ORIGINAL ARTICLE

Optimizing software defect prediction: a fusion of binary horse herd optimizer and machine learning methods

Bahman Arasteh[1,2,3] · Asgarali Bouyer[1,4] · Peri Gunes[5,6] · Reza Ghanbarzadeh[7] · Farhad Soleimanian Gharehchopogh[8]

Received: 29 August 2024 / Accepted: 30 May 2025 / Published online: 4 October 2025
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2025

Abstract
Predicting software defects is a vital component of software engineering, focused on improving software quality by detecting potential issues at an early stage. This process involves predicting modules that are likely to be defective before testing, thereby minimizing both the time and expenses associated with the testing phase. Machine learning has proven to be an effective tool in identifying software modules that are prone to defects. However, achieving accurate and precise classification remains a challenge due to the complexity of factors within the training data set. To address this, the selection of the most relevant features for classification becomes essential, a task commonly addressed through the use of metaheuristic algorithms. This paper presents a new approach through the introduction of a binary form of the horse herd optimization algorithm (bHOA), specifically designed to enhance the feature selection process from training data sets. By focusing on the most significant features, the new method aims to significantly improve the precision and accuracy of software defect classifiers. The key contributions of this study include the development of a binary variant of the bHOA for optimized feature selection and the creation of an effective model for classifying faulty software modules. The efficacy of this method was tested using five real-world and benchmark data sets during both the training and testing phases. The results show that among the twenty-one features present in the data sets, metrics such as basic cyclomatic complexity, programme difficulty, the number of operators and operands, and lines of code are the most predictive of software defects. These findings highlight the substantial improvements in accuracy, precision, recall, and $F1$ score achieved through the integration of the binary bHOA with machine learning methods for software defect prediction.

Keywords Horse herd optimization algorithm · Machine learning · Prediction · Feature selection · Software defect

1 Introduction

Defects in software systems present a considerable challenge to maintaining quality. Predicting these defects is an essential focus in software engineering, as it allows for early identification and resolution of issues during development [1]. Early detection, before a software's release, is critical for improving its overall quality. The Pareto principle suggests that most defects are concentrated in specific modules. Consequently, predicting defects early on can significantly enhance the final product's quality. This process involves determining whether software modules will likely contain defects or are defect-free. By forecasting defective modules before the testing phase, development teams can reduce the time and costs associated with testing. As software systems become more

complex, testing every module in detail becomes increasingly impractical. Accurately identifying which modules are prone to defects is essential for maintaining software quality. In this regard, machine learning (ML) techniques provide dependable models that enable software developers to efficiently classify faulty modules [2].

The employment of software defect prediction (SDP) techniques has seen a substantial increase, with the goal of enhancing software quality. This study categorizes software modules into two groups: those likely to be defective and those unlikely to contain defects [1, 2]. This classification relies on building a model from historical data of defective modules, which is then used to enhance defect prediction in new modules. Previous studies have demonstrated a strong connection between software module metrics and their likelihood of containing defects [3].

Historical datasets for software defect prediction contain various features (called attributes or metrics) pertaining to software development, maintenance, and testing. The likelihood of a defect in a particular software module can be predicted using these features. The typical features of historical datasets are: lines of code, the complexity of the programme's control flow, Halstead metrics such as difficulty, effort, volume, number of operators, operands, amount of comments in the code, and time spent developing the module. Various ML algorithms, such as Decision Trees (DT), K-Nearest Neighbour (KNN), Naive Bayes (NB), Support Vector Machines (SVM), and neural networks, are typically used for classifying software modules. One of the key challenges in this process is handling a large quantity of features, which can reduce the classification accuracy. Feature selection methods are commonly employed to address this issue by narrowing down the number of features. The results of previous studies [3, 4] indicate that different features of the datasets have different influences on the performance of the ML algorithms. For example, McCabe's design complexity, the total counts of operators and operands, programme length, and difficulty have different influences on the performance of the created defect predictors by the ML algorithms. However, selecting the most optimal features for classification is a problem classified as NP-hard, which is often tackled using evolutionary algorithms [4].

The current research presents a novel approach to SDP by integrating the binary horse herd optimisation algorithm (bHHOA) with different machine learning algorithms. The bHHOA is first utilized to select the most relevant features from the training data set, which are then applied to build classification models using different machine learning algorithms. The bHHOA is inspired by the collective intelligence displayed in horse herds, considering factors such as age and experience. It incorporates six key behaviours observed in horses: grazing, hierarchy, sociability, imitation, defence, and roaming, which are essential to the HHOA's operation. These behaviours replicate natural horse responses to environmental stimuli, such as the fight-or-flight response in dangerous situations, competition for resources, and instinctive avoidance of risk.

The main goal of this method is to leverage the bHHOA's ability to select critical features to predict software defects with ML algorithms, with the aim of detecting and addressing software defects before release. A variety of ML algorithms, including K-Nearest Neighbour (KNN), Decision Trees (DT), Support Vector Machines (SVM), Naive Bayes (NB), and Artificial Neural Networks (ANN), were employed to classify modules that may contain defects. By concentrating on the most important features, this method seeks to enhance the accuracy and precision of the software module classifier. The efficiency of this new feature selection approach was assessed during the testing phase, using test data to evaluate its impact on SDP. The study mainly aims at:

- Determining the most critical features in data sets used for predicting software defects.
- Enhancing the SDP models' accuracy, precision, and sensitivity.
- Improving the effectiveness and reliability of SDP models.

The proposed bHHOA demonstrates advantages over existing feature selection methods for software fault prediction. Existing ML-based defect prediction techniques often suffer from degraded performance due to the inclusion of irrelevant or redundant features. In contrast, bHHOA identifies a minimal and highly impactful subset of features, enhancing the learning process by eliminating noise and improving generalization. By targeting critical code and design-level features, bHHOA ensures that the most effective features are prioritized. This not only leads to a more efficient SDP, but also enhances the predictive performance, achieving high accuracy, precision, recall, and $F1$ score. Furthermore, bHHOA maintains robustness across different datasets; this

characteristic improves the scalability and adaptability of the method. The significant contributions of the current study include:

- Creating a binary horse herd optimization algorithm developed to identify the most influential features in defect prediction data sets.
- Creating a robust and efficient classification model for identifying defective software modules.
- Boosting the efficiency and stability of software defect detection approaches by focusing on the smallest yet most impactful subset of features.

The structure of this paper is organized as follows: Section 2 reviews the existing literature on SDP. Section 3 explains the methodology behind the proposed approach, divided into two parts. The first part focuses on the creation of the binary HHOA (bHHOA) and its function in selecting the most significant features from the training data. The second part discusses the construction of the classifier using these optimized features in conjunction with various ML algorithms. Section 4 presents the experimental results achieved through the application of the proposed method on real-world data sets and evaluates the approach based on certain performance metrics. Lastly, Sect. 5 concludes with a summary of the findings and offers recommendations for future research.

2 Related works

2.1 Software defect prediction without features selection

The testing phase plays a crucial role in the software development process [5–7]. The dependability of software is considerably impacted by the presence of defects (bugs or faults) [8, 9]. This stage frequently demands considerable investment of both time and resources, highlighting the necessity of identifying software modules that are prone to defects. Such predictions, made before the testing phase begins, aid in identifying and correcting modules prone to defects. By utilizing historical data from previous or similar projects with problematic software modules, a model is developed to accurately predict defects in new modules. Research on defect prediction and estimation has demonstrated that the effectiveness of the prediction model is heavily influenced by the foundational hypothesis used in its construction [10]. A range of methods exist for SDP, and this section explores four machine learning approaches: normalization-based, unbalanced learning-based, feature selection-based, and blended learning methods. Data normalization is a pre-processing step that involves standardizing the values of independent features in a data set before training and testing. The aim of normalization is to standardize the range of different feature values within data sets, which is essential as specific machine learning algorithms may struggle to perform effectively without it. This technique improves the algorithm's accuracy by rescaling the measured values to a new range [11].

Logarithmic normalization adjusts feature values by transforming them into their logarithmic counterparts. The min–max method identifies each feature's minimum and maximum values and normalizes these features within the identified range. The Z-standardization method uses feature values' mean and standard deviation to remap them to a new range, where the mean is set to 0 and the standard deviation to 1. Additionally, it is essential to recognize that in the problems of classification, the distribution of samples across different classes can be uneven. Several studies, including [12], have employed techniques such as logarithmic normalization, min–max scaling, and Z-standardization. In binary classification, imbalances occur when one class has more samples. This poses a problem since most learning algorithms are designed to train classifiers with equal samples across classes. This leads to inadequate predictions and estimations for the less-represented class when dealing with unbalanced data sets. To tackle this challenge, various strategies have been developed, falling into three main categories: data-level, algorithm-level, and cost-sensitive learning methods [13].

Data-level methods are pre-processing approaches that adjust the data distribution before it reaches the learning algorithm, effectively balancing the classes. These methods work by modifying the training samples to create a more balanced data set, allowing classifiers to function more effectively under standard conditions [13]. There are two key types of data sampling techniques: oversampling and undersampling. Oversampling increases the sampling rate of the minor class by adding more samples from the less-represented class, while undersampling decreases the sampling rate of the major class by removing samples from the more dominant class, thereby achieving a balance. Algorithm-level approaches, on the other hand, adapt the learning algorithms themselves to pay more attention to the minor class. This is achieved through methods like bagging and boosting, which are designed to focus the learning process more on the class with fewer samples. These techniques help the learning algorithm become more sensitive to the minor class, enhancing its capability to accurately predict in unbalanced scenarios.

2.2 Software defect prediction using features selection

Machine learning methods designed to work with high-dimensional data sets, which contain numerous features and face several obstacles. These challenges encompass increased computational demands, difficulties in extracting relevant insights, and the necessity of managing complexity in order to avoid overfitting during the training phase [14]. To overcome these challenges, dimensionality decrease techniques are utilized, aiming to reduce the data set to a lower-dimensional space while retaining the critical attributes of the original data. These approaches can be classified into two types of methods: feature extraction and feature subset selection.

Feature extraction methods minimize dimensionality by merging existing feature values into a smaller, more streamlined feature set. A widely recognized method, principal component analysis (PCA), restructures the original features into a collection of linearly uncorrelated components, representing the most substantial variance within the data. In contrast, feature subset selection methods decrease dimensionality by choosing a subset of the most crucial features from the original data set. This approach maintains the original feature representation but limits the data set to the most informative features, enhancing the performance of the model and interpretability. These methods aim to reduce computational complexity and improve the generalizability of ML models by concentrating on the most critical features.

Feature subset selection is a commonly employed method in machine learning, particularly in classification and regression tasks. While these tasks often involve a large number of features, some may contribute little or negatively impact learning accuracy. Eliminating these irrelevant or less significant features can improve accuracy and lower computational costs. The goal of feature selection is to determine the minimal set of features necessary for the task at hand. In defect prediction, feature selection incorporates both filtering and classification techniques [15]. The filtering method functions separately from the machine learning algorithm, assessing feature subsets using predefined criteria without relying on a classification function. Initially, features are given a credit rating, ranked accordingly, and the least significant ones are removed on the basis of a defined threshold. The refined subset is then utilized within the classification system. This approach streamlines the data set, ensuring that only the most relevant features are considered, which can lead to improved model performance and efficiency.

Various filter-based feature ranking methods, such as Information Gain (IG), Gain Ratio (GR), Symmetric Uncertainty (SU), the Chi-Square test (CS), and two versions of the ReliefF algorithm, were examined in [16]. The CS filter assesses features by evaluating class distributions and their correlations. IG measures the extent of information a feature (Y) provides in predicting the target class about another feature (X). However, IG tends to prioritize features with numerous distinct values, which may not always be the most relevant for classification purposes. To mitigate this bias, GR and SU have been developed. GR introduces a penalty for features with numerous values. At the same time, SU counters this bias by calculating the combined entropies of X and Y. Additionally, the ReliefF filter, which extends the original Relief algorithm, demonstrates robustness in managing noisy data and multi-class problems. Figure 1 depicts the process utilized by filter-based methods for selecting features. These filter-based methods help identify and rank the most relevant features for the given task, ensuring

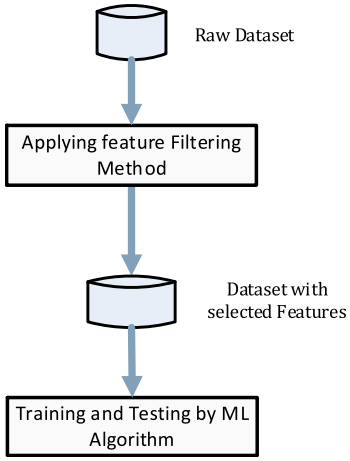


Fig. 1 Filter-based methods' procedure for feature selection

that the resulting feature subset contributes effectively to the model's performance. By employing these techniques, it is possible to improve the accuracy and efficiency of ML models used for SDP.

Classification, often referred to as black-box methods, depends on an evaluation function driven by a learning algorithm's error rate. In this method, new feature subsets are created via a generator function, trained through a machine learning technique, and assessed based on the error rate or the count of errors within the test set. Although classification methods tend to be more effective than filter-based methods, they typically demand greater computational resources. Within this framework, two prominent techniques are forward selection and backward selection, both designed to determine the optimal feature set. Forward selection evaluates features one by one at each stage for potential inclusion, while backward selection begins with all features and incrementally removes them until a predefined stopping criterion is satisfied, identifying the most crucial features for SDP [15]. Figure 2 depicts the steps involved in these classification techniques.

Both of the forward and backward selection techniques have been applied [17], with an alternative method called greedy forward selection, which begins with the first feature to enhance the accuracy and performance [18]. Figure 2 shows the process used by classification-based methods for feature selection. These methods, by systematically evaluating the contribution of each feature, help identify the most relevant features, thereby improving the model's accuracy and efficiency. The rigorous process ensures that only the essential features are retained, leading to more effective SDP.

Blended learning is recognized as an effective technique in machine learning [19]. This method enhances learning accuracy by integrating the results from multiple classifiers. Blended learning primarily involves two

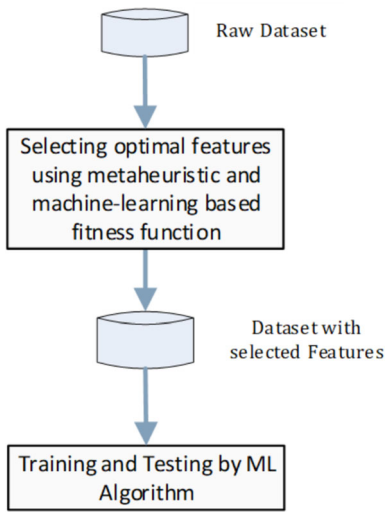


Fig. 2 The classification methods' procedure for feature selection

scenarios. The first scenario involves applying different classification algorithms to SDP problems, mainly focusing on finding the most suitable algorithm. However, this method does not benefit from integrating knowledge from other algorithms and faces difficulty in selecting the best option. In the second scenario, when feature sets are large and complex, utilizing all features within a single classifier becomes unfeasible. A crucial element in constructing an effective blended classifier lies in selecting the appropriate classification rule. Without the right choice of classifiers, their diversity contributes little to improving the blended model's performance. In summary, blended learning combines the strengths of multiple classifiers to enhance overall predictive accuracy. By integrating insights from various classification algorithms and effectively managing diverse feature sets, blended learning can significantly enhance the performance of SDP models. Table 1 compares the main merits and demerits of the existing methods.

3 The proposed feature selection-based defect predictor

3.1 The proposed predictor's structure

This section outlines the SDP method proposed in the current study, which incorporates the binary HHOA and machine learning algorithms. The bHHOA was applied to pinpoint the most important features within the SDP data sets, which were subsequently utilized to construct predictive models using different ML algorithms. The utilized algorithms in the current study include Artificial Neural Networks (ANN), Decision Trees (DT), K-Nearest Neighbours (KNN), Naive Bayes (NB), and Support Vector Machines (SVM). Figure 3 illustrates the overall process of this defect prediction framework. By combining bHHOA for feature selection with various machine learning techniques, this methodology seeks to improve both the accuracy and effectiveness of SDP.

