e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–44
DOI: 10.37190/e-Inf250103
BibTeX

# A Comparative Analysis of Metaheuristic Feature Selection Methods in Software Vulnerability Prediction

Deepali Bassi*, Hardeep Singh*
*Corresponding authors: deepalics.rsh@gndu.ac.in, hardeep.dcse@gndu.ac.in

**Article info**

Dataset link: https://drive.google.com/file/d/14qoY98ZPiWalupRUquMALR_HHChuSaPB/view?usp=sharing

**Keywords:**
software vulnerability prediction
metaheuristic feature selection
SMOTE
machine learning algorithms

Submitted: 6 Jun. 2024
Revised: 21 Sep. 2024
Accepted: 2 Oct. 2024
Available online: 18 Nov. 2024

**Abstract**

**Background:** Early identification of software vulnerabilities is an intrinsic step in achieving software security. In the era of artificial intelligence, software vulnerability prediction models (VPMs) are created using machine learning and deep learning approaches. The effectiveness of these models aids in increasing the quality of the software. The handling of imbalanced datasets and dimensionality reduction are important aspects that affect the performance of VPMs.
**Aim:** The current study applies novel metaheuristic approaches for feature subset selection.
**Method:** This paper performs a comparative analysis of forty-eight combinations of eight machine learning techniques and six metaheuristic feature selection methods on four public datasets.
**Results:** The experimental results reveal that VPMs productivity is upgraded after the application of the feature selection methods for both metrics-based and text-mining-based datasets. Additionally, the study has applied Wilcoxon signed-rank test to the results of metrics-based and text-features-based VPMs to evaluate which outperformed the other. Furthermore, it discovers the best-performing feature selection algorithm based on AUC for each dataset. Finally, this paper has performed better than the benchmark studies in terms of $F_1$-score.
**Conclusion:** The results conclude that GWO has performed satisfactorily for all the datasets.

## 1. Introduction

The prediction of software vulnerabilities composes an essential step to provide software quality and security. Vulnerability, according to ISO/IEC 27000:2018, is a flaw in a control or asset that one or more threats could exploit. A few instances that illustrate the harm caused by software vulnerabilities are the open-source programs Heartbleed, ShellShock, and Apache Commons; well-known web browser plugins like Adobe Flash Player and Oracle Java. Millions of internet users have had their security jeopardized by browser plugins, and thousands of businesses and clients worldwide have been put in danger by open-source software. Furthermore, 1.7 million USD in financial damages have also been reported [1] as a result of software malfunction. Organizations had to pay 1.4 million USD in 2017

and 1.3 million USD in 2018 to cope with cyberattacks as a result of cybercrimes [2]. The National Institute of Standards and Technology (NIST) documented an exponential growth in software vulnerabilities since 2016 [3]. Software developers' negligence regarding the security facets during the initial phases of development causes security issues in later stages. Vulnerabilities provide possibilities for attackers to perform criminal activities and their sales at a very high price [4]. They are known to be the subgroup of faults, as are less in number than faults [5]. A fault in the software specification, development, or configuration is considered a vulnerability if the security policy is violated during its execution [6]. Detection and fixing of vulnerabilities before the deployment stage aids in reducing testing costs and maintaining their market reputation.

The vulnerability prediction models (VPMs) are devised to predict vulnerable and non-vulnerable components and therefore, the quality of these models is essential for the security of the software systems. Researchers and engineers are striving to build accurate VPMs, thus ensuring the quality and security of the systems. Research studies have previously used software assurance techniques such as static analysis, dynamic analysis, fuzz-testing, code inspections, etc. to identify security vulnerabilities [7]. Due to the huge time consumption and high false positive rate problems in conventional techniques, machine learning and deep learning-based VPMs gained interest. Commonly, VPMs are executed in the testing phase of the overall Software Development Life-Cycle (SDLC) in order to prioritize the inspection efforts (e.g., static analysis testing, dynamic testing, etc.). The model will identify which files are likely to have vulnerabilities if their features are collected at the file level, while vulnerable methods may be detected if the dataset is generated at the method level. Some studies on vulnerability prediction used text mining data, while others used software measures similar to those used in fault prediction models [8].

Hence, VPMs can be modeled as metrics-based, text-mining-based, or a combination of both datasets. In metrics-based VPMs, the components are determined using software metrics, e.g., cohesion, coupling, and complexity metrics that predict the vulnerability [9]. In text-mining-based VPMs, source code is converted into tokens and these tokens or text features predict the vulnerable components [10]. Vulnerability datasets are imbalanced which leads to biases in the prediction models as the majority class is favored over the minority class [11]. Most studies have used data balancing methods to handle class imbalance problems [8,12] and hyperparameter optimization (HPO) methods for choosing the optimal hyperparameters of classifiers [13–17].

Machine learning techniques also face the problem of the high dimensionality of the dataset. The performance of classifiers degrades when the classification parameters are increased. Therefore, to improve the efficacy of the model, the feature subset size should be decreased. Dimensionality reduction is an effective method to remove redundant or irrelevant features and thereby upgrade the performance, lowering computational complexity, constructing generalized models, and reducing storage [18,19]. Two major approaches have been proposed for dimensionality reduction: feature synthesis and feature selection [20].

In the case of feature synthesis, dimensional space is transformed from high to low whereas, in feature selection, a subset of given features is chosen by removing redundant or features with no predictive information. Feature selection methods are categorized as a filter, wrapper, hybrid, and embedded. Further two methods exist to describe how the features can be evaluated such as feature ranking and subset selection. In the feature ranking method, each feature is given a score based on some criteria, and the features with insufficient scores are removed [21]. In the subset selection method, an optimal subset is found out of all possible subsets. If there are n initial features, the search space for the best

subset comprises all feature subsets, which is equal to 2n different states. In other words, the value of each property is evaluated independently in the property ranking algorithms, and the relationship between characteristics is not taken into account.

Feature selection algorithms based on metaheuristics are emerging in the field of vulnerability prediction [22–24]. These methods presume that the features are independent of one another and that any potential relationship between the features is ignored. Although this basic assumption decreases the computational complexity of the feature selection approach, it may reduce its performance in many circumstances. Choosing a feature subset is an NP-Hard task. The best subset can be identified simply by assessing all feasible subsets using an exhaustive search approach. Although this method guarantees an optimal feature subset, even for medium-sized datasets, finding the optimal answer is time-consuming and even impractical. Because evaluating all feasible subsets is prohibitively expensive, a feature subset must be searched that is acceptable in terms of both computing complexity and suitability. Metaheuristic algorithms are one technique to solve complex optimization and NP-Hard issues. Metaheuristic approaches are categorized as evolutionary algorithms and swarm intelligence (SI). SI algorithms used approximate and non-deterministic strategies to explore and exploit the search space to obtain near-optimal solutions. Swarm-based approaches are the most prevalent type of nature-inspired metaheuristic group [22].

### 1.1. Motivation

Efficient VPMs are important for ensuring the security and quality of the software [8]. Their performance is affected by the imbalanced datasets, the selection of optimal hyperparameters for machine learning algorithms, and the dimensionality of the datasets. Recent studies have worked on improving it by incorporating data balancing methods, HPO, reducing the dimensionality through feature synthesis, filter-based feature ranking, and also using metaheuristic algorithms for feature subset selection. In [23], the researchers have applied the dual HPO, where the problem of imbalanced datasets is handled and the selection of appropriate hyperparameters is done, to optimize VPMs. Research studies related to feature selection or dimensionality reduction have been explored for the past few years but metaheuristic feature selection methods have not been explored much.

To the best of our knowledge, we have come across two papers [24] and [25] that use such techniques. So, exploring such an area could be beneficial for the researchers to know the impact of the combination of metaheuristic feature selection and machine learning techniques on VPMs. The research paper [24] uses the grey-wolf optimization (GWO), particle swarm optimization (PSO), and genetic algorithm (GA) metaheuristic feature subset selection methods, random forest (RF) machine learner, and SMOTE resampling technique on metrics PHP dataset. In [25], Diploid Genetic algorithms with deep learning networks (Long Short Term Memory and Gradient Recurrent Unit) on software vulnerability prediction are applied. The deep SYMbiotic-based genetic algorithm model (DNN-SYMbiotic GAs) is used in this suggested method to solve challenges involving the prediction of software vulnerabilities. The current study is motivated to extend the paper [24] where it performs a comparative analysis of the various combinations of three other metaheuristic algorithms and seven other machine learning techniques. Moreover, it has not only worked on metrics datasets but also on text-features-based datasets.

### 1.2. Contributions

The contributions are as follows:

- This paper performs the experiments using two datasets; one is in PHP language and other in JavaScript. The PHP dataset [10] consists of three projects (Drupal, PHPMyAdmin and Moodle) and is released in two forms, i.e., software metrics and text features. The JavaScript dataset [12, 26] contains software components (methods) from several projects and contains also both metrics and text features. Since all the datasets are imbalanced therefore SMOTE resampling technique is applied to balance the datasets.
- The current study performs a comparative analysis of six metaheuristic algorithms such as PSO, GA, GWO, salp swarm algorithm (SSA), harris hawk optimization (HHO), and whale optimization algorithm (WOA) combined with eight machine learning algorithms namely random forest (RF), naïve bayes (NB), adaboost (AB), support vector machine (SVM), $k$-nearest neighbor (KNN), decision tree (DT), logistic regression (LR), and multilayer perceptron (MLP).

  The paper has framed the following research questions:

**RQ 1.** Has all the metaheuristic feature selection and machine learning combinations improved the efficacy of VPMs?
Previous studies have experimented with improving the efficiency of VPMs using optimal hyperparameters settings [23], applying data balancing techniques and dimensionality reduction methods [20]. The research study [24] used feature selection algorithms namely PSO, GA, and GWO on only metrics-based PHP datasets with random forest machine learner and SMOTE technique. Second [25] has proposed a new VPM based on the SYMbiotic genetic algorithm and deep learning techniques on metrics-based PHP datasets. Through this question, the current study will explain whether all metaheuristic feature selection and machine learning combinations have improved the performance of VPMs.

**RQ 2.** Which one statistically performed better metrics-based or text-mining-based VPMs in the context of feature selection?
Walden et al. [10] have mentioned in their paper that text-mining-based VPMs have performed better than metrics-based VPMs for within-project prediction. In the case of cross-project prediction, metrics-based VPMs performed slightly better. The motive of this research question is to observe that on applying a metaheuristic feature selection method which VPM (metrics and text-mining) has significantly performed better. For the significant comparison, the paper applied Wilcoxon signed rank test.

