying the bagging process, x features are selected randomly from the full feature set. Then, one feature is selected as a split node. The splitting process is repeated, with a fresh selection of features, until reaching a specified depth d where a decision tree is completed [94]. After the multiple splits, a random forest of decision trees is constituted. Every new instance is passed to all trees in

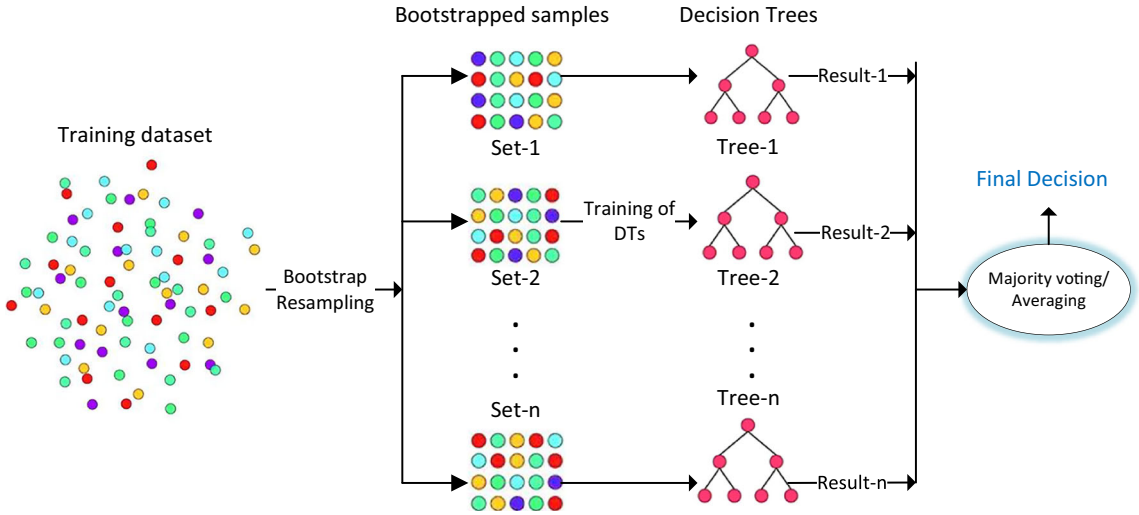

Fig. 1 Ensemble learning (bagging method)

the forest, and a class label is predicted (termed as a vote), Then the majority voting strategy is applied to select the class label for this instance.

3.2 Data sampling for imbalanced classification

The quality of data is considered a significant factor that has a profound impact on the performance of ML techniques. Imbalanced datasets are distinguished as a challenging aspect that may degrade the prediction quality of classification methods. This issue emerges in most real-world problems in which the target classes are not represented equally. In other words, in binary class datasets, most of the instances are labeled with the first class (called majority class), while few of them are labeled with the other one (called minority class). In such a case, the classifier is trained on highly imbalanced data and thus tends to pick up the patterns in the dominant classes, which leads to inaccurate prediction of the minority class [95].

The class imbalance problem poses a significant challenge in the field of software defect prediction since the available datasets are highly imbalanced. That is to say, the occurrences of defective cases are very low compared to normal cases (see Fig. 3). Various strategies can be employed to handle this problem, such as cost-sensitive, kernel-based, and sampling methods [95]. Sampling methods are categorized into two types: oversampling, which increases the rate of the minority class, and under-sampling, which reduces the frequency of the minority class. The latter causes information to be lost, which leads to poor prediction quality. In this research, we utilized an oversampling technique called SMOTE to rebalance the used SFP datasets.

The SMOTE is a promising oversampling method that proves its superiority in dealing with imbalanced data. It is originally introduced by Chawla, Nitesh V., et al. [96]. This technique preserves the original data without losing information, and it increases the rate of the minority class without duplication. New synthetic samples ($\hat{x}_{ij}$) labeled with the minority class are generated using the k-nearest neighbors' method for each minority sample (x_i) using the Euclidean distance, where $j = 1, 2, ..., k$. The new synthetic samples are generated along the lines joining the minority sample and its j selected neighbors as in (1).

$$x_{new} = x_i + (\hat{x}_i - x_i) * r \tag{1}$$

where r is a random vector between 0 and 1, $\hat{x}_i$ denotes one of the k neighbors. The value of k depends on the desired amount of oversampling.

3.3 Feature Selection (FS)

One of the most common questions when applying ML algorithms is whether all features (factors) are relevant to the classification rule. As a response to this question, a problem called FS emerged. FS is defined as the process of reducing the dimension of data by eliminating irrelevant, noisy, and redundant features. In other words, it is the task of searching for the most informative subset of features. It is an essential pre-processing technique that aims to enhance the performance of ML tasks [25, 26, 97].

FS approaches are classified into wrapper and filter based on the evaluation function used to measure the selected subset of features [98]. In wrapper-based methods, a search algorithm (deterministic or heuristic) is employed to generate subsets of features for examination. Then

the effectiveness of each suggested subset of features is evaluated by a given classifier (learning algorithm). The evaluation is conducted in terms of several measures such as accuracy, the area under the ROC, etc. FS is treated as a binary optimization problem in which the search algorithm is guided using the reported error by a classifier [99].

In the filter-based approach, the learning algorithm is not involved in the evaluation function. The effectiveness of a subset of features can be evaluated based on the intrinsic properties of the data. Statistical measures are used to measure the dependency or correlation between features, which can be filtered to select the most informative features. Several ranking techniques have been introduced for feature evaluations, such as gain ratio and information gain [100]. The filter-based approach is more effective compared to a wrapper-based approach in terms of the required computational time.

In this paper, we propose a wrapper FS approach that considers WOA as a selection mechanism and RF classifier as an evaluation method. In the following subsection, the WOA is introduced, followed by the description of the enhanced approach of the original WOA.

3.4 An overview of the WOA

WOA is a recent Swarm Intelligence (SI) algorithm that mimics the behavior of humpback whales in hunting fish in the oceans. The hunting process starts by constructing bubble nets to constrict the prey, and then the whale swims towards them in a spiral shape to attack them. According to [60], WOA can balance its stochastic exploratory and exploitative tendencies effectively. In the exploration phase, WOA simulates the encircling mechanism of the whales in nature. Where the prey represents the best solution, found so far, and the other solutions in the population represent the candidate whales. The whales change their positions by moving spirally toward the prey's location as modeled in (2) and (3):

$$D = \mid C.\vec{X}^{*}(t) - \vec{X}\,(t) \mid \tag{2}$$

$$\vec{X}\,(t+1) = \vec{X}^{*}(t) - \vec{A}\,.D \tag{3}$$

where t represents the current iteration, X^{*} represents the prey's location, X represents the locations of the candidate solution (whale). Vectors A and C are defined in (4) and (5).

$$\vec{A} = 2\vec{a}\,.\vec{r} - \vec{a} \tag{4}$$

$$\vec{C} = 2.\vec{r} \tag{5}$$

where $\vec{r}$ is generated randomly in the interval [0,1], and $\vec{a}$ simulates the shrinking encircling behavior of the whales and decreases linearly in the interval [2, 0] as in (6)).

$$a = 2\left(1 - \frac{t}{T}\right) \tag{6}$$

According to (7) which represents the bubble-net attacking process (exploitation phase), a solution's position is changed based on two different approaches; shrinking encircling (when $p < 0.5$) and spiral updating mechanism (when $p \geq 0.5$), where a probability of 50% is used to select between these two approaches.

$$\vec{X}\,(t+1) = \begin{cases} \vec{X}^{*}(t) - \vec{A}\,.D & p < 0.5 \\ D'.e^{bl}.\cos(2\pi l) + \vec{X}^{*}(t) & p \geq 0.5 \end{cases} \tag{7}$$

where D' represents the distance between the ith solution and the prey's location, b is a constant, and l is a random number in the interval [-1,1].

Based on the variation of $\vec{A}$, which takes a value in the interval [-1, 1], a solution is forced to move towards or far away from the best solution. If $\vec{A} < 1$, then a solution is moved towards the prey's location (exploitation), while it is moved towards a randomly selected solution from the population (represented as $\vec{X_{rand}}$ in (8) and (9)) when $\vec{A} > 1$ (exploration).

$$D = \mid C.\vec{X_{rand}}\,(t) - \vec{X}\,(t) \mid \tag{8}$$

$$\vec{X}\,(t+1) = \vec{X_{rand}}\,(t) - \vec{A}\,.D \tag{9}$$

As for all population-based metaheuristic algorithms (MAs), WOA starts the optimization process by generating N random solution, each of which represents a whale in nature. Then, each solution is evaluated using the adopted fitness function. The solution with the lowest fitness value is denoted as the best solution since this type of this problem is a minimization problem, and the coefficients of the algorithm are calculated. The algorithm then moves according to the parameter a which is decreased from 2 to 0. Each solution is updated based on the value of $\vec{A}$, where it moved towards a randomly selected solution from the current population when $\vec{A} > 1$. Also, it is moved towards the best solution when $\vec{A} < 1$. The WOA switches between a spiral or circular movement based on the value of p. The pseudo-code of WOA is shown in Algorithm 1.

4 The proposed methodology

The main objective of this paper is to build a well-performing classification model that is able to predict

```
Generate a random population
Initialize all coefficients in the algorithm
Evaluate all solutions in the population using a proper fitness function
Denote the best solution so far as X*
while (t < L) do
    for each solution in the population do
        Update all coefficients (i.e., a, A, C, l, and p)
        if (p ≤ 0.5) then
            if (| A |< 1) then
                Use Eq. (3) to update the position of the current solution
            else if (| A |> 1) then
                Select one random solution from the population
                Update the position of X(t) by Eq. (9)
            else if (p > 0.5) then
                Update the position of X(t) by Eq. (7)
    Reevaluate all solutions in the populations
    Update X* with a better solution if any.
    t = t + 1
return X*
```

Algorithm 1 Pseudo-code of WOA.

faulty software components. The datasets were selected from the PROMISE repository and normalized to set a proper scale for all data. Moreover, some techniques were applied to balance the data to get more accurate results. Then, more experiments were conducted to select the most appropriate classifier for this problem. After that, extensive experiments were conducted to tune the parameters of the selected classifier. In the last phase, a set of wrapper feature selection methods were applied to enhance the performance of the adopted classier. Figure 2 represents the proposed methodology.

4.1 Preprocessing techniques

Data preprocessing is a vital step in the mining process. It aims to prepare the dataset to be suitable for the mining techniques to achieve high performance. The datasets are investigated using a 2D visualization using Principle Component Analysis (PCA) as shown in Fig. 3. The figure demonstrates that the datasets are highly imbalanced and not linearly separable. Therefore, all datasets should be balanced before applying proper results and adopting a learning algorithm. Moreover, a complex learning algorithm is required to provide better performance because the data in datasets is not linearly separable.

– **Data Normalization**: The collected datasets are complete, and no missing data are there. Their structures are well to be mined, and all attributes are numeric. However, the numeric data are of different scales. Therefore, to avoid bias towards some dominant features, the Min-max normalization method (as can be seen in (10)) was applied to standardize the data in the interval of [0, 1].

$$x^n = \frac{x - \min}{\max - \min} \tag{10}$$

where x^n is the normalized value of x within the interval [min, max].

– Data Balancing After investigating the adopted datasets, we noticed that they are highly imbalanced as the rate of faulty instances is very low compared to normal ones (see Fig. 3). Thus, the datasets should be balanced before using them with the classification technique to avoid any decrease in their performance. In this paper, we applied three variants of the SMOTE oversampling technique (i.e., SMOTE, Borderline SMOTE, and SVM

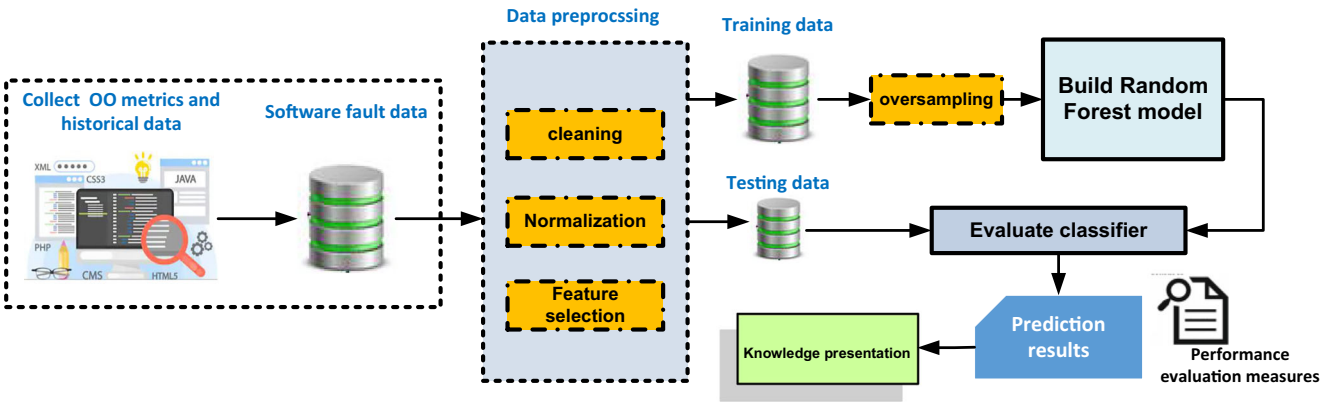

Fig. 2 Software fault prediction process

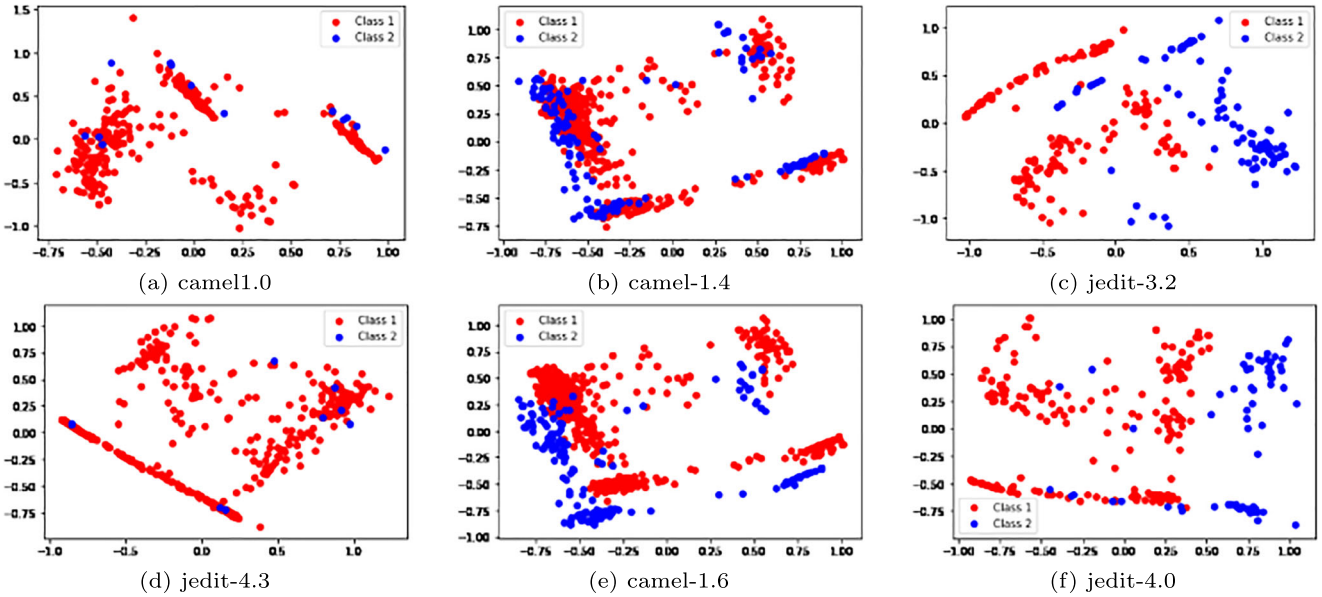

Fig. 3 Visualization of target class distribution based on the first 2 principal components of the dataset features

SMOTE) to select the best one that positively affects the performance of the learning algorithm.

4.2 Classifier selection

Investigating data visualization in Fig. 3, it can be seen that data in most datasets are not linearly separable. Thus, simple classifiers may not be suitable to handle this kind of problem. Therefore, we did extensive experiments to compare the performance of different classifiers on the same datasets and under the same computational system.

4.3 Binary variant of WOA (BWOA)

As mentioned earlier, FS is a binary optimization problem, while the WOA was originally designed to deal with continuous optimization problems. This requires using a conversion function that converts the continuous solutions to binary to make them suitable for binary problems. We used the Transfer Functions (TFs) that were widely used to convert the continuous metaheuristics population to binary [101, 102]. TFs can be categorised based on their shapes [103] into S-shaped and V-shaped functions (see Fig. 4). The proposed Binary WOA for FS is called BWOA.

The S-shape TF [104] is used to convert the continuous PSO algorithm into binary. The TF is usually used to produce a probability of flipping a future value from 0 to 1 or from 1 to 0 as in (11). It takes the elements of the step vector (solution x) that was generated by the algorithm.

$$S(x_i^j(t)) = \frac{1}{1 + \exp^{-x_i^j(t)}} \tag{11}$$

where x_i^j represents the j_{th} element in the i_{th} solution x, and t indicates the current iteration. An element of a solution in the next iteration is updated by (12).

$$X_i^j(t+1) = \begin{cases} 0 \text{ If } rand < S(x_i^j(t+1)) \\ 1 \text{ If } rand \geq S(x_i^j(t+1)) \end{cases} \tag{12}$$

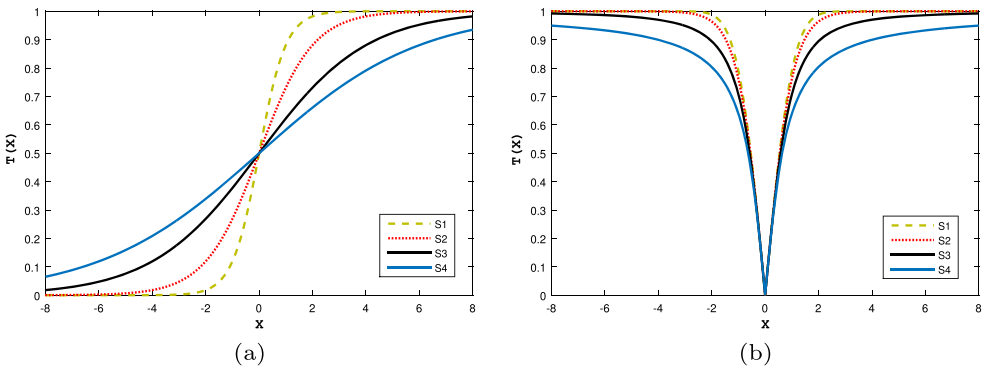

Fig. 4 Transfer functions families (a) S-shaped and (b) V-shaped

Table 1 S-shaped and V-shaped transfer functions

S-shaped family		V-shaped family	
Name	Transfer function	Name	Transfer function
S1	$T(x) = \frac{1}{1+e^{-2x}}$	V1	$T(x) = \left\|\text{erf}\left(\frac{\sqrt{\Pi}}{2}x\right)\right\| = \left\|\frac{\sqrt{2}}{\Pi}\int_0^{(\sqrt{\Pi}/2)x} e^{-t^2}dt\right\|$
S2 [104]	$T(x) = \frac{1}{1+e^{-x}}$	V2 [105]	$T(x) = \left\|\tanh(x)\right\|$
S3	$T(x) = \frac{1}{1+e^{(-x/2)}}$	V3	$T(x) = \left\|(x)/\sqrt{1+x^2}\right\|$
S4	$T(x) = \frac{1}{1+e^{(-x/3)}}$	V4	$T(x) = \left\|\frac{2}{\Pi}\text{arc tan}\left(\frac{\Pi}{2}x\right)\right\|$

where $X_i^j(t+1)$ is the binary value of the real x_i^j, $S(x_i^j(t))$ is the probability value, which can be obtained via (11).

Another TF that belongs to the V-shaped family [105] is used to convert the continuous version of the GSA algorithm into binary. Equation (13) represents the V-shape TF and (14) represents the rule to convert to binary.

$$V(x_i^j(t)) = |\tanh(x_i^j(t))| \tag{13}$$

$$X_i^j(t+1) = \begin{cases} \neg X_i^j(t) & r < V(\Delta x_i^j(t+1)) \\ X_i^j(t) & r \geq V(\Delta x_i^j(t+1)) \end{cases} \tag{14}$$

In this paper, eight TFs were adopted to convert the original WOA into binary. The original TFs that were proposed in [104], which are *S2 TF* and *V2 TF* [105]. In addition, the six TFs that were proposed in [103] are evaluated. The mathematical formulation of all TFs is shown in Table 1.