Standard data sets were utilized for training the machine learning algorithms and measuring the effectiveness of the software defect prediction models. The role of bHHOA is pivotal in selecting features that substantially boost

Table 1 Comparison of the software fault prediction methods

Paper	Description	Advantage	Drawback
[20]	Decision tree, bagging, random forest, gradient boosting	Higher prediction performance	Lack of feature selection and insufficient accuracy
[21]	Combination of differentiable and non-differentiable ML algorithms	Stable performance	Slower, especially for large datasets
[22]	A combination of wagging and AdaBoost	Appropriate for imbalanced datasets higher and more stable performance than the AdaBoost	Higher computational costs due to the large number of models generated and the complexity of hyperparameter tuning
[23]	Combination of PSO and bagging ensemble methods	Eliminating some irrelevant features and increasing the training performance	Impractical for very large or complex datasets and sensitivity to PSO initialization
[24]	Harmony search-based optimization algorithm and cost-sensitive boosting	The utilized metaheuristic algorithm (for parameter tuning) improves the performance of the created fault predictor	Heavily depends on tuning parameters, risk of local optima and lower performance in the imbalanced datasets
[25]	Combination of the salp swarm algorithm (SSA) combined with a backpropagation neural network (BPNN) to create a software fault predictor	Higher prediction accuracy	Slow convergence of SSA in the software fault prediction problem
[26]	Three-way decisions and ensemble learning to predict software defect	Higher accuracy and a lower decision cost.	Difficulty in threshold selection for three-way decisions and need a trade-off between precision and recall

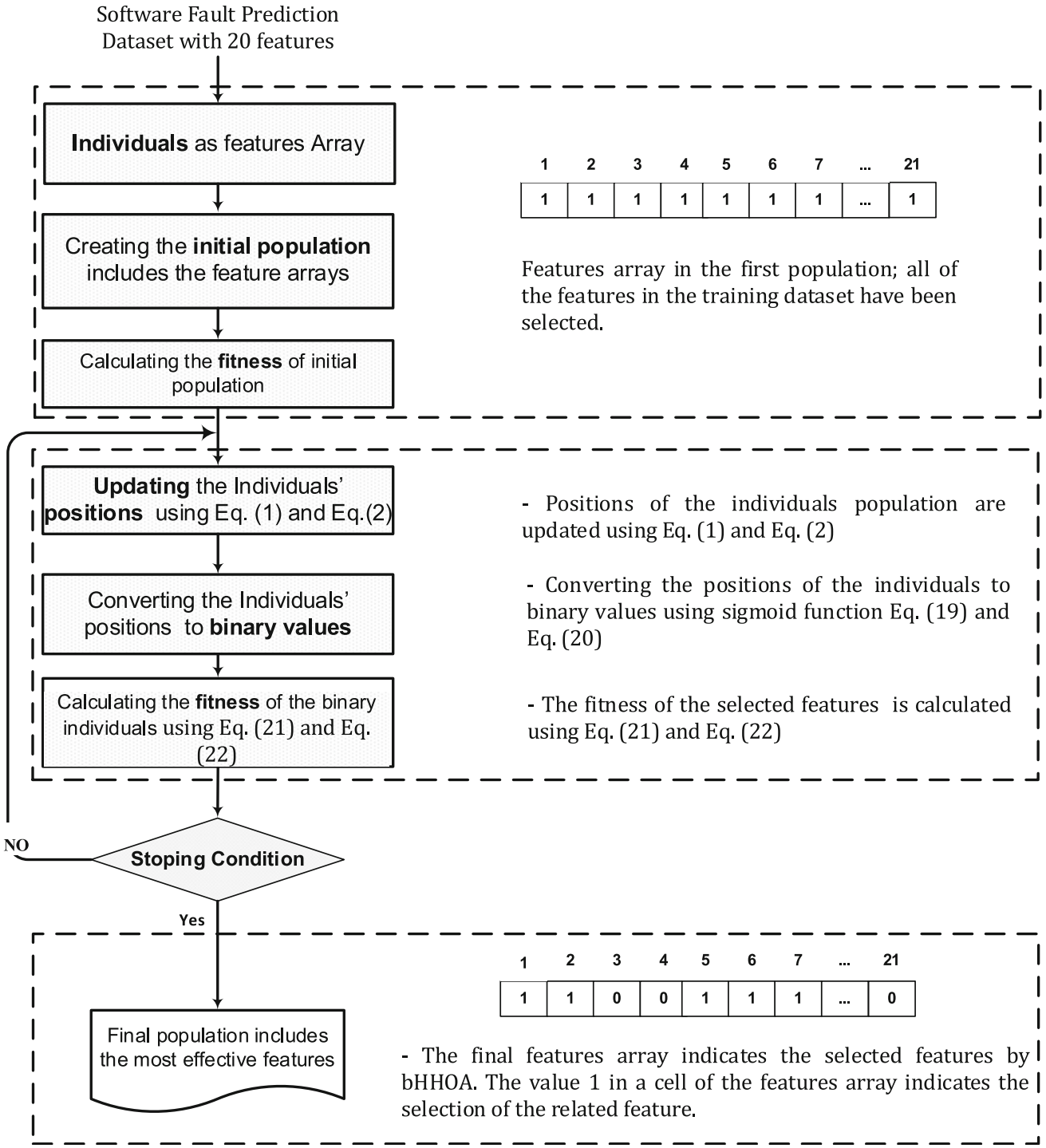


Fig. 3 The flowchart of the proposed bHHOA-based software defect predictor

the predictor's accuracy and precision, thus enhancing its overall classification effectiveness. In the bHHOA framework, each entity, symbolized as a horse, is represented by a binary vector with 21 positions, corresponding to the features in the training data set, labelled from $F1$ to $F21$. Each position in the binary vector aligns with a specific feature in the training data set, as shown in Fig. 4. A bit value of 0 signifies that the feature is excluded from the training process, whereas a bit value of 1 indicates its inclusion.

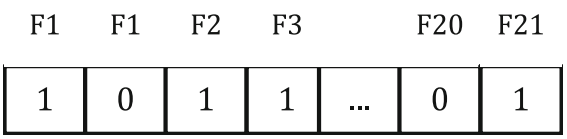


Fig. 4 The binary structure of an individual in bHHOA for specifying features of the training data set

Table 2 Overview of features utilized in the data sets (adapted from [28])

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

After applying bHHOA to a randomly selected feature, a fitness score is calculated for each binary feature vector. This fitness score is derived from the error of training and the quantity of features chosen, aiming to reduce both factors with each iteration. Consequently, the algorithm consistently works towards achieving the lowest possible training error with each new combination of selected features. By iteratively refining the feature selection, bHHOA enhances the classifier's performance, ensuring that the most relevant features are utilized while reducing unnecessary complexity.

3.2 Datasets for training

The data sets for this research were obtained from NASA's publicly available repository, established in 2005 [27]. These data sets incorporated a range of software metrics, such as McCabe's metrics, Halstead's metrics, branch count, and five unique measures related to lines of code. Table 2 outlines the features of the utilized datasets.

3.3 The features selection method

3.3.1 Horse Herd Optimisation Algorithm

Once the training data set is ready, the binary variation of the horse herd optimization algorithm (HHOA) is implemented for feature selection. This step focuses on pinpointing the most relevant features from the data set before constructing the classification model with machine learning algorithms. HHOA, a swarm-based method, utilizes both local and global search strategies, with each population member simulating the behaviour of a horse within a herd [29]. Each horse is depicted as a 21-length vector in this configuration, reflecting the number of features in the data set. The solution, or horse vector, is part of a population matrix that includes n rows and 21 columns. This population evolves over time as the members (horses) iteratively move and interact within the matrix. Figure 5 provides a flowchart that depicts the operation of HHOA.

The HOA was selected based on its merits against other heuristic algorithms. HOA makes a healthy balance between exploitation (local refining) and exploration (global search). The algorithm simulates how horses forage for food in a broad sense (exploration) and then concentrates on a rich food source (exploitation). This equilibrium lessens the possibility of becoming trapped in local optima. Another advantage of the HHOA is its dynamic population behaviour. The dynamics of the herd's movement enable optimization that is flexible. To obtain better answers, horses dynamically change their stance in response to their surroundings and one another. This flexibility can perform better than static methods like Genetic Algorithms, which could need more manual parameter tuning. Particularly for high-dimensional problems, HOA frequently converges more quickly than algorithms like Ant Colony Optimization or genetic algorithms because of its cooperative search approach. HOA generally uses fewer and simpler parameters than algorithms like GA, ACO PSO BAT, SFLA, and GWO. By employing HHOA for feature selection, the algorithm effectively narrows down the most impactful features, optimizing the training process and enhancing the overall performance of the classification model. This method leverages the collective intelligence of the horse herd, balancing exploration, and exploitation to achieve optimal feature selection.

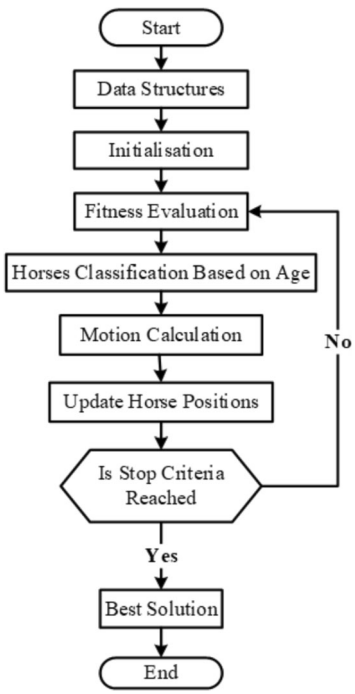


Fig. 5 Flowchartillustrating the binary HHOA applied to feature selection problem

HHOA simulates the movement of horses in each iteration according to Eq. 1. In this Equation, $X_m^{Iter,AGE}$ represents the position of the mth horse for a given age (AGE) and iteration ($Iter$), while $\vec{V}_m^{Iter,AGE}$ denotes velocity and the age range of the current horse, and $Iter$ signifies the current cycle. Considering that the average lifespan of horses ranges from 25 to 30 years, they display various behavioural patterns at different stages of their lives. In HHOA, horses' lifespans are divided into four distinct age categories to reflect these varying behaviours:

- δ: horses at age 0–5 years
- γ: horses at age 5–10 years
- β: horses at age 10–15 years
- α: horses at age 15 and older

$$X_m^{Iter,AGE} = \vec{V}_m^{Iter,AGE} + X_m^{(Iter-1),AGE}, AGE = \alpha, \beta, \gamma, \delta \tag{1}$$

Horses in HHOA are ranked according to their optimal responses. As a result, the top 10% of horses from a sorted matrix are selected as the leading group. The rest of the horses are distributed into three groups, each comprising 20%, 30%, and 40% of the population. The algorithm implements six distinct behaviours observed in horse herds to determine the velocity vector for horses of various ages. Equation 2 is used to define the velocity vector corresponding to horses of various ages during each iteration of the method.

$$\begin{aligned} \vec{V}_m^{Iter,\alpha} &= \vec{G}_m^{Iter,\alpha} + \vec{D}_m^{Iter,\alpha} \\ \vec{V}_m^{Iter,\beta} &= \vec{G}_m^{Iter,\beta} + \vec{H}_m^{Iter,\beta} + \vec{S}_m^{Iter,\beta} + \vec{D}_m^{Iter,\beta} \\ \vec{V}_m^{Iter,\gamma} = \vec{G}_m^{Iter,\gamma} &+ \vec{H}_m^{Iter,\gamma} + \vec{S}_m^{Iter,\gamma} + \vec{I}_m^{Iter,\gamma} + \vec{D}_m^{Iter,\gamma} + \vec{R}_m^{Iter,\gamma} \\ \vec{V}_m^{Iter,\delta} &= \vec{G}_m^{Iter,\delta} + \vec{I}_m^{Iter,\delta} + \vec{R}_m^{Iter,\delta} \end{aligned} \tag{2}$$

Horses, being grazing animals, graze throughout their lives at any age. This behaviour is modelled in HHOA by defining a grazing zone around each horse, represented by a *'g'* coefficient. This ensures that each horse grazes in specific areas. Equations 3 and 4 in the study mathematically describe how this grazing behaviour of horses is executed within the algorithm.

$$\vec{G}_m^{Iter,AGE} = g_{Iter}\left(\overset{\vee}{u} + \rho \overset{\vee}{l}\right) + [X_m^{(Iter-1)}], AGE = \alpha, \beta, \gamma, \delta \tag{3}$$

$$g_m^{Iter,AGE} = g_m^{(Iter-1),AGE}\omega_g \tag{4}$$

In Eqs. 3 and 4, $\vec{G}_m^{Iter,AGE}$ represents the movement parameter of the ith horse, illustrating the inclination of the ith horse to graze. During each iteration, this grazing component decreases linearly in accordance with the *'g'* coefficient. The upper bound and lower bound for grazing space in the Equation are represented by *'u'* and *'l'*, respectively, and are recommended to be set at 1.05 and 0.95. In this Equation, *'r'* would be a randomly generated value within the range of 0 to 1. The coefficient *'g'*, representing the grazing propensity, is also suggested to be set at 1.5 for all four age groups of horses.

In their natural behaviour, horses often follow a leader, who could be the most experienced and strongest horse or even a human in their herd. HHOA incorporates this natural tendency as a hierarchy, denoted by the coefficient *'h'*. This hierarchical behaviour is particularly observed in horses during their middle ages, from 5 to 15 years. Equations 5 and 6 are employed to mathematically express this behaviour.

$$\vec{H}_m^{Iter,AGE} = h_m^{Iter,AGE}\left[X_*^{(Iter-1)} - X_m^{(Iter-1)}\right], AGE = \alpha, \beta \text{ and } \gamma \tag{5}$$

$$h_m^{Iter,AGE} = h_m^{(Iter-1),AGE}\omega_h \tag{6}$$

In Eqs. 5 and 6, $\vec{H}_m^{Iter,AGE}$ represents the influence of the position of the best-performing horse on the velocity vector while and $X_*^{(Iter-1)}$ denotes the position of the best-performing horse from the previous iteration. As social creatures, horses naturally seek interaction and often live alongside other animals. The herd's presence offers protection against predators, enhancing their survival chances and reducing vulnerability. Due to their social nature, horses may sometimes engage in conflicts with each other, and their unique personalities can lead to occasional irritability.

It has been observed that horses generally flourish in the company of others. Particularly between the ages of 5 and 15, horses show a preference for staying within the herd. This behaviour is defined by a tendency to move towards the average positions of the other horses. HHOA models and captures this behaviour using the variable S, as detailed in Eqs. 7 and 8. This variable represents the tendency of horses to congregate and align their movements with the average position of the herd, which is called sociability.

$$\vec{S}_m^{Iter,AGE} = s_m^{Iter,AGE}\left[\left(\frac{1}{N}\sum_{j=1}^{N} X_j^{(Iter-1)}\right) - X_m^{(Iter-1)}\right], AGE = \beta, \gamma \tag{7}$$

$$S_m^{Iter,AGE} = s_m^{(Iter-1),AGE}\omega_s \tag{8}$$

In Eqs. 7 and 8, $\vec{S}_m^{Iter,AGE}$ is the social motion vector of the ith horse, indicating its movement in relation to the herd during the $Iter$th iteration, while $s_m^{Iter,AGE}$ refers to the orientation of the horse towards the herd during the same iteration. Together, these elements describe the horse's social behaviour during each iteration. The component ω_s gradually decreases the orientation towards the herd with each cycle. The variable N represents the total number of herd horses.

Horses tend to imitate each other, adopting both negative and positive behaviours and habits, such as seeking the best grazing spots. Young horses often mimic older ones, and this tendency continues until they mature. HHOA also incorporates this imitative behaviour (imitation), which is represented by variable i. This imitative tendency of horses can be mathematically modelled using Eqs. 9 and 10.

$$\vec{I}_m^{Iter,AGE} = i_m^{Iter,AGE}\left[\left(\frac{1}{pN}\sum_{j=1}^{pN} \hat{X}_j^{(Iter-1)}\right) - X^{(Iter-1)}\right], AGE = \gamma \tag{9}$$

$$i_m^{Iter,AGE} = i_m^{(Iter-1),AGE}\omega_i \tag{10}$$

In Eqs. 9 and 10, $\vec{I}_m^{Iter,AGE}$ is the movement vector of ith horse in the direction of the top horse with $\hat{X}$ locations. pN presents the number of horses positioned in the optimal locations. Parameter p is defined as ten per cent of the total number of horses, while ω_i presents a reducing factor per iteration for i_{Iter}. When faced with danger, horses present a fight-or-flight response, with their natural instinct leaning towards fleeing or running. Additionally, they may display defensive behaviour like bucking when captured. Horses inherently strive to secure food and water resources, driven by a need to avoid competition and dangerous encounters, such as those with predators like wolves. In the HHOA context, horses' defence mechanism is represented mathematically in Eqs. 11 and 12. In these equations, the behaviour of horses is modelled as moving away from individuals displaying incorrect

behaviours, with the parameter d describing this behaviour. This behaviour is represented by a negative coefficient, which effectively prevents horses from adopting unsuitable postures or engaging in inappropriate actions.

$$\vec{D}_m^{Iter,AGE} = -d_m^{Iter,AGE}\left[\left(\frac{1}{qN}\sum_{j=1}^{pN}\overset{\vee}{X}_j^{(Iter-1)}\right) - X^{(Iter-1)}\right], AGE = \alpha, \beta \text{ and } \gamma \tag{11}$$

$$d_m^{Iter,AGE} = d_m^{(Iter-1),AGE}\omega_d \tag{12}$$

In Eqs. 11 and 12, $\vec{D}_m^{Iter,AGE}$ represents the scape factor of the ith horse from the average of the horses in the worst position denoted by the $\overset{\vee}{X}$ vector. qN represents the total quantity of horses in the worst positions. ω_d refers to the decrement factor per d_{Iter}, and it is recommended that q be set to 20% of the total number of horses.