**RQ3.** Which metaheuristic feature selection algorithm has performed the best?
Rhmann [24] has shown that PSO-RF has performed better than other benchmark studies. It has concluded that GA, PSO, and GWO-based RF outperformed the existing machine-learning algorithms. The current study applied all the possible combinations of eight machine learners with the six metaheuristic feature selection methods. The research question aims to find out which feature selection method has performed the best to help the researchers in improving the efficiency of VPMs.

### 1.3. Paper organization

The rest of the paper is structured as: Section 2 represents the research works related to the current study, Section 3 explains the background knowledge, Section 4 provides

the research methodology, Section 5 provides the results, Section 6 discusses the results, Section 7 defines the threats to validity, and Section 8 concludes the paper. The appendix includes the detailed results tables and metrics description of the datasets.

## 2. Related work

To build effective VPMs research studies have emphasized the early prediction of vulnerable components using machine learning algorithms. Therefore, handling the factors affecting the performance of VPMs such as optimal hyperparameters, imbalanced datasets, feature selection, etc. is essential. In this section, we have discussed the research works that include VPMs and feature selection techniques.

Ghaffarian and Shahriari [7] have reviewed machine learning and data-mining techniques to curb the effect of software vulnerability. It has stated various software vulnerabilities. In addition, it has been mentioned that the vulnerability datasets are imbalanced and affect the efficiency of machine learning algorithms. Kaya et al. [8] show the impact of feature types, classifiers, and data-balancing techniques on VPMs. It has covered three feature types such as metrics, text, and a combination of metrics and text features. Experiments are performed using seven machine classifiers and four data sampling techniques on the PHP datasets namely Drupal, Moodle, and PHPMyAdmin. Evaluation is done through the performance metrics precision, recall, AUC, $F_1$-score, and specificity. The conclusion states that for smaller datasets Drupal and PHPMyAdmin, the random forest has outperformed other classifiers, and for larger datasets, i.e., Moodle, Rusboost has outshined.

Walden et al. [10] have created vulnerability datasets based on PHP open-source projects as most of the vulnerabilities are observed in web applications. The datasets include two feature types such as software metrics and text features. Random forest and under-sampling techniques are used for classification and balancing the dataset, respectively. The performance metrics used are recall and inspection ratio. The paper has experimented on both software metrics-based and text mining-based VPMs. In addition, it has been found that text-mining-based models outperformed metrics-based models. Ferenc et al. [12] predict the vulnerabilities in JavaScript programs using static code metrics. It has proposed the JavaScript dataset by extracting the vulnerability information from public databases such as Node Security Project, GitHub code fixing patches, and the Snyk platform. The paper has applied a grid search algorithm for parameter tuning, resampling techniques to balance the data, and eight machine learning algorithms including deep learning algorithms to find the best-performing VPMs. Finally, it concludes that the $k$-nearest neighbor has performed best with an $F$-measure of 0.76, over-sampling has improved recall but diminishes precision, and under-sampling increases precision and decreases recall.

Stuckman et al. [20] analyzed the impact of dimensionality reduction techniques (feature selection, principal component analysis, and confirmatory factor synthesis) on the productivity of VPMs. It has used Smote and under-sampling techniques for balancing the data and resulted in smote showing low recall, low inspection rate, and high f-measure therefore it is better than the under-sampling method. In addition, dimensionality reduction techniques performed well for cross-project prediction rather than within-project prediction. Chen et al. [21] have empirically analyzed the effect of feature selection methods on machine learning. It has used filter-based ranking methods due to the high cost of computation of other methods. The paper has applied ChiSquared, $F$-score, GainRatio, InfoGain, GiniIndex, FisherScore, and ReliefF filter-based ranking methods on three PHP

datasets using a random forest machine classifier. The paper has shown an increase in the performance of VPMs.

Bassi and Singh [23] examine the effect of dual HPO on metrics-based VPMs. This study proposes an approach for optimizing hyperparameters for machine learners and data balancing strategies using the Python framework Optuna. It compared six combinations of five machine learners and five resampling approaches using default values and optimized hyperparameters for experimentation. The article discovered that dual HPO outperforms HPO on learners and HPO on data balancers. Furthermore, it investigated the impact of data complexity measures and concluded that HPO did not increase the performance of datasets with substantial overlap.

Rhmann [24] has proposed a new technique by combining the grey-wolf metaheuristic technique and random forest. The paper has emphasized finding the best subset of relevant features. It has shown that metaheuristic algorithms combined with random forest performed better than other machine learning algorithms. Particle swarm optimization with random forest outperforms the baseline methods. Sahin et al. [25] suggest a unique deep learning method and SYMbiotic Genetic algorithm for software vulnerability prediction. They have applied Diploid Genetic algorithms with deep learning networks on software vulnerability prediction. The deep SYMbiotic-based genetic algorithm model (DNN-SYMbiotic GAs) is employed in this suggested method to solve challenges involving the prediction of software vulnerabilities. On many benchmark datasets from the PHPMyAdmin, Moodle, and Drupal projects, extensive experiments are carried out. According to the results, the suggested approach (DNN-SYMbiotic GAs) improved vulnerability prediction, which implies better software quality prediction.

Viszkok et al. [26] have worked on the dataset produced by [12] by including the process metrics and observed that $F$-measure has improved by 8.4%, precision by 3.5%, and recall by 6.3%. Kalouptsoglou et al. [27] have used three feature types of metrics, text-tokens, and a combination of both to model the VPMs. It has proposed a text token-based dataset of JavaScript programs used in [12] and a new metric $F_2$-score. The paper concludes that text-tokens-based models perform better than metrics-based models in terms of $F_2$-score and the combination has not made much difference in the predictive performance.

Wang and Yao [28] tackled the class imbalance problem in defect prediction models by using under-sampling techniques, ensemble-learning techniques, and threshold moving techniques on naive bayes and random forest classifiers. The paper concludes that balanced random under-sampling shows a better defect prediction rate. Adaboost has performed best in increasing the efficacy of SDP models. The overall performance is measured using G-mean, AUC, and balance metrics. Shin et al. [29] evaluated the VPMs using code churn, complexity, and developer activity metrics on Linux and Firefox projects. The model has an inspection rate of less than 30% and a recall of 70%. Shin and Williams [30] analyze whether fault prediction models also work for vulnerability prediction by performing experiments on Mozilla Firefox which contains 21% of files with faults and 3% of files with vulnerabilities. It concluded that fault prediction models are equally capable as VPMs in predicting vulnerabilities. Furthermore, it suggested the models attain better recall and low false positives.

Lagerstrom et al. [31] experimented on the Google Chromium project to inspect the relationship that software vulnerabilities have with two types of metrics namely architecture coupling metrics and component-level metrics. The results reveal that vulnerable files are in correlation with both types of metrics. Zhang et al. [32] proposed a VULPREDICTOR that works on the combination of software metrics and text features. The paper performed

experiments on three PHP datasets Drupal, Moodle, and PHPMyAdmin. It has achieved the $F_1$-score of 0.683 and EffectivenessRatio@20% to 75%. Abunadi et al. [33] explain how cross-project prediction techniques are useful in software vulnerability prediction. This paper results in the high recall, precision, and $F$-measure of J48 and random forest but has not applied data balancing techniques.

Khalid et al. [34] proposed NMPREDICTOR which consists of two tiers. The first tier contains 6 classifiers that are built on the training set of labeled metrics and text features files. In the second tier, a meta-classifier combining all 6 classifiers random forest, J48, and naïve bayes (both metrics and text) is built. This paper has experimented on PHP datasets and resulted in the highest $F_1$-score of 0.848. Catal et al. [35] implemented a web service for software vulnerability prediction. The paper uses the Azure machine learning platform to build the web service. Several machine learning algorithms are applied to the PHP datasets. For the performance evaluation, the Area under the ROC (AUC) metric is used. This paper concludes that the multilayer perceptron model has produced the best results.

Li and Shao [36] surveyed the prediction of software vulnerabilities using feature selection-based machine learning. This paper has classified the existing research works into 4 different feature types such as metrics, text mining features, graph, and taint analysis. It has discussed the advantages and challenges of machine learning in software vulnerability prediction. Solutions to the three main challenges namely selection of relevant features, class imbalance, and label data high cost are illustrated and further work has been discussed for the future. Rostami et al. [22] compares and categorizes various feature selection methods. The paper focuses on increasing the accuracy of prediction models through the use of metaheuristic algorithms. It has analyzed the performance of eleven swarm intelligence-based feature selection methods on six medical datasets from the UCI repository and three machine learning algorithms namely support vector machine, naïve bayes, and adaboost. In addition, it has discussed the pros and cons of each metaheuristic algorithm.

Apart from the above research studies, there are deep learning methods that are less sensitive to feature selection methods. Sonekalb et al. [37] provide an SLR which aims to do a detailed analysis and comparison of 32 primary works on DL-based vulnerability analysis of program code. It discovered a wide range of proposed analysis methods, code embeddings, and network topologies. They go over these strategies and alternatives in great depth and identify the current level of research in this field and suggest future directions by collating commonalities and contrasts in the techniques. To facilitate a stronger benchmarking of approaches, it also presents an overview of publicly available datasets. This SLR serves as an overview and jumping-off point for researchers interested in performing deep vulnerability analysis on program code.

Li et al. [38] have introduced VulDeePecker, the first deep learning-based vulnerability detection system, to relieve human experts from the tiresome and subjective labor of manually defining features and lowering the false negatives that are experienced by previous vulnerability detection systems. To assess the performance of VulDeePecker and other deep learning-based vulnerability detection systems that will be created in the future, they have gathered and made publicly available a relevant dataset. According to experimental findings, VulDeePecker can yield significantly fewer false negatives (with acceptable false positives) than other methods. They also applied VulDeePecker to 3 software products (Xen, Seamonkey, and Libav) and find 4 vulnerabilities that were "silently" patched by the vendors when they released later versions of these products but are not listed in the National Vulnerability Database; in contrast, these vulnerabilities are almost entirely missed by the other vulnerability detection systems they tested.

Zhou et al. [39] propose Devign, a general graph neural network-based model for graph-level classification through learning on a rich set of code semantic representations, which is inspired by the work on manually-defined patterns of vulnerabilities from various code representation graphs and the most recent development in graph neural networks. To effectively extract meaningful features from the learned rich node representations for graph-level classification, it contains a unique Conv module. The model is trained on manually labeled datasets constructed from four diverse, large-scale open-source C projects that use real source code with high levels of complexity and variation rather than the synthesis code employed in earlier efforts.