4.4 Formulation of FS problem

One of the critical aspects that should be considered when designing an optimization algorithm is the fitness function. Since FS is a multi-objective optimization problem, both objectives should be considered when evaluating a solution. The suitability of a feature subset is determined by the number of selected features (minimize) and the classification accuracy (maximize). Aggregation is one of the most popular techniques for multi-objective formulation [106]. In this technique, the objectives are integrated into a single objective formula such that a preset weight identifies each objective importance. In this work, we adopt a fitness function that combines both objectives of FS as shown in (15).

$$Fitness(X) = \alpha \cdot E(X) + \beta * \left(1 - \frac{|R|}{|N|}\right) \tag{15}$$

where $Fitness(X)$ represents the fitness value of a subset X, $E(X)$ represents the classification error rate by using the selected features in the X subset, $|R|$ and $|N|$ are the number of selected features and the number of original features in the dataset respectively, α and β are the weights of the classification error and the reduction ratio, $\alpha \in [0, 1]$ and $\beta = (1-\alpha)$ adopted from [36, 97, 107].

Another aspect that should be considered when designing an optimization approach to tackle an FS problem is the solution representation. In this research, the feature subset is represented as a binary vector of N elements where N is the total number of features in the original dataset. Each dimension in that vector contains a binary value (0 or 1). The 0 value indicates that the corresponding feature is not selected in that feature subset, while the 1 value indicates the corresponding feature is selected. Figure 5 shows a sample solution for a dataset of N features.

4.4.1 Enhanced BWOA (BEWOA)

The performance of the binary version of WOA (BWOA) can be improved to deal with the complex nature of the SFP problem more efficiently. This is because the SFP search space is highly dimensions and rigid. Therefore, the enhanced version of BWOA (BEWOA) extracts efficient exploration strategies from GWO [108] and HHO [109]. The flowchart of the proposed BEWOA is illustrated in Fig. 6.

We propose a new approach that employs multi-exploration strategies instead of having one strategy as follows:

– The exploration strategy that was used in GWO, where the best three solutions are selected and updated based on the values of A and C parameters as shown in (16–19).

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1.|\vec{C}_1.\vec{X}_\alpha - \vec{X}(t)| \tag{16}$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2.|\vec{C}_2.\vec{X}_\beta - \vec{X}(t)| \tag{17}$$

$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3.|\vec{C}_3.\vec{X}_\delta - \vec{X}(t)| \tag{18}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \tag{19}$$

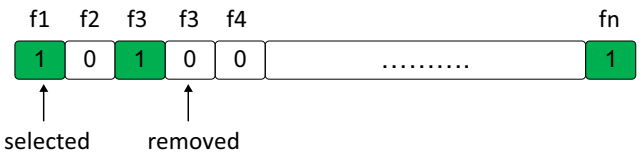

Fig. 5 Binary solution representation

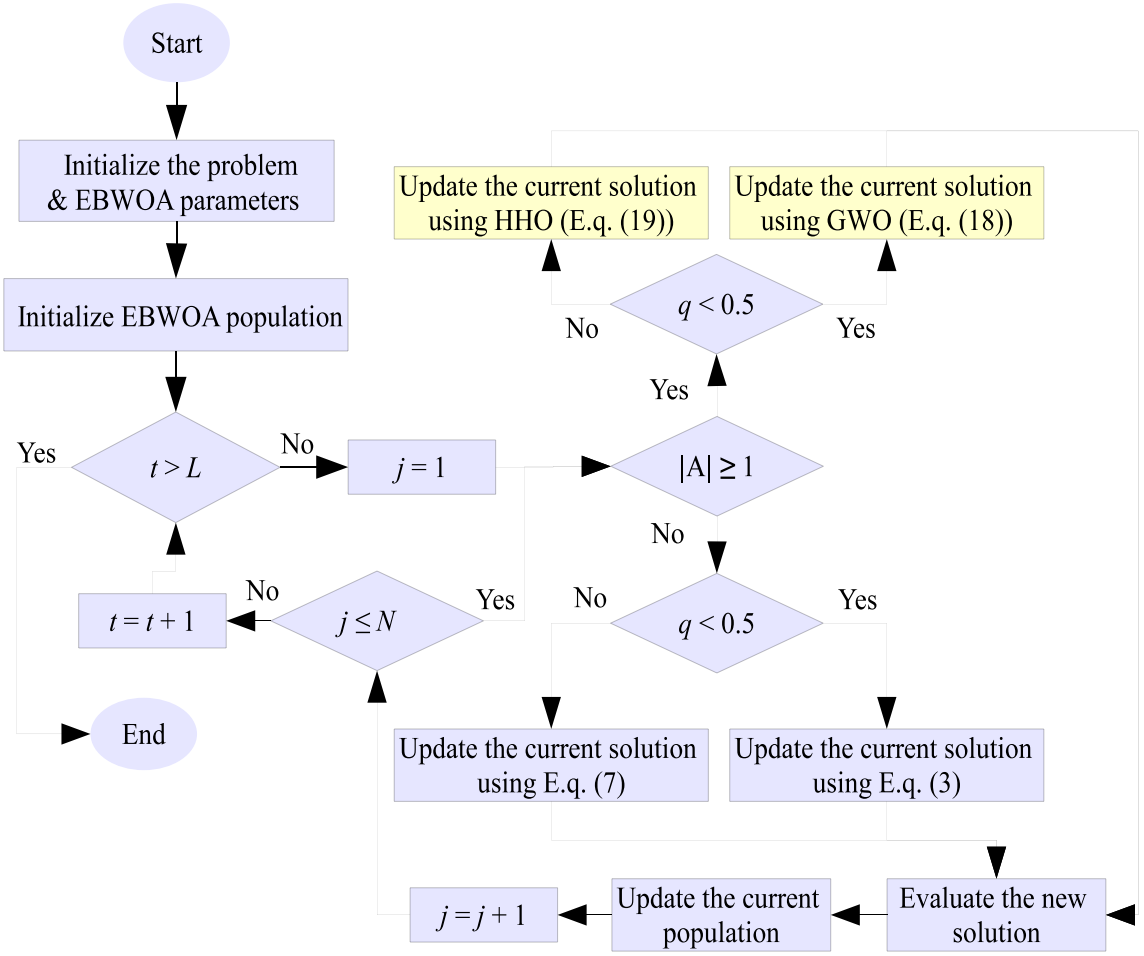

Fig. 6 The flowchart of the proposed BEWOA

Note that the $\vec{X}_\alpha$, $\vec{X}_\beta$ and $\vec{X}_\delta$ are the best, second-best, third-best solutions in the population. The vectors $\vec{C}_1$, $\vec{C}_2$, $\vec{C}_2$ are calculated based on the (5) while vectors $\vec{A}_1$, $\vec{A}_2$, $\vec{A}_2$ are calculated based on the (4).

– The average of all positions was used in HHO to guide the solution being updated. This strategy is used as shown in (20, 21).

$$\vec{X}(t+1) = r.\vec{X}^*(t) - \vec{X}_{avg}(t) \tag{20}$$

$$X_{avg}(t) = \frac{1}{N}\sum_{i=1}^{N} X_i(t) \tag{21}$$

Algorithm 2 shows the new modifications. Now, instead of updating the position of the current solution based on the WOA ((3) in Algorithm 1), the positions in BEWOA of the current solution are updated based on either GWO updating (19) or HHO updating (20).

The main rationale of this enhancement is to improve the exploration stage of the original WOA by employing the survival-of-the-fittest principle rather than random search. Since the search space of SFP is very deep, the search requires a more guided exploration to reach the best possible solution. Therefore, concentrating on the best solutions in the search space regions can converge the search to the optimal solution quickly.

In order to provide more insight about the proposed BEWOA, the time complexity is analyzed. As can be noticed from Algorithm 2, it has several statements with loops. The statement in Line 1 needs a $\mathcal{O}(N \times n)$ where N is the population size while n is the solution dimension . In line 3, the complexity depends on the evaluation function complexity as calculated in (15). In the main loop starting from Line 5 to Line 18, the time complexity is $\mathcal{O}(L \times N \times n)$. Therefore, the time complexity of the proposed method is $\mathcal{O}(LNn)$ which is similar to the original WOA.

5 Experimental results and discussion

A set of experiments are conducted in this research to prove the efficiency of the proposed approach. This section

Generate a random population
Initialize all coefficients in the algorithm
Evaluate all solutions in the population using a proper fitness function
Denote the best solution so far as X^*
while $(t < L)$ **do**
  for each solution in the population **do**
    Update all coefficients (i.e., a, A, C, l, and p)
    if $(|A| >= 1)$ **then** ▷ `Exploration phase`
      if $(q \leq 0.5)$ **then**
        Update the current solution using (19)
      else if $(q \geq 0.5)$ **then**
        Update the current solution using (20)
      Evaluate the new solution
    else if $(|A| < 1)$ **then** ▷ `Exploitation phase`
      if $(p < 0.5)$ **then**
        Use (3) to update the position of the current solution
      else if $(p > 0.5)$ **then**
        Update the position of $X(t)$ by (7)
      Evaluate the new solution
  Update X^* with a better solution if any.
  $t = t + 1$
return X^*

Algorithm 2 Pseudo-code of the proposed BEWOA.

presents these results in detail. The experiments are carried out in four phases as follows:

- First, different SMOTE techniques are adopted to solve the imbalance dataset issues.
- Second, the hyper-parameter settings of the RF classifier are investigated, and those parameters that provided the optimal results are selected.
- Third, the performance is tested using 7 popular machine learning methods (RF, KNN, NB, LDA, DT, LR, and SVM).
- Fourth, a novel BEWOA approach is introduced as a feature selection method to tackle the data dimensionality problem. In addition, the proposed approach is compared with the other 9 state-of-the-art methods.

In the experiments, the train/test model is adopted. The datasets are split randomly into two parts; 66% are used for training purposes, while the remaining 34% of the dataset is used for blind testing. Due to the availability of random factors in the classification algorithm (i.e., RF) and the optimization technique (i.e., WOA), we only report the average values of 30 runs for each employed method. Please note that we use the **boldface** to represent the best results. All the experiments are conducted on a computer under the same environment and conditions to guarantee a fair comparison. The details of the utilized system are presented in Table 2. It is worth mentioning that Python programming language is used to implement the proposed classification framework, together with the open-source libraries (e.g., Panda, Numpy, Matplotlib, and SKlearn (Scikit-learn)).

Table 2 The system properties

Name	Setting
Hardware	
CPU	Intel Core(TM) i7-8550U
Frequency	2.2 GHz
RAM	8 GB
Hard drive	1 TB
Software	
Operating system	Windows 10 64bit
Language	Python 3.8.3

5.1 Datasets: investigated software projects

In this work, we adopt 16 well-known datasets related to the SFP problem to assess the performance of the proposed approach. The datasets are collected from the PROMISE repository [110, 111] and have been recommended by many researchers in the field. The adopted datasets consist of several software metrics (e.g., object-oriented metrics) that are usually employed to investigate the quality of a software project (see Table 4 for more information about these metrics) [112]. The details of the adopted datasets are presented in Table 3. It can be seen that these datasets have various sizes (from 109 to 909 instances), and all of them have 20 object-oriented metrics.

From Table 3 it can be noticed that most of the datasets are imbalanced, where the rate of the defected instances is much lower than the rate of the normal instances. In some of the datasets, the rate of the defective instances is only 2.2% from all instances. This observation motivates us to use an oversampling technique to re-balance the datasets before applying them to the ML approaches. Table 4 illustrates the main features provided in the dataset given in Table 3.

Table 3 Details of the 16 software projects (datasets) from PROMISE repository

Dataset	version	No. metrics	No. instances	No. normal instances	No. defective instances	Rate of defective instances
Ant	1.7	20	745	579	166	0.223
Camel	1.0	20	339	326	13	0.038
	1.2	20	608	392	216	0.355
	1.4	20	872	727	145	0.166
	1.6	20	965	777	188	0.195
Jedit	3.2	20	272	182	90	0.331
	4.0	20	306	231	75	0.245
	4.1	20	312	233	79	0.253
	4.2	20	367	319	48	0.131
	4.3	20	492	481	11	0.022
Log4j	1.0	20	135	101	34	0.252
	1.1	20	109	72	37	0.339
Lucene	2.0	20	195	104	91	0.467
Xalan	2.4	20	723	613	110	0.152
	2.5	20	803	416	387	0.482
	2.6	20	885	474	411	0.464

5.2 Evaluation measures

Since SFP is a binary classification problem, we use the confusion matrix to calculate the evaluation measures. Table 5 represents the confusion matrix, followed by the equations for calculating the evaluation metrics. The True Positive Rate (TPR) is calculated according to (22), True Negative Rate (TNR) is calculated as in (23), while the Area

Table 4 Description of object-oriented metrics

Metrics	Descriotion
WMC	Number of methods defined in a class.
DIT	provides a measure of the inheritance levels from the object hierarchy top for each class .
NOC	Number of immediate descendants of a class.
CBO	Count the number of classes coupled to a given class.
RFC	Count the number of distinct methods invoked by a class in response to a received message.
LCOM	Count the number of methods that do not share a field to the method pairs that do.
CA	Count the number of dependent classes for a given class.
CE	Count the number of classes on which a class depends.
NPM	Number of public methods defined in a class.
LCOM3	Count the number of connected components in a method graph.
LOC	Count the total number of lines of code of a class.
DAM	Computes the ratio of private attributes in a class.
MOA	Count the number of data members declared as class type.
MFA	Shows the fraction of the methods inherited by a class to the methods accessible by the functions defined in the class.
CAM	Computes the cohesion among methods of a class based on the parameters list.
IC	Count the number of coupled ancestor classes of a class.
CBM	Count the number of new or redefined methods that are coupled with the inherited methods.
AMC	Measures the average method size for each class.
MAX_CC	Maximum counts of the number of logically independent paths in a method.
AVG_CC	Average counts of the number of logically independent paths in a method.

Table 5 Confusion matrix for binary classification

	Predicted positive	Predicted negative
Actual positive	True Positive (TP)	False Negative (FN)
Actual negative	False Positive (FP)	True Negative (TN)

Under the Curve (AUC) is calculated based on (24). Since the available datasets are imbalanced, and thus we do not consider the accuracy as the evaluation measure because it can be profoundly misleading of judging a model.

- TPR: The percentage of the positive cases that are correctly classified as positive.

$$TPR = TP/(TP + FN) \tag{22}$$

- TNR: The percentage of negative cases that are correctly classified as negative.

$$TNR = TN/(TN + FP) \tag{23}$$

- AUC: a measure of how well a model can distinguish between defected and normal groups.

$$AUC = (TPR + TNR)/2 \tag{24}$$

5.3 Handling imbalanced data using different SMOTE variants

In this section, we are interested in assessing the effect of rebalancing the datasets on the RF classifier. Three SMOTE variants (namely: SMOTE, BorderlineSMOTE, and SVMSMOTE) are applied, and the AUC, TPR, and TNR results are reported in Table 6. Observing the results in Table 6, one can conclude that SMOTE recorded the best performance. Moreover, the prediction quality of the RF classifier has been enhanced after rebalancing the datasets. Without applying the SMOTE technique, the TPR was very high compared with the TNR. However, after applying the SMOTE techniques, the results became more reasonable, and the TNR became more realistic.

Table 7 displays the AUC rates obtained by the RF classifier using different SMOTE variants. Inspecting the result in Table 7, it is clear that the AUC rate with SMOTE techniques was much better than the original one. Among the SMOTE variants, the BorderlineSMOTE was the best (rank of 1.88), followed by SMOTE (rank of 2.06). Our findings indicate that the BorderlineSMOTE was the most suitable technique for re-balancing the datasets. In sum, we can say that applying a resampling technique is important to the imbalanced datasets since it can significantly improve the performance of the classifier. On the whole, we apply BorderlineSMOTE technique on all datasets in the subsequent experiments.

Table 6 Evaluation results of RF classifier using different SMOTE variants in terms of testing TPR and TNR

Dataset	True Positive Rate				True negative Rate			
	without	SMOTE	BorderlineSMOTE	SVMSMOTE	without	SMOTE	BorderlineSMOTE	SVMSMOTE
Ant-1.7	**0.933**	0.866	0.840	0.892	0.418	0.811	**0.872**	0.833
Camel-1.0	**1.000**	0.975	0.975	0.965	0.000	0.957	0.968	**0.981**
Camel-1.2	**0.880**	0.809	0.718	0.840	0.300	**0.669**	0.661	0.613
Camel-1.4	**0.972**	0.893	0.901	0.942	0.156	0.839	**0.865**	0.813
Camel-1.6	**0.937**	0.876	0.832	0.876	0.176	**0.895**	0.844	0.793
Jedit-3.2	**0.927**	0.885	0.869	0.885	0.722	0.793	0.828	**0.897**
Jedit-4.0	**0.915**	0.895	0.908	0.882	0.333	**0.773**	0.747	0.733
Jedit-4.1	0.933	**0.963**	0.901	0.864	0.433	**0.845**	0.746	0.746
Jedit-4.2	**0.982**	0.902	0.919	0.919	0.143	**0.953**	**0.953**	0.894
Jedit-4.3	**1.000**	0.976	0.988	0.982	0.000	**0.979**	**0.979**	0.960
Log4j-1.0	**0.882**	**0.882**	**0.882**	**0.882**	0.333	0.688	**0.813**	0.594
Log4j-1.1	**0.917**	0.870	0.652	0.783	0.462	**0.792**	0.708	**0.792**
Lucene-2.0	0.619	0.694	0.750	**0.861**	0.542	**0.563**	0.531	**0.563**
Xalan-2.4	**0.937**	0.862	0.867	0.862	0.250	0.818	**0.862**	0.829
Xalan-2.5	0.746	0.734	**0.811**	0.741	0.563	**0.688**	0.586	0.680
Xalan-2.6	**0.852**	0.815	0.828	0.803	0.634	0.671	**0.691**	0.684
Overall Rank (F-Test)	**1.41**	2.97	2.78	2.84	3.94	**1.88**	1.91	2.28

Table 7 AUC rates obtained by RF classifier using different SMOTE variants

Dataset	without	SMOTE	BorderlineSMOTE	SVMSMOTE
Ant-1.7	0.676	0.839	0.856	**0.863**
Camel-1.0	0.500	0.966	0.971	**0.973**
Camel-1.2	0.590	**0.739**	0.689	0.726
Camel-1.4	0.564	0.866	**0.883**	0.878
Camel-1.6	0.557	**0.885**	0.838	0.834
Jedit-3.2	0.825	0.839	0.848	**0.891**
Jedit-4.0	0.624	**0.834**	0.827	0.807
Jedit-4.1	0.683	**0.904**	0.824	0.805
Jedit-4.2	0.562	0.928	**0.936**	0.906
Jedit-4.3	0.500	0.978	**0.984**	0.971
Log4j-1.0	0.608	0.785	**0.847**	0.738
Log4j-1.1	0.689	**0.831**	0.680	0.787
Lucene-2.0	0.580	0.628	0.641	**0.712**
Xalan-2.4	0.593	0.840	**0.864**	0.846
Xalan-2.5	0.655	**0.711**	0.699	0.710
Xalan-2.6	0.743	0.743	**0.759**	0.743
Rank (F-Test)	3.88	2.06	**1.88**	2.19

5.4 Random forest hyperparameter tuning

This section investigates the hyperparameter settings of the RF classifier. As well-known, the performance of the RF classifier is highly affected by the used parameter values. Therefore, we consider a set of comprehensive experiments to find out the most appropriate parameters that can best reveal the performance of the classifier.

The following hyperparameters of the RF classifier were examined:

- n_estimators: number of trees in the forest
- max_depth: max number of levels in each decision tree
- min_samples_split: min number of data points placed in a node before the node is split
- min_samples_leaf: min number of data points allowed in a leaf node
- bootstrap: a method for sampling data points (with or without replacement)
- max_samples: If bootstrap is True, the number of samples to draw from X to train each base estimator

Table 8 presents the hyperparameter settings of the RF classifier. As can be observed, we test the RF classifier using multiple combinations of parameters Initially, we tune the n_estimators and fixed the other parameters. The

Table 8 Hyperparameter tuning of the RF classifier

n_estimators	max_depth	min_samples_split	min_samples_leaf	bootstrap	max_samples
5,10,20,30,40,50,70 100,15,200,300,400	None	2	1	TRUE	1
10	1,2,5,7,10,15 20,25,30,35 40,50	2	1	TRUE	1
10	7	2,3,4,5,7,10,12 15,20,25,50,70,100	1	TRUE	1
10	7	2	1,2,3,4,5,6,7 8,9,10,12,14	TRUE	1
10	7	2	1	TRUE, FALSE	1
10	7	2	1	TRUE	0.2,0.4,0.5,0.6 0.7,0.8,0.9,1
10*	**7***	**2***	**1***	**TRUE***	**1***

process of hyperparameter tuning is repeated by replacing the n_estimators with max_depth, min_samples_split, min_samples_leaf, bootstrap, and max_samples, respectively.