The sixth horse behaviour integrated into HHOA is their tendency to wander. Occasionally, a horse may decide to graze in a different location. Horses are inherently curious creatures, driven by a desire to explore new areas and become acquainted with their environment. For this reason, stable walls are often constructed in a way that allows horses to see each other, enabling them to satisfy their curiosity about their fellow horses. Roaming is commonly observed in young horses, but this behaviour diminishes as they mature. In HHOA, this behaviour is simulated as random movement, represented by factor r, as described in Eqs. 13 and 14.

$$\vec{R}_m^{Iter,AGE} = r_m^{Iter,AGE} pX^{(Iter-1)}, AGE = \gamma \text{ and } \delta \tag{13}$$

$$r_m^{Iter,AGE} = r_m^{(Iter-1),AGE}\omega_r \tag{14}$$

In Eqs. 13 and 14, $\vec{R}_m^{Iter,AGE}$ is the random velocity vector of the ith horse to local search, while ω_r refers to the decrement factor of $r_m^{Iter,AGE}$ per cycle. The velocities of the δ, γ, β, and α horses are calculated using Eqs. 15 through 18, respectively.

$$\begin{aligned}\vec{V}_m^{Iter,\delta} = &\left[g_m^{(Iter-1),\delta}\omega_g\left(\overset{\vee}{u} + \rho\overset{\vee}{l}\right) + [X_m^{(Iter-1)}]\right] + \left[i_m^{(Iter-1),\delta}\omega_i\left[\left(\frac{1}{pN}\sum_{j=1}^{pN}\hat{X}_j^{(Iter-1)}\right) - X^{(Iter-1)}\right]\right] \\ &+ \left[r_m^{(Iter-1),\delta}\omega_r pX^{(Iter-1)}\right]\end{aligned} \tag{15}$$

$$\begin{aligned}\vec{V}_m^{Iter,\gamma} = &\, g_m^{(Iter-1),\gamma}\omega_g(\check{u} + \rho\check{u}) + [X_m^{(Iter-1)}] + \left[h_m^{(Iter-1),\gamma}\omega_h\left[X_*^{(Iter-1)} - X_m^{(Iter-1)}\right]\right] \\ &+ \left[s_m^{(Iter-1),\gamma}\omega_s\left[\left(\frac{1}{N}\sum_{j=1}^{N}X_j^{(Iter-1)}\right) - X_m^{(Iter-1)}\right]\right] + \left[i_m^{(Iter-1),\gamma}\omega_i\left[\left(\frac{1}{pN}\sum_{j=1}^{pN}\hat{X}_j^{(Iter-1)}\right) - X^{(Iter-1)}\right]\right] \\ &- d_m^{(Iter-1),\gamma}\omega_d\frac{1}{qN}\sum_{j=1}^{pN}\check{X}_j^{(Iter-1)} - X^{(Iter-1)}\left[r_m^{(Iter-1),\gamma}\omega_r pX^{(Iter-1)}\right]\end{aligned} \tag{16}$$

$$\vec{V}_m^{Iter,\beta} = \left[g_m^{(Iter-1),\beta} \omega_g \left(\overset{\vee}{u} + \rho \overset{\vee}{I} \right) + [X_m^{(Iter-1)}] \right] + \left[h_m^{(Iter-1),\beta} \omega_h \left[X_*^{(Iter-1)} - X_m^{(Iter-1)} \right] \right] + \left[s_m^{(Iter-1),\beta} \omega_s \left[\left(\frac{1}{N} \sum_{j=1}^{N} X_j^{(Iter-1)} \right) - X_m^{(Iter-1)} \right] \right] - \left[d_m^{(Iter-1),\beta} \omega_d \left[\left(\frac{1}{qN} \sum_{j=1}^{pN} \overset{\vee}{X}_j^{(Iter-1)} \right) - X^{(Iter-1)} \right] \right] \tag{17}$$

$$\vec{V}_m^{Iter,\alpha} = \left[g_m^{(Iter-1),\alpha} \omega_g \left(\overset{\vee}{u} + \rho \overset{\vee}{I} \right) + [X_m^{(Iter-1)}] \right] - \left[d_m^{(Iter-1),\alpha} \omega_d \left[\left(\frac{1}{qN} \sum_{j=1}^{pN} \overset{\vee}{X}_j^{(Iter-1)} \right) - X^{(Iter-1)} \right] \right] \tag{18}$$

3.3.2 Binary HHOA

This section presents an innovative binary method for the horse herd optimization algorithm (HHOA), utilizing the sigmoid function. In this approach, each solution (horse) in the population initially has a continuous numerical value. However, as the task involves determining whether to include or exclude a feature, the new binary solution (horse) must be composed of binary values, either 0 or 1, as shown in Fig. 6. A feature is retained in the new data set if its corresponding value is 1 and omitted if the value is 0.

The Sigmoid function [35, 86] is utilized to convert the continuous values within the individual array of the HHOA into binary values. As depicted in Fig. 6, the new binary HHOA (bHHOA) is employed to select relevant features for ML algorithms. A binary version is essential for tasks like feature selection, where solutions are

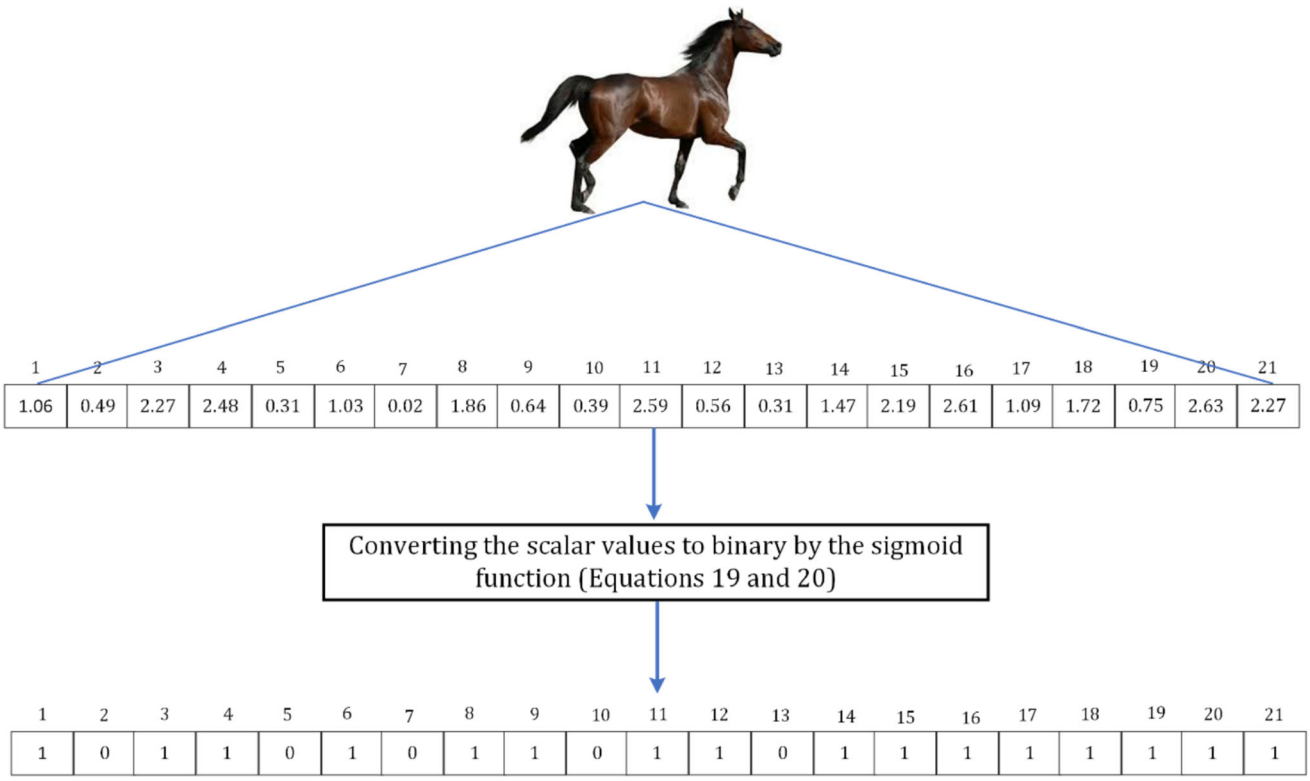


Fig. 6 The created binary vector by bHHOA for selecting the training set's optimal features

restricted to binary values (0 or 1). In bHHOA, Eq. 1 updates the position of each horse with $X_i^{Iter,AGE}$ indicating the *i*th horse's position in iteration Iter.

In the new model, the binary form of the Sigmoid function is utilized to adjust the solution's position in HHOA, as detailed in Eq. 19. The Sigmoid function's output ranges between zero and one, and a threshold is applied to convert this output into a binary (discrete) value. Equation 20 applies the Sigmoid function with a random threshold to transform the continuous solution into a binary one. In this context, 'rand' represents a number that is uniformly distributed between 0 and 1. By applying Eqs. 19 and 20, the solutions within the bHHOA population are constrained to operate within a binary (discrete) search space, thus integrating these equations precisely into the bHHOA framework. In Eq. 19, variable c is a coefficient and adapted to 0.5. This binary approach ensures that the feature selection process is appropriately adapted for tasks requiring binary decisions, thereby improving the efficiency and effectiveness of the ML algorithms utilized in this study.

$$\mathrm{SG}\left(X_i^{Iter,AGE}\right) = \frac{1}{1+\mathrm{e}^{-X_i(AGE-c)}} \tag{19}$$

$$\mathrm{Binary}\ (X_i^{Iter,AGE}) = \begin{cases} 0\text{if rand} < \mathrm{SG}\left(X_i^{Iter,AGE}\right) \\ 1\text{if rand} \geq \mathrm{SG}\left(X_i^{Iter,AGE}\right) \end{cases} \tag{20}$$

Equations 21 and 22 are introduced to assess the fitness of feature vectors, focusing on minimizing both errors of training and the number of selected features. In Eq. 21, the training error is determined by subtracting the accuracy rate from 100 and then normalizing it to a scale ranging from 0 to 1. Similarly, Eq. 22 accounts for the quantity of features used in training, that is similarly scaled to fit within the [0, 1] range. These equations together evaluate the fitness of the selected feature set by balancing the model's error rate with the complexity added by the number of features included.

$$\mathrm{Fitness}(i) = \mathrm{ErrorPercentageTest}(i) + \mathrm{Used\ Features} \tag{21}$$

$$\mathrm{ErrorPercentageTest}(i) = (100 - \mathrm{Accuracy}(i))/100 \tag{22}$$

These equations serve to assess the fitness of the selected feature set by taking into account both the error rate of the model and the complexity introduced by the number of features used. By minimizing the combination of these two factors, the algorithm aims to find a balance between model accuracy and simplicity, ensuring that the most relevant features are selected without introducing unnecessary complexity.

4 Experimental setup and findings

4.1 The platform of the experiments

This section examines the outcomes generated by the new SDP approach. At first, machine learning algorithms (ANN, DT, KNN, NB, and SVM) are applied traditionally using the entire set of features. The results are then contrasted with those produced by the binary horse herd optimization algorithm (bHHOA), specifically designed for optimized feature selection. A comparison between both methods is subsequently carried out. The proposed method, alongside the binary genetic algorithm (bGA), binary Ant Colony Optimizer (bACO), binary Particle Swarm Optimizer (bPSO), and a hybrid binary particle swarm optimizer, and genetic algorithm (bPSOGA), was executed using MATLAB 2022 on a machine with Windows 10 equipped with a 2.0-GHz Intel Core i7 processor

and 8 GB of RAM. Table 3 provides details on the calibration parameters for both the bHHOA and bACO techniques within the context of SDP. It is worth noting that binary ACO had already been presented as a tool for feature selection in the study by [30]. This comparative approach provides a comprehensive evaluation of the new method's effectiveness in SDP. The study seeks to demonstrate the advantages of bHHOA by comparing its performance with other binary optimization algorithms, focusing on improvements in accuracy, precision, and overall predictive effectiveness.

As explained in Table 4, five data sets were used for training and testing. Initially, 70% of the data were randomly selected to train different ML models, including ANN, DT, KNN, NB, and SVM. The remaining 30% was reserved for testing to evaluate the models' performance. Table 4 details the attributes of the data used in both phases. This approach enables a thorough evaluation of the models across different data sets and performance indicators.

4.2 Evaluation criteria

The current study evaluates software module classification problems using four primary metrics: accuracy, precision, recall, and the $F1$ score. These metrics are calculated using specific equations, identified as Eqs. 23–26, for determining accuracy, precision, recall, and the $F1$ score, respectively. These metrics thoroughly assess the classification models' effectiveness in predicting software defects.

Table 3 The implemented feature selector algorithms' configuration parameters and their values

	Value
bGA configuration parameters	
Length of chromosome	21
Crossover rate	0.80
Mutation rate	0.05
Termination condition	100 Iterations
bACO configuration parameters	
Pheromone (τ)	1.0
The Q parameter's value	1.0
Pheromone weight (α)	1.0
Evaporation rate (ρ)	0.1
Stop condition	100 Iterations
bPSO configuration parameters	
Inertiaweight	0.8
$C1$	2
$C2$	2
Num. of particles	25
Termination condition	100 Iterations
bHHOA configuration parameters	
Quantity of horses	30
$h\beta$	0.90
h_γ	0.50
S_β	0.20
S_γ	0.10
d_β	0.20
S_γ, R_δ	0.10
R_γ	0.05

Table 4 Characteristics of the training and test data set

Number	Project name	Num. of records (modules)	Num. of records in training dataset	Num. of records in testing dataset	Num. of features
1	KC2	522	365	157	21
2	PC1	1109	776	333	21
3	KC1	2109	1476	633	21
4	CM1	498	349	149	21
5	JM1	10,885	7619	3266	21

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{All test samples}} \tag{23}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{24}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{25}$$

$$F_1 = \frac{2 * (\text{Precision} * \text{Recall})}{\text{Precision} + \text{Recall}} \tag{26}$$

The confusion matrix is frequently used to evaluate the accuracy and performance of classification models. Within classification and prediction tasks, it distinguishes between the following four possible outcomes:

- True Positive (TP): quantity of positive instances correctly identified as belonging to the positive class.
- False Positive (FP): quantity of negative instances incorrectly classified as belonging to the positive class.
- False Negative (FN): quantity of positive instances incorrectly classified as belonging to the negative class.
- True Negative (TN): quantity of negative instances correctly identified as belonging to the negative class.

The confusion matrix plays a vital role in generating key performance metrics, including accuracy, precision, recall, and $F1$ score, which are fundamental for assessing the effectiveness of software module classification models. Extensive tests were carried out on classifiers developed using five different machine learning algorithms. These experiments address the following research questions across the data sets under consideration:

RQ1 Does the new approach improve the accuracy, precision, recall, and $F1$ score of machine learning algorithms in SDP?

RQ2 Is the new method successful in identifying and removing irrelevant features from the training data sets?

RQ3 Does the bHHOA demonstrate superior speed of convergence and rate of success compared to other techniques?

4.3 Evaluating the new feature selector's performance on the CM1 data set

The first experiment sought to answer RQ1 by applying the CM1 data set for training and testing purposes. In the first experiment, the accuracy, precision, recall, and $F1$ score of the classifiers generated by (ANN, DT, KNN, NB, and SVM) algorithms were calculated. At the outset, the experiments were carried out using the complete training set, encompassing all available features. Overfitting in machine learning algorithms can occur due to

irrelevant features in the training dataset. The key aim of feature selection is to determine a minimal yet effective set of features capable of accurately predicting the target variables. While these features can distort the ML models' performance, feature selection techniques streamline the process by removing unnecessary data, thus reducing computational costs. Additional experiments were carried out across each data set using bHHOA in combination with various ML algorithms. The purpose of bHHOA was to isolate the most relevant features in the training data. The results indicate that including bHHOA leads to more effective SDP outcomes. For instance, the ANN classifier's accuracy and precision increased from 88.15% and 88.46%, respectively, without bHHOA, to 99.259% and 95.120% with its inclusion. These results demonstrate that incorporating bHHOA significantly improves the performance of ML algorithms in SDP.

Figure 7 compares performance metrics between standard ML algorithms and those improved by bHHOA, a feature selection technique applied to the CM1 data set. The results of the simulation show that machine learning algorithms achieve superior performance when trained on data sets with features optimally selected by bHHOA. By incorporating bHHOA for effective feature selection, the resulting ML+bHHOA approach shows improved performance compared to the ML algorithms alone. In summary, the proposed bHHOA method for feature selection is highly effective when combined with machine learning algorithms on CM1.