Li et al. [40] introduced a vulnerability detector, which can simultaneously achieve a high detection capability and a high locating precision, powered by deep learning called VulDeeLocator. The challenges while designing VulDeeLocator include how to support accurate control flows and variable define use relations, how to achieve high locating precision, and how to support semantic relations between the definitions of types as well as macros and their uses across files. They overcome these challenges by utilizing two novel concepts: (i) using intermediate code to accept more semantic information, and (ii) using the idea of granularity refinement to identify vulnerable areas. VulDeeLocator finds 18 verified vulnerabilities (also known as true-positives) when applied to 200 files randomly chosen from three different real-world software applications. Sixteen of these correspond to known vulnerabilities; the other two are not documented in the National Vulnerability Database (NVD) but have been "silently" corrected by Libav's manufacturer when fresh versions are released.

Kalouptsoglou et al. [41] explores if combining deep learning and software metrics might improve cross-project vulnerability prediction. Several machine learning models, including deep learning, are assessed and contrasted using a dataset of prominent real-world PHP software applications. Feature selection is evaluated for its impact on cross-project prediction. There investigation suggests that using software metrics and deep learning can improve vulnerability prediction models' performance across projects. The study found that cross-project prediction models perform better when projects share similar software metrics.

### 2.1. Comparisons with existing works

Table 1 compares our work with the existing works where PHP and JavaScript datasets are used. It describes the feature selection techniques, performance metrics applied, machine learning techniques used, whether data balancing is performed or not, features for modeling VPMs, i.e., metrics, text features, or combination of both. The current work is highly inspired by Rhmann et al. [24] and tried to work on different combinations of machine learning methods and metaheuristic feature selection methods. It is observed that the proposed work has tried to incorporate maximum performance metrics which suit the imbalanced datasets.

Previous works have included the PHP dataset and JavaScript datasets separately but this paper includes both to validate the work. In addition to this, it includes four performance metrics, eight machine learning algorithms, six feature selection methods, SMOTE resampling technique, two feature types, and finally two different language datasets with distinct granularity levels.

Table 1: Comparisons with existing works

| Research papers | Feature selection techniques | Performance metrics | Features | Machine learning techniques used | Resampling techniques | Datasets used |
|---|---|---|---|---|---|---|
| Kaya et al. [8] | NO | AUC, Precision, Recall, $F_1$-score, Specificity | Software Metrics, Text Features, Combination of software metrics and text features | RF, AB, Linear Discriminant, Linear SVM, Weighted KNN, Subspace Discriminant, Rusboost | Smote, Adasyn, ClusterSmote, BLSmote | Drupal, Moodle, PHPMyAdmin |
| Walden et al. [10] | NO | Recall, Precision | Software Metrics, Text Features | RF | Under-sampling technique used (Weka SpreadSubsample) | Drupal, Moodle, PHPMyAdmin |
| Ferenc et al. [12] | NO | $F$-Measure | Software Metrics | Simple Deep Neural Network (DNNs), Complex Deep Neural Network (DNNc), KNN, SVM, RF, LR, Linear Regression, Gaussian NB(GNB), DT | Over-sampling with ratios (25%, 50%, 75%, 100%) and Under-sampling with ratios (25%, 50%, 75%, 100%) | JavaScript Dataset |
| Kudjo et al. [13] | NO | Precision, Recall, Accuracy | Software Metrics | RF, KNN, SVM, DT | No Resampling | Drupal, Moodle, PHPMyAdmin |
| Stuckman et al. [20] | Feature subset selection, Entropy Reduction, Principal Component Analysis (PCA), Sparse PCA, Confirmatory Factor Analysis (CFA) | Recall, $F_1$-score, Inspection Ratio | Software Metrics, Text Features | RF | Under-sampling, Smote | Drupal, Moodle, PHPMyAdmin |
| Rhmann et al. [24] | PSO, GWO, GA | Precision, Recall, $F$-Measure | Software Metrics | RF | SMOTE | Drupal, Moodle, PHPMyAdmin |
| Sahin et al. [25] | Symbiotic Genetic Algorithm (I and II) | Accuracy, $F_1$-score, Precision | Software Metrics | Artificial Neural Networks, Long-Short-Term-Memory (LSTM), Gated Recurrent Unit (GRU) | No Resampling | Drupal, Moodle, PHPMyAdmin |

Table 1 continued

| Research papers | Feature selection techniques | Performance metrics | Features | Machine learning techniques used | Resampling techniques | Datasets used |
|---|---|---|---|---|---|---|
| Viszkok et al. [26] | NO | Accuracy, Precision, Recall, $F$-Measure | Software Metrics | Simple Deep Neural Network (DNNs), Complex Deep Neural Network (DNNc), KNN, SVM, RF, LR, Linear Regression, Gaussian NB(GNB), DT | Over-sampling with ratios (25%, 50%, 75%, 100%) and Under-sampling with ratios (25%, 50%, 75%, 100%) | JavaScript Dataset |
| Kalouptsoglou et al. [27] | Point-BiSerial Correlation (PBSC) | Accuracy, Precision, Recall, $F_1$-score, $F_2$-score | Software Metrics, Text Features, Combination of software metrics and text features | DT, RF, NB, SVM, KNN, MLP | Random resampling | JavaScript Dataset |
| Zhang et al. [32] | NO | Precision, Recall, $F_1$-score | Combination of software metrics and text features | RF, NB, DT | No Resampling | Drupal, Moodle, PHPMyAdmin |
| Abunadi et al. [33] | NO | Precision, Recall, $F$-Measure | Software Metrics | NB, LR, SVM, RF, DT | No Resampling | Drupal, Moodle, PHPMyAdmin |
| Khalid et al. [34] | NO | Accuracy, Precision, Recall, $F_1$-score | Combination of software metrics and text features | RF, NB, DT | SMOTE | Drupal, Moodle, PHPMyAdmin |
| Catal et al. [35] | NO | AUC | Software Metrics | Averaged Perceptron, Bayes point machine, Boosted Decision Tree, Decision Forest, Decision jungle, Deep SVM, SVM, Logistic Regression, Multilayer Perceptron | No Resampling | Drupal, Moodle, PHPMyAdmin |
| Kalouptsoglou et al. [41] | Tree Based Elimination | Recall, Inspection Rate | Software Metrics | RF, SVM, MLP, XGBoost, Ensemble | Random UnderSampler | Drupal, Moodle, PHPMyAdmin |
| Current Work | JavaScript Dataset PSO, GA, GWO, HHO, SSA, WOA | AUC, Precision, Recall, $F_1$-score | Software Metrics, Text Features | DT, RF, NB, SVM, KNN, LR, AB, MLP | SMOTE | Drupal, Moodle, PHPMyAdmin, JavaScript Dataset |

## 3. Background knowledge

This section contains a brief explanation of the machine learning techniques (Section 3.1), resampling techniques used for balancing the datasets (Section 3.2), feature selection algorithms (Section 3.3), and performance evaluation metrics (Section 3.4).

### 3.1. Machine learning techniques

Machine Learning Techniques majorly are supervised and unsupervised. Supervised machine learning algorithms work on labeled data and unsupervised works on unlabeled data. The current paper uses eight supervised machine learning algorithms namely, random forest, support vector machine, $k$-nearest neighbor, adaboost, naïve bayes, logistic regression, decision tree, and multilayer perceptron. For comparisons with baseline methods, we are using these algorithms.

#### 3.1.1. Decision tree (DT)

Decision trees (DT) are non-parametric supervised machine learning algorithms [42]. This classifier is structured as a tree where internal nodes are the features, branches depict decision rules and each leaf node gives the outcome.

#### 3.1.2. Random forest (RF)

The random forest (RF) algorithm gives the output by taking a majority of the votes from numerous decision trees. RF is simple and can handle large datasets efficiently [43].

#### 3.1.3. Naïve Bayes (NB)

Naive Bayes is the supervised machine learning algorithm based on Bayes' theorem, assuming that there is independence among the features of the class. NB models are of four categories: Gaussian NB (GNB), Multinomial NB (MNB), Bernoulli NB (BNB), and Complement NB (CNB) [44].

#### 3.1.4. Adaboost (AB)

Adaboost, called adaptive boosting, is the boosting algorithm where weak learners are sequentially added and trained by weighted training data to build strong classifiers. The classification output is predicted by calculating the mean weights of the weak classifiers. It can use different base learners to boost its performance but is affected by noisy data and outliers [45].

#### 3.1.5. Support Vector Machine (SVM)

SVM is used to construct the best decision boundary called a hyperplane that separates multidimensional space into classes to place new data points in the correct class. SVM consists of various kernel functions (linear, polynomial, radial basis function, and sigmoid) used for the decision function. It has the overfitting issue, which arises when the number of features is much larger than the number of samples [46].

#### 3.1.6. $K$-Nearest Neighbor (KNN)

KNN classifies the data points by calculating the distance among them. New data points are classified by comparing them with the stored data. The value of $k$ is crucial to determine as a smaller value may cause underfitting and a larger value may cause overfitting [47].

#### 3.1.7. Logistic regression (LR)

Logistic regression predicts the probability of the target variable. In other words, dependent variables are predicted through the independent variables set [48].

#### 3.1.8. Multilayer perceptron (MLP)

Multilayer perceptron (MLP) is a feed-forward neural network that has three layers namely the input layer, hidden layer, and output layer. The input layer receives the signal for processing, the hidden layer acts as the computational engine and the output layer performs the prediction and classification tasks. MPs are used for non-linearly separable problems [49].

### 3.2. Resampling techniques

Resampling techniques are used to handle the class imbalance problem by balancing the datasets. The results may favor the majority class if the datasets are imbalanced. Therefore, to avoid biased results data balancing is important. Resampling Techniques are further classified as Under-sampling, Over-sampling, and Hybrid Sampling.

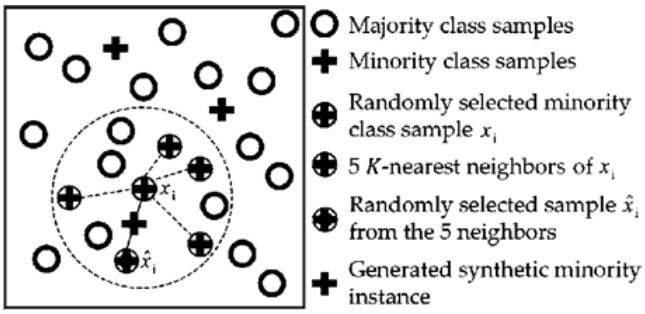


Figure 1. SMOTE process

The current study balances the dataset using SMOTE technique because it may not lead to loss of data and is highly used in previous studies. SMOTE [50] executes the $k$-nearest neighbor algorithm for synthetic sample generation. First, the minority class vector is found; second, the value of $k$ is selected; third, compute the distance between minority data points and any neighbor to plot a synthetic sample; and Lastly, the above step is repeated until the dataset is balanced. SMOTE obviates overfitting problems and increases the performance of the classifiers by generalizing the decision boundaries [51]. Figure 1 explains the SMOTE[1] process.