Figure 7 illustrates the training and testing curves of the AUC score for different hyperparameter settings. With proper tuning of these parameters, one can see that the training AUC score has been substantially increased. However, in some cases, an optimal training AUC performance may lead to the over-fitting issue, which provided a very low AUC score on the testing set. In this regard, we evaluate the hyperparameter using the testing AUC score rather than the training AUC in the present work. Based on the result obtained, the optimal performance of the RF classifier can be achieved with n_estimators=10, max_depth=7, min_samples_split=2, min_samples_leaf=1, bootstrap=TRUE, and max_samples=1.

5.5 Comparison with other classification techniques

After performing the hyperparameters tuning of the RF classifier, we compare it to a set of well-known classification algorithms (i.e., KNN, NB, LDA, DT, LR, and SVM) in solving the SFP problems. We adopt this set of classifiers because they are from different categories of the supervised ML techniques, and each one differs from the others in its structure. For example, the KNN classifier does not build the model, and it predicts the output based on the

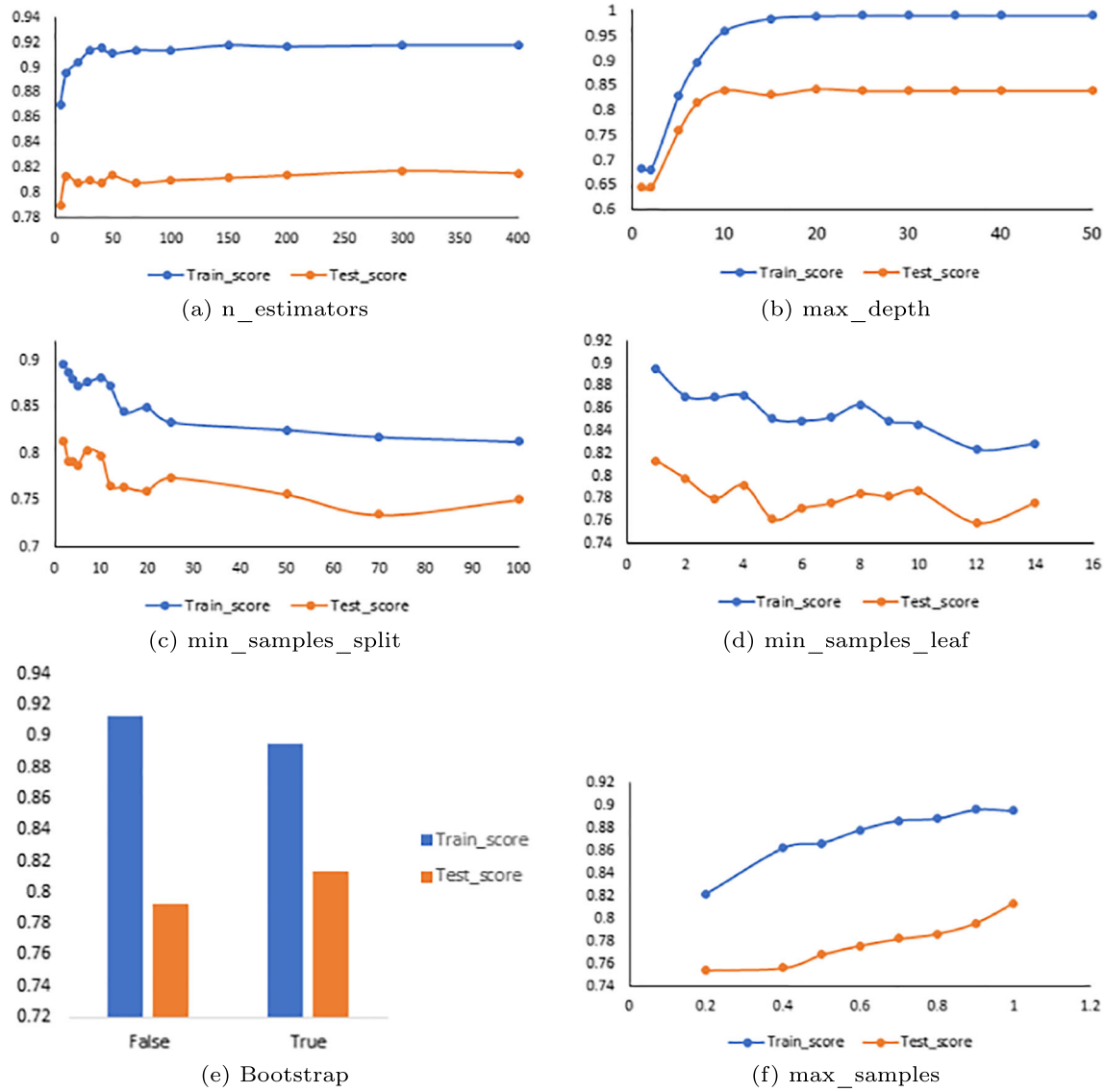

Fig. 7 The performance of RF model after adjusting the considered set of hyperparameters in terms of training and testing curves of AUC score

Table 9 Comparison of RF against traditional classifiers in terms of testing AUC rates

Dataset	RF	KNN	NB	LDA	LR	SVM	DT
Ant-1.7	**0.8465**	0.7926	0.6461	0.7554	0.7449	0.7653	0.8158
Camel-1.0	**0.9671**	0.9190	0.8526	0.9047	0.8479	0.8772	0.9503
Camel-1.2	0.6392	0.6312	0.5641	0.6073	0.5284	0.5212	**0.6509**
Camel-1.4	**0.8572**	0.7897	0.5773	0.7239	0.7399	0.7160	0.7939
Camel-1.6	**0.8136**	0.8031	0.5866	0.6549	0.6014	0.6330	0.7724
Jedit-3.2	0.7982	**0.8512**	0.7089	0.7896	0.7831	0.7679	0.7646
Jedit-4.0	**0.8077**	0.8015	0.6480	0.7486	0.7555	0.7426	0.8076
Jedit-4.1	0.8194	**0.8845**	0.6997	0.7992	0.7410	0.7269	0.7595
Jedit-4.2	**0.9313**	0.8866	0.8247	0.9019	0.8355	0.8676	0.9047
Jedit-4.3	**0.9778**	0.9732	0.6721	0.8737	0.8511	0.8501	0.9475
Log4j-1.0	**0.8474**	0.8235	0.7390	0.8024	0.7417	0.7868	0.7426
Log4j-1.1	0.6803	**0.7663**	0.6178	0.7228	0.6812	0.7029	0.7237
Lucene-2.0	0.5833	0.5660	0.6181	0.6424	0.6563	**0.6840**	0.4965
Xalan-2.4	0.8392	**0.8503**	0.6347	0.7477	0.7417	0.7353	0.8108
Xalan-2.5	**0.6697**	0.5920	0.5688	0.5892	0.6011	0.5892	0.6484
Xalan-2.6	**0.7307**	0.6890	0.6754	0.6871	0.6818	0.6967	0.7043
Rank (F-Test)	**1.81**	2.69	6.63	3.97	5.00	4.91	3.00

distance between training and testing samples. Also, the LR, NB, and LDA are easy-to-implement algorithms since they do not contain extra parameters.

In this study, the min-max normalization technique is applied to all datasets, in addition to the oversampling technique. After investigating the performance of each classifier, a deep comparison in terms of AUC rates among the classifiers is presented in Table 9. In order to obtain the overall rank, the average ranking values of the Friedman test (F-Test) are calculated.

From Table 9, it is observed that the RF classifier performed better than other classifiers in most cases. Among rivals, RF obtained the highest AUC scores in 63% of the datasets while the KNN, which came in second place, obtained the best results in 25% of the datasets. Besides, the results of the F-Test reveal that the RF classifier outperformed other classifiers with the rank of 1.81, and it was ranked as the best classifier in terms of the AUC measurement. Ultimately, it can be inferred that the RF classifier is a useful and powerful learning algorithm for SFP problems.

5.6 Feature selection based on proposed BWOA approaches

In the previous sections, the impact of the re-sampling technique on this work is presented. Accordingly, the RF classifier perceived the best AUC value in most of the SFP problems. Nevertheless, the appearance of irrelevant and redundant features might be degrading the performance of the classifier, which limits the learning ability of the algorithm in the SFP prediction. As such, feature selection can be an excellent way to resolve the above-mentioned issue. In this work, we propose the new variant of the BWOA algorithm as a feature selection method to select the significant features.

5.6.1 Performance of BWOA using different TFs

In the first sub-section, we study the performance of the BWOA algorithm with different TFs. According to [61], we adopt four S-shaped TFs (S1 to S4) and four V-shaped TFs (V1 to V4). In total, eight binary versions of WOA, namely: SBWOA1, SBWOA2, SBWOA3, SBWOA4, VBWOA1, VBWOA2, VBWOA3, and VBWOA4 are introduced. Table 10 reports the AUC results of the BWOA using different TFs. From Table 10, it is seen that the SBWOA1 achieved the optimal AUC score in at least 11 datasets. Our findings reveal that the SBWOA1 overtook other variants in selecting the most informative feature subset. This argument is supported by the results of the F-Test in Table 10. By observing the results in Table 11, one can see that the SBWOA1 was not very good at reducing the feature size. However, the SBWOA1 can often retain the significant

Table 10 Evaluation results of BWOA using 8 transfer functions in terms of AUC rates

Dataset	Measure	SBWOA1	SBWOA2	SBWOA3	SBWOA4	VBWOA1	VBWOA2	VBWOA3	VBWOA4
Ant-1.7	AVG	**0.8790**	0.8768	0.8754	0.8764	0.8722	0.8705	0.8718	0.8729
	STD	0.0042	0.0039	0.0040	0.0035	0.0059	0.0059	0.0054	0.0043
Camel-1.0	AVG	**0.9888**	0.9877	0.9879	0.9876	0.9870	0.9870	0.9858	0.9847
	STD	0.0021	0.0019	0.0023	0.0019	0.0031	0.0026	0.0015	0.0022
Camel-1.2	AVG	**0.7391**	0.7345	0.7354	0.7379	0.7254	0.7271	0.7273	0.7288
	STD	0.0080	0.0075	0.0055	0.0093	0.0095	0.0118	0.0128	0.0103
Camel-1.4	AVG	**0.8941**	0.8918	0.8907	0.8918	0.8857	0.8857	0.8856	0.8822
	STD	0.0032	0.0037	0.0046	0.0052	0.0074	0.0046	0.0066	0.0050
Camel-1.6	AVG	**0.8398**	0.8376	0.8339	0.8342	0.8349	0.8320	0.8304	0.8299
	STD	0.0041	0.0042	0.0065	0.0051	0.0062	0.0070	0.0076	0.0073
Jedit-3.2	AVG	**0.9060**	0.9044	0.9029	0.9049	0.8977	0.8936	0.8922	0.8956
	STD	0.0046	0.0056	0.0082	0.0091	0.0080	0.0083	0.0098	0.0098
Jedit-4.0	AVG	**0.8983**	0.8940	0.8924	0.8914	0.8864	0.8845	0.8857	0.8844
	STD	0.0044	0.0080	0.0057	0.0045	0.0092	0.0098	0.0071	0.0120
Jedit-4.1	AVG	**0.8879**	0.8803	0.8809	0.8819	0.8761	0.8734	0.8736	0.8728
	STD	0.0060	0.0067	0.0052	0.0069	0.0071	0.0096	0.0104	0.0087
Jedit-4.2	AVG	0.9492	0.9491	0.9482	**0.9501**	0.9456	0.9463	0.9454	0.9469
	STD	0.0033	0.0027	0.0047	0.0037	0.0047	0.0047	0.0037	0.0046
Jedit-4.3	AVG	**0.9922**	0.9917	0.9912	0.9909	0.9891	0.9898	0.9893	0.9888
	STD	0.0020	0.0017	0.0026	0.0019	0.0021	0.0029	0.0028	0.0029
Log4j-1.0	AVG	0.9402	0.9404	**0.9408**	0.9358	0.9364	0.9297	0.9336	0.9347
	STD	0.0097	0.0094	0.0112	0.0070	0.0134	0.0099	0.0088	0.0112
Log4j-1.1	AVG	0.8195	0.8240	0.8303	0.8355	0.8406	0.8411	**0.8450**	0.8440
	STD	0.0130	0.0169	0.0170	0.0175	0.0210	0.0185	0.0103	0.0123
Lucene-2.0	AVG	0.7625	0.7477	0.7559	0.7560	0.7510	0.7467	**0.7700**	0.7585
	STD	0.0136	0.0144	0.0157	0.0248	0.0262	0.0143	0.0375	0.0318
Xalan-2.4	AVG	**0.8889**	0.8873	0.8867	0.8853	0.8820	0.8818	0.8808	0.8795
	STD	0.0043	0.0033	0.0050	0.0033	0.0063	0.0066	0.0050	0.0052
Xalan-2.5	AVG	0.7278	0.7265	0.7241	**0.7284**	0.7194	0.7180	0.7179	0.7161
	STD	0.0051	0.0061	0.0053	0.0076	0.0102	0.0071	0.0085	0.0058
Xalan-2.6	AVG	**0.8070**	0.8031	0.8011	0.8028	0.7978	0.7972	0.7973	0.7983
	STD	0.0069	0.0071	0.0070	0.0065	0.0072	0.0074	0.0095	0.0089
Rank (F-Test)		**1.75**	3.16	3.63	3.16	5.44	6.44	6.13	6.31

feature that can best describe the target classes, which benefits the learning process. In Table 12, the SBWOA1 again ranked first (rank of 2.41) in terms of the fitness value. All in all, we can conclude that the BWOA with an s-shaped transfer function S1 is the most appropriate for SFP problems.

5.6.2 Performance of enhanced WOA

After investigating the best TFs in the BWOA algorithm, we inspect the performance of the proposed SBEWOA (the enhanced EWOA with s-shaped TF S1) against SBWOA (the original WOA with s-shaped TF S1) in all datasets. Table 13 outlines the performance comparison of the SBEWOA and SBWOA in terms of the AUC rate, feature size, and fitness value. From Table 13, the proposed SBEWOA has achieved the optimal AUC rate and fitness for most of the datasets (15 datasets). As for the feature size, the SBEWOA can always select a minimal number of features during the selection process. In comparison with SBWOA, the SBEWOA is more capable of finding the positive features that can contribute to the highest AUC rate.

Table 11 Evaluation results of BWOA using 8 transfer functions in terms of number of selected features

Dataset	Measure	SBWOA1	SBWOA2	SBWOA3	SBWOA4	vBWOA1	vBWOA2	vBWOA3	vBWOA4
Ant-1.7	AVG	15.00	11.80	11.50	11.55	9.45	8.90	**8.60**	9.45
	STD	2.3170	2.3530	1.9331	2.1145	2.7810	2.6931	2.4581	3.2521
Camel-1.0	AVG	12.40	11.20	10.60	11.40	8.95	**8.15**	8.55	8.75
	STD	2.4149	1.9894	2.3261	1.9574	2.4165	2.3681	2.9105	2.6132
Camel-1.2	AVG	14.15	12.80	11.55	11.85	8.75	9.05	**7.95**	**7.95**
	STD	2.0844	2.3974	2.5021	1.7852	2.5105	2.7810	2.4165	2.7043
Camel-1.4	AVG	15.00	12.35	11.90	11.70	10.30	11.70	8.90	**8.25**
	STD	1.8918	1.8715	1.8890	2.1300	2.9576	2.5976	2.6338	1.9967
Camel-1.6	AVG	15.10	13.10	11.80	11.30	8.75	**7.80**	**7.80**	8.40
	STD	2.7701	2.4039	1.5424	2.2965	2.9890	2.8023	3.1722	3.2671
Jedit-3.2	AVG	13.25	12.15	10.45	10.90	8.15	7.00	**6.50**	7.55
	STD	2.7314	2.5603	2.0384	2.3373	2.7582	2.7145	2.7242	3.8179
Jedit-4.0	AVG	13.75	12.70	10.75	11.70	8.75	10.20	9.20	**8.15**
	STD	2.2213	1.9222	1.4824	2.3864	2.7886	2.3974	2.1176	3.2163
Jedit-4.1	AVG	15.20	12.65	10.65	10.85	8.20	**6.25**	6.95	6.60
	STD	3.4121	3.4224	2.5808	1.9270	3.5924	2.3592	2.3050	2.2572
Jedit-4.2	AVG	13.05	11.10	10.20	11.55	8.20	8.10	8.05	**7.40**
	STD	2.7999	2.7125	1.7351	1.9595	2.5047	2.4900	2.4165	2.1374
Jedit-4.3	AVG	12.85	11.35	10.10	10.95	7.85	7.75	8.40	**7.10**
	STD	2.6011	2.5603	1.6827	2.3278	2.0590	2.0743	2.5423	2.2919
Log4j-1.0	AVG	12.55	11.95	9.60	10.10	7.40	6.45	6.20	**5.75**
	STD	2.1145	1.7313	2.2100	1.8035	2.5423	1.9861	1.4364	1.8317
Log4j-1.1	AVG	10.60	8.00	7.85	7.95	3.10	3.45	3.25	**2.40**
	STD	3.7753	1.7472	2.0072	2.6453	2.5526	1.8489	2.0229	0.8208
Lucene-2.0	AVG	13.55	10.90	10.55	10.50	4.75	**3.50**	6.25	4.35
	STD	2.9820	3.0762	1.7313	1.8778	2.7506	2.2827	4.0246	2.5397
Xalan-2.4	AVG	13.90	12.10	10.60	12.25	8.95	8.85	8.80	**8.10**
	STD	2.4473	2.3373	2.0876	2.7314	2.7429	2.6011	2.5047	2.8451
Xalan-2.5	AVG	13.45	12.40	10.75	11.70	9.15	9.15	**8.20**	8.25
	STD	3.0345	2.3033	1.8883	1.6255	2.5603	2.9607	2.0926	2.4468
Xalan-2.6	AVG	11.35	10.25	11.00	9.95	7.70	**7.00**	7.90	7.35
	STD	2.9429	2.6730	2.0520	2.6651	2.2501	2.6754	2.3147	2.4767
Rank (F-Test)		8.00	6.75	5.31	5.91	3.38	2.53	2.31	**1.81**

The foremost cause for the improved efficiency of the SBEWOA is that it employs the multi-exploration strategies rather than the random operator in the global search phase. Thus, in case of immature convergence, the search agents can explore the untried areas and escape the local optimum. Besides, the SBEWOA enables the search agents to learn from the best three solutions, thereby increasing the chance of exploring the promising areas.