A comparison of the average performance of classifiers built using machine learning algorithms, with and without feature selection, is depicted in Fig. 8. Classifiers developed with no feature selection achieve an average accuracy of 87.02%, which drastically enhances to 94.71% when the bHHOA feature selection is incorporated. As depicted in Fig. 8, the proposed bHHOA+ML approach demonstrates considerably higher accuracy than the other methods. This enhancement is evident in other metrics of performance as well. For example, the average precision of classifiers increases from 92.63% without feature selection to 94.87% when bHHOA is applied. The

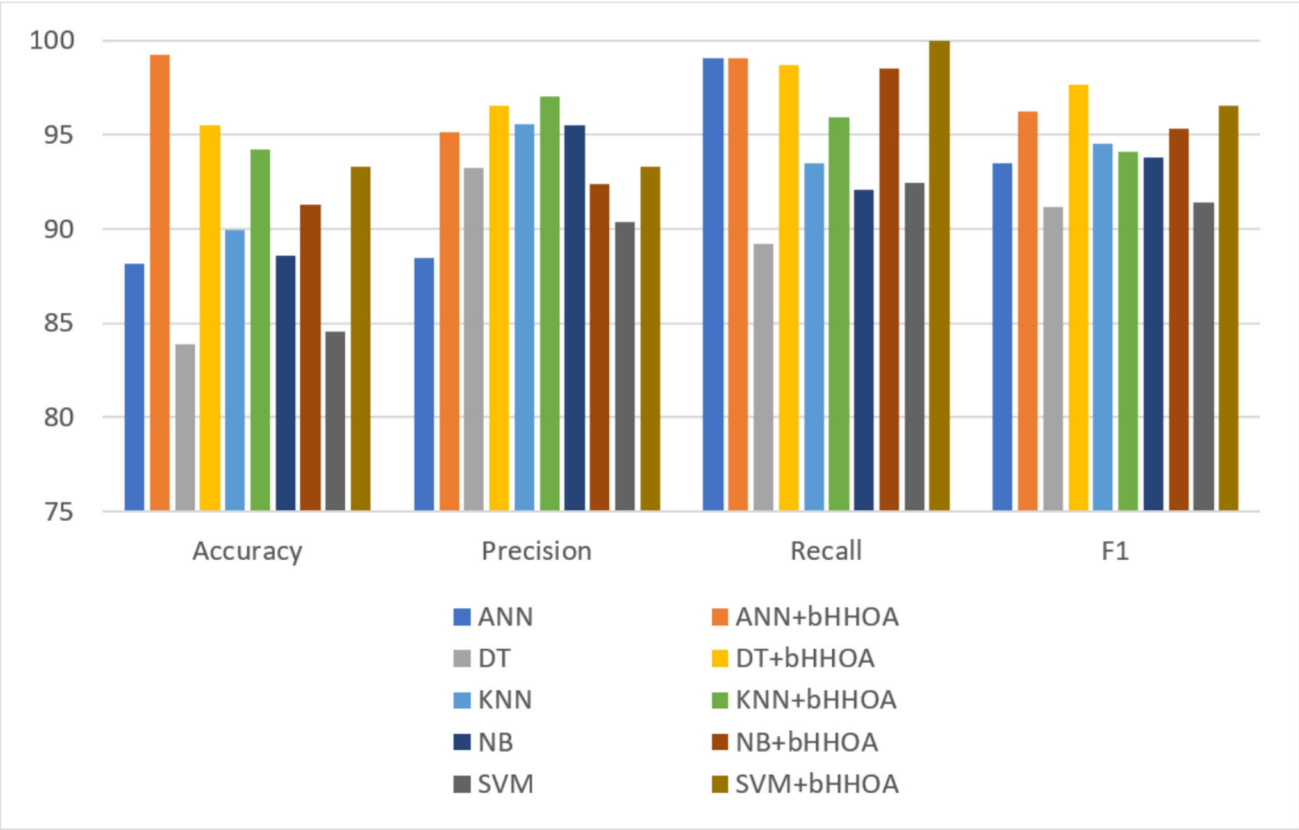


Fig. 7 The influence of bHHOA on machine learning algorithm performance in the CM1 Data set

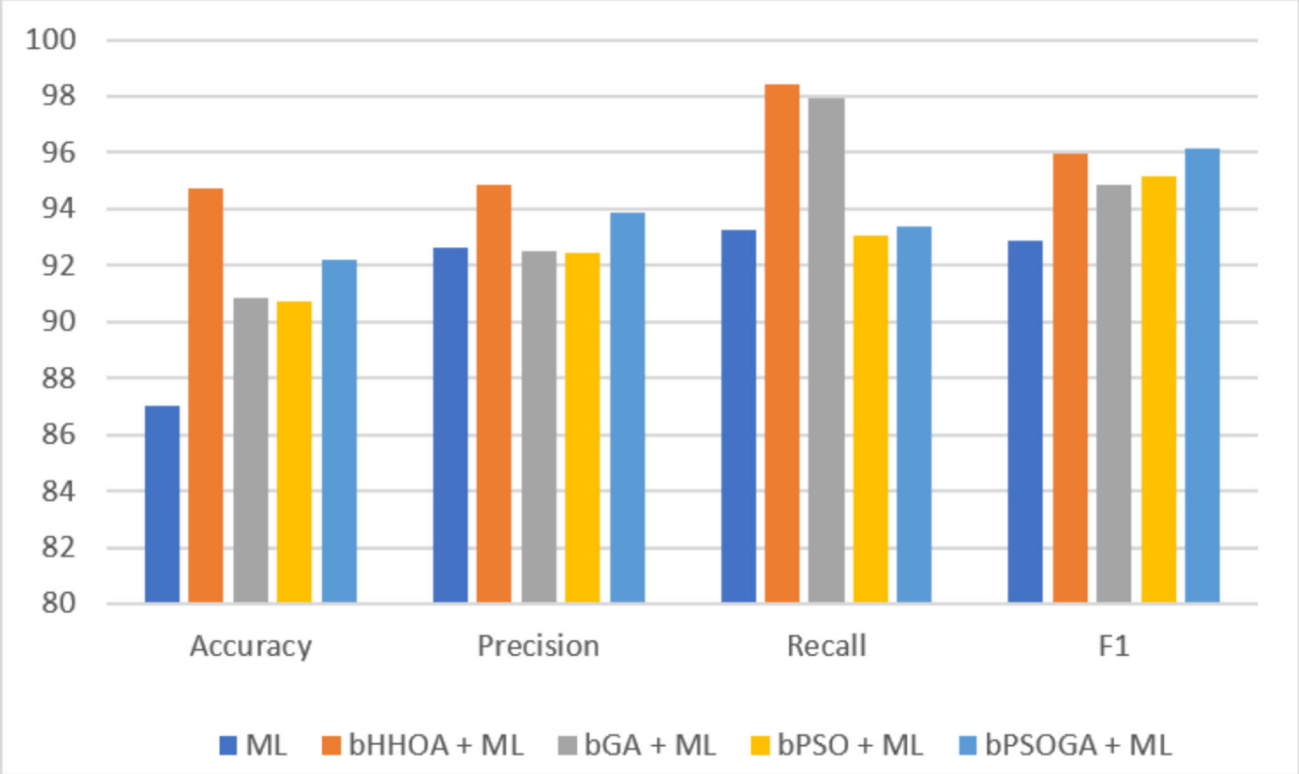


Fig. 8 Average performance metrics of ML algorithms, comparing results with and without feature selection on CM1 data set

proposed method also excels in metrics like average recall and $F1$ score. As a result, the bHHOA combined with machine learning (bHHOA+ML) proves to be more effective than classifiers produced by alternative methods.

The number of features chosen by the algorithm plays a critical role in defect prediction methods that rely on feature selection. Irrelevant features could hinder a model's generalization capabilities and decrease its overall accuracy. Furthermore, more variables lead to increased model complexity. The new feature selection algorithm effectively narrows down the most critical features, enhancing the defect prediction's accuracy. Figure 9 shows the quantity of essential features determined by bHHOA for different ML algorithms. In the CM1 data set, which is a real-world defect prediction data set containing 21 independent features and one dependent feature, bHHOA identifies around eight features as the ideal subset. In contrast, the average number of features selected by bGA, bACO, bPSO, and bPSOGA are 9, 9, 12, and 10, respectively.

The average performance of the new method was evaluated across ten iterations, showcasing its reliable effectiveness in defect detection. Figure 10 compares the average number of features selected by bGA, bACO, bPSO, bPSOGA, and bHHOA across ten runs with different machine learning algorithms. Specifically, in the context of ANN classification, the average quantity of selected features by the five feature selectors is 11. Conversely, the same feature selectors chose an average of 14 features. For NB classification, an average of 8 features were selected. According to the results shown in Figs. 9 and 10, bHHOA selected 3 and 5 features when used with NB and ANN, respectively. Notably, bHHOA consistently selected fewer features than the other feature selection algorithms, indicating its greater efficiency in identifying the most impactful features.

Table 5 highlights the features selected from the CM1 data set by bHHOA in the most successful execution out of ten runs. Ten iterations were conducted for each ML algorithm and feature selector, with the features that contributed to the highest fitness values being considered the most effective. The table lists the distinct features selected by bHHOA, highlighting differences in feature selection. In conclusion, the analysis identifies the most impactful features chosen by bHHOA in the CM1 data set, as detailed below:

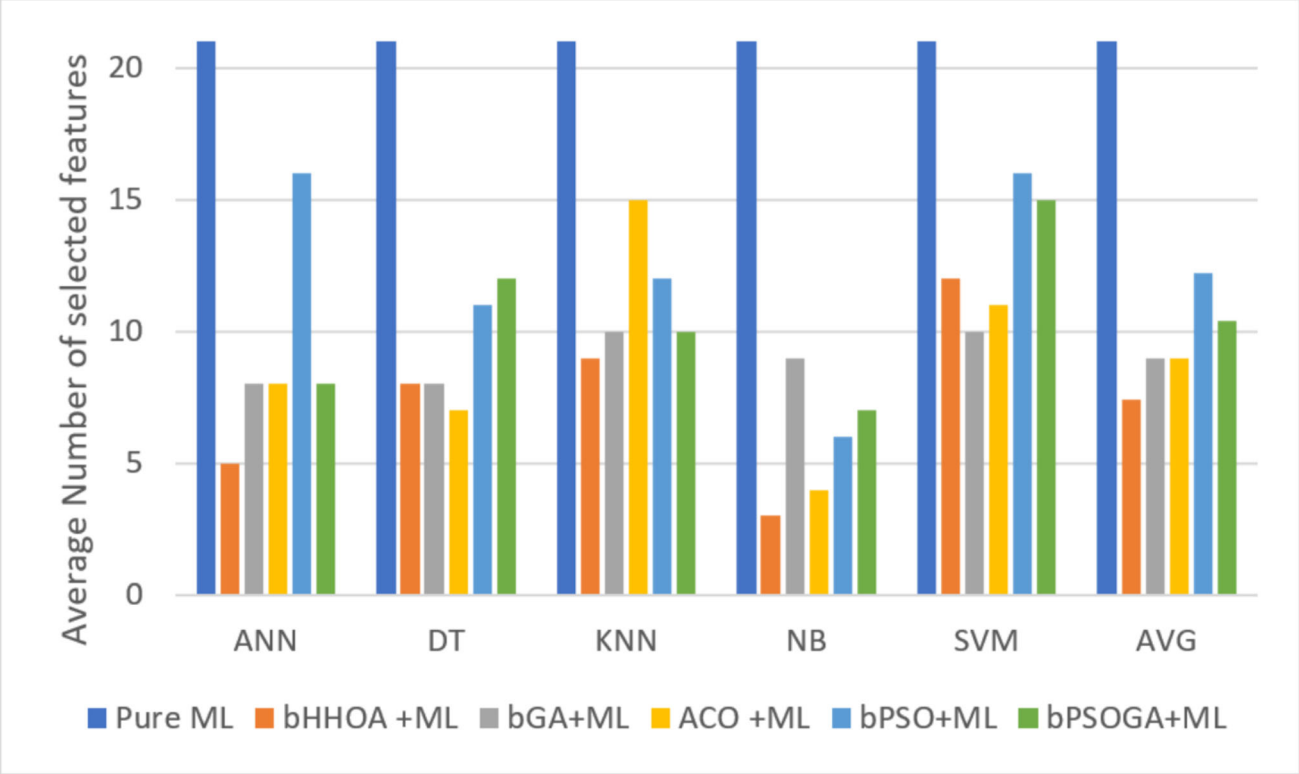


Fig. 9 The average number of features selected by different feature selection methods across different machine learning algorithms in CM1 data set over the course of ten iterations

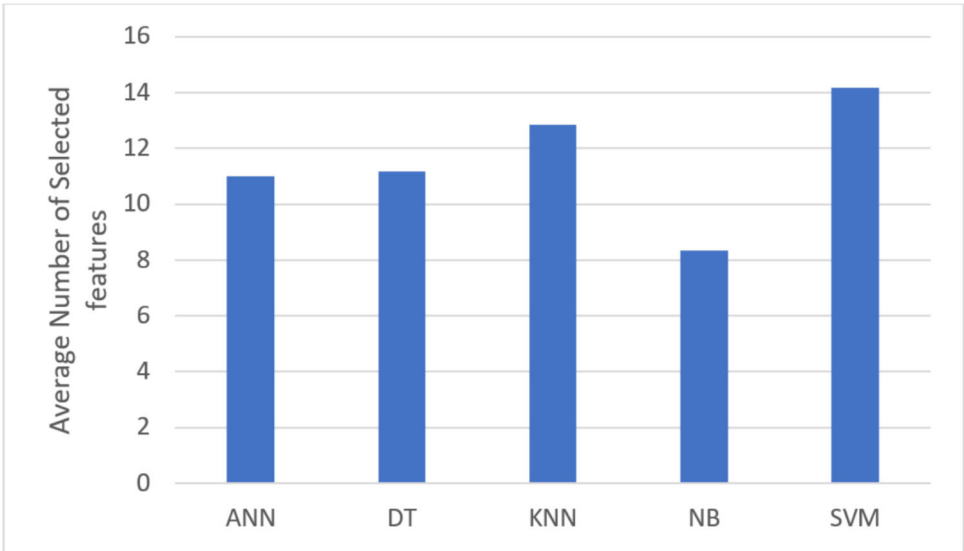


Fig. 10 The average number of features selected by various feature selectors (bGA, bACO, bPSO, bPSOGA, and bHHOA) in different machine learning algorithms in the CM1 data set

1. Cyclomatic complexity (McCabe)
2. Essential complexity (McCabe)
3. Operators + Operands (McCabe)
4. Design complexity (McCabe)

Table 5 Features selected from the CM1 data set by bHHOA in its most successful run

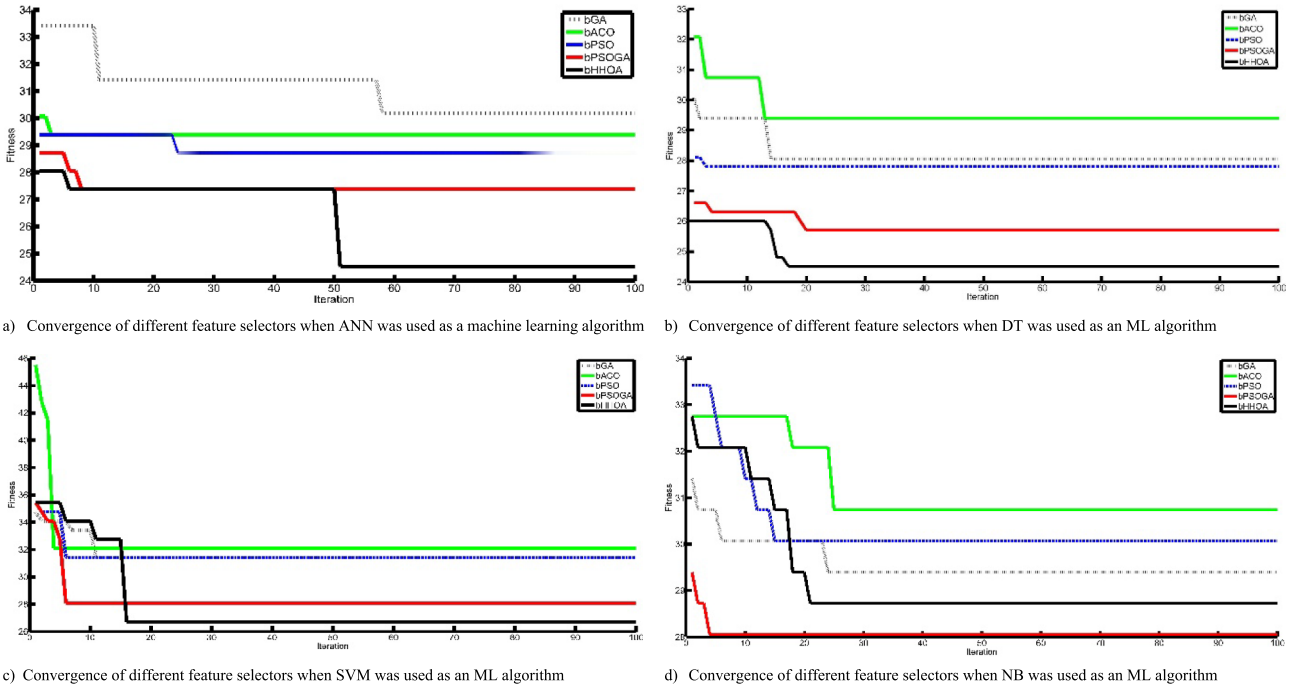
	ANN	DT	KNN	NB	SVM
LOC	1	0	1	0	0
v(g)	1	1	0	0	1
eV(g)	0	1	1	0	1
iv(g)	0	1	1	1	0
N	0	1	1	0	1
v	0	0	1	0	0
L	0	1	1	0	0
D	0	1	1	0	1
I	1	0	0	1	0
B	0	1	1	0	1
E	0	0	0	0	1
T	1	0	1	0	1
LOCode	1	0	0	0	0
LOComment	0	1	1	0	1
LOBlank	0	0	0	0	0
LOCodeAndComment	0	1	0	1	0
uniq_Op	0	1	1	0	1
uniq_Opnd	1	0	1	0	1
total_Op	0	0	0	0	0
total_Opnd	0	1	0	0	1
branchCount	1	0	1	1	0