[1]https://rikunert.com/smote_explained

### 3.3. Feature selection algorithms

Previous studies have used feature synthesis algorithms for dimensionality reduction. Metaheuristic algorithms are of two types: Evolutionary Algorithms (EA) and Swarm Intelligence (SI). EA includes mechanisms based on biological evolution like mutation, reproduction, recombination, and selection. In EA, the initial population of individual solutions is generated randomly and a fitness function is used which is responsible for the quality of the solutions. After multiple iterations, the initial population evolved and reaches global optimization. Genetic algorithms are one example of evolutionary algorithms. SI is inspired by nature where each factor takes a simple task and exhibits a global intelligent behavior by having a factoring relationship with one another and their random reactions. There exists a plethora of SI-based feature selection algorithms such as particle swarm optimization (PSO), Gravitational Search Algorithm (GSA), Ant Colony Optimization (ACO), Differential Evolution (DE), Artificial Bee Colony Optimization (ABC), Firefly Algorithm (FA), Cuckoo Optimization Algorithm (COA), Bat Algorithm (BA), Grey Wolf Optimization (GWO), Salp Swarm Algorithm (SSA), Whale Optimization Algorithm (WOA), and Harris–Hawk Optimization (HHO).

This paper has incorporated 6 feature subset selection metaheuristic algorithms. It has applied three metaheuristic algorithms GA, PSO, and GWO from the base paper [24] but with seven other machine learning algorithms which are highly popular [2]. In addition to this, whale optimization (fish-based), harris hawk (bird-based), and salp swarm(sea-based) algorithms are mostly used in prediction areas like fault and defect prediction [52–55]]. Furthermore, the study used the feature selection code from GitHub[2] which contained 13 metaheuristic algorithms and we tried to implement the latest and swarm-based algorithms which were HHO, WOA, and SSA.

#### 3.3.1. Genetic algorithms (GA)

A genetic algorithm is a population-based metaheuristic algorithm that imitates natural evolutionary mechanisms. It involves initial population production, selection of good solutions, fitness function definition, crossover, and mutation. Initial population generation includes all the possible solutions to the given problem. The fitness function assigns the fitness score to each individual and the individual with a higher fitness score has a higher chance of being selected. The selection phase creates a region with a high probability of producing the best solution. Reproduction has two operators: crossover and mutation. Crossover interchanges the genetic information of two parents for reproduction. The child population generated is the same in size as the parent population. New genetic information is added to a new child population by changing some bits in the chromosome called Mutation [22].

#### 3.3.2. Particle swarm optimization (PSO)

Particle swarm optimization is inspired by the food search of a flock of birds or a school of fish. A bird flying in search of food randomly, and sharing its discovery can help in getting the best hunt for the entire flock. Each particle dynamically adjusts its velocity depending on its flying experiences and others in the group. Each particle keeps a record of its best result called pbest (personal best) and the best value of any particle called gbest (global

[2]https://github.com/JingweiToo/Wrapper-Feature-Selection-Toolbox-Python/tree/main/FS

best). The position of every particle is modified depending on its current position and velocity, the distance between pbest and its current position, and finally distance between gbest and its current position [22].

#### 3.3.3. Grey Wolf optimization (GWO)

Grey Wolf is inspired by grey wolves and mimics the hunting process of grey wolves in nature. Grey wolves live in groups of 5–12 individuals and follow a hierarchical management system. The social hierarchy of grey wolves consists of Alpha ($\alpha$) the head of the group and their orders are directions followed by other wolves. Beta ($\beta$) are the subordinates that aid $\alpha$ in making decisions. Delta ($\delta$) are scouts which report to $\alpha$ and $\beta$. Lastly, Omega ($\omega$) is at the bottom of the management system and is accountable for internal relationships. Grey Wolf Hunting has three phases: chasing and approaching the prey, encircling and harassing the prey, and attacking the prey [22].

#### 3.3.4. Whale optimization algorithm (WOA)

A whale optimization algorithm is a metaheuristic algorithm inspired by the hunting mechanisms of humpback whales. It is easy to implement and robust. Humpback whales search for food in multidimensional space. The algorithm imitates the bubble-net foraging method of searching and attacking the prey by the whales. When the whale locates its prey, it forms a bubble-net spiral path and reaches upwards to the prey. There are three stages to explain predation behavior: surrounding the prey, bubble net attack, and hunting the prey [22].

#### 3.3.5. Salp swarm algorithm (SSA)

A salp swarm algorithm is stimulated by the sea salps' swarming behavior [55]. The salp population is divided into leaders and followers. The salp chain is created when salps shape the swarm in heavy oceans. The leader lies in the front of the chain and the rest are the followers. The SSA algorithm starts with the initialization of the group of solutions randomly. Further, the iterative improvement process tries to reach the global optimum. Follower salps update their position based on the leader's position. To reach the desired solution the process is executed repeatedly [22].

#### 3.3.6. Harris Hawk optimization (HHO)

Harris Hawk optimization algorithm is stimulated by the cooperative hunting behavior of harris hawks. They work in groups and attack the prey from all directions to surprise it. It has three stages which include surprise pounce, trailing the prey, and other attacking mechanisms. The first stage is Exploration which is to find and discover the prey or best candidate solution, second stage is the transformation from exploration to exploitation depending on the external/escaping energy of the prey. The third and final stage is Exploitation in which the prey is attacked hard or soft depending on the energy left with the prey [22].

### 3.4. Performance measures

The efficacy of the machine learning algorithms is evaluated using performance measures. Accuracy is measured by adding all correct predictions divided by the total predictions made by a machine learning algorithm. Since our datasets are imbalanced therefore we tend to use performance metrics that provide results without any biases [56]. The performance metrics Area under ROC curve (AUC), Precision, Recall, and $F_1$-score are highly used in previous studies so keeping in mind the comparative analysis, the current study also used the same metrics.

## 4. Research methodology

This section includes the experimental datasets, setup, performance results, statistical comparisons, and discussions of the results. Experiments are executed to evaluate the various combinations of metaheuristic feature selection and machine learning methods to analyze which combination has shown the highest performance.

### 4.1. Datasets

This paper uses eight datasets out of which six datasets (metrics and text-features) belong to PHP language and two datasets (metrics and text features) are based on JavaScript language. These are publicly available labeled datasets with labeling "NO" and "YES". The paper focuses on binary classification hence the datasets are suitable for the purpose. The granularity level for PHP datasets is "file" whereas, for a JavaScript dataset, it is a method/function. This indicates whether the components for vulnerability prediction are files or methods. The experiments are first performed on the PHP dataset then to check the validity of the work, it is implemented on another publicly available JavaScript dataset.

- PHP Dataset[3]: Drupal is the content management system with 202 total files and 62 vulnerable files. Moodle is a learning management system with 2924 total files and 24 vulnerable files. PHPMyAdmin is an open-source administration for MySQL with 322 total files and 27 vulnerable files.
- JavaScript Dataset[4,5]: The JSVulnerability dataset is collected from the Node Security Platform and the Snyk Vulnerability Database. It consists of 12 125 total functions and 1496 vulnerable functions.
- All the metrics-based and text-mining-based datasets are downloaded and saved in CSV files. VPMs are trained on numerical features called software metrics, which are the characteristics of source code. Text-mining-based VPMs are trained on text features called tokens gathered from source code using various text mining techniques such as Bag-of-words (BOW), Term Frequency (TF), Term Frequency-Inverse Document Frequency (TF-IDF), etc.
- Each CSV file that contains metrics datasets shows the dependent and independent variables. The dependent variable determines whether the file/function is vulnerable or not, depending on the values of independent variables. PHP dataset has 13 independent

[3]http://seam.cs.umd.edu/webdata
[4]http://www.inf.u-szeged.hu/~ferenc/papers/JSVulnerabilityDataSet/
[5]https://sites.google.com/view/vulnerability-prediction-data/home

variables and one dependent variable. The JavaScript dataset contains 35 independent variables and one dependent variable.
– Table 2 describes the versions, no. of total files, no. of vulnerable files, no. of vulnerabilities, metrics features, and text features of the datasets used for experimentation
– Table 9 (refer to Appendix) gives the metrics description of the PHP and JavaScript datasets.

Table 2. Dataset descriptions

| Dataset | Version | Total files /functions | Vulnerable files/functions | Vulnerabilities | Metrics | Text features |
|---|---|---|---|---|---|---|
| Drupal | 6.0 | 202 | 62 | 97 | 13 | 3811 |
| PHPMyAdmin | 3.3.0 | 322 | 27 | 75 | 13 | 5232 |
| Moodle | 2.0.0 | 2924 | 24 | 51 | 13 | 18306 |
| JSVulnerability | – | 12 125 | 1496 | – | 35 | 12 942 |

### 4.2. Text mining

Text mining is the preprocessing of textual features and converting them into vector form which is the input for machine learning algorithms [57, 58]. There exist a lot of methods for performing text mining such as Bag-of-words (BOW), Term-Frequency (TF), Term-Frequency inverse document frequency (TF-IDF), sequence of tokens, etc.

Recent studies [8, 10, 20, 32, 34]] have mainly used BOW for preprocessing PHP dataset text features and for JavaScript datasets BOW and sequence of tokens have been used [12, 26, 27]. BOW calculates the number of occurrences of each word/token in the whole file/method. In text-mining-based VPMs, each token is considered a feature in the source code. To obtain text features from the source code of the Drupal dataset, the study first saved the textual dataset in Microsoft Access. There is a total of 202 files for the Drupal dataset and each file is labeled as vulnerable or not. To convert the source code into vector form, textual analysis is performed on the source code to remove redundant features, white spaces, punctuation marks, arithmetic, and logical operators. Then a dictionary/vocabulary is created which includes all the tokens associated with a key (see: Fig. 2). Each row in the MS-Access table represents a file which is compared against the dictionary to find out the occurrence of each token in the file. Finally, the CSV file is generated that contains the value of each token in the file. CSV file is further processed in MS-Excel to create each column heading indicating text-feature shown in Figure 3.

### 4.3. Experimental setup

In this paper, experiments are performed on four metrics and four text-features datasets. In addition to this, six metaheuristic feature selection approaches and eight machine learning algorithms are used. SMOTE technique is applied, keeping in view the imbalanced nature of the datasets. The experiments are performed in Jupyter notebook python by replicating the pseudocode mentioned in [24] with some additions mentioned in Algorithm below.