5.6.3 Population diversity analysis

In this sub-section, the diversity measurement has been considered to check the behaviour of the proposed method during the search. Maintaining the diversity during the search leads the optimization algorithm to escape the local optima and enhance the final solution. To measure the diversity capability, the hamming distance between the population members in each iteration of the proposed method (i.e., SBEWOA) as well as the original method (i.e., SBWOA) are recorded and plotted as shown in Fig. 8. As can be seen from the Figure, the enhanced version of BWOA (i.e., SBEWOA) are able to preserve a good diversity during the search where the hamming distance keep converging the search to the optimal results. On the other hand, the SBWOA has almost stable hamming distance results during the search and converge slowly. From Fig. 8, one can see that the SBEWOA has maintained a good diversity for all datasets with smaller Hamming distance. At the initial stage,

Table 12 Evaluation results of BWOA using 8 transfer functions in terms of fitness values

Dataset	Measure	SBWOA1	SBWOA2	SBWOA3	SBWOA4	VBWOA1	VBWOA2	VBWOA3	VBWOA4
Ant-1.7	AVG	**0.1273**	0.1278	0.1291	0.1281	0.1313	0.1327	0.1312	0.1306
	STD	0.0037	0.0038	0.0040	0.0033	0.0056	0.0052	0.0048	0.0037
Camel-1.0	AVG	0.0173	0.0178	0.0173	0.0180	0.0173	**0.0170**	0.0183	0.0196
	STD	0.0017	0.0015	0.0024	0.0018	0.0027	0.0030	0.0022	0.0025
Camel-1.2	AVG	**0.2654**	0.2693	0.2678	0.2654	0.2762	0.2747	0.2739	0.2725
	STD	0.0080	0.0068	0.0051	0.0095	0.0092	0.0119	0.0126	0.0097
Camel-1.4	AVG	**0.1123**	0.1133	0.1142	0.1129	0.1183	0.1190	0.1177	0.1207
	STD	0.0030	0.0033	0.0044	0.0054	0.0065	0.0045	0.0058	0.0048
Camel-1.6	AVG	**0.1662**	0.1674	0.1703	0.1697	0.1679	0.1702	0.1718	0.1726
	STD	0.0045	0.0043	0.0068	0.0052	0.0066	0.0070	0.0074	0.0076
Jedit-3.2	AVG	0.0997	0.1008	0.1014	**0.0996**	0.1054	0.1089	0.1099	0.1071
	STD	0.0048	0.0059	0.0079	0.0086	0.0078	0.0078	0.0086	0.0096
Jedit-4.0	AVG	**0.1075**	0.1113	0.1119	0.1133	0.1168	0.1195	0.1177	0.1185
	STD	0.0044	0.0076	0.0057	0.0045	0.0092	0.0093	0.0071	0.0113
Jedit-4.1	AVG	**0.1186**	0.1248	0.1232	0.1224	0.1268	0.1285	0.1286	0.1292
	STD	0.0050	0.0063	0.0049	0.0068	0.0073	0.0093	0.0104	0.0084
Jedit-4.2	AVG	0.0568	0.0559	0.0564	**0.0552**	0.0580	0.0573	0.0581	0.0563
	STD	0.0036	0.0025	0.0043	0.0036	0.0045	0.0050	0.0036	0.0047
Jedit-4.3	AVG	0.0141	0.0139	**0.0138**	0.0145	0.0147	0.0140	0.0148	0.0146
	STD	0.0021	0.0016	0.0023	0.0018	0.0022	0.0025	0.0028	0.0025
Log4j-1.0	AVG	0.0655	0.0650	**0.0634**	0.0687	0.0667	0.0728	0.0688	0.0675
	STD	0.0093	0.0095	0.0110	0.0068	0.0130	0.0099	0.0088	0.0109
Log4j-1.1	AVG	0.1840	0.1782	0.1720	0.1668	0.1594	0.1591	**0.1551**	0.1556
	STD	0.0129	0.0170	0.0166	0.0171	0.0204	0.0184	0.0104	0.0123
Lucene-2.0	AVG	0.2419	0.2553	0.2469	0.2468	0.2488	0.2525	**0.2309**	0.2413
	STD	0.0138	0.0141	0.0155	0.0246	0.0258	0.0139	0.0378	0.0317
Xalan-2.4	AVG	**0.1170**	0.1176	0.1175	0.1197	0.1213	0.1215	0.1224	0.1233
	STD	0.0041	0.0036	0.0052	0.0038	0.0062	0.0062	0.0046	0.0044
Xalan-2.5	AVG	0.2762	0.2770	0.2785	**0.2747**	0.2823	0.2838	0.2834	0.2852
	STD	0.0049	0.0062	0.0055	0.0074	0.0098	0.0064	0.0082	0.0056
Xalan-2.6	AVG	**0.1967**	0.2001	0.2024	0.2002	0.2041	0.2043	0.2046	0.2034
	STD	0.0072	0.0066	0.0071	0.0067	0.0069	0.0075	0.0091	0.0090
Rank (F-Test)		**2.41**	3.38	3.56	3.28	5.38	6.00	6.13	5.88

the SBEWOA keeps exploring the search regions to find the global minimum. Then, the SBEWOA searches locally for the local optimum. Even though SBEWOA was less robust compared to SBWOA, but it can offer better solution that provide high AUC performance.

5.6.4 Deep analysis on the modifications

This sub-section conducts a deep analysis to study the impacts of the modifications on the proposed method. Table 14 presents the findings of proposed modifications in terms of fitness values. In Table 14, the SBWOA-G denotes the modification using GWO algorithm (19), SBWOA-H refers to the modification using HHO algorithm (20), and the SBEWOA is the modification using both GWO and HHO algorithms. Based on the results obtained, the SBEWOA has achieved the lowest fitness in most datasets. The convergence analysis in Figs. 9 and 10 again verify the convergence power of the SBEWOA in finding the near optimal feature subset. By observing the AUC performance in Table 15, it shows that SBEWOA can usually score the highest AUC values, followed by SBWOA-H.

Table 13 Comparison of SBEWOA versus the conventional SBWOA in terms of AUC rates, number of selected features, and fitness values

Dataset	Measure	AUC	No. of features	Fitness			
		SBWOA	SBEWOA	SBWOA	SBEWOA	SBWOA	SBEWOA
Ant-1.7	AVG	0.8790	**0.8806**	15.00	**13.55**	0.12728	**0.12497**
	STD	0.0042	0.0036	2.3170	2.5849	0.00370	0.00314
Camel-1.0	AVG	0.9888	**0.9892**	12.40	**12.35**	0.01729	**0.01682**
	STD	0.0021	0.0027	2.4149	1.8994	0.00171	0.00268
Camel-1.2	AVG	0.7391	**0.7437**	**14.15**	14.40	0.26541	**0.26091**
	STD	0.0080	0.0054	2.0844	1.7889	0.00799	0.00535
Camel-1.4	AVG	0.8941	**0.8951**	15.00	**14.00**	0.11234	**0.11080**
	STD	0.0032	0.0050	1.8918	2.7910	0.00299	0.00438
Camel-1.6	AVG	0.8398	**0.8418**	15.10	**14.90**	0.16616	**0.16407**
	STD	0.0041	0.0035	2.7701	3.3071	0.00446	0.00426
Jedit-3.2	AVG	0.9060	**0.9115**	**13.25**	13.35	0.09970	**0.09424**
	STD	0.0046	0.0061	2.7314	2.9429	0.00477	0.00603
Jedit-4.0	AVG	0.8983	**0.9043**	13.75	**13.65**	0.10753	**0.10154**
	STD	0.0044	0.0087	2.2213	2.8335	0.00444	0.00903
Jedit-4.1	AVG	0.8879	**0.8914**	**15.20**	16.60	0.11862	**0.11583**
	STD	0.0060	0.0069	3.4121	2.0365	0.00496	0.00652
Jedit-4.2	AVG	0.9492	**0.9512**	13.05	**12.80**	0.05680	**0.05471**
	STD	0.0033	0.0036	2.7999	2.9308	0.00365	0.00300
Jedit-4.3	AVG	0.9922	**0.9926**	12.85	**12.45**	0.01414	**0.01360**
	STD	0.0020	0.0017	2.6011	2.9105	0.00206	0.00157
Log4j-1.0	AVG	0.9402	**0.9493**	12.55	**12.35**	0.06547	**0.05640**
	STD	0.0097	0.0112	2.1145	2.3458	0.00934	0.01093
Log4j-1.1	AVG	0.8195	**0.8395**	10.60	**9.55**	0.18398	**0.16363**
	STD	0.0130	0.0145	3.7753	2.9996	0.01290	0.01455
Lucene-2.0	AVG	0.7625	**0.7702**	**13.55**	15.05	0.24190	**0.23500**
	STD	0.0136	0.0143	2.9820	2.9643	0.01379	0.01515
Xalan-2.4	AVG	**0.8889**	0.8886	**13.90**	14.90	**0.11696**	0.11773
	STD	0.0043	0.0020	2.4473	2.0749	0.00406	0.00261
Xalan-2.5	AVG	0.7278	**0.7334**	**13.45**	15.10	0.27616	**0.27151**
	STD	0.0051	0.0069	3.0345	2.3147	0.00494	0.00707
Xalan-2.6	AVG	**0.8070**	0.8058	**11.35**	13.05	**0.19671**	0.19878
	STD	0.0069	0.0039	2.9429	2.6253	0.00721	0.00382

On the other side, the proposed SBEWOA did not show the best results in feature reduction. From Table 16, the number of features chosen by SBEWOA was slightly higher than BGWO. However, the feature subsets produced by SBEWOA were able to maintain high AUC results.

5.6.5 Comparison of SBEWOA with other optimizors

In this sub-section, we further compare the performance of the proposed SBEWOA with the other 9 state-of-the-art feature selection methods. These comparison methods are binary firefly algorithm (BFFA), binary moth flame optimization (BMFO), binary multiverse optimizer (BMVO), binary grey wolf optimization (BGWO), binary bat algorithm (BBAT), binary cuckoo search (BCS), binary harris hawk optimization (BHHO), binary Jaya algorithm (BJAYA), and genetic algorithm (GA). In this study, we adjusted the optimizers' parameters in accordance with the settings recommended in preliminary publications and related studies on feature selection [99, 113–116]. The details of parameter configurations used in this paper are outlined in Table 17.

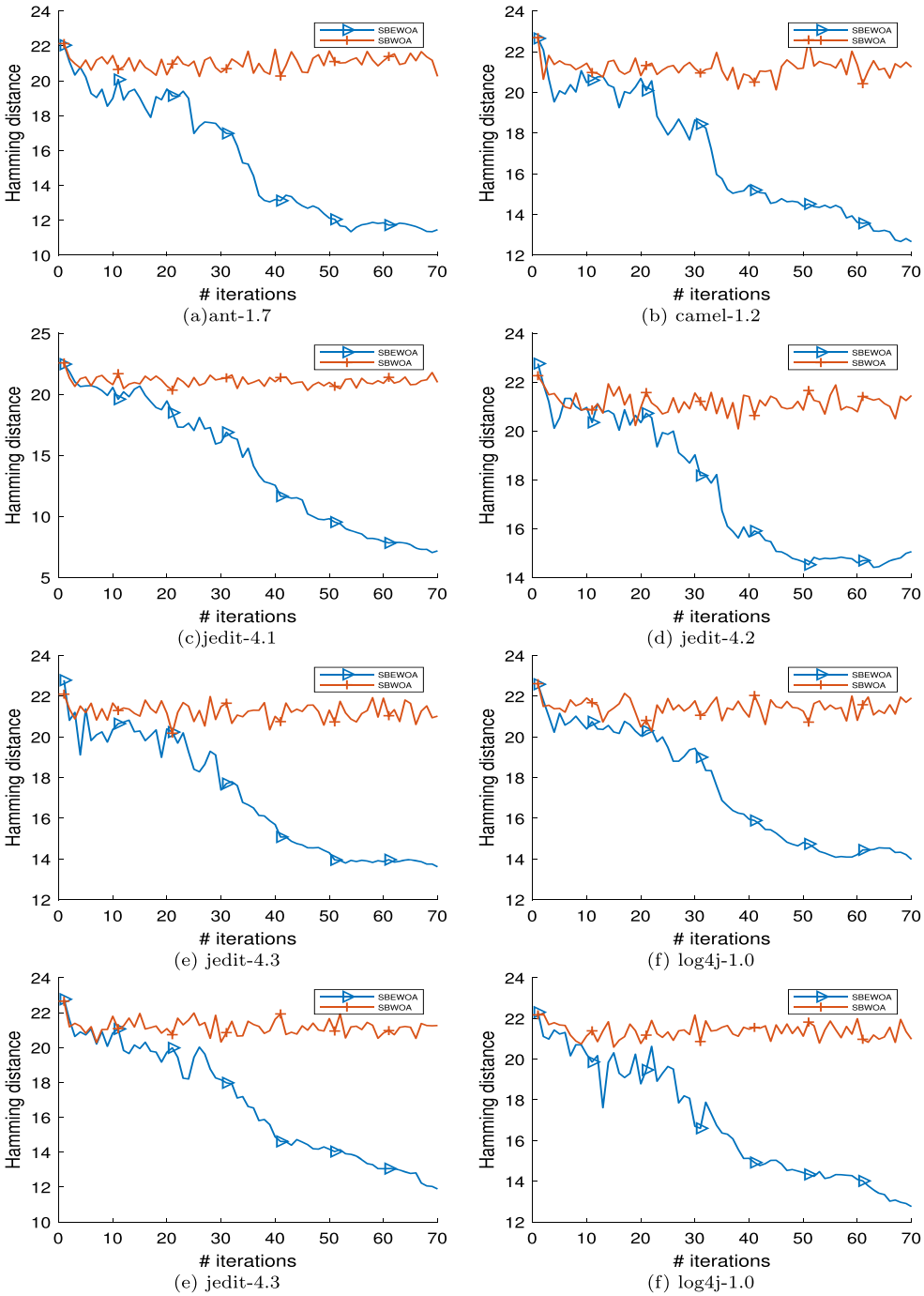

Fig. 8 The population diversity curves of BWOA and BEWOA

Table 18 presents the AUC rates of the SBEWOA and other methods. As can be seen, the SBEWOA overwhelmed its competitors by scoring the highest AUC rates in 12 datasets, with the minimum best ranking of 1.47. On the other side, the second-best algorithm was BHHO (rank of 2.41), followed by the BMFO method (rank of 3.59). When observing the result of the Wilcoxon test in Table 19, one can see that the performance of the SBEWOA was significantly better than other methods for most of the datasets.

Table 20 reports the number of selected features for SBEWOA and other methods. Meanwhile, the results of the Wilcoxon test based on the number of selected feature is shown in Table 21. As can be observed, the BBAT

Table 14 Impact of proposed modifications in terms of fitness results

Dataset	Measure	BGWO	BHHO	SBWOA-G	SBWOA-H	SBEWOA
Ant-1.7	AVG	0.12885	0.12559	**0.12414**	0.12485	0.12497
	STD	0.00349	0.00361	0.00483	0.00351	0.00314
Camel-1.0	AVG	0.01678	0.01624	0.01622	**0.01537**	0.01682
	STD	0.00229	0.00189	0.00219	0.00154	0.00268
Camel-1.2	AVG	0.26320	0.26525	0.26399	**0.26005**	0.26091
	STD	0.01113	0.00866	0.00667	0.00780	0.00535
Camel-1.4	AVG	0.11229	0.11256	0.11193	0.11099	**0.11080**
	STD	0.00630	0.00448	0.00300	0.00321	0.00438
Camel-1.6	AVG	0.16788	0.16628	0.16090	**0.15932**	0.16407
	STD	0.00519	0.00542	0.00515	0.00407	0.00426
Jedit-3.2	AVG	0.09908	0.09738	0.10034	0.09602	**0.09424**
	STD	0.01129	0.00983	0.00421	0.00535	0.00603
Jedit-4.0	AVG	0.10716	0.10360	0.10624	0.10519	**0.10154**
	STD	0.00593	0.00530	0.00526	0.00371	0.00903
Jedit-4.1	AVG	0.11975	0.11831	0.11644	**0.11387**	0.11583
	STD	0.00786	0.00673	0.00686	0.00484	0.00652
Jedit-4.2	AVG	0.05583	**0.05392**	0.05511	0.05460	0.05471
	STD	0.00363	0.00416	0.00314	0.00251	0.00300
Jedit-4.3	AVG	0.01391	**0.01324**	0.01376	0.01330	0.01360
	STD	0.00215	0.00173	0.00134	0.00121	0.00157
Log4j-1.0	AVG	0.06493	0.05980	0.05952	**0.05607**	0.05640
	STD	0.00874	0.01162	0.01248	0.00729	0.01093
Log4j-1.1	AVG	0.17868	0.17123	0.16538	0.16883	**0.16363**
	STD	0.01673	0.01999	0.02794	0.01370	0.01455
Lucene-2.0	AVG	0.24637	0.23872	0.24458	0.24538	**0.23500**
	STD	0.01165	0.01601	0.02020	0.01758	0.01515
Xalan-2.4	AVG	0.11875	**0.11562**	0.11607	0.11628	0.11773
	STD	0.00309	0.00426	0.00367	0.00435	0.00261
Xalan-2.5	AVG	0.27629	0.27433	0.27325	0.27282	**0.27151**
	STD	0.00401	0.00714	0.00639	0.00673	0.00707
Xalan-2.6	AVG	0.20064	**0.19731**	0.20133	0.20006	0.19878
	STD	0.00711	0.00633	0.00704	0.00531	0.00382
Rank (F-Test)		4.63	3.00	3.13	2.06	2.19

outperformed other methods in minimal feature selection, while the SBEWOA ranked 8th with the F-Test rank of 7.72. Our findings reveal that the SBEWOA was not very good at reducing the feature numbers. However, it is worth noting that an algorithm that worked perfectly in feature reduction might be degrading the performance of the classifier due to extensive elimination of the positive features (refer Tables 18 and 20). Accordingly, the SBEWOA is excellent at maintaining the positive features while removing the negative features. Hence, the SBEWOA can often offer better AUC rates when dealing with SFP problems.

Table 22 shows the fitness values of the SBEWOA and other methods. Judging the Table 22, the SBEWOA contributed to the best fitness values in 11 datasets. Besides, the SBEWOA can often offer consistent results due to a smaller standard deviation. As compared to other methods, the SBEWOA has retained the best rank of 1.69, which exhibited a better search tendency when dealing with FS problems in SFP analysis. The results of Wilcoxon test in Table 23 support the arguments. On the one hand, Table 24 shows the results of the computational time among the proposed SBEWOA and other optimizers. Based on the results obtained, GA was the fastest algorithm in terms of running time. Although SBEWOA is not the fastest algorithm, it can be computed faster than BGWO, BBAT, BCS, and BJaya in many cases. This argument is further supported by the statistical results shown in Table 25.

Figures 11 and 12 illustrate the convergence curves of the SBEWOA for all datasets. It is seen that the SBEWOA showed an excellent acceleration rate in most cases. Taking jedit-3.2 and lucene-2.0 datasets as examples, we can observe that the SBEWOA converged faster and deeper to

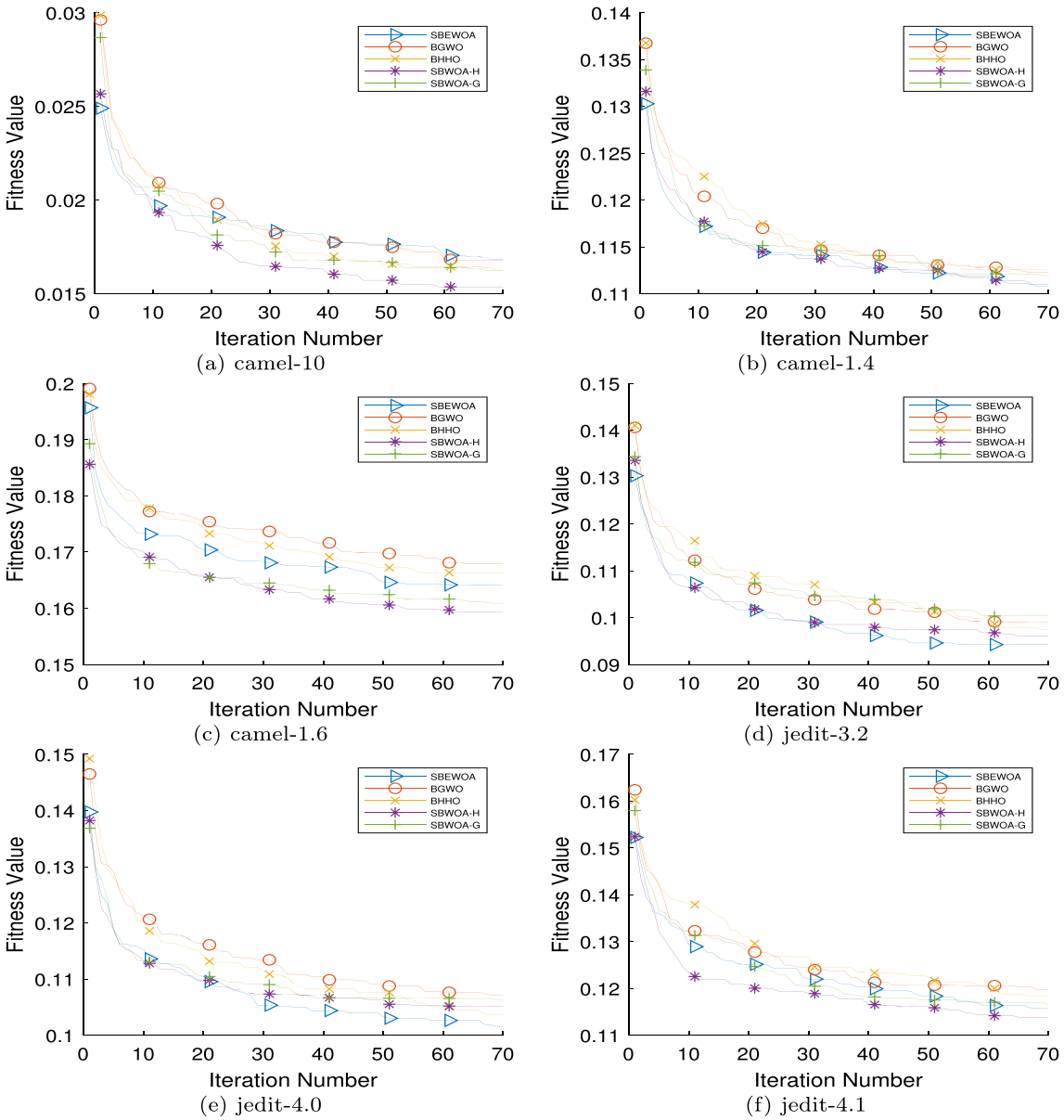

Fig. 9 Convergence analysis of basic and modified versions on camel-10, camel-1.4, camel-1.6, jedit-3.2, jedit-4.0, and jedit-4.1 datasets

explore the global minimum solution. The results imply that SBEWOA giving good convergence ability in solving the FS problem, thus leading to satisfactory achievements.