5. Difficulty (Halstead)
6. Quantity of delivered bugs (Halstead)
7. Time to write the programme (Halstead)
8. Quantity of branches in the flow graph (Halstead)
9. unique operators (Halstead)
10. unique operands (Halstead)

To validate the consistency of the new method, its execution on CM1 was replicated ten times, with results meticulously documented. The findings revealed drastic enhancements in both accuracy and precision. Figure 11 illustrates the convergence of various implemented feature selectors in their best cases over ten executions. As depicted in Fig. 11, the proposed feature selector exhibited a higher convergence speed and a better (lower) fitness value compared to bGA, bACO, bPSO, and bPSOGA. By employing the proposed algorithm, only the most effective features out of the 21 available were utilized for training and testing. This method not only reduced the run time of the learning algorithm but also enhanced its overall efficiency. Some of the features were repeatedly chosen by bHHOA during training, suggesting they were more effective than the rest. Figure 10 provides a visual representation of each feature's influence on the predictions made within the CM1 data set, highlighting the relative importance of these selected features.

4.4 Evaluating the new feature selector's performance on the KC1 data set

A second set of experiments was conducted with the KC1 dataset to address the research questions. This dataset, sourced from a real-world software application, comprises 2107 instances and 21 features, with 1475 instances allocated for training and 633 for testing. The developed classifiers by ANN, DT, KNN, NB, and SVM were trained on all 21 features and evaluated based on accuracy, precision, recall, and $F1$ score. The KNN-based classifier that employed bHHOA for feature selection achieved the greatest accuracy and precision in the initial tests using the full training dataset. Conversely, the ANN-based defect predictor excelled in recall and $F1$ score



a) Convergence of different feature selectors when ANN was used as a machine learning algorithm

b) Convergence of different feature selectors when DT was used as an ML algorithm

c) Convergence of different feature selectors when SVM was used as an ML algorithm

d) Convergence of different feature selectors when NB was used as an ML algorithm

Fig. 11 The convergence of different feature selectors with various machine learning algorithms in CM1 data set

compared to the other models. Redundant features in training data sets can cause overfitting in machine learning models, potentially skewing their predictive accuracy. The KC1 data set was used as a benchmark to assess feature selection's influence on ML performance. In a follow-up experiment, the KC1 data set was processed using bHHOA alongside various ML algorithms. For example, the ANN classifier, without bHHOA, achieved 84.65% accuracy and 85.01% precision. With bHHOA, these values increased significantly to 86.57% accuracy and 87.76% precision. These results show that integrating bHHOA improves the performance of ML algorithms for SDP when used in the KC1 data set. The obtained results highlight that selecting key features with bHHOA enhances defect prediction accuracy compared to methods with no feature selection.

Figure 12 compares the performance metrics for machine learning algorithms on the KC1 data set, highlighting those enhanced with bHHOA for feature selection versus traditional ML methods. The results indicate that ML algorithms using feature-selected training data sets outperform conventional approaches. The figure compares standard ML models and those optimized with the ML+bHHOA approach on the KC1 data set. Integrating bHHOA for feature selection significantly boosts performance by utilizing the most relevant features. Applying bHHOA as a feature selector in ML algorithms yields impressive results on the KC1 data set. Machine learning models optimized using bHHOA achieve higher accuracy, precision, recall, and $F1$ scores compared to traditional models. In conclusion, incorporating bHHOA for effective feature selection significantly improves the performance of ML algorithms on the KC1 data set.

Figure 13 shows the average performance of classifiers built employing ML algorithms on the KC1 data set, with and without feature selection. The combination of bHHOA with ML algorithms results in higher average accuracy, with standard ML classifiers achieving 84.59% compared to 87.78% for those enhanced by bHHOA (ML+bHHOA). This trend extends across other performance metrics as well. For example, average precision improves from 87.05% with standalone ML algorithms to 89.36% when bHHOA is applied. Classifiers created using the proposed method also exhibit significantly higher recall compared to those developed solely with ML algorithms. In summary, the classifiers using bHHOA consistently outperform those without feature selection, as bHHOA effectively improves defect prediction models by removing irrelevant features. After ten iterations, the average performance of the new approach on KC1 data set confirms its strong defect detection capability.

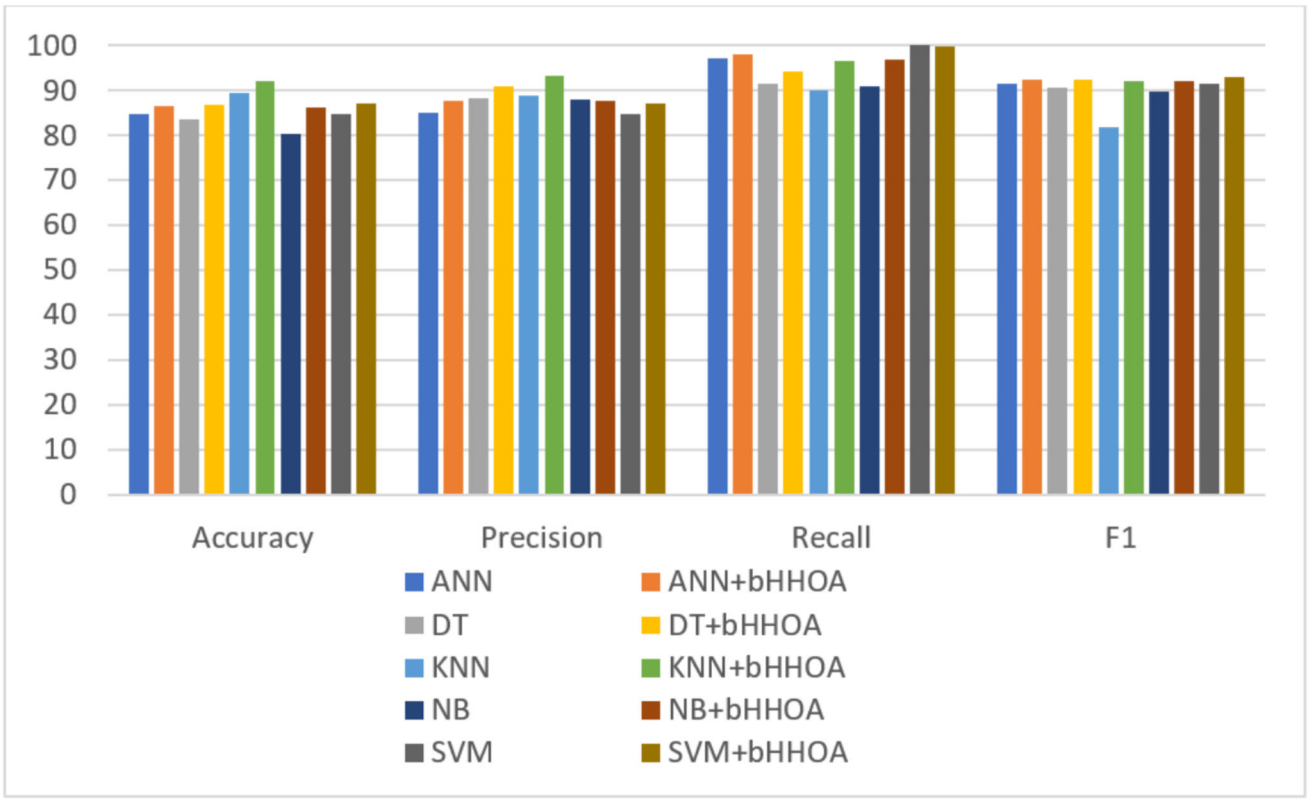


Fig. 12 The influence of bHHOA on machine learning algorithm performance in the KC1 data set

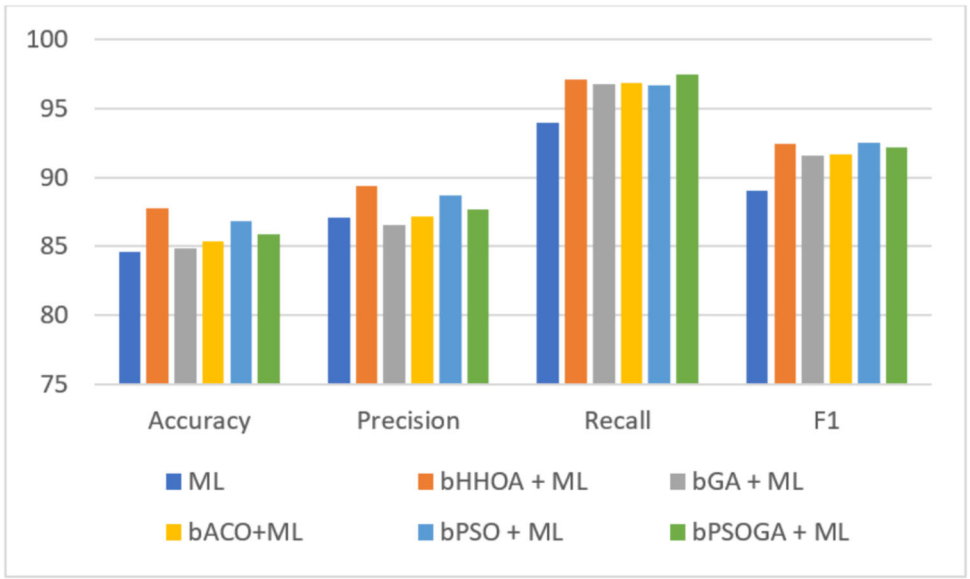


Fig. 13 Average performance metrics of machine learning algorithms on the KC1 data set, with and with no feature selection methods

Figure 14 shows the average number of selected features by various feature selectors (bGA, bACO, bPSO, bPSOGA, and bHHOA) over ten runs within the KC1 data set. For the ANN classifier, bHHOA selected 20 features, while bPSO selected 16. For the DT classifier, bHHOA and bPSO selected 7 and 10 features, respectively. In the context of KNN, bHHOA identified eight features, while bPSO identified 14 features. For the NB classification algorithm, bHHOA identified six features, whereas bACO selected nine. In general, bHHOA consistently chooses fewer features compared to bACO, bPSO, and bPSOGA. The results shown in Fig. 13

suggest that the overall effectiveness (accuracy, precision, recall, and $F1$ score) of the new method (bHHOA+ML) is greater than that of standalone ML and other feature selection algorithms. The bHHOA+ML method excels in identifying the most crucial features, thereby creating the most efficient fault predictor with a minimal number of features. According to Fig. 14, while bGA selects the fewest features (8), the accuracy and precision of the fault predictor developped by bGA are lower than those of the other predictors. This indicates that the features selected by bGA and bACO are not the most effective ones.

Figure 15 depicts the quantity of selected features from the KC1 data set by various feature selectors (bGA, bACO, bPSO, bPSOGA, and bHHOA) across different ML algorithms over ten executions. On average, the feature selectors chose about eight features for ANN, while for NB, the average number of selected features was approximately 5. The lower performance of the fault predictors created by NB suggests that some effective features were eliminated during the selection process. Indeed, a balance must be struck between the quantity of features selected and the performance of the resulting fault predictor. The fitness function (Eq. 21) takes into account both the rate of error and the quantity of features selected to achieve this balance.

Table 6 provides an overview of the most efficient features identified by bHHOA in its best-performing runs. In conclusion, the key features in KC1 data set, pinpointed by bHHOA, are as below:

1. Programme length (Halstead)
2. Difficulty (Halstead)
3. Number of Delivered Bugs (Halstead)
4. Effort: Halstead effort to write a programme
5. Time to write the programme (Halstead)
6. Total operands (Halstead)
7. line count of code (McCabe)
8. Cyclomatic complexity (McCabe)
9. Operators + Operands (McCabe)
10. Quantity of branches in the flow graph (Halstead)

The proposed method was applied to the KC1 data set ten times to verify its consistency, showing marked improvements in precision and accuracy. This technique considers 21 features and selectively employs the most impactful ones for classifier training. Throughout the training phase with bHHOA, certain features were consistently selected, underscoring their crucial role in enhancing the resulting classifier's accuracy and precision. Figure 16 provides the convergence speed of different heuristic algorithms in finding the optimal features.

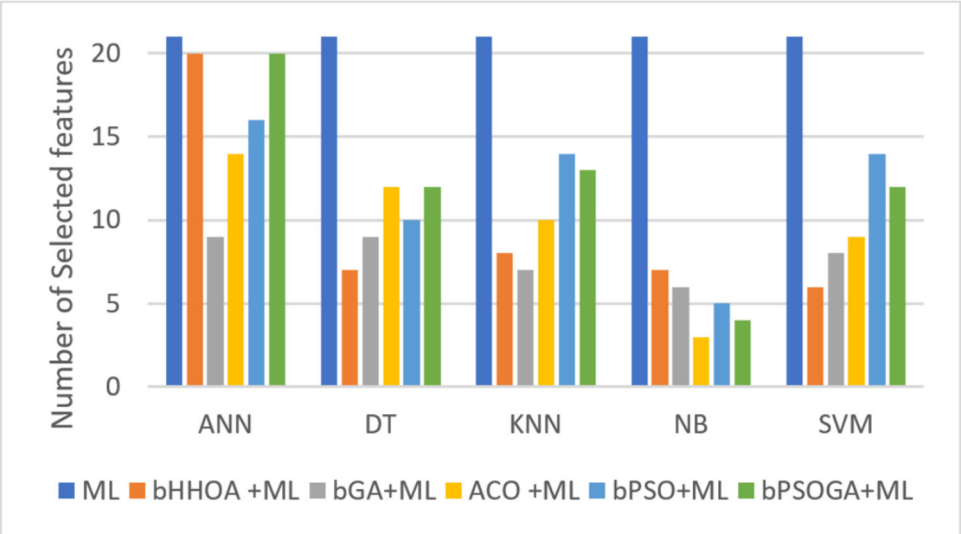


Fig. 14 The average quantity of selected features by various feature selectors (bGA, bACO, bPSO, bPSOGA, and bHHOA) across different machine learning algorithms in KC1 data set throughout ten runs

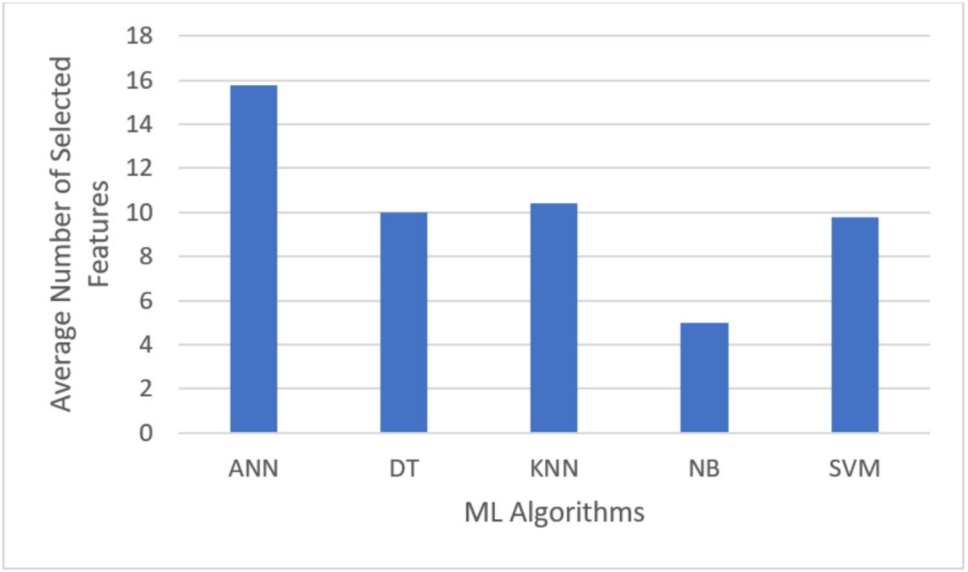


Fig. 15 Average quantity of features selected by various feature selectors (bGA, bACO, bPSO, bPSOGA, and bHHOA) across various machine learning algorithms on the KC1 data set