The new pseudocode presented in Table 3, includes eight datasets and four evaluation metrics (precision, recall, AUC, $F_1$-score). Every metaheuristic and machine learning algorithm uses this pseudocode to get the desired results. The number of generations and Population size is set to 100 and 10, respectively. The hyperparameters of machine learning,

| | ID | Tokens | IsVulnerable |
|---|---|---|---|
| 0 | 12 | 'T_OPEN_TAG T_FUNCTION(db_status_report) ( ) {... | no |
| 1 | 13 | 'T_OPEN_TAG T_STRING ( , T_LNUMBER ) ; T_STRIN... | yes |
| 2 | 14 | 'TOPENTAG TFUNCTIONdrupalgetform TVARIABLEfor... | yes |
| 3 | 15 | 'T_OPEN_TAG T_FUNCTION(image_gd_info) ( ) { T_... | no |
| 4 | 16 | 'T_OPEN_TAG T_FUNCTION(image_get_available_too... | no |
| ... | ... | ... | ... |
| 197 | 7 | 'TOPENTAG TSTRING TLNUMBER TSTRING TLNUM... | yes |
| 198 | 8 | 'T_OPEN_TAG T_STRING ( , ) ; T_FUNCTION(update... | yes |
| 199 | 9 | 'T_OPEN_TAG T_FUNCTION(db_query) ( T_VARIABLE(... | no |
| 200 | 10 | 'T_OPEN_TAG T_REQUIRE_ONCE ; T_FUNCTION(db_sta... | no |
| 201 | 11 | 'T_OPEN_TAG T_REQUIRE_ONCE ; T_FUNCTION(db_sta... | no |

| | {'tandequal': 2 | 'tarray': 3 | 'tarraycast': 4 | 'tas': 5 | 'tboolcast': 6 | 'tbooleanand': 7 | 'tbooleanor': 8 | 'tbreak': 9 | 'tcase': 10 | 'tclone': 11 | ... | 'tvariableyear': 3804 | 'tvariableyes': 3805 | 'tvariablezel |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 12 | 0 | 6 | 2 | 7 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |
| 1 | 0 | 38 | 0 | 5 | 0 | 19 | 6 | 2 | 9 | 0 | ... | 0 | 0 | |
| 2 | 0 | 89 | 0 | 18 | 0 | 54 | 19 | 6 | 12 | 0 | ... | 0 | 0 | |
| 3 | 0 | 6 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | ... | 0 | 0 | |
| 4 | 0 | 11 | 0 | 1 | 0 | 3 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 197 | 0 | 239 | 3 | 44 | 1 | 55 | 23 | 16 | 27 | 1 | ... | 0 | 0 | |
| 198 | 0 | 13 | 0 | 6 | 0 | 3 | 1 | 1 | 14 | 0 | ... | 0 | 0 | |
| 199 | 0 | 7 | 0 | 4 | 0 | 3 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |
| 200 | 0 | 4 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 | ... | 0 | 0 | |
| 201 | 0 | 4 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | ... | 0 | 0 | |

Figure 2. Conversion of source code into vector form

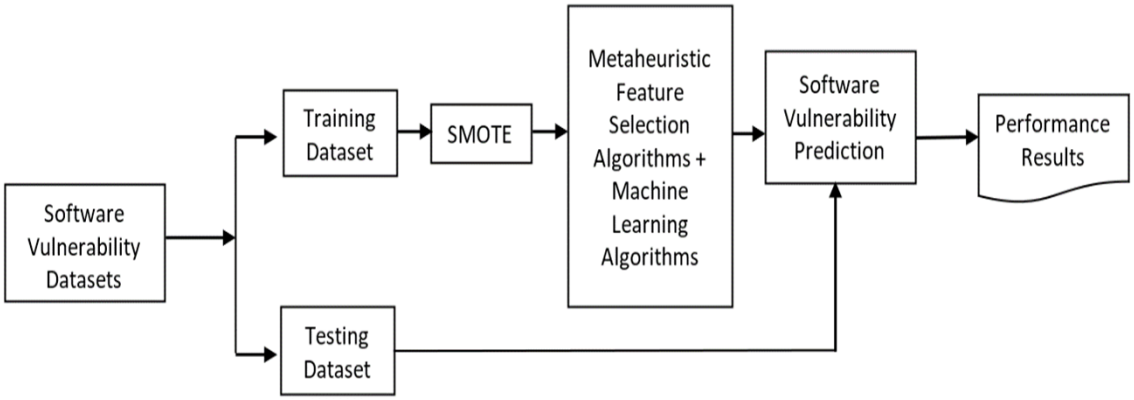


Figure 3. Working methodology

Table 3. Algorithm

**Input:** Dataset (D)
**Output:** Optimized cross-validated results (precision, recall, AUC, $F_1$-score)
1. Initialize the initial population of metaheuristic as: $\{a_1, a_2, a_3, \ldots, a_n\}$ where $a_i = [0, 1]$, 0 means feature is not selected, 1 means feature is selected and $n$ is the number of features in the dataset
2. Take the fitness function as $F_1$-score of ML on the subset of D.
3. Repeat step 1 and 2 until the desired number of iterations or we get maximum $F$-measure = 1.

Pseudo-code of Metaheuristics-ML

**Input:** Vulnerability Dataset = $D\{Drupal_m, Drupal_t, Moodle_m, Moodle_t, PHPMyAdmin_m, PHPMyAdmin_t, JavaScript_m, JavaScript_t\}$
**Output:** Optimal values of precision, recall, AUC, $F_1$-score
1. Initialize the values: Number of dimension = independent features of dataset, Number of generation = $n$, population size = $N$, Take initial candidate solution as $\{a_1, a_2, a_3, \ldots, a_n\}$ where $a_i = [0, 1]$ n is the number of features in the dataset
2. For each iteration:
   - Selection features set $D'$ from $D$
   - Divide $D'$ into 80:20 ratio $\{D_tr, D_te\}$
   - Preprocess and standardize the dataset $D'$
   - Train a ML with $D_tr$
   - Evaluate the ML with 10-fold cross validation.
   - Return fitness = $F_1$-score

   After $n$ generation or $F_1$-score = 1
   Best $F_1$-score, precision, recall, AUC on 10-cross validation.
**End**

data balancing, and metaheuristic feature selection techniques are set to default. The main focus of this study is to compare the different combinations of ML and metaheuristic techniques hence the optimized hyperparameters are to be considered in the future. Figure 3 represents the working methodology of the VPMs. 80% of the dataset is split to train the machine learning classifier, and the classifier's performance is measured using 10-cross validation. The obtained $F$-measure serves as a fitness function for metaheuristic algorithms. The metaheuristic algorithm selects the best features based on the optimized AUC, precision, recall, and $F$-measures.

## 5. Results

The results are gathered after performing experiments and are represented in Tables A2–A9 of the Appendix. Tables A2–A5 show metrics-based results and Tables A6–A9 show text-mining-based results. Each table gives the evaluation metrics (AUC, Precision, Recall, $F_1$-score) values of each combination of machine learning algorithms and feature selection methods. Furthermore, there lies a column named "N_features" that depicts the count of features selected for every combination. In addition to this, for metrics-based datasets, the index of features selected is also mentioned but for text-features-based datasets, describing the index would be cumbersome. The bold values indicate the highest value of each performance metric among each machine learning algorithm and the yellow shaded + bold values indicates the highest value of each performance metric across all machine learning algorithm per dataset. Tables 4 and 5 represent the best-performing metaheuristic technique for each machine learning algorithm in each dataset.

Table 4. Best AUC values of metaheuristic feature selection for metrics-based VPMs in all machine learning algorithms

| Dataset | Machine learning technique | Best performing metaheuristic technique | AUC |
|---|---|---|---|
| Drupal | RF | SSA | 0.9643 |
| | SVM | SSA | 0.8928 |
| | KNN | HHO | 0.9658 |
| | DT | SSA | 0.9652 |
| | AB | GWO | 0.9656 |
| | NB | GWO | 0.8939 |
| | LR | GA | 0.9286 |
| | MLP | WOA | 0.8927 |
| Moodle | RF | SSA | 0.9573 |
| | SVM | GWO | 0.8659 |
| | KNN | PSO | 0.9616 |
| | DT | SSA | 0.9968 |
| | AB | GWO | 0.9745 |
| | NB | GA | 0.8031 |
| | LR | WOA | 0.8756 |
| | MLP | GA | 0.9439 |
| PHPMyAdmin | RF | HHO | 0.9661 |
| | SVM | GA | 0.8306 |
| | KNN | GWO | 0.9161 |
| | DT | GWO | 0.9833 |
| | AB | SSA | 0.9609 |
| | NB | SSA | 0.7638 |
| | LR | SSA | 0.8649 |
| | MLP | GWO | 0.9149 |
| JavaScript | RF | GA | 0.9705 |
| | SVM | GA | 0.8035 |
| | KNN | SSA | 0.9322 |
| | DT | PSO | 0.9605 |
| | AB | PSO | 0.8923 |
| | NB | PSO | 0.6929 |
| | LR | GA | 0.7362 |
| | MLP | HHO | 0.8745 |

### 5.1. Results for metrics-based VPMs

Table 4 describes the best-performing metaheuristic feature selection algorithm for each machine learning technique in the metrics-based VPMs:
- for Drupal, KNN-HHO has performed highest with AUC 0.9658,
- for Moodle, DT-SSA has performed highest with AUC 0.9968,
- for PHPMyAdmin, DT-GWO performed highest with AUC 0.9833,
- for JavaScript, RF-GA performed best with AUC 0.9705.

### 5.2. Results for text-features-based VPMs

Table 5 describes the best-performing metaheuristic feature selection algorithm for each machine learning technique in the text-features-based VPMs:
- for Drupal, MLP-GWO has performed highest with AUC 0.9986,
- for Moodle, DT-GWO has performed highest with AUC 0.9561,
- for PHPMyAdmin, AB-GWO performed highest with AUC 0.9879,

– for JavaScript, NB-HHO performed best with AUC 0.9998.