Based on the results obtained, it can be inferred that the proposed SBEWOA is a powerful and useful FS tool for SFP analysis. The superior performance of the SBEWOA can be attributed to the multi-exploration strategies which take full advantage of the best three leaders in exploring the feature spaces. Moreover, the utilization of the average of all positions and the current best position allows the search agents to explore the untried regions effectively. In case of premature convergence, the search agents can escape the local optimum and seek out promising solutions. Hence, SBEWOA can usually achieve better results than other methods.

5.6.6 Relevant features selected by SBEWOA

In the final sub-section, we are interested to investigate the top features of SFP analysis. Table 26 depicts the details of features selected by SBEWOA that scored the best AUC results. From Table 26, it is noticed that different set of features has been chosen by SBEWOA on different datasets. On the one hand, Table 27 outlines the number of times each feature has been selected by the SBEWOA algorithm. The importance of features is demonstrated in Fig. 13. Accordingly, the top five features were ca (80.938%), dit (77.188%), moa (75.938%), max_cc (75.625%), and mfa (75%). On the contrary, the worst features are found to be wmc and lcom (54.063%).

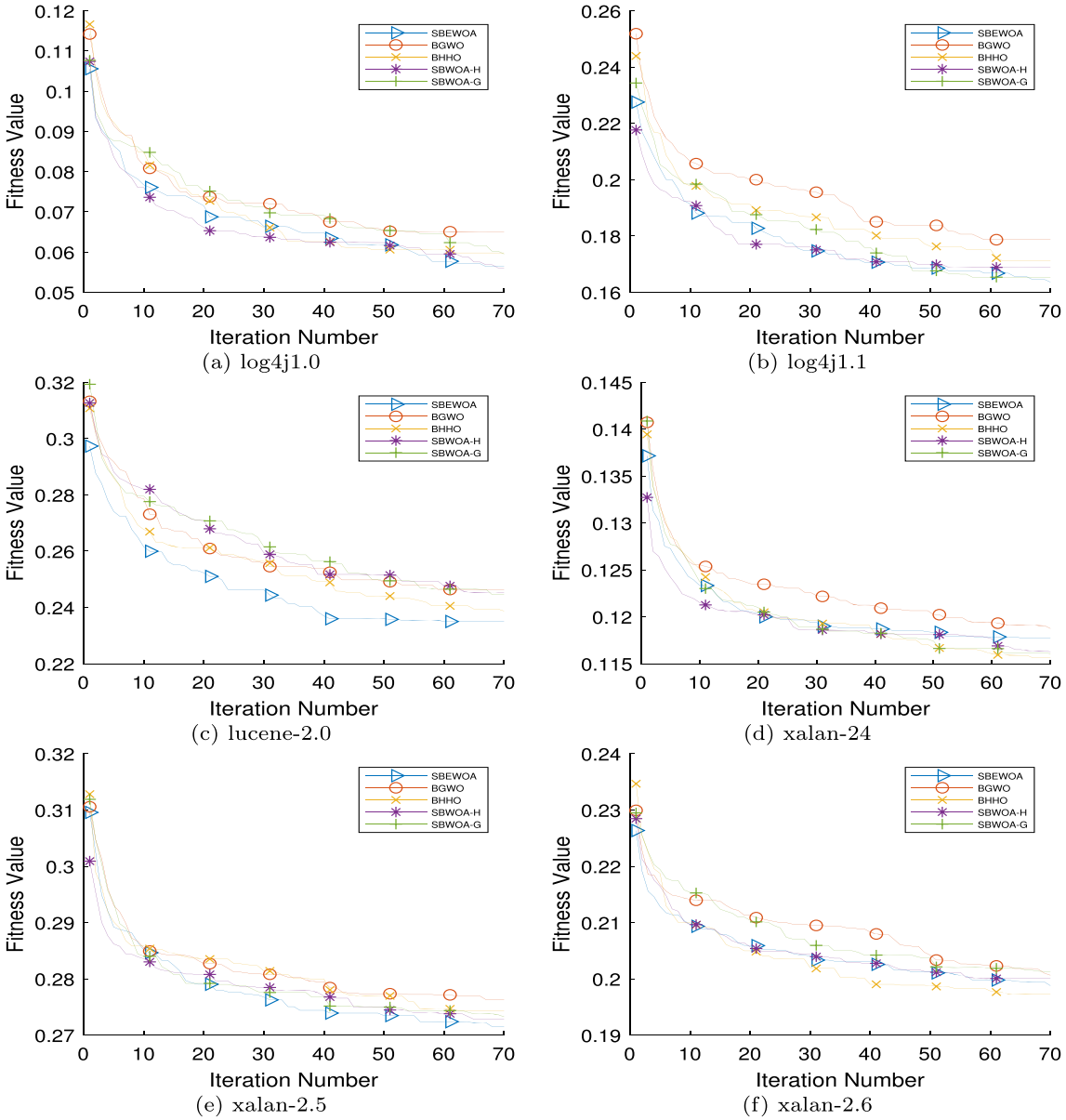

Fig. 10 Convergence analysis of basic and modified versions on log4j1.0, log4j1.1, lucene-2.0, xalan-24, xalan-2.5 , and xalan-2.6 datasets

Table 28 shows the details of the features chosen for each dataset. In ant-1.7 dataset, the most relevant features was cbo, followed by cam. As for xalan-2.6 dataset, the mos feature was frequently selected by the algorithm. Across all datasets, it is seen that the ca features have been selected with 259 times, while the second-best feature, dit, has been selected with 247 times. Our findings suggest that these features are having high discriminative power when dealing with the SFP problem.

6 Conclusion and future works

Software Fault Prediction (SFP) helps developers in identifying the faulty components of the software prior to system deployment. In this article, we developed a well-performing classification model that is able to predict faulty software components. The 16 software project datasets were selected from the PROMISE repository. The datasets are then normalized to set a proper scale for all data.

Table 15 Impact of proposed modifications in terms of AUC results

Dataset	Measure	BGWO	BHHO	SBWOA-G	SBWOA-H	SBEWOA
Ant-1.7	AVG	0.8765	0.8805	**0.8816**	0.8814	0.8806
	STD	0.0036	0.0038	0.0049	0.0034	0.0036
Camel-1.0	AVG	0.9893	0.9899	0.9897	**0.9904**	0.9892
	STD	0.0024	0.0016	0.0025	0.0011	0.0027
Camel-1.2	AVG	0.7410	0.7391	0.7407	**0.7444**	0.7437
	STD	0.0108	0.0089	0.0068	0.0076	0.0054
Camel-1.4	AVG	0.8938	0.8935	0.8944	**0.8952**	0.8951
	STD	0.0065	0.0050	0.0029	0.0029	0.0050
Camel-1.6	AVG	0.8374	0.8396	0.8446	**0.8464**	0.8418
	STD	0.0052	0.0053	0.0049	0.0040	0.0035
Jedit-3.2	AVG	0.9063	0.9085	0.9053	0.9094	**0.9115**
	STD	0.0116	0.0101	0.0048	0.0056	0.0061
Jedit-4.0	AVG	0.8983	0.9027	0.9003	0.9010	**0.9043**
	STD	0.0062	0.0053	0.0054	0.0033	0.0087
Jedit-4.1	AVG	0.8867	0.8880	0.8896	**0.8935**	0.8914
	STD	0.0086	0.0069	0.0074	0.0042	0.0069
Jedit-4.2	AVG	0.9499	**0.9520**	0.9508	0.9514	0.9512
	STD	0.0036	0.0036	0.0033	0.0033	0.0036
Jedit-4.3	AVG	0.9918	0.9923	0.9922	**0.9928**	0.9926
	STD	0.0025	0.0020	0.0017	0.0014	0.0017
Log4j-1.0	AVG	0.9399	0.9455	0.9456	0.9492	**0.9493**
	STD	0.0095	0.0121	0.0131	0.0075	0.0112
Log4j-1.1	AVG	0.8242	0.8312	0.8375	0.8334	**0.8395**
	STD	0.0170	0.0203	0.0281	0.0134	0.0145
Lucene-2.0	AVG	0.7580	0.7657	0.7601	0.7589	**0.7702**
	STD	0.0117	0.0157	0.0204	0.0182	0.0143
Xalan-2.4	AVG	0.8870	**0.8903**	0.8895	0.8892	0.8886
	STD	0.0025	0.0043	0.0042	0.0043	0.0020
Xalan-2.5	AVG	0.7281	0.7297	0.7310	0.7316	**0.7334**
	STD	0.0039	0.0074	0.0066	0.0066	0.0069
Xalan-2.6	AVG	0.8034	**0.8068**	0.8033	0.8042	0.8058
	STD	0.0073	0.0057	0.0069	0.0048	0.0039
Rank (F-Test)		4.63	3.06	3.19	2.00	2.13

Firstly, the imbalance problem in the dataset is resolved by applying and comparing several SMOTE techniques to get more accurate results. The results revealed that the BorderlineSMOTE technique offered the optimal AUC rate as it can properly re-balance the datasets.

The hyper-parameter settings of the Random Forest (RF) classifier are investigated, in which the parameters that provide the optimal results are selected. Additionally, The RF classifier is compared against other traditional classifiers. In most cases, it obtains superior outcomes. Furthermore, eight transfer functions (TFs) were adopted to convert the original WOA into binary search space. Based on the result obtained, BWOA with S-shaped transfer function S1 is the most appropriate for SFP problems. Moreover, the performance of BWOA is improved by integrating the exploration strategies from GWO and HHO algorithms. This newly proposed method is called SBEWOA. The main rationale of this enhancement is to improve the exploration stage of the WOA by employing the survival-of-the-fittest principle rather than random search. This modification enables the proposed SBEWOA to select a set of positive features from the complex dataset.

Table 16 Impact of proposed modifications in terms of number of features

Dataset	Measure	BGWO	BHHO	SBWOA-G	SBWOA-H	SBEWOA
Ant-1.7	AVG	**13.20**	14.60	13.80	14.80	13.55
	STD	2.2148	1.9304	1.9358	2.2148	2.5849
Camel-1.0	AVG	12.45	12.50	12.10	**11.70**	12.35
	STD	2.0384	2.4387	2.1250	2.9037	1.8994
Camel-1.2	AVG	**13.55**	13.95	14.50	14.00	14.40
	STD	2.2118	2.3278	1.6059	2.3396	1.7889
Camel-1.4	AVG	14.30	14.30	14.70	14.40	**14.00**
	STD	2.5567	2.3193	1.6890	2.4366	2.7910
Camel-1.6	AVG	**13.75**	14.90	14.10	14.55	14.90
	STD	2.5314	2.3598	2.5319	1.6694	3.3071
Jedit-3.2	AVG	12.70	13.65	13.25	**12.60**	13.35
	STD	2.3193	2.4979	3.3067	3.1187	2.9429
Jedit-4.0	AVG	**13.00**	14.50	15.05	14.30	13.65
	STD	2.2005	2.0391	2.5438	2.1788	2.8335
Jedit-4.1	AVG	15.20	14.85	**14.25**	16.80	16.60
	STD	3.0366	3.0997	3.4925	2.3974	2.0365
Jedit-4.2	AVG	**12.40**	12.85	12.90	12.95	12.80
	STD	2.7985	2.7582	2.1740	2.3946	2.9308
Jedit-4.3	AVG	11.65	**11.20**	12.15	12.30	12.45
	STD	2.4121	2.4623	3.0483	2.3642	2.9105
Log4j-1.0	AVG	**10.85**	11.60	11.30	11.60	12.35
	STD	2.9069	2.6636	2.7739	2.4149	2.3458
Log4j-1.1	AVG	9.25	8.15	9.00	**7.75**	9.55
	STD	3.0240	2.7004	2.2243	2.2682	2.9996
Lucene-2.0	AVG	13.55	13.55	14.10	**13.30**	15.05
	STD	2.9105	3.0689	3.0245	2.4730	2.9643
Xalan-2.4	AVG	13.70	14.10	13.25	**13.10**	14.90
	STD	2.8303	2.0749	2.0995	2.6931	2.0749
Xalan-2.5	AVG	14.30	**13.55**	13.80	14.10	15.10
	STD	2.2501	2.5021	2.6477	2.1001	2.3147
Xalan-2.6	AVG	**12.05**	12.10	13.15	12.40	13.05
	STD	2.9285	3.3388	2.6011	2.6036	2.6253
Rank (F-Test)		2.13	3.00	3.19	2.91	3.78

Lastly, the proposed SBEWOA is compared against 9 state-of-the-art feature selection methods. The results show that the proposed SBEWOA is a powerful and useful FS tool for the SFP problem. The superior performance of the SBEWOA can be attributed to the multi-exploration strategies that help in exploring the features space. Among the rivals, the SBEWOA not only gives the highest AUC score but also the minimum number of features. The proposed multi-stage approach helps in producing a fruitful solution to tackling SFP problems.

There are several limitations in this work. First, the proposed SBEWOA suffers from low feature reduction power, but it can often attain high accuracy. Second, the proposed SBEWOA is highly complex in its structure, thus resulting in high computation cost. In the future, the SBEWOA can be applied to other applications such as parameter estimation photovoltaic solar cells and intrusion detection system. Furthermore, powerful mechanisms such as chaotic map and opposite-based learning can be integrated into SBEWOA for performance enhancement.

Table 17 Parameter settings for the optimization algorithms

Algorithm	parameter	value
Common parameters		
	Population size	10
	Maximum No. of iterations	70
	No. of runs	20
	Dimension	#features
	Fitness function	alpha=0.99 , Beta=0.01
Internal parameters		
BHHO	Convergence parameter E	decreased linearly from 2 to 0
BWOA	convergence constant a	decreased linearly from 2 to 0
	Spiral factor b	1
BBAT	Qmin Frequency minimum	0
	Qmax Frequency maximum	2
	loudness A	0.5
	Pulse rate r	0.5
BGWO	parameter a	decreased linearly from 2 to 0
BFFA	Gamma Absorption coefficient	1
	Alpha (randomness)	0.5
	Beta min (initial attractiveness)	0.2
BMFO	Spiral factor b	1
	convergence constant	decreased linearly from -1 to -2
BMVO	wormhole existence probability (WEP)	increased linearly from 0.2 to 1
	travelling distance rate (TDR)	decreased linearly from 0.6 to 0
BCS	Discovery rate of alien solutions Pa	0.25
GA	crossover Probability	0.8
	mutation Probability	0.001
	selection operator	Roulette Wheel

Table 18 Comparison between the proposed SBEWOA and other optimizers based on AUC rates

Dataset	Measure	SBEWOA	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	AVG	**0.8806**	0.8764	0.8803	0.8770	0.8765	0.8134	0.8773	0.8805	0.8802	0.8668
	STD	0.0036	0.0037	0.0041	0.0039	0.0036	0.0204	0.0033	0.0038	0.0029	0.0066
Camel-1.0	AVG	0.9892	0.9878	0.9891	0.9883	0.9893	0.9409	0.9878	**0.9899**	0.9886	0.9815
	STD	0.0027	0.0023	0.0021	0.0023	0.0024	0.0303	0.0021	0.0016	0.0022	0.0046
Camel-1.2	AVG	**0.7437**	0.7347	0.7427	0.7378	0.7410	0.6429	0.7342	0.7391	0.7411	0.7189
	STD	0.0054	0.0052	0.0079	0.0077	0.0108	0.0251	0.0091	0.0089	0.0065	0.0143
Camel-1.4	AVG	**0.8951**	0.8908	0.8932	0.8903	0.8938	0.8298	0.8880	0.8935	0.8944	0.8771
	STD	0.0050	0.0038	0.0024	0.0032	0.0065	0.0225	0.0034	0.0050	0.0044	0.0059
Camel-1.6	AVG	**0.8418**	0.8355	0.8397	0.8394	0.8374	0.7500	0.8318	0.8396	0.8399	0.8196
	STD	0.0035	0.0061	0.0037	0.0076	0.0052	0.0331	0.0045	0.0053	0.0039	0.0095
Jedit-3.2	AVG	**0.9115**	0.9002	0.9065	0.9054	0.9063	0.7994	0.9051	0.9085	0.9073	0.8872
	STD	0.0061	0.0077	0.0084	0.0084	0.0116	0.0354	0.0089	0.0101	0.0068	0.0124
Jedit-4.0	AVG	**0.9043**	0.8951	0.8997	0.8970	0.8983	0.7887	0.8967	0.9027	0.8993	0.8723
	STD	0.0087	0.0087	0.0106	0.0063	0.0062	0.0509	0.0089	0.0053	0.0060	0.0189
Jedit-4.1	AVG	**0.8914**	0.8801	0.8880	0.8811	0.8867	0.8032	0.8804	0.8880	0.8876	0.8657
	STD	0.0069	0.0093	0.0067	0.0082	0.0086	0.0218	0.0071	0.0069	0.0092	0.0129
Jedit-4.2	AVG	0.9512	0.9475	0.9506	0.9497	0.9499	0.8954	0.9489	**0.9520**	0.9512	0.9400
	STD	0.0036	0.0018	0.0030	0.0038	0.0036	0.0235	0.0030	0.0036	0.0036	0.0052

Table 18 (continued)

Dataset	Measure	SBEWOA	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Jedit-4.3	AVG	**0.9926**	0.9907	0.9925	0.9911	0.9918	0.9608	0.9909	0.9923	0.9918	0.9870
	STD	0.0017	0.0022	0.0022	0.0016	0.0025	0.0155	0.0019	0.0020	0.0020	0.0029
Log4j-1.0	AVG	**0.9493**	0.9349	0.9425	0.9433	0.9399	0.8394	0.9402	0.9455	0.9410	0.9126
	STD	0.0112	0.0112	0.0120	0.0132	0.0095	0.0372	0.0107	0.0121	0.0102	0.0130
Log4j-1.1	AVG	**0.8395**	0.8248	0.8188	0.8228	0.8242	0.6837	0.8324	0.8312	0.8186	0.7922
	STD	0.0145	0.0170	0.0174	0.0173	0.0170	0.0506	0.0195	0.0203	0.0160	0.0262
Lucene-2.0	AVG	**0.7702**	0.7532	0.7583	0.7557	0.7580	0.6144	0.7596	0.7657	0.7565	0.7137
	STD	0.0143	0.0128	0.0106	0.0150	0.0117	0.0493	0.0166	0.0157	0.0163	0.0226
Xalan-2.4	AVG	0.8886	0.8877	0.8890	0.8883	0.8870	0.8227	0.8861	**0.8903**	0.8884	0.8715
	STD	0.0020	0.0035	0.0038	0.0043	0.0025	0.0253	0.0037	0.0043	0.0033	0.0059
Xalan-2.5	AVG	**0.7334**	0.7232	0.7316	0.7282	0.7281	0.6406	0.7282	0.7297	0.7292	0.7098
	STD	0.0069	0.0087	0.0063	0.0071	0.0039	0.0224	0.0072	0.0074	0.0046	0.0091
Xalan-2.6	AVG	0.8058	0.8014	0.8027	0.8062	0.8034	0.7284	0.8021	**0.8068**	0.8034	0.7833
	STD	0.0039	0.0082	0.0041	0.0076	0.0073	0.0154	0.0058	0.0057	0.0067	0.0096
Rank (F-Test)		**1.47**	7.34	3.59	5.53	5.13	10.00	6.44	2.41	4.09	9.00