Table 6 The features selected by bHHOA across various machine learning algorithms in KC1 achieved the best results from 10 runs

	ANN	DT	KNN	NB	SVM
LOC	1	0	1	0	1
v(g)	1	1	0	0	1
eV(g)	1	0	0	0	0
iv(g)	1	0	0	0	0
N	1	0	1	1	0
V	1	0	0	0	0
L	1	1	0	1	1
D	1	1	1	1	0
I	1	0	0	0	0
B	1	1	1	1	0
E	1	1	1	1	0
T	1	0	1	1	1
LOCode	1	0	0	0	0
LOComment	1	1	1	1	1
LOBlank	1	0	1	1	0
LOCodeAndComment	1	0	0	0	1
uniq_Op	1	0	0	0	0
uniq_Opnd	1	0	0	0	0
total_Op	1	1	0	0	0
total_Opnd	1	0	1	1	1
branchCount	0	1	1	0	1

4.5 Evaluating the new feature selector's performance on the JM1 data set

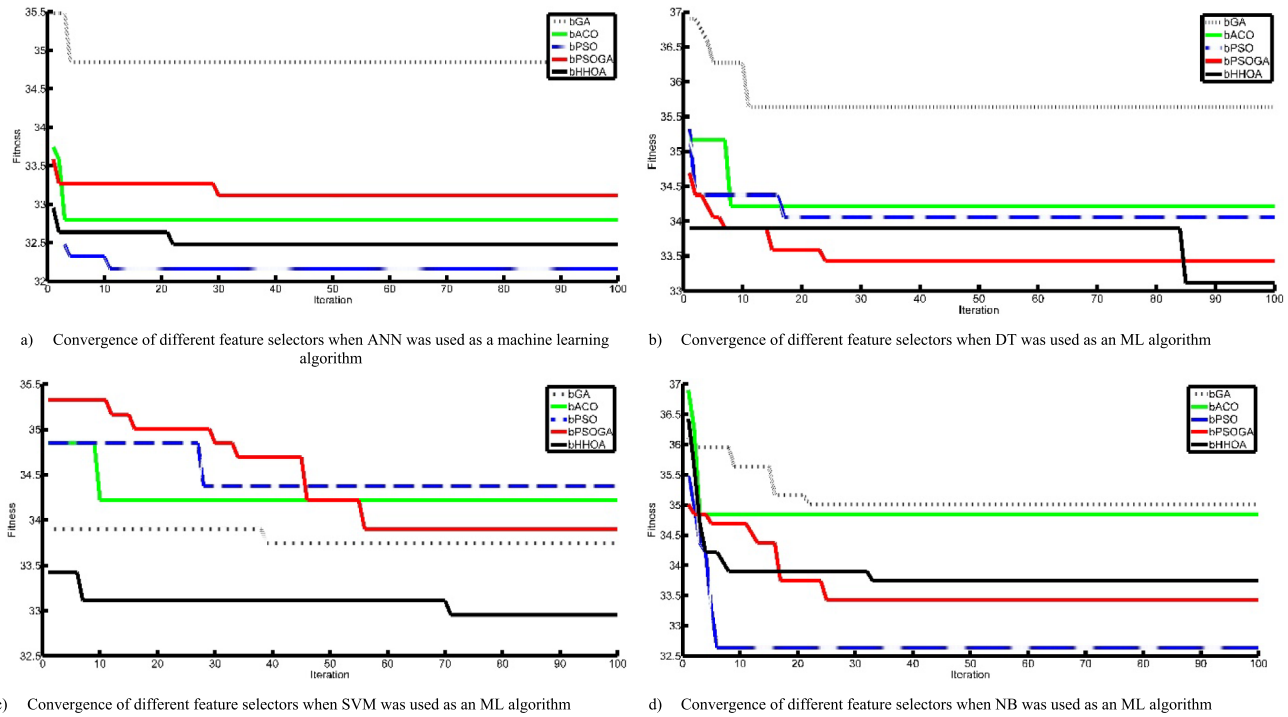
In the third series of experiments aimed at answering specific research questions, the JM1 dataset was utilized for both training and testing. This dataset comprises 21 features and 10,885 records, with 7619 records allocated for training and 3266 for testing. In this experiment, the created classifiers were trained using the entire dataset of 21 features and 7619 training records. The machine learning algorithms applied included ANN, DT, KNN, NB, and SVM, all using complete features. Among these models, the KNN-based classifier demonstrated the best accuracy and precision. In the second phase of the third experiment, the JM1 data set was examined using bHHOA

alongside various machine learning algorithms. Classifiers employing KNN and SVM with bHHOA achieved an accuracy of 90.11% and 82.21%, respectively. The integration of bHHOA notably boosted the accuracy and precision of these models. Specifically, for the ANN-based defect predictor, recall, and $F1$ scores improved from 85.82% and 84.89% to 98.19% and 88.76%, respectively, when bHHOA was applied. These findings, consistent with other data sets, show that machine learning algorithms' performance is significantly better when bHHOA is utilized for feature selection. The results show that defect predictors using bHHOA outperform those developed with previous methods, highlighting the substantial enhancement of the proposed feature selector.

Machine learning (ML) algorithms that incorporate selected features from the JM1 data set exhibit significantly enhanced performance in comparison with the traditional methods, particularly with regard to accuracy, precision, recall, and $F1$ score. These algorithms outperform those built with standard techniques when equipped with carefully chosen features. The ML+bHHOA approach achieves significantly better results by using carefully selected features. bHHOA effectively selects key features and enhances various machine learning algorithms used in the JM1 data set. Classifiers developed using the optimized ML approach consistently deliver better accuracy, precision, recall, and $F1$ scores. Overall, the result underscores the effectiveness of bHHOA in feature selection, significantly boosting ML algorithm performance.

Figure 17 shows the average performance of defect predictors developed with machine learning algorithms on the JM1 data set, comparing those that utilize feature selection with those that do not. The average accuracy of predictors from standard machine learning algorithms is 79.57%, which improves to 84.51% when using the ML+bHHOA method. This enhancement is mirrored in other metrics: the average precision rises from 83.14% to 86.19% with bHHOA. Additionally, classifiers using ML+bHHOA exhibit a notably higher average recall compared to those from standard ML methods. Overall, classifiers developed with the ML+bHHOA approach consistently outperform those built with traditional ML techniques, demonstrating that the bHHOA method effectively identifies and eliminates non-essential features, thus improving the defect predictors' accuracy.

Additionally, Fig. 18 illustrates the average number of effective features selected by bHHOA across various machine learning algorithms, demonstrating its ability to determine the most relevant features from the 21 available JM1 data sets. This selective approach highlights bHHOA's role in enhancing the performance of ML-



a) Convergence of different feature selectors when ANN was used as a machine learning algorithm

b) Convergence of different feature selectors when DT was used as an ML algorithm

c) Convergence of different feature selectors when SVM was used as an ML algorithm

d) Convergence of different feature selectors when NB was used as an ML algorithm

Fig. 16 The convergence of different feature selectors with various machine learning algorithms in the KC1 data set

based defect predictors. According to Fig. 18, when using the ANN algorithm, bHHOA selected 11 features, while bPSO chose 20. For the DT algorithm, bHHOA selected 14 features, compared to 9 by bPSO. In the NB classification scenario, bHHOA selected 11 features, while bACO selected 6. For SVM, bHHOA and bPSO selected 13 and 11 features, respectively. Overall, the feature selection algorithms tend to select fewer features on average with DT and NB.

Table 7 details the features from the JM1 data set optimized by bHHOA across different ML algorithms, showcasing the top results from ten iterations. Each ML algorithm was run ten times, and the features that led to the highest performance were considered the most effective. According to this analysis, the ten most valuable features selected by bHHOA in the JM1 data set are:

1. line count of code (McCabe)
2. Time to write a programme (Halstead)
3. Essential complexity (McCabe)
4. Programme length (Halstead)
5. Difficulty (Halstead)
6. Effort: Halstead effort to write a programme
7. Number of lines which contain both code and comments (Halstead)
8. Branch Count
9. Total operands (Halstead)
10. unique operators (Halstead)

To examine the reliability of the new method, it was employed in the JM1 data set across ten iterations. The results showed improvements in key metrics, including accuracy, precision, recall, and $F1$ scores. The method used a subset of the entire 21 features, selectively identified by bHHOA, to train the ML algorithm. bHHOA consistently selected the same features for building the defect predictor, which significantly impacted the accuracy and precision of the resulting classifier. Figure 19 illustrates how different feature selectors converge, with bHHOA generally outperforming most other methods.

4.6 Evaluating the new feature selector's performance on the KC2 data set

In the fourth set of experiments, the KC2 dataset was utilized to explore specific research questions, serving both training and testing purposes. This dataset contains 522 records and 21 features, with 365 records reserved for

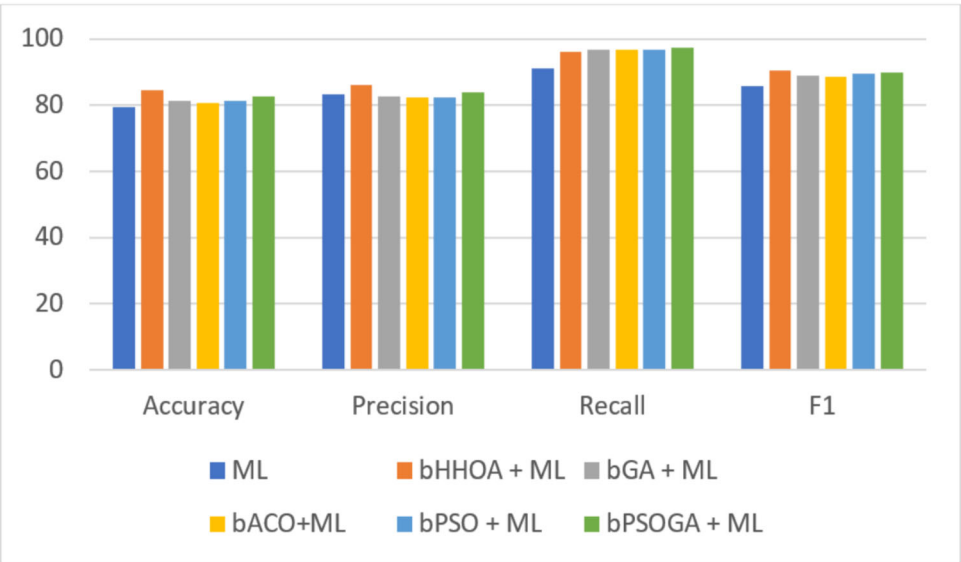


Fig. 17 The average performance metrics of machine learning algorithms both with and without feature selectors on JM1 data set

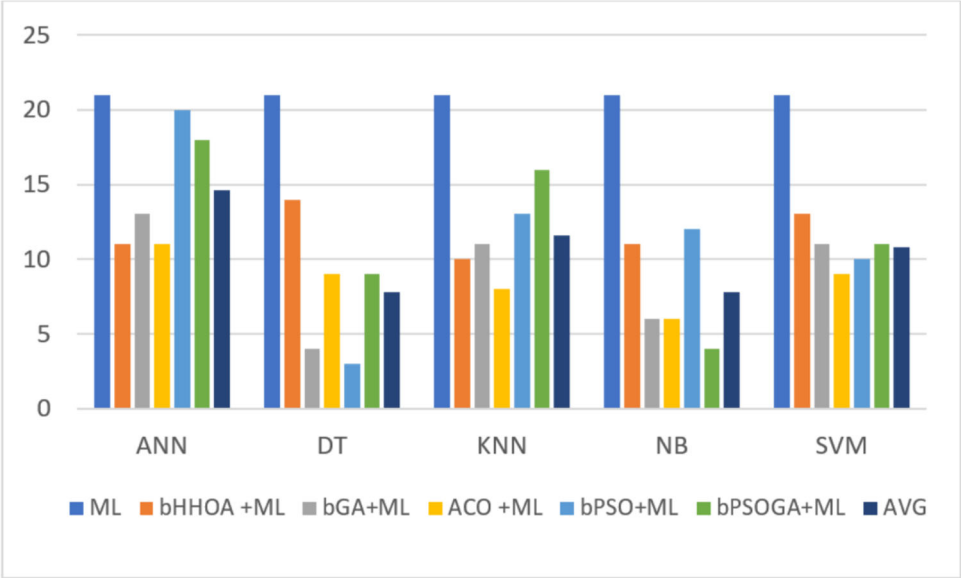


Fig. 18 The average quantity of features selected by various feature selectors across different machine learning algorithms in the JM1 data set

Table 7 The selected features by bHHOA across different machine learning algorithms in the JM1 data set, based on the best results from ten runs

	ANN	DT	KNN	NB	SVM
LOC	1	1	1	1	1
v(g)	0	1	1	1	0
eV(g)	1	1	0	1	1
iv(g)	1	0	0	0	1
N	0	1	1	0	1
v	1	1	0	0	1
L	1	1	0	1	1
D	1	1	1	0	1
I	0	1	0	1	1
B	1	0	0	0	1
E	1	0	1	1	1
T	1	1	1	1	1
LOCode	0	0	0	0	0
LOComment	0	1	0	0	0
LOBlank	0	0	1	0	1
LOCodeAndComment	0	1	1	1	1
uniq_Op	1	1	0	1	0
uniq_Opnd	1	0	0	1	0
total_Op	1	0	0	0	1
total_Opnd	0	0	1	1	1
branchCount	0	1	1	1	1

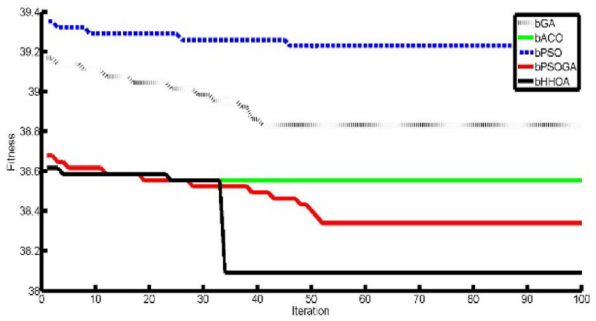
training and 157 for testing. The results highlight the performance of these models without the inclusion of bHHOA, with the KNN-based classifier outperforming others in terms of accuracy and precision. In the KC2 training data set, non-essential features can cause ML algorithms to overfit. To tackle this, the second phase of the fourth set of experiments applied bHHOA with ML algorithms on KC2 data set. When bHHOA was applied, the accuracy of classifiers using KNN and NB improved to 91.00% and 89.80%, respectively, marking a significant enhancement. Incorporating bHHOA into the ML algorithms notably increased both precision and accuracy. For instance, the recall and $F1$ scores for the ANN-based defect predictor rose from 90.16% and 86.27% to 94.57% and 93.12% with bHHOA. This trend, seen in other data sets, is also apparent in the KC2 data set, where bHHOA

improves performance. Overall, the bHHOA feature selection method drastically improves the performance of defect predictors across different machine learning algorithms.

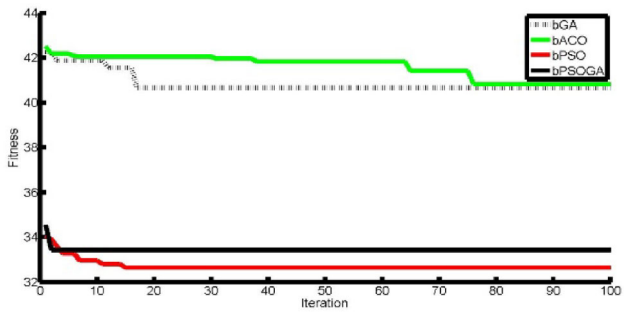
Figure 20 depicts the average performance comparison of defect predictors built with machine learning algorithms on the KC2 data set, evaluating both with feature selection and with no feature selection. The average accuracy of predictors using standard machine learning algorithms is 82.76%, which increases to 88.99% when incorporating the ML+bHHOA method. This enhancement is evident in other key metrics as well. For example, the average precision of classifiers using standard machine learning algorithms is approximately 88.14%, whereas it improves to 91.95% with the bHHOA-enhanced approach. The ML+bHHOA method shows superior results in accuracy, precision, recall, and $F1$ scores, thanks to the effective feature selection from KC2 data set. The figure clearly shows the substantial performance gains achieved with ML+bHHOA over traditional ML methods, highlighting the benefits of using selectively chosen features.