Table 5. Best AUC values of metaheuristic feature selection for text-feature-based VPMs in all machine learning algorithms

| Dataset | Machine learning technique | Best performing metaheuristic technique | AUC |
|---|---|---|---|
| Drupal | RF | GWO | 0.9666 |
| | SVM | GA | 0.9289 |
| | KNN | GWO | 0.9929 |
| | DT | GA | 0.9648 |
| | AB | GA | 0.9648 |
| | NB | PSO | 0.9289 |
| | LR | GWO | 0.9982 |
| | MLP | GWO | 0.9986 |
| Moodle | RF | HHO | 0.9469 |
| | SVM | SSA | 0.8031 |
| | KNN | PSO | 0.9421 |
| | DT | GWO | 0.9561 |
| | AB | GWO | 0.9315 |
| | NB | GA | 0.8631 |
| | LR | HHO | 0.9144 |
| | MLP | GWO | 0.9144 |
| PHPMyAdmin | RF | GWO | 0.9833 |
| | SVM | GWO | 0.9152 |
| | KNN | GA | 0.9833 |
| | DT | GWO | 0.9859 |
| | AB | GWO | 0.9879 |
| | NB | SSA | 0.8474 |
| | LR | PSO | 0.9666 |
| | MLP | GWO | 0.9878 |
| JavaScript | RF | WOA | 0.9995 |
| | SVM | HHO | 0.9788 |
| | KNN | WOA | 0.9896 |
| | DT | WOA | 0.9994 |
| | AB | GWO | 0.9995 |
| | NB | HHO | 0.9998 |
| | LR | GWO | 0.9912 |
| | MLP | WOA | 0.9995 |

The findings show that different machine learning algorithms have different best-performing metaheuristic feature selection techniques. The No-Free-Lunch (NFL) theorem states that no optimization or machine learning algorithm is good enough to solve all issues [59]. As a result, there is no guarantee that a single metaheuristic will uncover the best set of characteristics across all problem domains. Given these considerations, there is always the possibility of generating superior results with novel feature selection metaheuristics.

### 5.3. Statistical tests and results

Tables 6–9 show whether the feature selection methods have improved the performance of VPMs by comparing the values of each performance metric for both software metrics and text features-based datasets. The highest value of performance metrics is considered among different feature selection algorithms. Furthermore, the Wilcoxon signed rank statistical

test [60] is applied to identify whether text-mining-based VPMs perform better than metrics-based VPMs. The authors have performed hundred iterations of machine learning algorithm on different (metrics and text mining) datasets without feature selection and with feature selection. Considering Tables 6–9, for instance, hundred performance values (X-Samples) of RF(metrics) is compared with X-Samples of RF(text) in the without feature selection case. Similarly, the ML+FS combination with highest performance values are selected and their iterative values (X-Samples) are compared using Wilcoxon signed rank test. The significant $p$-value is considered to be 0.05.

The hypothesis is as follows:
$H_0$: Text-mining-based VPMs are better than metrics-based VPMs.
If the $p$-value is less than 0.05 then accept $H_0$, otherwise reject $H_0$. Accepting the hypothesis indicates that text-mining-based VPMs are better than metrics-based VPMs and rejecting the hypothesis indicates that metrics-based VPMs are better than text-mining-based VPMs. Tables 6–9, highlight the cases where the $p$-value is less than 0.05, therefore null hypothesis is accepted in them indicating text-mining-based VPMs are better than metrics-based VPMs.

## 6. Discussion

Imbalanced datasets, hyperparameter settings, and dimensionality of the dataset have degraded the performance of VPMs. This study is performed to find whether the combination of multiple metaheuristic feature selection and machine learning algorithms increases the efficacy of VPMs. In addition to this, the performance of text-features-based and metrics-based VPMs are compared. Furthermore, the focus is on finding the best metaheuristic technique and if not stating the reason behind it, also which technique has performed satisfactorily for all the datasets. These are illustrated through the answers to the research questions mentioned below.

### 6.1. Illustration of research questions

**RQ 1.** Has all the metaheuristics feature selection and machine learning combinations improved the efficacy of VPMs?

The comparison of various machine learning methods based on the usage of feature selection methods for each dataset is shown in Tables 6–9. The findings have shown that the feature selection method has improved the efficacy for both metrics-based and text-features-based VPMs with maximum performance metrics (AUC, Precision, Recall, and $F_1$-score) values of 0.9833, 0.9962, 0.9974, 0.9962 and 0.9986, 0.9994, 0.9996, 0.9997, respectively.

Table 6 shows the results for the Drupal dataset.

– It has been observed that for metrics-based VPMs AUC, Precision, Recall, and $F_1$-score have improved by 15.9%, 34.92%, 25.58%, and 34.98%, respectively.
– For text-mining-based VPMs AUC, Precision, Recall, and $F_1$-score have improved by 13.06%, 34.6%, 25.69%, and 31.94%, respectively.

Table 7 shows the results for Moodle dataset.

– For metrics-based VPMs, the performance metrics AUC, Precision, Recall, and $F_1$-score have improved by 14.37%, 96.56 %, 42.47%, and 93.69%, respectively.

Table 6. Comparison of various machine learning methods based on the usage of feature selection methods for the Drupal dataset

| Machine learning techniques | Without feature selection | | | | With feature selection | | | |
|---|---|---|---|---|---|---|---|---|
| | AUC | Precision | Recall | $F_1$-score | AUC | Precision | Recall | $F_1$-score |
| RF(metrics) | **0.8122** | 0.5993 | 0.6751 | **0.6341** | 0.9643 | 0.991 | 0.9912 | 0.9587 |
| RF(text) | **0.8681** | 0.6225 | **0.7422** | **0.6771** | 0.9666 | 0.9587 | 0.9289 | 0.9435 |
| $p$-value | 0.0014 | 0.0015 | 0.0051 | 0.0052 | 0.2541 | – | – | – |
| SVM(metrics) | 0.8006 | 0.5332 | 0.6316 | 0.5782 | 0.8928 | 0.9166 | 0.9925 | 0.9001 |
| SVM(text) | 0.8745 | 0.6397 | 0.6859 | 0.6619 | 0.9289 | 0.9145 | 0.9286 | 0.8848 |
| $p$-value | 0.0014 | 0.0001 | 0.0026 | 0.0001 | 0.0022 | – | – | – |
| KNN(metrics) | 0.7693 | 0.5327 | **0.7099** | 0.6086 | **0.9658** | 0.9941 | 0.9947 | **0.9753** |
| KNN(text) | 0.7732 | 0.6951 | 0.7215 | 0.7080 | 0.9929 | **0.9912** | 0.9899 | 0.9903 |
| $p$-value | 0.3321 | 0.0001 | 0.0018 | 0.0001 | 0.0009 | – | – | 0.0 |
| DT(metrics) | 0.6726 | 0.5196 | 0.5567 | 0.5091 | 0.9652 | 0.9898 | 0.9925 | 0.9582 |
| DT(text) | 0.6939 | 0.5301 | 0.5741 | 0.5196 | 0.9648 | 0.9333 | 0.9485 | 0.9608 |
| $p$-value | 0.0045 | 0.0042 | 0.0452 | 0.0412 | – | – | – | 0.0456 |
| AB(metrics) | 0.7747 | 0.5393 | 0.6661 | 0.5961 | 0.9656 | 0.9337 | 0.9957 | 0.9634 |
| AB(text) | 0.7915 | **0.6482** | 0.7011 | 0.6736 | 0.9648 | 0.9745 | 0.9486 | 0.9614 |
| $p$-value | 0.0014 | 0.0013 | 0.0036 | 0.0001 | – | 0.0035 | – | – |
| NB(metrics) | 0.7773 | **0.6475** | 0.4214 | 0.4952 | 0.8939 | **0.9948** | 0.8571 | 0.8888 |
| NB(text) | 0.8765 | 0.7088 | 0.7286 | 0.7185 | 0.9289 | 0.9231 | 0.8788 | 0.8888 |
| $p$-value | 0.0001 | 0.0036 | 0.0001 | 0.0001 | 0.0013 | – | 0.0012 | – |
| LR(metrics) | 0.6386 | 0.4941 | 0.5811 | 0.5341 | 0.9286 | 0.9915 | **0.9974** | 0.9194 |
| LR(text) | 0.7154 | 0.5715 | 0.6215 | 0.5954 | 0.9982 | 0.9911 | **0.9988** | **0.9949** |
| $p$-value | 0.0004 | 0.0001 | 0.0004 | 0.0051 | 0.0042 | – | 0.3156 | 0.0015 |
| MLP(metrics) | 0.7094 | 0.4274 | 0.5283 | 0.4725 | 0.8927 | 0.9090 | 0.9286 | 0.8965 |
| MLP(text) | 0.7615 | 0.5668 | 0.5507 | 0.5586 | **0.9986** | 0.9901 | 0.9928 | 0.9914 |
| $p$-value | 0.0026 | 0.0001 | 0.0365 | 0.0016 | 0.0001 | 0.0015 | 0.0015 | 0.0001 |

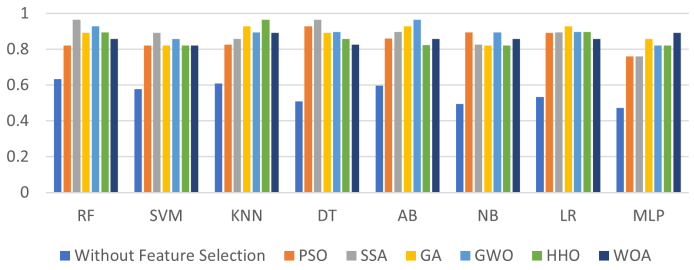


Figure 4. AUC Performance results for Drupal (metrics) dataset

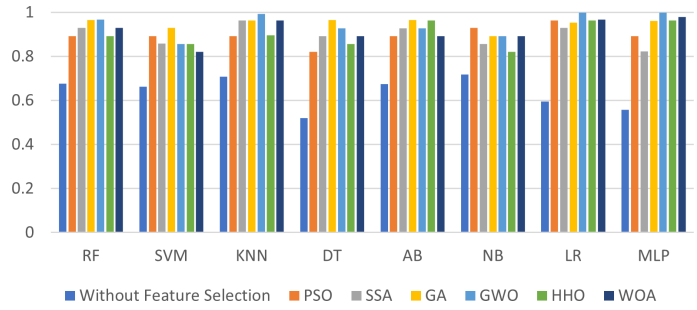


Figure 5. AUC Performance results for Drupal (text) dataset

Table 7. Comparison of various machine learning methods based on the usage of feature selection methods for the Moodle dataset