Table 19 2-tailed P-values of the Wilcoxon signed ranks test based on AUC results reported in Table 18 (P-values $\leq$ 0.05 are in bold and significant)

Dataset	SBEWOA (as a best performing method) vs								
	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	**1.46E-03**	6.64E-01	**4.27E-03**	**1.47E-03**	**6.66E-08**	**2.65E-03**	7.34E-01	5.68E-01	**2.89E-07**
Camel-1.0	1.09E-01	7.79E-01	2.70E-01	9.18E-01	**5.50E-08**	7.58E-02	4.48E-01	3.83E-01	**1.65E-06**
Camel-1.2	**3.28E-05**	5.25E-01	**4.49E-03**	5.13E-02	**6.78E-08**	**7.19E-04**	5.13E-02	1.55E-01	**2.38E-07**
Camel-1.4	**2.65E-03**	1.84E-01	**1.77E-04**	1.89E-01	**6.66E-08**	**1.15E-05**	1.55E-01	3.79E-01	**7.77E-08**
Camel-1.6	**1.63E-04**	2.81E-01	1.79E-01	**6.34E-03**	**6.26E-08**	**2.21E-06**	3.00E-01	4.98E-01	**6.25E-08**
Jedit-3.2	**5.26E-05**	**2.26E-02**	**1.45E-02**	**2.25E-02**	**6.52E-08**	**1.13E-03**	1.25E-01	**1.44E-02**	**2.21E-06**
Jedit-4.0	**1.83E-03**	**1.07E-02**	**2.27E-03**	**1.24E-02**	**6.53E-08**	**6.16E-03**	8.69E-01	**4.59E-02**	**2.77E-06**
Jedit-4.1	**4.27E-04**	9.54E-02	**2.65E-04**	1.00E-01	**5.84E-08**	**1.47E-04**	1.57E-01	1.93E-01	**5.63E-07**
Jedit-4.2	**3.82E-04**	4.21E-01	8.59E-02	1.55E-01	**6.43E-08**	**3.32E-02**	6.43E-01	7.85E-01	**1.09E-06**
Jedit-4.3	**1.41E-03**	7.35E-01	**2.26E-03**	1.73E-01	**5.78E-08**	**1.51E-03**	4.83E-01	1.59E-01	**1.93E-06**
Log4j-1.0	**2.21E-04**	5.10E-02	1.18E-01	**3.68E-04**	**6.82E-08**	**1.05E-02**	3.01E-01	**2.29E-02**	**2.43E-07**
Log4j-1.1	**3.89E-03**	**1.17E-03**	**4.83E-03**	**5.98E-03**	**5.99E-08**	2.08E-01	6.75E-02	**1.50E-04**	**3.40E-06**
Lucene-2.0	**4.83E-05**	**3.46E-03**	**4.06E-03**	**2.97E-02**	**6.59E-08**	5.26E-02	4.46E-01	**1.95E-03**	**6.60E-08**
Xalan-2.4	1.76E-01	8.07E-01	9.78E-01	8.51E-02	**6.73E-08**	**3.24E-02**	2.23E-01	6.55E-01	**6.72E-08**
Xalan-2.5	**4.95E-04**	2.02E-01	**5.24E-03**	**6.25E-04**	**6.41E-08**	5.60E-02	**4.34E-02**	**3.21E-02**	**1.17E-07**
Xalan-2.6	**6.27E-03**	**3.13E-02**	9.14E-01	1.40E-01	**6.74E-08**	**1.99E-02**	9.24E-01	2.79E-01	**6.69E-08**

Table 20 Comparison between the proposed SBEWOA and other optimizers based on the number of selected features

Dataset	Measure	SBEWOA	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	AVG	13.55	11.10	14.90	12.60	13.20	**8.45**	10.45	14.60	15.30	10.20
	STD	2.5849	1.8325	1.9974	2.0365	2.2148	2.5438	1.9050	1.9304	1.3416	2.19089
Camel-1.0	AVG	12.35	11.00	13.70	11.25	12.45	**8.10**	9.40	12.50	12.55	11.00
	STD	1.8994	2.0000	1.1743	1.8028	2.0384	3.0591	2.1126	2.4387	2.1879	2.051957
Camel-1.2	AVG	14.40	12.30	14.90	12.25	13.55	**8.10**	10.90	13.95	15.20	10.10
	STD	1.7889	2.6178	1.3727	2.4682	2.2118	2.3598	1.4832	2.3278	1.0563	2.100125
Camel-1.4	AVG	14.00	12.00	15.55	12.60	14.30	**8.35**	11.10	14.30	15.70	11.40
	STD	2.7910	1.8064	1.4681	1.8180	2.5567	1.7554	1.2096	2.3193	0.9234	1.875044
Camel-1.6	AVG	14.90	11.90	15.40	12.40	13.75	**7.85**	10.70	14.90	15.95	10.90
	STD	3.3071	1.7442	2.3930	2.2337	2.5314	2.4339	1.6890	2.3598	2.0641	2.35975
Jedit-3.2	AVG	13.35	10.45	14.95	11.60	12.70	**9.05**	9.50	13.65	14.50	10.45
	STD	2.9429	1.7614	1.3169	1.3139	2.3193	2.8924	2.3731	2.4979	2.1643	1.848897
Jedit-4.0	AVG	13.65	12.45	14.65	11.35	13.00	**8.50**	10.45	14.50	15.00	10.60
	STD	2.8335	2.4597	1.8994	1.4965	2.2005	3.3166	1.8771	2.0391	1.5218	1.231174
Jedit-4.1	AVG	16.60	11.30	16.80	12.45	15.20	**8.45**	9.55	14.85	16.10	10.90
	STD	2.0365	1.8382	1.6416	2.4597	3.0366	2.0641	2.4165	3.0997	2.2455	2.04939
Jedit-4.2	AVG	12.80	10.55	14.20	11.65	12.40	**8.55**	10.60	12.85	14.55	9.95
	STD	2.9308	1.2344	1.9358	2.6413	2.7985	2.1145	1.7290	2.7582	1.9595	2.064104
Jedit-4.3	AVG	12.45	10.15	13.40	10.40	11.65	**8.00**	9.60	11.20	13.30	10.00
	STD	2.9105	1.4609	2.5423	2.2804	2.4121	2.0520	1.3139	2.4623	2.1546	1.685854
Log4j-1.0	AVG	12.35	9.90	12.95	10.65	10.85	9.45	**9.20**	11.60	12.65	9.60
	STD	2.3458	2.1250	1.6694	2.4554	2.9069	2.0125	1.9894	2.6636	1.4965	2.779625
Log4j-1.1	AVG	9.55	8.90	12.65	9.40	9.25	9.45	7.90	8.15	13.70	**7.80**
	STD	2.9996	1.3727	2.7582	2.7796	3.0240	2.5849	1.7442	2.7004	1.8382	2.166734
Lucene-2.0	AVG	15.05	10.95	14.75	11.55	13.55	**9.40**	10.40	13.55	16.00	10.60
	STD	2.9643	2.0125	2.7697	1.7911	2.9105	2.7222	2.1374	3.0689	1.9735	1.729009
Xalan-2.4	AVG	14.90	10.75	14.95	12.00	13.70	**7.95**	10.65	14.10	14.85	10.70
	STD	2.0749	1.9702	1.4681	1.5894	2.8303	2.7810	2.2542	2.0749	1.6944	1.657519
Xalan-2.5	AVG	15.10	11.05	14.60	12.30	14.30	**8.70**	11.00	13.55	15.70	11.05
	STD	2.3147	2.2821	2.1374	1.5252	2.2501	2.0026	2.1764	2.5021	1.4546	1.637553
Xalan-2.6	AVG	13.05	10.65	13.60	11.55	12.05	**8.50**	8.65	12.10	13.50	9.85
	STD	2.6253	1.8432	1.8750	1.8771	2.9285	2.1398	2.0072	3.3388	2.1885	1.814416
Rank (F-Test)		7.72	3.97	9.31	4.94	6.31	1.44	2.19	6.91	9.44	2.78

Table 21 2-tailed P-values of the Wilcoxon signed ranks test based on number of features reported in Table 20 (P-values ≤ 0.05 are in bold and significant)

Dataset	SBEWOA (as a best performing method) vs								
	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	**1.61E-03**	6.97E-02	1.46E-01	5.36E-01	**8.26E-06**	**4.16E-04**	2.51E-01	**2.92E-02**	**3.10E-04**
Camel-1.0	**2.67E-02**	**1.10E-02**	6.31E-02	8.69E-01	**4.08E-05**	**1.57E-04**	4.34E-01	5.73E-01	5.73E-02
Camel-1.2	**5.83E-03**	4.31E-01	**4.02E-03**	1.56E-01	**2.17E-07**	**5.98E-06**	6.98E-01	1.48E-01	**1.90E-06**
Camel-1.4	**3.17E-03**	**4.51E-02**	**1.64E-02**	6.48E-01	**4.44E-06**	**2.49E-04**	7.94E-01	**3.29E-02**	**1.01E-03**
Camel-1.6	**4.51E-03**	7.90E-01	**1.83E-02**	1.74E-01	**7.64E-07**	**1.72E-04**	6.38E-01	5.10E-01	**4.51E-04**
Jedit-3.2	**1.24E-03**	1.15E-01	**1.08E-02**	3.13E-01	**1.78E-04**	**2.55E-04**	8.59E-01	2.29E-01	**1.10E-03**
Jedit-4.0	1.41E-01	3.86E-01	**8.10E-03**	3.20E-01	**3.13E-05**	**1.13E-03**	4.98E-01	2.52E-01	**1.90E-03**
Jedit-4.1	**1.16E-06**	9.54E-01	**1.43E-05**	3.09E-01	**7.32E-08**	**2.36E-07**	6.93E-02	4.31E-01	**6.88E-07**

Table 21 (continued)

Dataset	SBEWOA (as a best performing method) vs								
	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Jedit-4.2	**4.45E-03**	1.15E-01	1.37E-01	6.23E-01	**7.58E-05**	**5.44E-03**	9.89E-01	**2.70E-02**	**1.58E-03**
Jedit-4.3	**1.36E-02**	3.17E-01	**3.01E-02**	3.81E-01	**2.79E-05**	**2.48E-03**	1.81E-01	3.73E-01	**8.44E-03**
Log4j-1.0	**2.34E-03**	4.04E-01	**4.59E-02**	1.48E-01	**5.55E-04**	**2.12E-04**	4.05E-01	6.59E-01	**3.83E-03**
Log4j-1.1	3.98E-01	**2.67E-03**	6.72E-01	5.01E-01	8.80E-01	**1.88E-02**	9.73E-02	**3.06E-05**	**3.44E-02**
Lucene-2.0	**1.55E-04**	8.13E-01	**9.79E-04**	1.50E-01	**8.10E-06**	**3.00E-05**	1.61E-01	4.43E-01	**2.72E-05**
Xalan-2.4	**5.93E-06**	8.55E-01	**9.43E-05**	1.63E-01	**2.53E-07**	**6.76E-06**	2.21E-01	7.27E-01	**3.76E-06**
Xalan-2.5	**3.11E-05**	3.24E-01	**3.54E-04**	2.06E-01	**2.56E-07**	**1.91E-05**	**4.03E-02**	5.85E-01	**1.18E-05**
Xalan-2.6	**3.89E-03**	5.03E-01	**4.36E-02**	1.91E-01	**8.64E-06**	**1.01E-05**	2.58E-01	6.42E-01	**1.89E-04**

Table 22 Comparison between the proposed SBEWOA and other optimizers based on fitness values

Dataset	Measure	SBEWOA	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	AVG	**0.12497**	0.12793	0.12595	0.12812	0.12885	0.14246	0.12674	0.12559	0.12621	0.13697
	STD	0.00314	0.00367	0.00396	0.00353	0.00349	0.00621	0.00373	0.00361	0.00298	0.00601
Camel-1.0	AVG	0.01682	0.01760	0.01767	0.01722	0.01678	0.02339	0.01677	**0.01624**	0.01757	0.02383
	STD	0.00268	0.00192	0.00180	0.00232	0.00229	0.00369	0.00199	0.00189	0.00195	0.00419
Camel-1.2	AVG	**0.26091**	0.26877	0.26217	0.26572	0.26320	0.29067	0.26858	0.26525	0.26392	0.28338
	STD	0.00535	0.00533	0.00787	0.00783	0.01113	0.01690	0.00890	0.00866	0.00643	0.01387
Camel-1.4	AVG	**0.11080**	0.11415	0.11352	0.11488	0.11229	0.12702	0.11646	0.11256	0.11241	0.12734
	STD	0.00438	0.00372	0.00262	0.00306	0.00630	0.00779	0.00337	0.00448	0.00431	0.00555
Camel-1.6	AVG	**0.16407**	0.16879	0.16636	0.16521	0.16788	0.18458	0.17183	0.16628	0.16646	0.18405
	STD	0.00426	0.00584	0.00295	0.00769	0.00519	0.00857	0.00419	0.00542	0.00374	0.00945
Jedit-3.2	AVG	**0.09424**	0.10398	0.10008	0.09941	0.09908	0.12482	0.09866	0.09738	0.09900	0.11694
	STD	0.00603	0.00785	0.00807	0.00842	0.01129	0.01382	0.00864	0.00983	0.00649	0.01214
Jedit-4.0	AVG	**0.10154**	0.11008	0.10666	0.10762	0.10716	0.13828	0.10751	0.10360	0.10716	0.13175
	STD	0.00903	0.00869	0.01079	0.00617	0.00593	0.01218	0.00896	0.00530	0.00609	0.01859
Jedit-4.1	AVG	**0.11583**	0.12439	0.11924	0.12392	0.11975	0.14811	0.12320	0.11831	0.11932	0.13840
	STD	0.00652	0.00924	0.00601	0.00776	0.00786	0.01076	0.00729	0.00673	0.00856	0.01271
Jedit-4.2	AVG	0.05471	0.05729	0.05598	0.05567	0.05583	0.06827	0.05586	**0.05392**	0.05556	0.06437
	STD	0.00300	0.00202	0.00296	0.00362	0.00363	0.00539	0.00290	0.00416	0.00402	0.00494
Jedit-4.3	AVG	0.01360	0.01433	0.01415	0.01403	0.01391	0.02078	0.01378	**0.01324**	0.01475	0.01790
	STD	0.00157	0.00183	0.00198	0.00174	0.00215	0.00286	0.00193	0.00173	0.00174	0.00290
Log4j-1.0	AVG	**0.05640**	0.06942	0.06339	0.06147	0.06493	0.10045	0.06379	0.05980	0.06470	0.09133
	STD	0.01093	0.01076	0.01184	0.01333	0.00874	0.01729	0.01015	0.01162	0.00981	0.01308
Log4j-1.1	AVG	**0.16363**	0.17788	0.18576	0.18010	0.17868	0.22671	0.16985	0.17123	0.18647	0.20966
	STD	0.01455	0.01687	0.01705	0.01727	0.01673	0.02128	0.01934	0.01999	0.01612	0.02601
Lucene-2.0	AVG	**0.23500**	0.24980	0.24662	0.24760	0.24637	0.28572	0.24316	0.23872	0.24905	0.28872
	STD	0.01515	0.01261	0.01015	0.01482	0.01165	0.02018	0.01626	0.01601	0.01653	0.02264
Xalan-2.4	AVG	0.11773	0.11660	0.11741	0.11663	0.11875	0.13122	0.11806	**0.11562**	0.11791	0.13257
	STD	0.00261	0.00379	0.00392	0.00418	0.00309	0.00804	0.00370	0.00426	0.00358	0.00561
Xalan-2.5	AVG	**0.27151**	0.27954	0.27302	0.27519	0.27629	0.29987	0.27456	0.27433	0.27592	0.29286
	STD	0.00707	0.00857	0.00663	0.00691	0.00401	0.01225	0.00693	0.00714	0.00471	0.00876
Xalan-2.6	AVG	0.19878	0.20195	0.20210	0.19766	0.20064	0.22345	0.20021	**0.19731**	0.20134	0.21950
	STD	0.00382	0.00804	0.00390	0.00775	0.00711	0.01007	0.00570	0.00633	0.00696	0.00942
Rank (F-Test)		**1.69**	6.88	4.88	5.06	5.09	9.75	4.94	2.19	5.28	9.25

Table 23 2-tailed P-values of the Wilcoxon signed ranks test based on fitness results reported in Table 22 (P-values ≤ 0.05 are in bold and significant)

Dataset	SBEWOA (as a best performing method) vs								
	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	**8.35E-03**	2.33E-01	**3.63E-03**	**9.19E-04**	**1.23E-07**	**2.47E-02**	3.64E-01	1.44E-01	**5.20E-07**
Camel-1.0	5.50E-01	2.40E-01	8.28E-01	8.49E-01	**9.97E-06**	7.86E-01	3.99E-01	4.47E-01	**1.81E-06**
Camel-1.2	**1.16E-04**	3.50E-01	**2.22E-02**	6.19E-02	**1.80E-06**	**1.48E-03**	8.09E-02	1.07E-01	**2.95E-07**
Camel-1.4	**9.03E-03**	**2.83E-02**	**1.11E-03**	1.13E-01	**9.07E-07**	**1.29E-04**	9.88E-02	1.13E-01	**6.75E-08**
Camel-1.6	**4.22E-03**	**3.10E-02**	5.41E-01	**1.11E-02**	**8.47E-08**	**5.54E-06**	1.77E-01	1.25E-01	**1.54E-07**
Jedit-3.2	**1.01E-03**	**8.29E-03**	5.30E-02	**3.84E-02**	**3.92E-07**	**4.38E-02**	1.40E-01	**2.06E-02**	**3.97E-06**
Jedit-4.0	**2.33E-03**	**1.43E-02**	**1.66E-02**	6.00E-02	**6.75E-08**	**4.83E-02**	7.97E-01	6.77E-02	**3.06E-06**
Jedit-4.1	**1.64E-03**	7.02E-02	**7.59E-04**	9.38E-02	**1.11E-07**	**9.72E-04**	3.31E-01	1.81E-01	**1.10E-06**
Jedit-4.2	**2.54E-03**	1.98E-01	3.30E-01	2.34E-01	**9.11E-08**	2.85E-01	7.45E-01	2.39E-01	**5.14E-06**
Jedit-4.3	**3.00E-02**	1.36E-01	3.36E-01	1.59E-01	**2.90E-07**	3.29E-01	5.97E-01	**3.11E-02**	**2.02E-05**
Log4j-1.0	**4.48E-03**	**4.22E-02**	4.90E-01	**1.72E-02**	**6.76E-08**	2.18E-01	4.90E-01	**1.85E-02**	**7.91E-07**
Log4j-1.1	**7.59E-03**	**1.48E-05**	**5.30E-03**	**8.68E-04**	**9.05E-08**	4.49E-01	3.23E-01	**1.37E-05**	**5.11E-06**
Lucene-2.0	**3.19E-04**	**6.37E-03**	**1.60E-02**	**4.79E-02**	**6.71E-08**	1.72E-01	5.78E-01	**1.98E-03**	**6.72E-08**
Xalan-2.4	3.37E-01	7.05E-01	2.23E-01	3.79E-01	**1.79E-06**	8.60E-01	1.23E-01	6.36E-01	**6.77E-08**
Xalan-2.5	**3.28E-03**	3.08E-01	**3.34E-02**	**2.04E-02**	**6.61E-07**	1.63E-01	1.13E-01	**2.36E-02**	**1.58E-07**
Xalan-2.6	**1.73E-02**	**1.33E-02**	6.55E-01	1.80E-01	**1.65E-07**	2.67E-01	5.79E-01	1.76E-01	**7.85E-08**

Table 24 Comparison between the proposed SBEWOA and other optimizers based on running time