The bHHOA method, designed for optimal feature selection, proves highly effective when used with machine learning algorithms on the KC2 data set. The defect predictors created using the ML+bHHOA approach show superior $F1$ scores, recall, accuracy, and precision compared to traditional models. Overall, the KC2 data set results highlight bHHOA's efficiency in selecting the most relevant features, greatly enhancing the performance of ML-based defect predictors. Classifiers built with ML+bHHOA consistently show greater average recall than those using standard ML algorithms. In essence, defect predictors created with the ML+bHHOA method are more effective than those developed without feature selection, with bHHOA's ability to identify and discard non-essential features markedly improving predictor accuracy.

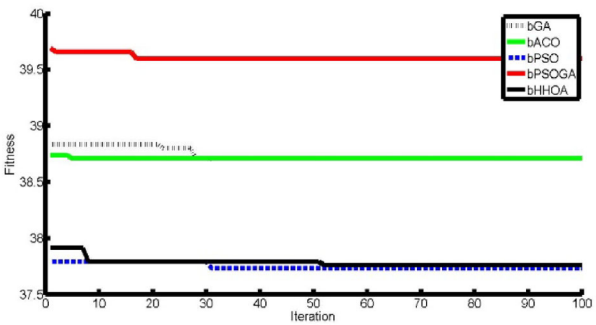
Figure 21 illustrates the number of practical features determined by bHHOA across different machine learning techniques, highlighting its ability to select the most crucial features from the 21 available in the KC2 data set. This selective process underscores bHHOA's role in improving ML-based defect predictors. After ten runs, the average performance of the recommended method on KC2 data set was assessed. Figure 21 details the average number of features selected by binary bACO and bHHOA over these runs. For the ANN algorithm, bHHOA



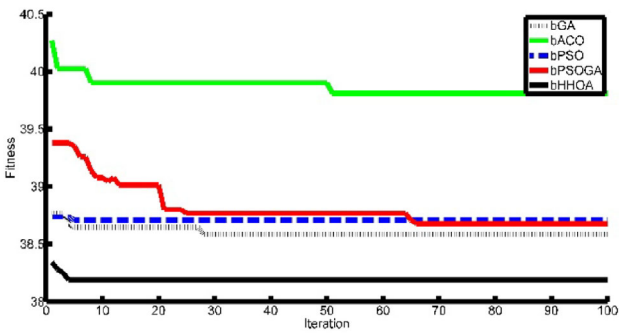
a) Convergence of different feature selectors when ANN was used as an ML algorithm



b) Convergence of different feature selectors when DT was used as an ML algorithm



c) Convergence of different feature selectors when SVM was used as an ML algorithm



d) Convergence of different feature selectors when NB was used as an ML algorithm

Fig. 19 The convergence of different feature selectors with different machine learning algorithms in the JM1 data set

selected 8 features, while bPSOGA selected 20 features. With the KNN algorithm, bHHOA chose 5 features, while bPSOGA selected 11. On average, bHHOA selects 9 features, fewer than the number selected by bGA, bPSO, and bPSOGA.

Table 8 presents the features from the KC2 data set that bHHOA optimized through various machine learning algorithms over ten runs. Each ML technique was run ten times, with the most successful features identified based on their performance. The top ten features deemed most effective by bHHOA in the KC2 data set are listed below:

1. Difficulty (Halstead)
2. Quantity of branches in flow graph (Halstead)
3. Design complexity (McCabe)
4. Number of Delivered Bugs (Halstead)
5. Essential complexity (McCabe)
6. Operators + Operands (McCabe)
7. unique operators (Halstead)
8. Number of lines of comments (Halstead)
9. line count of code (McCabe)
10. Total operands (Halstead)

The new method was tested ten times on the KC2 data set to assess its reliability. The results indicated notable improvements in accuracy, precision, recall, and $F1$ scores. While all 21 features were initially considered, only those selected by bHHOA were used for training the machine leaning algorithm. A consistently chosen subset of features from bHHOA's output was employed to build the defect predictor, significantly impacting its accuracy and precision. Figure 22 demonstrates that the proposed bHHOA method surpasses other feature selectors in both performance and convergence speed.

4.7 Evaluating the new feature selector's performance on the PC1 data set

The last experiment series, designed to answer the research questions, used the PC1 data set. This data set, containing 21 features and 1109 records, was split into 776 records for training and 333 for testing. Notably, the defect predictor built with the KNN algorithm achieved the greatest accuracy and precision among the models

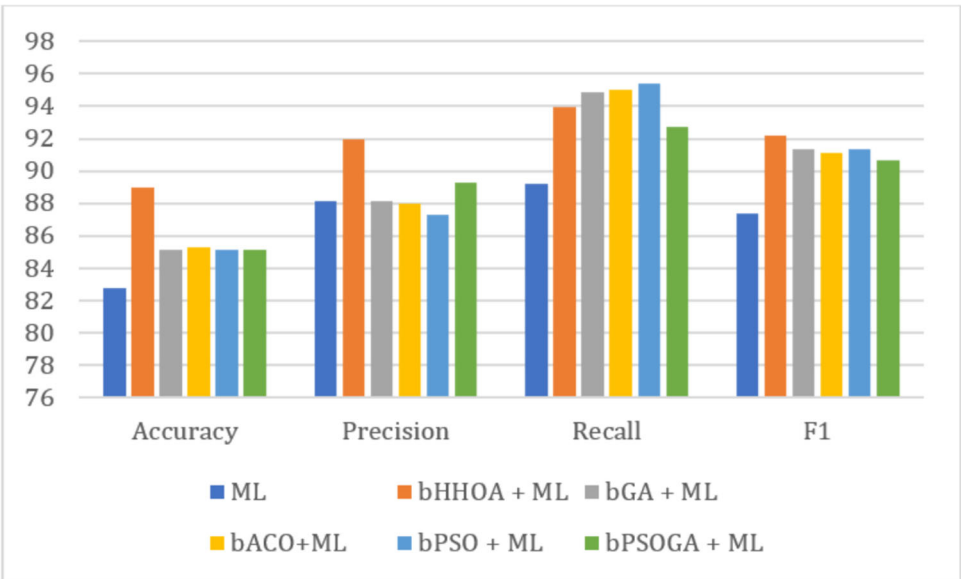


Fig. 20 The average performance metrics of machine learning algorithms using different feature selectors on KC2 data set, over the course of ten runs

https://doi.org/10.1007/s00521-025-11669-6

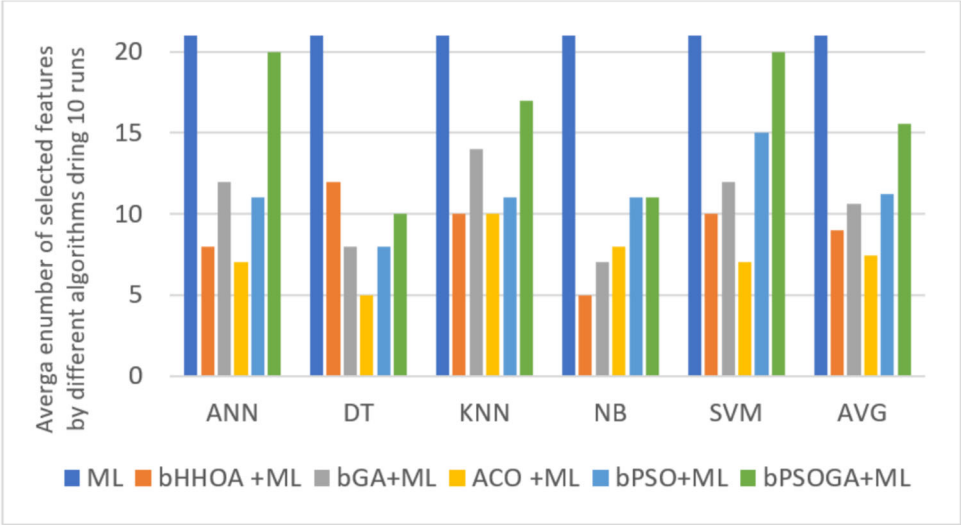


Fig. 21 The average number of features selected by various feature selectors across various machine learning algorithms on KC2 data set, throughout ten executions

tested. In the second phase of the experiments on the PC1 data set, bHHOA was combined with various ML algorithms to assess the performance of defect predictors. For instance, classifiers built with ANN and DT achieved accuracy rates of 93.19% and 94.29%, respectively, when combined with bHHOA. Specifically, the defect predictor originally produced using only ANN had precision and recall scores of 91.74% and 99.66%. However, these metrics significantly increased to 92.04% and 100%, with bHHOA utilized for feature selection. These results underscore the effectiveness of bHHOA as a feature selector for ML algorithms on PC1 data set, aligning with its performance in other data sets. Regarding the results shown in Fig. 23, it is clear that software defect predictors incorporating bHHOA outperform those developed using the previous method without feature selection.

Table 8 The features selected by bHHOA across various machine learning algorithms in KC2 data set based on the best outcome from 10 executions

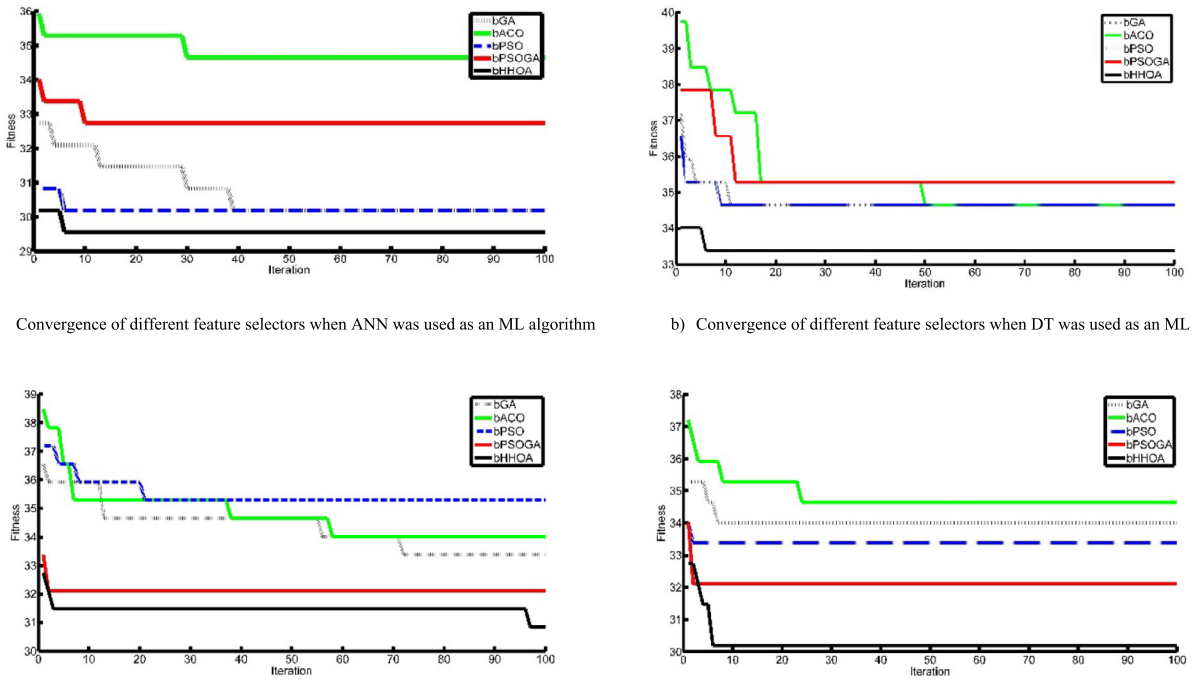
	ANN	DT	KNN	NB	SVM
LOC	0	1	0	0	1
v(g)	1	0	0	0	0
eV(g)	0	1	1	1	0
iv(g)	1	1	1	1	0
N	0	1	1	0	1
V	0	1	1	0	0
L	0	0	1	0	1
D	1	1	1	1	1
I	0	1	0	0	1
B	1	1	0	1	1
E	0	1	0	1	1
T	0	0	1	0	1
LOCode	1	0	0	0	0
LOComment	0	1	0	1	1
LOBlank	0	1	0	0	0
LOCodeAndComment	0	0	1	0	1
uniq_Op	1	1	0	1	0
uniq_Opnd	1	0	0	0	0
total_Op	1	0	1	0	0
total_Opnd	0	0	0	1	1
branchCount	1	1	1	1	1

By leveraging optimally selected features from the PC1 data set, the ML+bHHOA methods significantly outperform standard machine learning algorithms in accuracy, precision, recall, and $F1$ scores. The experimental results reveal a marked improvement in defect predictors created with the ML+bHHOA approach compared to those using conventional ML methods. This boost in performance is due to the effective feature selection, which enhances results. When applied for optimal feature selection, bHHOA proves highly effective on the PC1 data set. Defect predictors developed with the ML+bHHOA approach consistently achieve better $F1$ scores, recall, accuracy, and precision than earlier models. By accurately identifying and discarding non-essential features, bHHOA markedly enhances the overall performance of defect predictors.

Figure 23 compares the average performance of defect predictors developed with machine learning algorithms on the PC1 data set, contrasting those with feature selection and with no feature selection. The defect predictors using the ML+bHHOA method achieve an average accuracy of 93.54%, surpassing the accuracy of those from standalone ML methods. This improvement is also reflected in other performance metrics. Specifically, there is a notable increase in the average recall for defect predictors using the ML+bHHOA approach compared to those from standard ML methods. Overall, the analysis demonstrates that defect predictors developed with the ML+bHHOA method outperform those created with no feature selection.

Following ten iterations, the proposed method's average performance on the PC1 data set was evaluated. Figure 24 demonstrates the average number of selected features by bACO and bHHOA throughout these iterations. For the KNN algorithm, bHHOA selected ten features, while bPSO selected 17. With the NB algorithm, bHHOA identified nine optimal features compared to eight features selected by NB+ACO. Overall, bHHOA consistently chose fewer features on average, at 9.4, compared to the numbers selected by bACO, bPSO, and bPSOGA.

Table 9 provides a detailed summary of the features from the PC1 data set that bHHOA selected and refined over 10 iterations with various machine learning algorithms. Each machine learning method underwent ten runs, and the features that consistently performed best were deemed the most effective. Following ten runs with the



a) Convergence of different feature selectors when ANN was used as an ML algorithm

b) Convergence of different feature selectors when DT was used as an ML algorithm

c) Convergence of different feature selectors when SVM was used as an ML algorithm

d) Convergence of different feature selectors when NB was used as an ML algorithm

Fig. 22 The convergence of different feature selectors with different machine learning algorithms in the KC2 data set

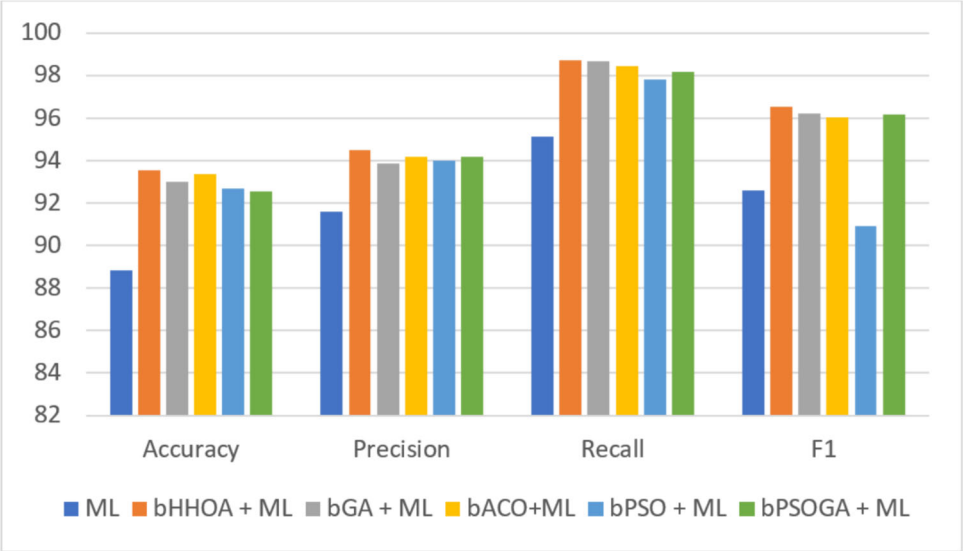


Fig. 23 The average performance metrics of machine learning algorithms on the PC1 data set, both with and without feature selectors