| Machine learning techniques | Without feature selection | | | | With feature selection | | | |
|---|---|---|---|---|---|---|---|---|
| | AUC | Precision | Recall | $F_1$-score | AUC | Precision | Recall | $F_1$-score |
| RF(metrics) | 0.7453 | 0.0207 | 0.0732 | 0.0294 | 0.9573 | 0.9406 | 0.9966 | 0.9581 |
| RF(text) | 0.6502 | 0.0159 | 0.1903 | 0.0295 | 0.9469 | 0.9364 | 0.9794 | **0.9475** |
| $p$-value | – | – | 0.0001 | 0.4851 | – | – | – | – |
| SVM(metrics) | 0.7833 | 0.0218 | 0.4708 | 0.0417 | 0.8659 | 0.8977 | 0.8459 | 0.8611 |
| SVM(text) | 0.8245 | 0.0358 | 0.5891 | 0.0676 | 0.8031 | **0.9941** | 0.6198 | 0.7589 |
| $p$-value | 0.0052 | 0.0152 | 0.0001 | 0.0152 | – | 0.0001 | – | – |
| KNN(metrics) | 0.6983 | 0.0229 | 0.3801 | 0.0432 | 0.9616 | 0.9269 | **0.9968** | 0.9605 |
| KNN(text) | 0.7035 | 0.0343 | 0.4014 | 0.0632 | 0.9421 | 0.9126 | 0.9829 | 0.9394 |
| $p$-value | 0.0452 | 0.0452 | 0.0452 | 0.0452 | – | – | – | – |
| DT(metrics) | 0.5292 | 0.0147 | 0.0784 | 0.0289 | 0.9968 | **0.9962** | 0.9966 | **0.9962** |
| DT(text) | 0.5708 | 0.0178 | 0.2423 | 0.0331 | **0.9561** | 0.9302 | 0.9623 | 0.9443 |
| $p$-value | 0.0052 | 0.0452 | 0.0001 | 0.0452 | – | – | – | – |
| AB(metrics) | 0.7419 | 0.0171 | 0.2553 | 0.0343 | **0.9745** | 0.9823 | 0.9863 | 0.9742 |
| AB(text) | 0.7689 | 0.0382 | 0.3641 | **0.0692** | 0.9315 | 0.8937 | 0.9897 | 0.9346 |
| $p$-value | 0.0098 | 0.0121 | 0.0001 | 0.0121 | – | – | 0.2465 | – |
| NB(metrics) | **0.8344** | **0.0342** | 0.3882 | **0.0628** | 0.8031 | 0.8969 | 0.7329 | 0.7882 |
| NB(text) | **0.8437** | **0.0487** | 0.3961 | 0.0867 | 0.8631 | 0.7849 | 0.9966 | 0.8707 |
| $p$-value | 0.0452 | 0.0016 | 0.0451 | 0.0021 | 0.0052 | – | 0.0001 | 0.0001 |
| LR(metrics) | 0.6501 | 0.0209 | **0.5734** | 0.0402 | 0.8756 | 0.8292 | 0.9589 | 0.8837 |
| LR(text) | 0.7276 | 0.0313 | **0.6166** | 0.0596 | 0.9144 | 0.8601 | **0.9978** | 0.9204 |
| $p$-value | 0.0004 | 0.0452 | 0.0098 | 0.0452 | 0.0041 | 0.0012 | 0.0013 | 0.0056 |
| MLP(metrics) | 0.6253 | 0.0237 | 0.3269 | 0.0442 | 0.9439 | 0.9218 | 0.9863 | 0.9449 |
| MLP(text) | 0.7121 | 0.0323 | 0.4256 | 0.0611 | 0.9144 | 0.8965 | 0.9966 | 0.9186 |
| $p$-value | 0.0001 | 0.0098 | 0.0001 | 0.0098 | – | – | 0.0425 | – |

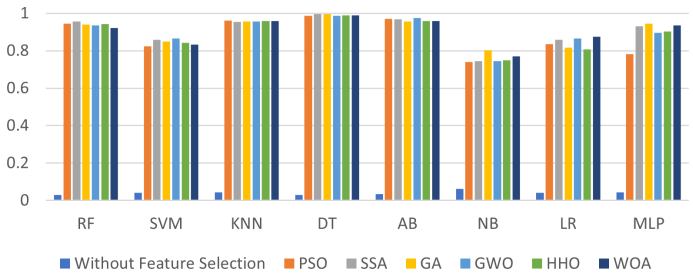


Figure 6. AUC Performance results for Moodle (metrics) dataset

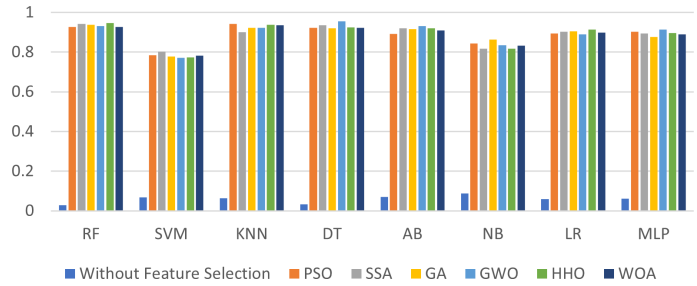


Figure 7. AUC Performance results for Moodle (text) dataset

Table 8. Comparison of various machine learning methods based on the usage of feature selection methods for the PHPMyAdmin dataset

| Machine learning techniques | Without feature selection | | | | With feature selection | | | |
|---|---|---|---|---|---|---|---|---|
| | AUC | Precision | Recall | $F_1$-score | AUC | Precision | Recall | $F_1$-score |
| RF(metrics) | **0.7536** | 0.2377 | 0.3562 | 0.2852 | 0.9661 | 0.9655 | 0.9489 | 0.9658 |
| RF(text) | **0.7651** | 0.3936 | 0.4253 | 0.4088 | 0.9833 | 0.9677 | 0.9945 | 0.9791 |
| $p$-value | 0.0251 | 0.0001 | 0.0001 | 0.0001 | 0.0026 | 0.3512 | 0.0045 | 0.0026 |
| SVM(metrics) | 0.7312 | 0.1949 | 0.54 | 0.2864 | 0.8306 | 0.8276 | 0.8279 | 0.8277 |
| SVM(text) | 0.7917 | 0.2559 | 0.5523 | 0.3497 | 0.9152 | 0.9311 | 0.90 | 0.9152 |
| $p$-value | 0.0098 | 0.0001 | 0.0251 | 0.0001 | 0.0001 | 0.0001 | 0.0004 | 0.0001 |
| KNN(metrics) | 0.6705 | 0.1332 | 0.5208 | 0.1988 | 0.9161 | 0.875 | 0.9655 | 0.9181 |
| KNN(text) | 0.6735 | 0.1588 | **0.5563** | 0.2971 | 0.9833 | 0.9666 | 0.9778 | 0.9722 |
| $p$-value | 0.3512 | 0.0251 | 0.0452 | 0.0001 | 0.0014 | 0.0001 | 0.0026 | 0.0056 |
| DT(metrics) | 0.6726 | **0.5196** | **0.5567** | **0.5375** | **0.9833** | **0.9777** | **0.9897** | **0.9831** |
| DT(text) | 0.7416 | 0.6441 | 0.6602 | 0.6521 | 0.9859 | 0.9677 | 0.9789 | 0.9736 |
| $p$-value | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.3516 | – | – | – |
| AB(metrics) | 0.6092 | 0.1398 | 0.3027 | 0.1913 | 0.9609 | 0.9666 | 0.9782 | 0.9665 |
| AB(text) | 0.6916 | 0.2854 | 0.3798 | 0.3259 | 0.9879 | 0.9667 | 0.9789 | 0.9727 |
| $p$-value | 0.0098 | 0.0001 | 0.0098 | 0.0001 | 0.0452 | 0.5462 | 0.5462 | 0.0452 |
| NB(metrics) | 0.7009 | 0.2165 | 0.3268 | 0.2605 | 0.7638 | 0.8184 | 0.8666 | 0.7762 |
| NB(text) | 0.7284 | 0.3514 | 0.4432 | 0.3919 | 0.8474 | 0.7631 | 0.8355 | 0.7976 |
| $p$-value | 0.0125 | 0.0001 | 0.0001 | 0.0001 | 0.0004 | – | – | 0.0452 |
| LR(metrics) | 0.6379 | 0.1661 | 0.2535 | 0.2007 | 0.8649 | 0.8846 | 0.90 | 0.8709 |
| LR(text) | 0.7782 | 0.2626 | 0.3101 | 0.2844 | 0.9666 | **0.9917** | 0.9333 | 0.9616 |
| $p$-value | 0.0001 | 0.0002 | 0.0041 | 0.0041 | 0.0001 | 0.0001 | 0.0452 | 0.0004 |
| MLP(metrics) | 0.6678 | 0.1532 | 0.4383 | 0.2271 | 0.9149 | 0.9285 | 0.90 | 0.9122 |
| MLP(text) | 0.6878 | 0.2733 | 0.5581 | 0.3669 | **0.9878** | 0.9756 | 0.9721 | 0.9693 |
| $p$-value | 0.0042 | 0.0452 | 0.0001 | 0.0001 | 0.0056 | 0.0056 | 0.0056 | 0.0065 |

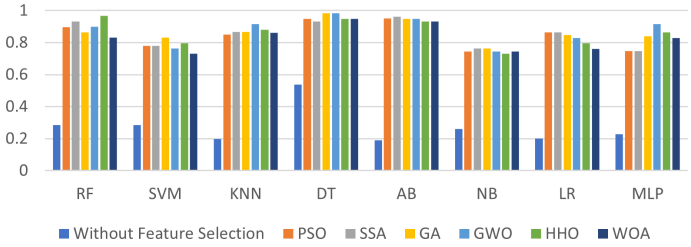


Figure 8. AUC Performance results for PHPMyAdmin (metrics) dataset

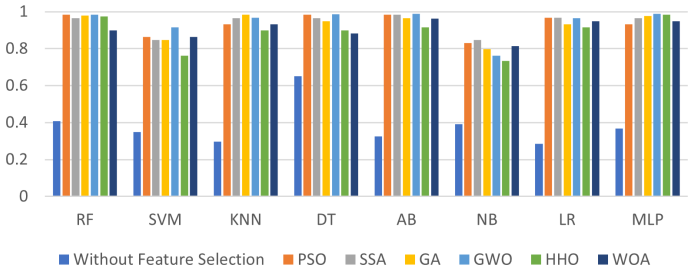


Figure 9. AUC Performance results for PHPMyAdmin (text) dataset

Table 9. Comparison of various machine learning methods based on the usage of feature selection methods for the JavaScript dataset