Dataset	Measure	SBEWOA	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	AVG	42.37	23.56	24.09	28.16	25.09	25.17	52.95	14.54	31.81	**8.56**
	STD	0.8091	0.6519	0.7328	2.7729	0.3925	1.6021	4.1888	0.4517	0.2031	0.581693
Camel-1.0	AVG	32.37	18.87	18.82	21.53	22.11	22.74	43.96	11.62	26.96	**6.68**
	STD	0.2938	0.4587	0.4384	0.2519	0.2793	1.7507	0.5911	0.6222	0.1231	0.21952
Camel-1.2	AVG	18.19	19.56	19.78	22.11	22.99	22.76	45.48	12.78	28.06	**7.14**
	STD	8.8163	0.4091	0.4876	0.2066	0.1430	0.3365	0.5128	0.5573	0.1162	0.424393
Camel-1.4	AVG	18.21	11.96	13.87	26.11	27.08	26.26	52.57	16.01	33.51	**9.27**
	STD	1.0107	2.6331	4.9142	0.1961	0.2194	0.8363	0.4391	0.4752	0.2471	0.753417
Camel-1.6	AVG	18.31	11.25	11.69	26.28	27.37	26.54	53.45	16.72	33.75	**10.25**
	STD	0.4799	0.1865	0.1926	0.2364	0.1863	0.8872	0.7123	1.0311	0.2548	1.03882
Jedit-3.2	AVG	11.39	7.47	**7.36**	20.32	21.14	21.07	41.53	10.42	25.40	7.56
	STD	0.1531	0.0537	0.0524	0.1327	0.1229	0.2213	0.5285	0.3130	0.0581	0.751342
Jedit-4.0	AVG	12.06	7.87	7.81	20.91	21.72	21.57	42.59	11.29	26.22	**7.59**
	STD	0.1428	0.1947	0.2001	0.1683	0.1280	0.3358	0.2930	0.4059	0.0694	0.82611
Jedit-4.1	AVG	12.25	7.92	7.92	21.04	21.87	21.75	43.07	11.39	26.41	**7.21**
	STD	0.1719	0.2095	0.2108	0.1722	0.1451	0.2789	0.6146	0.4089	0.1001	0.900599
Jedit-4.2	AVG	13.10	8.50	8.54	21.98	22.75	22.57	51.46	12.05	27.55	**7.15**
	STD	0.3220	0.2623	0.2771	0.2030	0.1389	0.4267	2.8950	0.2907	0.1372	0.382593
Jedit-4.3	AVG	13.65	8.79	8.80	22.52	23.37	22.98	53.98	13.08	28.18	**7.50**
	STD	0.1982	0.0556	0.0620	0.4389	0.3382	0.5758	0.0718	0.4732	0.1063	0.335046
Log4j-1.0	AVG	10.35	6.86	6.69	19.45	20.18	20.27	46.62	10.12	24.04	**6.02**
	STD	0.1611	0.0654	0.0600	0.3320	0.1244	0.2542	0.3118	0.6578	0.0408	0.484603
Log4j-1.1	AVG	9.86	6.60	6.40	19.00	19.78	19.81	45.64	9.46	23.60	**5.49**
	STD	0.1363	0.0614	0.0723	0.2388	0.3506	0.3296	0.0749	0.6620	0.0283	0.194603

Table 24 (continued)

Dataset	Measure	SBEWOA	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Lucene-2.0	AVG	10.50	6.96	6.77	19.48	20.25	20.32	46.88	23.63	24.24	**5.75**
	STD	0.1240	0.0654	0.0652	0.1119	0.1165	0.1727	0.0687	0.8396	0.0641	0.196377
Xalan-2.4	AVG	17.44	11.09	11.48	25.72	26.80	25.97	60.85	38.72	32.51	**9.13**
	STD	0.3575	1.0491	1.2458	0.2112	0.1837	0.7990	0.3791	1.1876	0.2004	0.256848
Xalan-2.5	AVG	14.14	8.99	9.07	22.62	23.47	23.01	54.15	31.32	28.46	**8.05**
	STD	0.2133	0.0891	0.0725	0.1844	0.1391	0.4627	0.1391	0.6955	0.1061	1.787095
Xalan-2.6	AVG	14.30	**9.24**	9.32	22.88	24.45	23.65	54.65	31.96	28.84	18.72
	STD	0.4048	0.2755	0.2397	0.3487	1.9162	0.5539	1.5794	0.6436	0.1487	0.714985
Rank (F-Test)		5.06	2.53	2.72	5.75	7.25	6.88	10.00	4.81	8.69	**1.31**

Table 25 2-tailed P-values of the Wilcoxon signed ranks test based on running time results reported in Table 24 (P-values ≤ 0.05 are in bold and significant)

Dataset	SBEWOA vs								
	BFFA	BMFO	BMVO	BGWO	BBAT	BCS	BHHO	BJAYA	GA
Ant-1.7	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**
Camel-1.0	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**
Camel-1.2	**6.04E-03**	**6.04E-03**	**1.23E-03**	**1.23E-03**	**1.23E-03**	**6.80E-08**	**8.60E-06**	**1.23E-03**	**6.80E-08**
Camel-1.4	**1.20E-06**	**1.61E-04**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**7.90E-08**	**6.80E-08**	**6.80E-08**
Camel-1.6	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**1.60E-05**	**6.80E-08**	**6.80E-08**
Jedit-3.2	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**7.90E-08**	**6.80E-08**	**6.80E-08**
Jedit-4.0	**6.80E-08**	**6.79E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**4.54E-06**	**6.80E-08**	**6.80E-08**
Jedit-4.1	**6.80E-08**	**6.79E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**1.38E-06**	**6.80E-08**	**6.80E-08**
Jedit-4.2	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**9.17E-08**	**6.80E-08**	**6.80E-08**
Jedit-4.3	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**7.41E-05**	**6.80E-08**	**6.80E-08**
Log4j-1.0	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	4.57E-01	**6.80E-08**	**6.80E-08**
Log4j-1.1	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**2.47E-04**	**6.80E-08**	**6.80E-08**
Lucene-2.0	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**
Xalan-2.4	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**
Xalan-2.5	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**1.20E-06**
Xalan-2.6	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**	**6.80E-08**

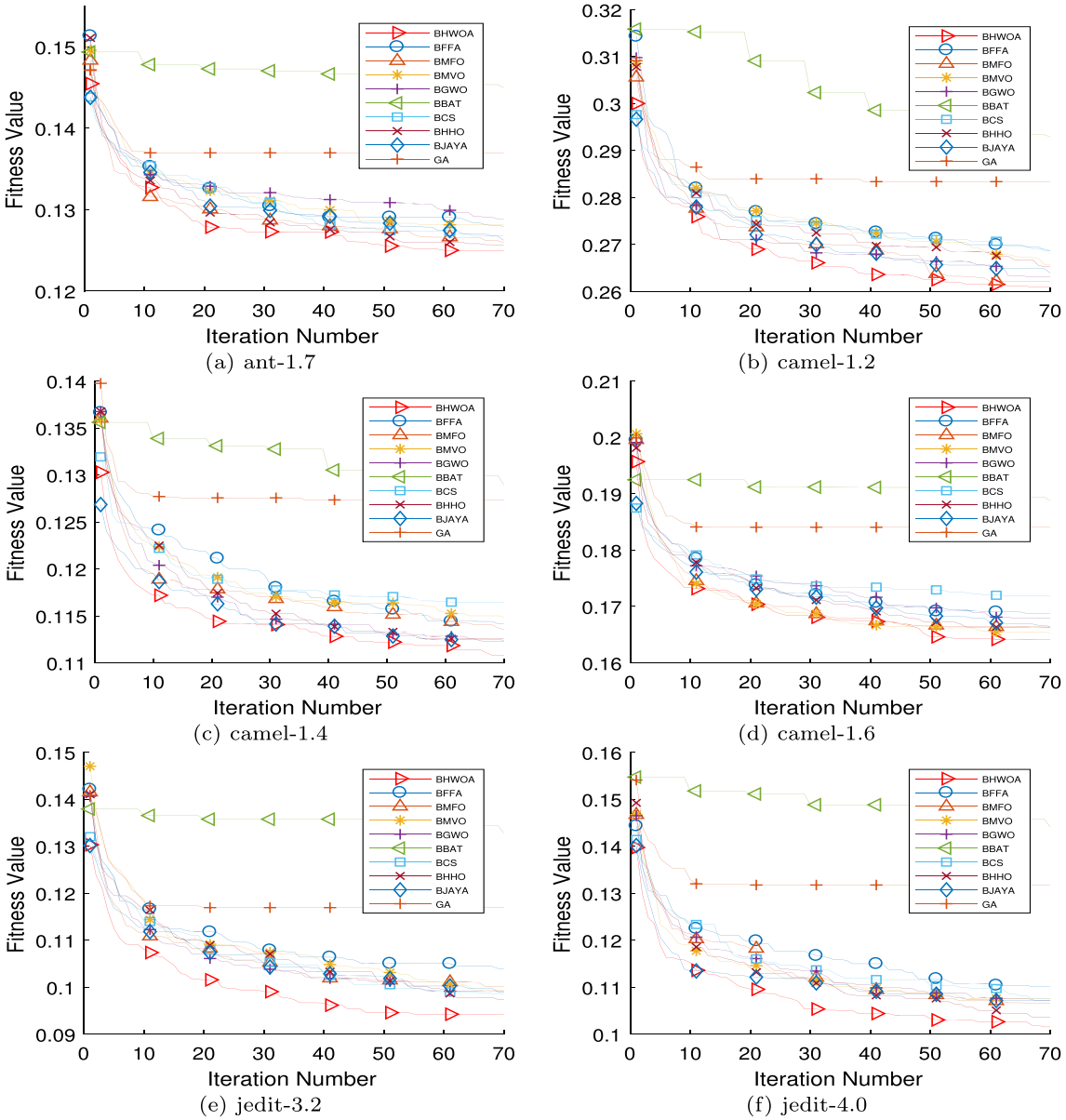

Fig. 11 Convergence curves for compared algorithms on ant-1.7, camel-1.2, camel-1.4, camel-1.6, edit-3.2, edit-4.0 datasets

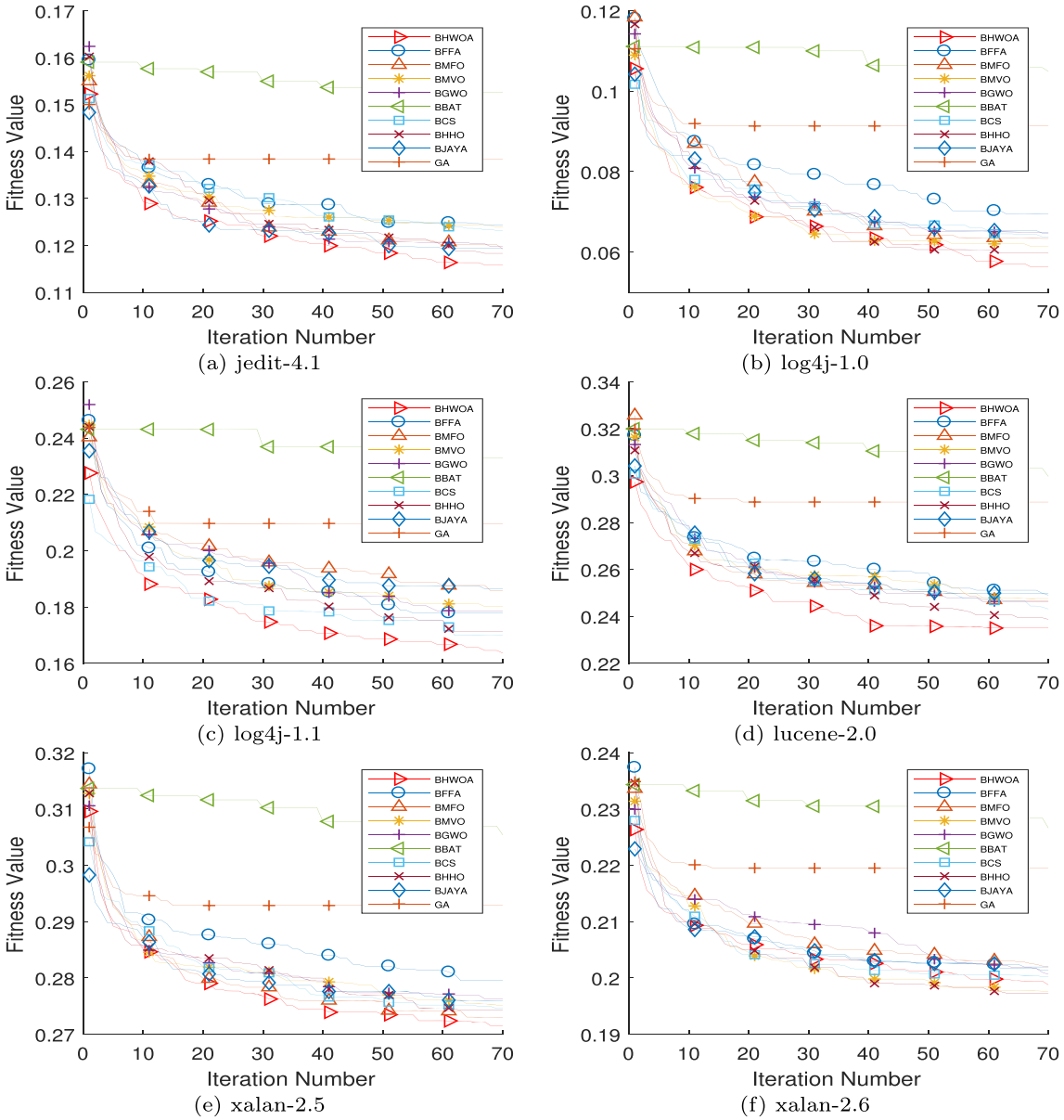

Fig. 12 convergence curves for compared algorithms on edit-4.1, og4j-1.0, log4j-1.1, ucene-2.0, xalan-2.5, xalan-2.6 datasets

Table 26 Details of selected features by SBEWOA that scores the best AUC results for each dataset [best result out of 20 runs]

Features	ant-1.7	camel-1.0	camel-1.2	camel-1.4	camel-1.6	jedit-3.2	jedit-4.0	jedit-4.1	jedit-4.2	jedit-4.3	log4j-1.0	log4j-1.1	lucene-2.0	xalan-2.4	xalan-2.5	xalan-2.6
wmc	1	0	1	1	1	1	0	1	1	0	0	1	1	0	1	1
dit	1	1	0	1	1	1	1	1	0	1	1	1	0	1	1	1
noc	1	1	1	1	1	1	0	1	1	0	1	0	1	0	1	0
cbo	1	1	1	1	1	1	1	1	0	1	1	0	1	1	0	0
rfc	1	1	0	0	0	0	1	1	1	0	1	1	1	0	0	1
lcom	0	0	1	1	1	0	0	1	0	0	0	1	0	0	1	1
ca	1	1	1	1	1	1	0	0	1	1	1	0	1	1	1	1
ce	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
npm	1	1	1	0	0	0	0	0	1	1	1	0	0	1	1	0
lcom3	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	0
loc	0	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1
dam	1	0	0	1	0	1	1	1	1	0	1	1	1	0	1	1
moa	0	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1
mfa	0	1	1	1	1	1	1	1	0	0	0	1	1	0	1	1
cam	0	0	1	1	0	0	0	1	1	1	1	1	1	1	1	1
ic	1	1	1	1	1	1	0	1	1	1	1	1	0	1	0	1
cbm	1	1	0	1	0	1	1	0	1	0	0	0	1	1	1	1
amc	1	0	1	1	0	0	0	1	0	0	0	1	0	1	1	0
max_cc	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0
avg_cc	1	0	1	1	1	1	1	1	0	0	0	1	1	0	1	0
Total	14	14	16	16	12	15	11	17	12	9	12	13	12	13	16	13
AUC	0.885767	0.995798	0.752647	0.909662	0.847781	0.924958	0.933772	0.899583	0.957532	0.994048	0.970588	0.874094	0.803819	0.893507	0.748306	0.811306

Table 27 The number of times each feature has been selected by SBEWOA for all datasets [over 320 runs]

Feature		Number of selections	Rate
f7	ca	259	80.938%
f2	dit	247	77.188%
f13	moa	243	75.938%
f19	max_cc	242	75.625%
f14	mfa	240	75.000%
f4	cbo	235	73.438%
f8	ce	232	72.500%
f3	noc	230	71.875%
f10	lcom3	225	70.313%
f15	cam	219	68.438%
f16	ic	217	67.813%
f9	npm	207	64.688%
f12	dam	207	64.688%
f5	rfc	206	64.375%
f18	amc	206	64.375%
f20	avg_cc	206	64.375%
f17	cbm	204	63.750%
f11	loc	190	59.375%
f1	wmc	173	54.063%
f6	lcom	173	54.063%

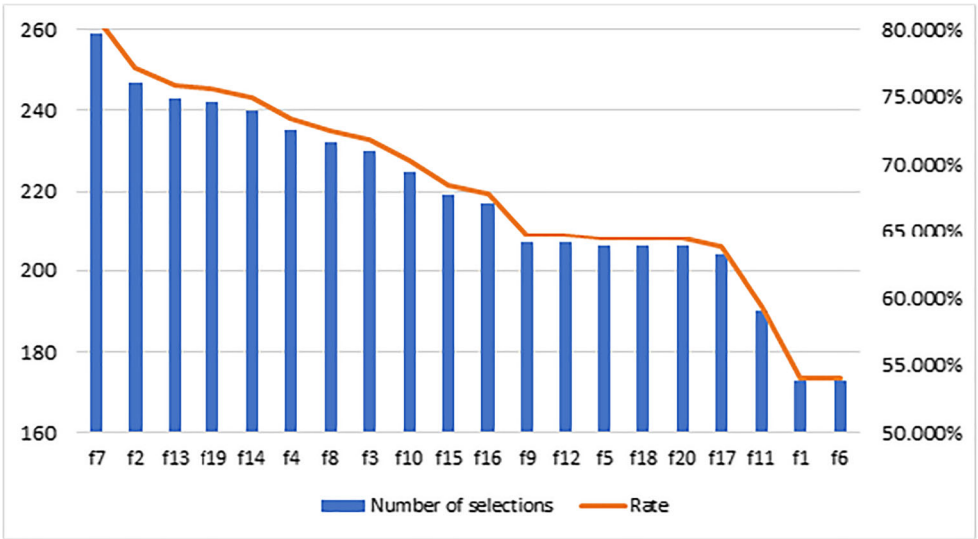

Fig. 13 Importance of features in terms of the number of times the SBEWOA algorithm has selected them

Table 28 Details of the number of times the features have been selected for each data

Features	ant-1.7	camel-1.0	camel-1.2	camel-1.4	camel-1.6	jedit-3.2	jedit-4.0	jedit-4.1	jedit-4.2	jedit-4.3	log4j-1.0	log4j-1.1	lucene-2.0	xalan-2.4	xalan-2.5	xalan-2.6	Total selections
wmc	9	9	18	11	16	10	10	15	6	8	1	6	11	12	18	13	173
dit	15	13	8	19	19	11	19	19	13	17	17	11	17	18	19	12	247
noc	16	13	13	17	19	17	12	18	13	14	16	8	14	13	14	13	230
cbo	19	17	16	15	13	18	17	18	9	15	14	6	18	16	12	12	235
rfc	16	11	13	14	12	4	13	20	15	7	10	10	18	13	13	17	206
lcom	8	9	10	10	15	14	2	19	10	10	13	11	10	10	16	6	173
ca	17	14	20	15	20	17	14	12	19	17	17	8	17	18	19	15	259
ce	15	14	15	13	20	17	11	18	18	12	3	14	16	14	18	14	232
npm	12	8	20	14	16	13	10	9	17	13	19	3	13	15	9	16	207
lcom3	16	11	15	18	13	15	13	16	13	16	13	17	10	12	18	9	225
loc	15	16	11	8	10	15	19	13	7	6	9	4	13	16	20	8	190
dam	9	12	15	13	16	10	13	18	19	7	17	9	10	16	10	13	207
moa	12	14	16	15	18	9	20	15	13	18	17	10	13	17	16	20	243
mfa	12	11	17	12	20	15	16	19	13	8	10	18	17	17	16	19	240
cam	18	9	16	11	10	8	17	18	12	18	16	13	17	11	18	7	219
ic	15	11	10	16	13	15	13	17	13	13	13	10	18	18	6	16	217
cbm	12	15	13	16	4	14	16	15	7	12	13	10	16	12	14	15	204
amc	12	12	8	10	10	11	12	17	12	16	8	9	17	18	19	15	206
max_cc	17	18	16	17	17	18	16	19	10	13	14	9	20	16	12	10	242
avg_cc	6	10	18	16	17	16	10	17	17	9	7	5	16	16	15	11	206