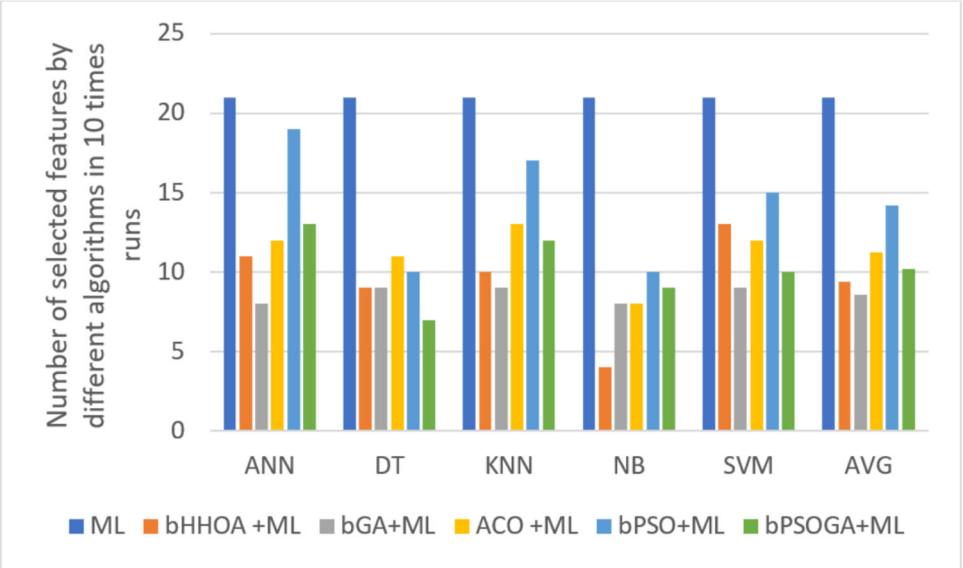


Fig. 24 The quantity of features selected by various feature selectors across different machine learning algorithms in the PC1 data set

new approach on the PC1 data set, bHHOA pinpointed a set of crucial features. The top ten most significant features identified by bHHOA in the PC1 data set are listed below:

1. Cyclomatic complexity (McCabe)
2. Difficulty (Halstead)
3. Number of Delivered Bugs (Halstead)
4. Programme length (Halstead)
5. Time to write programme (Halstead)
6. Line count (Halstead)
7. unique operands (Halstead)
8. Total operators (Halstead)
9. Quantity of branches in flow graph (Halstead)

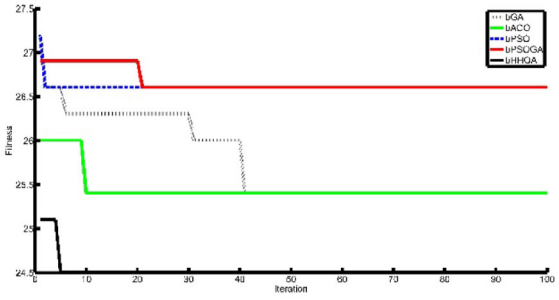
Table 9 The selected features by bHHOA across various machine learning algorithms on PC1 data set, based on the best outcomes from ten runs

	ANN	DT	KNN	NB	SVM
LOC	0	0	0	0	1
v(g)	1	1	0	1	1
eV(g)	0	0	1	0	1
iv(g)	1	0	1	0	0
N	0	1	0	0	1
v	0	0	1	0	0
L	1	0	1	0	1
D	1	0	1	1	1
I	1	0	0	0	0
B	1	1	1	0	1
E	0	1	0	0	1
T	1	1	0	0	1
LOCode	1	0	1	0	1
LOComment	0	1	0	1	0
LOBlank	1	1	1	1	0
LOCodeAndComment	0	1	0	0	1
uniq_Op	0	0	0		1
uniq_Opnd	1	1	0	0	1
total_Op	1	0	1	1	0
total_Opnd	0	0	1	0	0
Branch Count	1	0	1	0	1

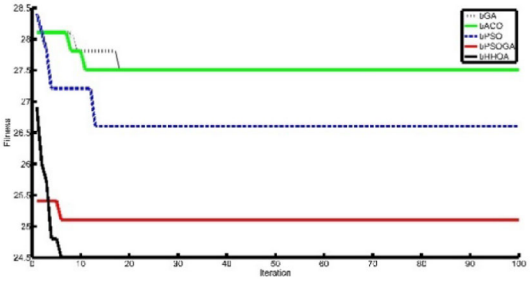
Figure 25 depicts the convergence trends of different feature selectors in identifying the most effective features in the benchmark data set. Although the performance of bHHOA varies with various machine learning algorithms, it consistently achieves the highest fitness value when combined with all ML methods, demonstrating superior performance in SDP. As a feature selector, bHHOA drastically boosts the effectiveness of ML algorithms for SDP tasks. The figure highlights that bHHOA converges more rapidly than other feature selectors across all data sets and ML algorithms. Despite variations in performance across different ML techniques, bHHOA consistently improves the performance of machine learning algorithms in SDP. Table 10 shows one of the confusion matrixes from ten times executions of the bHHOA+DT in different datasets.

5 Conclusion

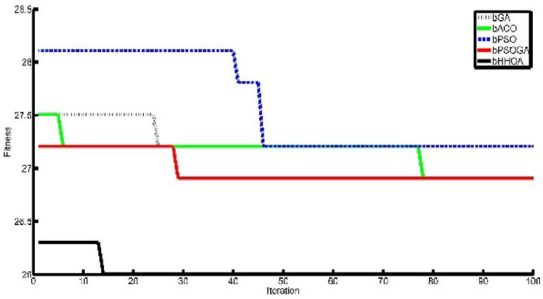
One significant challenge with ML-based defect predictors is their limited capability to assess the influence of each feature on the outcome, which can result in suboptimal training performance. This paper introduces a binary version of the horse herd optimization algorithm (bHHOA), specifically developed to identify the most pertinent features from software defect prediction (SDP) data sets; by employing bHHOA, a highly efficient and robust classifier was created, trained on a streamlined dataset that includes only the most crucial features. This method improves the defect predictor's performance by selecting a minimal yet impactful subset of features from traditional defect prediction datasets, as demonstrated with the PROMISE Software Engineering Repository dataset [27]. The findings underscore the effectiveness of bHHOA in selecting features for machine learning-based defect prediction models. The most important features identified by bHHOA include McCabe's design complexity, the total counts of operators and operands, Halstead's programme length and difficulty, the number of branches in the code, lines of code, comments, and the unique operators and operands. The defect predictor that integrates bHHOA with machine learning algorithms achieved an average accuracy of 89.91%, a precision of 91.37%, a recall of 96.87%, and an $F1$ score of 93.50%. The exclusion of certain features by bHHOA does not



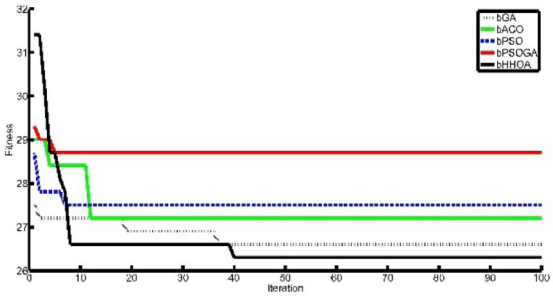
a) Convergence of different feature selectors when ANN was used as an ML algorithm



b) Convergence of different feature selectors when DT was used as an ML algorithm



c) Convergence of different feature selectors when SVM was used as an ML algorithm



d) Convergence of different feature selectors when NB was used as an ML algorithm

Fig. 25 The convergence of different feature selectors with different machine learning algorithms in the PC1 data set

Table 10 One of the selected confusion matrices from ten times executions of bHHOA+DT in each dataset

CM1		KC1		KC2		PC1	
131	2	504	35	117	10	293	6
10	6	53	41	8	22	23	11

suggest their lack of importance but indicates that other features have a more significant impact and better generalization for learning algorithms. The software defect predictors can be made based on a software product's code-level and design-level features; hence, utilizing the design-level and codes-level features for training and creating SDP is considered a future study. Utilizing fuzzy logic in the procedure of defect prediction is another future study. Evaluating the combination of novel metaheuristic algorithms with ML techniques for SDP and chaotic maps [31] further offers promising research directions. Additionally, examining recent metaheuristics algorithms as feature selectors in SDP methods [32–34] and integrating deep learning techniques present valuable avenues for future investigation. Many software developers use AI tools (like Copilot or ChatGPT) to write some parts of the code. This trend is traditional in software development and identifying and predicting defects in the AI-written code becomes crucial. Existing methods for predicting software bugs (based on patterns from human-written code) may not work well for the AI-written code. Investigating the methods to handle this challenge is another future study.

Author contributions All authors have contributed in all stages of this research.

Funding The authors declare that they did not receive any funding during the preparation of this manuscript.

Data availability The data sets produced in this study and the implemented code can be accessed freely via the following link: https://drive.google.com/drive/folders/1tA_hzNTsWdlp2DSZVWoCVGcv3lGtlVqF?usp=drive_link.

Declarations

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

Ethics approval The authors affirm that the current study is their original and unpublished work, not submitted or published elsewhere. The paper accurately and comprehensively reflects the authors' research and analysis.

References

1. Arasteh B (2018) Software fault-prediction using combination of neural network and naive bayes algorithm. J Netw Technol 9(3):94–101. https://doi.org/10.6025/jnt/2018/9/3/94-101
2. Al-Laham M, Kassaymeh S, Al-Betar MA, Makhadmeh SN, Albashish D, Alweshah M (2023) An efficient convergence-boosted salp swarm optimiser-based artificial neural network for the development of software fault prediction models. Comput Electr Eng 111:108923. https://doi.org/10.1016/j.compeleceng.2023.108923
3. Thirumoorthy K, Britto JJJ (2022) A clustering approach for software defect prediction using hybrid social mimic optimisation algorithm. Computing 104:2605–2633. https://doi.org/10.1007/s00607-022-01100-6
4. Hao W, Arasteh B, Arasteh K, Soleimanian F, Rouhi A (2024) A software defect prediction method using binary gray wolf optimiser and machine learning algorithms. Comput Electr Eng 118:109336. https://doi.org/10.1016/j.compeleceng.2024.109336
5. Ghaemi A, Arasteh B (2020) SFLA-based heuristic method to generate software structural test data. J Softw Evol Process 32:e2228. https://doi.org/10.1002/smr.2228
6. Shomali N, Arasteh B (2020) Mutation reduction in software mutation testing using firefly optimisation algorithm. Data Technol Appl 54(4):461–480. https://doi.org/10.1108/DTA-08-2019-0140
7. Hosseini MJ, Arasteh B, Isazadeh A, Mohsenzadeh M, Mirzarezaee M (2021) An error-propagation aware method to reduce the software mutation cost using genetic algorithm. Data Technol Appl 55(1):118–148. https://doi.org/10.1108/DTA-03-2020-0073
8. Arasteh B, Najafi BJ (2018) Programming guidelines for improving software resiliency against soft-errors without performance overhead. Computing 100:971–1003. https://doi.org/10.1007/s00607-018-0592-y
9. Arasteh B, Miremadi BSG, Rahmani AM (2014) Developing inherently resilient software against soft-errors based on algorithm level inherent features. J Electron Test 30:193–212. https://doi.org/10.1007/s10836-014-5438-8
10. Batool BI, Khan AKTA (2022) Software fault prediction using data mining, machine learning and deep learning techniques: a systematic literature review. Comput Electr Eng 100:107886. https://doi.org/10.1016/j.compeleceng.2022.107886
11. Jiang Y, Li M, Zhou Z, Member S (2011) Software defect detection with rocus. J Comput Sci Technol 26(2):328–342
12. Wang SS, Yao XX (2013) Using class imbalance learning for software defect prediction. IEEE Trans Reliab 62(2):434–443
13. Galar M, Fern A, Barrenechea E, Bustince H (2011) A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches mikel. IEEE Trans Syst MAN Cybern 42(4):1–22
14. Anbu M, Anandha GS (2019) Feature selection using firefly algorithm in software defect prediction. Cluster Comput 22:10925–10934. https://doi.org/10.1007/s10586-017-1235-3
15. Mafarja M, Thaher T, Al-Betar MA et al (2023) Classification framework for faulty-software using enhanced exploratory whale optimiser-based feature selection scheme and random forest ensemble learning. Appl Intell 53:18715–18757. https://doi.org/10.1007/s10489-022-04427-x
16. Yousef AH (2014) Extracting software static defect models using data mining. Ain Shams Eng J 6(1):133–144
17. Jayanthi R, Florence L (2019) Software defect prediction techniques using metrics based on neural network classifier. Cluster Comput 22(1):77–88. https://doi.org/10.1007/s10586-018-1730-1
18. Laradji IH, Alshayeb M, Ghouti L (2015) Software defect prediction using ensemble learning on selected features. Inf Softw Technol 58:388–402
19. Yucalar F, Ozcift A, Borandag E, Kilinc D (2020) Multiple-classifiers in software quality engineering: combining predictors to improve software fault prediction ability. Eng Sci Technol Int J 23(4):938–950. https://doi.org/10.1016/j.jestch.2019.10.005
20. Campos J, Costa E, Vieira M (2019) Improving failure prediction by ensembling the decisions of machine learning models: a case study. IEEE Access 7:177661–177674

21. Bal P, Kumar S (2018) Extreme learning machine based linear homogeneous ensemble for software fault prediction. In: ICSOFT. PP 103–112
22. He H, Zhang X, Wang Q, Ren J, Liu J, Zhao X, Cheng Y (2019) Ensemble multiboost based on ripper classifier for prediction of imbalanced software defect data. IEEE Access 7:110333–110343
23. Coelho RA, Dos R et al (2014) Applying swarm ensemble clustering technique for fault prediction using software metrics. In: 2014 13th International conference on machine learning and applications, IEEE, pp 356–361
24. Ryu D, Baik J (2018) Effective harmony search-based optimization of cost-sensitive boosting for improving the performance of cross-project defect prediction. KIPS Trans Softw Data Eng 7(3):77–90
25. Sofian K, Salwani A, Mohammed A, Mohammed A (2022) Salp swarm optimizer for modeling the software fault prediction problem. J King Saud Univ Comput Inf Sci 34(6):3365–3378. https://doi.org/10.1016/j.jksuci.2021.01.015
26. Li W, Huang Z, Li Q (2016) Three-way decisions-based software defect prediction. Knowl-Based Syst 91:263–274. https://doi.org/10.1016/j.knosys.2015.09.035
27. Promise software engineering repository. Online Available:http://promise.site.uottawa.ca/SERepository/data_sets-page.html.
28. Arasteh B, Arasteh K, Ghaffari A, Ghanbarzadeh R (2024) A new binary chaos-based metaheuristic algorithm for software defect prediction. Cluster Comput. https://doi.org/10.1007/s10586-024-04486-4
29. MiarNaeimi F, Azizyan G, Rashki M (2021) Horse herd optimisation algorithm: a nature-inspired algorithm for high-dimensional optimisation problems. Knowl-Based Syst 213:106711
30. Kanan HR, Faez K, Taheri SM (2007) Feature selection using ant colony optimization (ACO): a new method and comparative study in the application of face recognition system. Advances in data mining-theoretical aspects and applications. 7th industrial conference, ICDM 2007, vol 4597. pp 63–76
31. Arasteh B (2023) Clustered design-model generation from a program source code using chaos-based metaheuristic algorithms. Neural Comput Appl 35:3283–3305. https://doi.org/10.1007/s00521-022-07781-6
32. Shomali N, Arasteh B (2020) Mutation reduction in software mutation testing using firefly optimization algorithm. Data Technol Appl 54(4):461–480. https://doi.org/10.1108/DTA-08-2019-0140
33. Hanafi AV, Ghaffari A, Rezaei H et al (2024) Intrusion detection in internet of things using improved binary golden jackal optimization algorithm and LSTM. Cluster Comput 27:2673–2690. https://doi.org/10.1007/s10586-023-04102-x
34. Gharehchopogh FS, Rezaii R, Arasteh B (2015) A new approach by using Tabu search and genetic algorithms in software cost estimation. In: 9th International conference on application of information and communication technologies (AICT), Rostov on Don, Russia, 2015, pp 113–117

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

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Bahman Arasteh[1,2,3] · Asgarali Bouyer[1,4] · Peri Gunes[5,6] · Reza Ghanbarzadeh[7] · Farhad Soleimanian Gharehchopogh[8]

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

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

[2] Department of Computer Science, Khazar University, Baku, Azerbaijan

[3] Applied Science Research Center, Applied Science Private University, Amman, Jordan

[4] Department of Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran

[5] İnfina Software Inc., R&D Department, Istanbul, Turkey

[6] Department of Electerical and Electronics Engineering, Faculty of Engineering and Architecture, Istanbul Gelisim University, Istanbul, Turkey

[7] Faculty of Science and Engineering, Southern Cross University, Gold Coast, Australia

[8] Department of Computer Engineering, Urmia Branch, Islamic Azad University, Urmia, Iran