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

1. Honest N (2019) Role of testing in software development life cycle. Int J Comput Sci Eng 7(05):886–889
2. Turabieh H, Mafarja M, Li X (2018) Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. Expert Syst Appl 122:12
3. Tumar I, Hassouneh Y, Turabieh H, Thaher T (2020) Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction. IEEE Access pp(01):1–1
4. Rathore S, Kumar S (2017) A study on software fault prediction techniques. Artif Intell Rev 05:1–73
5. Fowler M, Highsmith J et al (2001) The agile manifesto. Softw Dev 9(8):28–35
6. Royce WW (1987) Managing the development of large software systems: concepts and techniques, pp 1–9, August 1970. In: Reprinted in proceedings of the ninth international conference on software engineering, pp 328–338
7. Hoda R, Salleh N, Grundy J, Tee HM (2017) Systematic literature reviews in agile software development: a tertiary study. Inf Softw Technol 85:60–70
8. Rathore SS, Kumar S (2017) A decision tree logic based recommendation system to select software fault prediction techniques. Computing 99:255–285
9. Gupta D, Saxena K (2017) Software bug prediction using object-oriented metrics. Sadhana - Acad Proc Eng Sci 42(05):655–669
10. Catal C, Diri B (2009) A systematic review of software fault prediction studies. Expert Syst Appl 36(05):7346–7354
11. Halstead MH (1977) Elements of software science (operating and programming systems series) USA: Elsevier science inc
12. McCabe TJ (1976) A complexity measure, IEEE. Trans Softw Eng SE-2(4):308–320
13. Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493
14. Lorenz M, Kidd J (1994) Object-oriented software metrics: a practical guide. Prentice-Hall, Inc
15. Bansiya J, Davis CG (2002) A hierarchical model for object-oriented design quality assessment. IEEE Trans Softw Eng 28(1):4–17
16. Deep Singh P, Chug A (2017) Software defect prediction analysis using machine learning algorithms. In: 2017 7th International conference on cloud computing, data science engineering - confluence, pp 775–781
17. Qasem O, Akour M (2019) Software fault prediction using deep learning algorithms. Int J Open Source Softw Process 10(10): 1–19
18. Oliveira J, Pontes K, Sartori I, Embiruçu M (2017) Fault detection and diagnosis in dynamic systems using weightless neural networks. Expert Syst Appl 84:05
19. Dong X, Yu Z, Cao W, Shi Y, Ma Q (2019) A survey on ensemble learning. Frontiers Comput Sci 14(08):241–258
20. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
21. Parmar A, Katariya R, Patel V (2019) A review on random forest: an ensemble classifier. In: Hemanth J, Fernando X, Lafata P, Baig Z (eds) International conference on intelligent data communication technologies and internet of things (ICICI) 2018. Springer international publishing, Cham, pp 758–763
22. Shaik AB, Srinivasan S (2019) A brief survey on random forest ensembles in classification model. In: Bhattacharyya S, Hassanien AE, Gupta D, Khanna A, Pan I (eds) International conference on innovative computing and communications. Singapore, Springer Singapore pp 253–260
23. Khoshgoftaar T, Van Hulse J, Napolitano A (2011) Comparing boosting and bagging techniques with noisy and imbalanced data. IEEE Trans Syst Man Cybern Part A 41(05):552–568
24. Dash M, Liu H (1997) Feature selection for classification. Intell Data Anal 1(3):131–156
25. Mafarja MM, Mirjalili S (2017) Hybrid whale optimization algorithm with simulated annealing for feature selection. Neurocomputing 260:302–312
26. Liu H, Motoda H (2012) Feature selection for knowledge discovery and data mining. Springer science & business media, vol 454
27. Talbi E-G (2009) Metaheuristics: from design to implementation. Wiley, vol 74
28. Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci 237:82–117
29. Zhang H, Sun G (2002) Feature selection using tabu search method. Pattern Recognit 35(3):701–711
30. Al-Betar MA, Hammouri AI, Awadallah MA, Doush IA (2021) Binary β-hill climbing optimizer with s-shape transfer function for feature selection. J Ambient Intell Humanized Comput 12(1):7637–7665
31. Boughaci D, Alkhawaldeh AA-S (2018) Three local search-based methods for feature selection in credit scoring. Vietnam J Comput Sci 5(2):107–121
32. Oreski S, Oreski G (2014) Genetic algorithm-based heuristic for feature selection in credit risk assessment. Expert Syst Appl 41(4):2052–2064
33. Ma J, Gao X (2020) A filter-based feature construction and feature selection approach for classification using genetic programming. Knowl-Based Syst 196:105806
34. Zhang Y, Gong D-W, Gao X-Z, Tian T, Sun X-Y (2020) Binary differential evolution with self-learning for multi-objective feature selection. Inf Sci 507:67–85
35. Wei B, Zhang W, Xia X, Zhang Y, Yu F, Zhu Z (2019) Efficient feature selection algorithm based on particle swarm optimization with learning memory. IEEE Access 7:166066–166078
36. Faris H, Mafarja MM, Heidari AA, Aljarah I, Ala'M A-Z, Mirjalili S, Fujita H (2018) An efficient binary salp swarm algorithm with crossover scheme for feature selection problems. Knowl-Based Syst 154:43–67
37. Kassaymeh S, Abdullah S, Al-Betar MA, Alweshah M (2022) Salp swarm optimizer for modeling the software fault prediction problem. J King Saud Univ-Comput Inf Sci 34(6):3365–3378
38. Hammouri AI, Mafarja M, Al-Betar MA, Awadallah MA, Abu-Doush I (2020) An improved dragonfly algorithm for feature selection. Knowl-Based Syst 203:106131. https://doi.org/10.1016/j.knosys.2020.106131
39. Awadallah MA, Al-Betar MA, Braik MS, Hammouri AI, Doush IA, Zitar RA (2022) An enhanced binary rat swarm optimizer based on local-best concepts of pso and collaborative crossover operators for feature selection. Comput Bio Med 147:105675. https://doi.org/10.1016/j.compbiomed.2022.105675
40. Mafarja MM, Mirjalili S (2019) Hybrid binary ant lion optimizer with rough set and approximate entropy reducts for feature selection. Soft Comput 23(15):6249–6265
41. Sawalha R, Doush IA (2012) Face recognition using harmony search-based selected features. Int J Hybrid Inf Technol 5(2): 1–16
42. Alweshah M, Alkhalaileh S, Al-Betar MA, Bakar AA (2022) Coronavirus herd immunity optimizer with greedy crossover

for feature selection in medical diagnosis. Knowl-Based Syst 235:107629
43. Paniri M, Dowlatshahi MB, Nezamabadi-pour H (2020) Mlaco: a multi-label feature selection algorithm based on ant colony optimization. Knowl-Based Syst 192:105285
44. Al-Betar MA, Hammouri AI, Awadallah MA, Abu Doush I (2021) Binary β-hill climbing optimizer with s-shape transfer function for feature selection. J Ambient Intell Humanized Comput 12(7):7637–7665
45. Sayed GI, Hassanien AE, Azar AT (2019) Feature selection via a novel chaotic crow search algorithm. Neural Comput Appl 31(1):171–188
46. Awadallah MA, Al-Betar MA, Hammouri AI, Alomari OA (2020) Binary jaya algorithm with adaptive mutation for feature selection. Arab J Sci Eng 45(12):10875–10890
47. Selvakumar B, Muneeswaran K (2019) Firefly algorithm based feature selection for network intrusion detection. Comput Security 81:148–155
48. Zhang Y, Cheng S, Shi Y, Gong D-W, Zhao X (2019) Cost-sensitive feature selection using two-archive multi-objective artificial bee colony algorithm. Expert Syst Appl 137:46–58
49. de Souza RCT, de Macedo CA, Dos Santos Coelho L, Pierezan J, Mariani VC (2020) Binary coyote optimization algorithm for feature selection. Pattern Recognit 107:107470
50. Aljarah I, Ala'M A-Z, Faris H, Hassonah MA, Mirjalili S, Saadeh H (2018) Simultaneous feature selection and support vector machine optimization using the grasshopper optimization algorithm. Cognit Computat 10(3):478–495
51. Karimi A, Irajimoghaddam M, Bastami E (2022) Feature selection using combination of genetic-whale-ant colony algorithms for software fault prediction by machine learning. Electr Cyber Defense, vol 10(1)
52. Rhmann W (2022) Software vulnerability prediction using grey wolf-optimized random forest on the unbalanced data sets. Int J Appl Metaheuristic Comput (IJAMC) 13(1):1–15
53. Kassaymeh S, Al-Laham M, Al-Betar MA, Alweshah M, Abdullah S, Makhadmeh SN (2022) Backpropagation neural network optimization and software defect estimation modelling using a hybrid salp swarm optimizer-based simulated annealing algorithm. Knowl-Based Syst 244:108511
54. Tameswar K, Suddul G, Dookhitram K (2022) A hybrid deep learning approach with genetic and coral reefs metaheuristics for enhanced defect detection in software. Int J Inf Manag Data Insights 2(2):100105
55. Zamani H, Nadimi-Shahraki MH, Gandomi AH (2022) Starling murmuration optimizer: a novel bio-inspired algorithm for global and engineering optimization. Comput Methods Appl Mech Eng 392:114616
56. Zamani H, Nadimi-Shahraki MH, Gandomi AH (2021) Qana: quantum-based avian navigation optimizer algorithm. Eng Appl Artif Intell 104:104314
57. Shayanfar H, Gharehchopogh FS (2018) Farmland fertility: a new metaheuristic algorithm for solving continuous optimization problems. Appl Soft Comput 71:728–746
58. Abdollahzadeh B, Gharehchopogh FS, Mirjalili S (2021) African vultures optimization algorithm: a new nature-inspired metaheuristic algorithm for global optimization problems. Comput Industr Eng 158:107408
59. Abdollahzadeh B, Soleimanian Gharehchopogh F, Mirjalili S (2021) Artificial gorilla troops optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems. Int J Intell Syst 36(10):5887–5958
60. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67
61. Mafarja M, Jaber I, Ahmed S, Thaher T (2021) Whale optimisation algorithm for high-dimensional small-instance feature selection. Int J Parallel Emergent Distributed Syst 36(2):80–96
62. Mafarja M, Mirjalili S (2018) Whale optimization approaches for wrapper feature selection. Appl Soft Comput 62:441–453
63. Mafarja M, Heidari AA, Habib M, Faris H, Thaher T, Aljarah I (2020) Augmented whale feature selection for iot attacks: structure, analysis and applications. Futur Gener Comput Syst 112:18–40
64. Nadimi-Shahraki MH, Zamani H, Mirjalili S (2022) Enhanced whale optimization algorithm for medical feature selection: a covid-19 case study. Comput Bio Med 148:105858
65. Zamani H, Nadimi-Shahraki M-H (2016) Feature selection based on whale optimization algorithm for diseases diagnosis. Int J Comput Sci Inf Security 14(9):1243
66. Wolpert D, Macready W (1997) No free lunch theorems for optimization. Evolutionary Computat IEEE 01:67–82
67. Singh A, Bhatia R, Singhrova A (2018) Taxonomy of machine learning algorithms in software fault prediction using object oriented metrics. Procedia Comput Sci 132:993–1001
68. Yogesh S, Arvinder K, Malhotra R (2009) Software fault proneness prediction using support vector machines. Lecture Notes Eng Comput Sci 2176:07
69. Khoshgoftaar TM, Seliya N (2003) Software quality classification modeling using the sprint decision tree algorithm. Int J Artif Intell Tools 12(03):207–225
70. Yuan X, Khoshgoftaar TM, Allen EB, Ganesan K (2000) An application of fuzzy clustering to software quality prediction. In: Application-specific systems and software engineering technology, 2000 proceedings. 3rd IEEE symposium on. IEEE, pp 85–90
71. Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng 33(1):2–13
72. kumar Dwivedi V, Singh MK (2016) Software defect prediction using data mining classification approach. Int J Tech Res Appl 4:31–35
73. Carrozza G, Cotroneo D, Natella R, Pietrantuono R, Russo S (2013) Analysis and prediction of mandelbugs in an industrial software system. In: Software testing, verification and validation (ICST), 2013 IEEE sixth international conference on. IEEE, pp 262–271
74. Bisi M, Goyal N (2015) Early prediction of software fault-prone module using artificial neural network. Int J Performability Eng 01:43–52
75. Caglayan B, Tosun A, Bener A, Miranskyy A (2014) Predicting defective modules in different test phases. Softw Qual J 23:06
76. Bowes D, Hall T, Petrić J (2018) Software defect prediction: do different classifiers find the same defects? Softw Qual J 26(2):525–552
77. Thaher T, Khamayseh F (2021) A classification model for software bug prediction based on ensemble deep learning approach boosted with smote technique. In: Sharma H, Saraswat M, Yadav A, Kim JH, Bansal JC (eds) Congress on intelligent systems. (Singapore), Springer Singapore, pp 99–113
78. Cahill J, Hogan JM, Thomas R (2013) Predicting fault-prone software modules with rank sum classification. In: 2013 22nd Australian software engineering conference. IEEE, pp 211–219
79. Erturk E, Sezer E (2016) Iterative software fault prediction with a hybrid approach. Appl Soft Comput 49:08
80. Khoshgoftaar T, Xiao Y, Gao K (2014) Software quality assessment using a multi-strategy classifier. Inf Sci - ISCI 259:02
81. Carrozza G, Cotroneo D, Natella R, Pietrantuono R, Russo S (2013) Analysis and prediction of mandelbugs in an industrial software system, 03

82. Rathore S, Kumar S (2017) Towards an ensemble based system for predicting the number of software faults. Expert Syst Appl 82:04
83. Choudhary GR, Kumar S, Kumar K, Mishra A, Catal C (2018) Empirical analysis of change metrics for software fault prediction. Comput Electr Eng 67:15–24
84. Shatnawi R (2017) The application of roc analysis in threshold identification, data imbalance and metrics selection for software fault prediction. Innov Syst Softw Eng 13:201–217
85. Chantar H, Mafarja M, Alsawalqah H, Heidari AA, Aljarah I, Faris H (2020) Feature selection using binary grey wolf optimizer with elite-based crossover for arabic text classification. Neural Comput Appl 32(16):12201–12220
86. Catal C, Diri B (2009) Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Inf Sci 179(03):1040–1058
87. Balogun AO, Basri S, Abdulkadir SJ, Hashim AS (2019) Performance analysis of feature selection methods in software defect prediction: a search method approach. Appl Sci 9(13):2764
88. Jia L (2018) A hybrid feature selection method for software defect prediction. In: IOP conference series: materials science and engineering. IOP publishing, vol 394, p 032035
89. Wahono RS, Herman NS (2014) Genetic feature selection for software defect prediction. Adv Sci Lett 20(1):239–244
90. Wahono RS, Suryana N, Ahmad S (2014) Metaheuristic optimization based feature selection for software defect prediction. J Softw 9(5):1324–1333
91. Thaher T, Arman N (2020) Efficient multi-swarm binary harris hawks optimization as a feature selection approach for software fault prediction. In: 2020 11th International conference on information and communication systems (ICICS), pp 249–254
92. Larranaga P, Calvo B, Santana R, Bielza C, Galdiano J, Inza I, Lozano J, Ar-mañanzas R, Santafé G, Pérez A, Robles V (2006) Machine learning in bioinformatics. Briefings bioinformatics 7(04):86–112
93. Wang F, Ma S, Wang H, Li Y, Qin Z, Zhang J (2018) A hybrid model integrating improved flower pollination algorithm-based feature selection and improved random forest for nox emission estimation of coal-fired power plants. Measurement 125:303–312
94. Malekipirbazari M, Aksakalli V (2015) Risk assessment in social lending via random forests. Expert Syst Appl 42(10):4621–4631
95. He H, Garcia E (2009) Learning from imbalanced data. Knowl Data Eng IEEE Trans 21(10):1263–1284
96. Chawla N, Bowyer K, Hall LO, Kegelmeyer WP (2002) Smote: synthetic minority over-sampling technique. J Artif Intell Res (JAIR) 16(01):321–357
97. Mafarja M, Aljarah I, Heidari AA, Hammouri AI, Faris H, Ala'M A-Z, Mirjalili S (2018) Evolutionary population dynamics and grasshopper optimization approaches for feature selection problems. Knowl-Based Syst 145:25–45
98. Kumar V, Minz S (2014) Feature selection: a literature review. Smart Comput Rev 4(01):211–229
99. Mafarja MM, Mirjalili S (2017) Hybrid whale optimization algorithm with simulated annealing for feature selection. Neurocomputing 260:302–312
100. Amrieh E, Hamtini T, Aljarah I (2016) Mining educational data to predict student's academic performance using ensemble methods. Int J Database Theory Appl 9(09):119–136
101. Mafarja M, Aljarah I, Heidari AA, Faris H, Fournier-Viger P, Li X, Mirjalili S (2018) Binary dragonfly optimization for feature selection using time-varying transfer functions. Knowl-Based Syst 161:185–204
102. Faris H, Ala'M A-Z, Heidari AA, Aljarah I, Mafarja M, Hassonah MA, Fujita H (2019) An intelligent system for spam detection and identification of the most relevant features based on evolutionary random weight networks. Inf Fusion 48:67–83
103. Mirjalili S, Lewis A (2013) S-shaped versus v-shaped transfer functions for binary particle swarm optimization. Swarm Evolutionary Computat 9:1–14
104. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: Systems, man, and cybernetics, 1997. Computational cybernetics and simulation, 1997 IEEE international conference on. IEEE, vol 5, pp 4104–4108
105. Rashedi E, Nezamabadi-Pour H, Saryazdi S (2010) Bgsa: binary gravitational search algorithm. Nat Comput 9(3):727–745
106. Mirjalili S, Dong J (2020) Multi-objective optimization using artificial intelligence techniques. 01
107. Emary E, Zawbaa HM (2016) Impact of chaos functions on modern swarm optimizers. PloS One 11(7):e0158738
108. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
109. Heidari AA, Mirjalili S, Faris H, Aljarah I, Mafarja M, Chen H (2019) Harris hawks optimization: algorithm and applications. Future Generation Comput Syst 97:849–872
110. (2017). Tera-Promise. http://openscience.us/repo. Last Accessed 24 Nov 2017
111. Jureczko M, Madeyski L (2010) Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th international conference on predictive models in software engineering, PROMISE '10, (New York). ACM, pp 9:1–9:10
112. Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20:476–493
113. Thaher T, Heidari AA, Mafarja M, Dong JS, Mirjalili S (2020) Binary harris hawks optimizer for high-dimensional, low sample size feature selection. Singapore: Springer Singapore, pp 251–272
114. Thaher T, Chantar H, Too J, Mafarja M, Turabieh H, Houssein EH (2022) Boolean particle swarm optimization with various evolutionary population dynamics approaches for feature selection problems, vol 195
115. Thaher T, Awad M, Aldasht M, Sheta A, Turabieh H, Chantar H (2022) An enhanced evolutionary based feature selection approach using grey wolf optimizer for the classification of high-dimensional biological data. JUCS - J Univ Comput Sci 28(5):499–539
116. Chantar H, Thaher T, Turabieh H, Mafarja M, Sheta A (2021) Bhho-tvs: a binary harris hawks optimizer with time-varying scheme for solving data classification problems. Applied Sci 11:14

Affiliations

Majdi Mafarja[1] · Thaer Thaher[2,3] · Mohammed Azmi Al-Betar[4] · Jingwei Too[5] · Mohammed A. Awadallah[6,7] · Iyad Abu Doush[8,9] · Hamza Turabieh[10]

Majdi Mafarja
mmafarja@birzeit.edu

Thaer Thaher
thaer.thaher@aaup.com

Jingwei Too
jamesjames868@gmail.com

Mohammed A. Awadallah
ma.awadallah@alaqsa.edu.ps

Iyad Abu Doush
idoush@auk.edu.kw

Hamza Turabieh
h.turabieh@tu.edu.sa

1 Department of Computer Science, Birzeit University, Birzeit, Palestine

2 Department of Computer Systems Engineering, Arab American University, Jenin, Palestine

3 Information Technology Engineering, Al-Quds University, Abu Dies, Jerusalem, Palestine

4 Artificial Intelligence Research Center (AIRC), College of Engineering and Information Technology, Ajman University, Ajman, United Arab EmiratesDeepSinghML2017, Irbid, Jordan

5 Faculty of Electrical Engineering, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100, Durian Tunggal, Melaka, Malaysia

6 Department of Computer Science, Al-Aqsa University, P.O. Box 4051, Gaza, Palestine

7 Artificial Intelligence Research Center (AIRC), Ajman University, Ajman, United Arab Emirates

8 Department of Computing, College of Engineering and Applied Sciences, American University of Kuwait, Salmiya, Kuwait

9 Computer Science Department, Yarmouk University Irbid, Jordan

10 Department of Health Management and Informatics, University of Missouri, Columbia, 5 Hospital Drive, Columbia, MO 65212, USA