| Machine learning techniques | Without feature selection | | | | With feature selection | | | |
|---|---|---|---|---|---|---|---|---|
| | AUC | Precision | Recall | $F_1$-score | AUC | Precision | Recall | $F_1$-score |
| RF(metrics) | **0.9437** | **0.7261** | 0.7655 | **0.7458** | **0.9705** | **0.9798** | 0.9689 | **0.9701** |
| RF(text) | **0.9591** | 0.8792 | 0.8272 | 0.8523 | **0.9995** | **0.9994** | 0.9991 | 0.9995 |
| $p$-value | 0.0452 | 0.0001 | 0.0015 | 0.0001 | 0.0452 | 0.0452 | 0.0452 | 0.0452 |
| SVM(metrics) | 0.6024 | 0.5255 | 0.2404 | 0.3297 | 0.8035 | 0.8721 | 0.9322 | 0.7872 |
| SVM(text) | 0.7878 | 0.5335 | 0.2871 | 0.3733 | 0.9788 | 0.9593 | 0.9991 | 0.9793 |
| $p$-value | 0.0001 | 0.0551 | 0.0452 | 0.0245 | 0.0001 | 0.0001 | 0.0245 | 0.0001 |
| KNN(metrics) | 0.8742 | 0.5054 | 0.7406 | 0.5995 | 0.9322 | 0.9105 | **0.9671** | 0.9345 |
| KNN(text) | 0.9248 | 0.4897 | 0.8386 | 0.6201 | 0.9896 | 0.9869 | 0.9944 | 0.9897 |
| $p$-value | 0.0045 | – | 0.0004 | 0.0056 | 0.0023 | 0.0016 | 0.0056 | 0.0045 |
| DT(metrics) | 0.8611 | 0.6146 | **0.7713** | 0.6868 | 0.9605 | 0.9595 | 0.9633 | 0.9606 |
| DT(text) | 0.9598 | 0.8894 | 0.8474 | 0.8678 | 0.9994 | 0.9988 | 0.9978 | 0.9972 |
| $p$-value | 0.0001 | 0.0001 | 0.0035 | 0.0001 | 0.0045 | 0.0045 | 0.0045 | 0.0045 |
| AB(metrics) | 0.8777 | 0.4565 | 0.7149 | 0.5572 | 0.8923 | 0.9273 | 0.8561 | 0.8877 |
| AB(text) | 0.9003 | 0.5664 | 0.7689 | 0.6552 | 0.9995 | 0.9981 | 0.9958 | 0.9964 |
| $p$-value | 0.0452 | 0.0004 | 0.0452 | 0.0001 | 0.0001 | 0.0023 | 0.0001 | 0.0001 |
| NB(metrics) | 0.7772 | 0.6475 | 0.4214 | 0.4952 | 0.6929 | 0.6414 | 0.8749 | 0.7401 |
| NB(text) | 0.9435 | **0.9526** | **0.8985** | **0.9194** | 0.9998 | 0.9994 | 0.9992 | 0.9996 |
| $p$-value | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |
| LR(metrics) | 0.6772 | 0.2803 | 0.5371 | 0.3646 | 0.7362 | 0.7307 | 0.8118 | 0.7462 |
| LR(text) | 0.7231 | 0.3002 | 0.5518 | 0.3888 | 0.9912 | 0.9879 | 0.9884 | 0.9881 |
| $p$-value | 0.0343 | 0.0452 | 0.0452 | 0.0452 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |
| MLP(metrics) | 0.7913 | 0.3856 | 0.7449 | 0.5051 | 0.8745 | 0.8916 | 0.8758 | 0.8526 |
| MLP(text) | 0.8124 | 0.4152 | 0.7664 | 0.5386 | 0.9995 | 0.9995 | **0.9996** | **0.9997** |
| $p$-value | 0.0452 | 0.0452 | 0.0452 | 0.0452 | 0.0001 | 0.0001 | 0.0001 | 0.0001 |

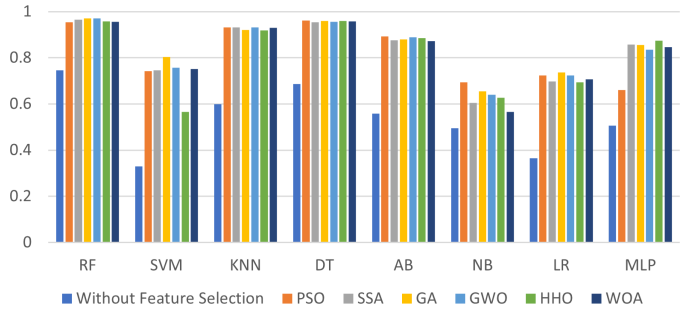


Figure 10. AUC Performance results for JavaScript (metrics) dataset

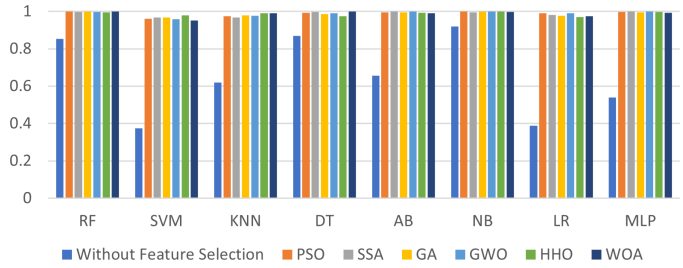


Figure 11. AUC Performance results for JavaScript(text) dataset

- For text-mining-based VPMs AUC, Precision, Recall, and $F_1$-score have improved by 11.75%, 42.47%, 38.21%, and 92.69%, respectively.
  Table 8 shows the results for the PHPMyAdmin dataset.
- For metrics-based VPMs, the performance metrics AUC, Precision, Recall, and $F_1$-score have improved by 23.3%, 46.85%, 43.75%, and 66.5%, respectively.
- For text-mining-based VPMs, the performance metrics AUC, Precision, Recall, and $F_1$-score have improved by 25.58%, 82.52%, 44.06%, and 79.37%.
  Table 9 shows the results for the JavaScript dataset,
- For metrics-based VPMs, the performance metrics AUC, Precision, Recall, and $F_1$-score have improved by 2.7%, 25.89%, 21.06%, and 23.12%, respectively.
- For text-mining-based VPMs, the performance metrics AUC, Precision, Recall, and $F_1$-score have improved by 4.04%, 4.68%, 10.11%, and 8.03%, respectively.

Figures 4–11 clearly show that after applying feature selection there is an improvement in the productivity of VPMs.

**RQ 2.** Which one statistically performed better, metrics-based or text-mining-based VPMs in the context of feature selection?
Tables 6–9 describe the statistical difference between metrics-based and text-mining-based VPMs after applying Wilcoxon signed rank test. For Drupal, without feature selection in all the cases text-mining-based VPMs have performed statistically better than metrics-based VPMs. After applying feature selection, in 13 out of 32 cases text-mining performed statistically better. For Moodle, without feature selection in 29 cases and with feature selection in 9 out of 32 cases text-mining-based VPMs performed better than metrics-based VPMs. For PHPMyAdmin, without feature selection, in 31 cases and with feature selection in 24 out of cases text-mining-based VPMs performed better than metrics-based VPMs. For JavaScript, without feature selection in 30 cases and with feature selection in all the cases text-mining-based VPMs performed better than metrics-based VPMs. Therefore, it is evident that text-mining-based VPMs statistically performed better than metrics-based VPMs in 60.9% of the cases in the context of feature selection.

**RQ 3.** Which metaheuristic feature selection algorithm has performed the best?
Different machine learning algorithms have different feature-selection methods that are performing best for each dataset. The No-Free-Lunch (NFL) theorem states that no optimization or machine learning algorithm is good enough to solve all issues [59]. As a result, there is no guarantee that a single metaheuristic will uncover the best set of characteristics across all problem domains. Given these considerations, there is always the possibility of generating superior results with novel feature selection metaheuristics. AUC performance metric is considered for describing the best-performing metaheuristic feature selection algorithm depicted in Figures 4–11. Figures 4, 6, 8, and 10 show results about metrics-based and Figures 5, 7, 9, and 11 text-features-based datasets.

For metrics-based VPMs, in the case of Drupal SSA, Moodle SSA, GWO, GA, PHPMyAdmin SSA, GWO, JavaScript GA, PSO has performed maximum for all machine learning algorithms (refer to Table 4). For text-features-based VPMs, in the case of Drupal GWO, Moodle GWO, PHPMyAdmin GWO, and JavaScript WOA have performed maximum times (refer to Table 5). Overall, it can be noticed that GWO has performed satisfactorily for all the datasets.

### 6.2. Comparison with benchmark studies

Figures 12 and 13 show the comparison of the proposed work with the existing benchmark studies for the PHP and JavaScript datasets, respectively. The paper has considered the $F_1$-score metric as the comparison criterion since it is preferably used in previous research studies. Figure 12 shows that Drupal and Moodle have the highest $F_1$-score for the proposed work whereas $F_1$-score for PHPMyAdmin is slightly less than Sahin et al. [25]. Sahin et al. [25] has not applied any data balancing technique which may produce biased results. Figure 13 shows that the proposed work's $F_1$-score has outperformed the benchmark studies.

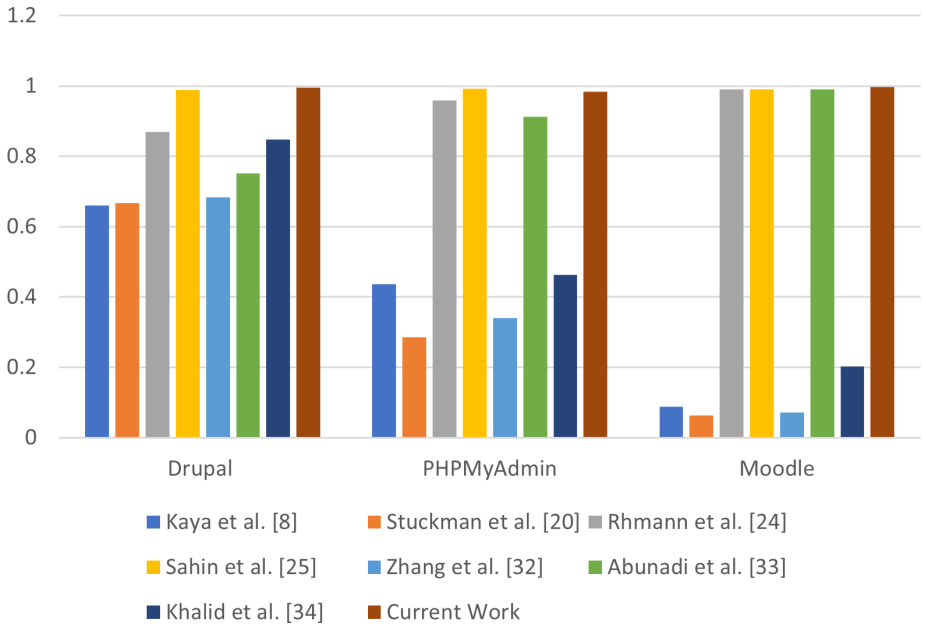


Figure 12. $F_1$-score performance comparison with existing studies for PHP dataset

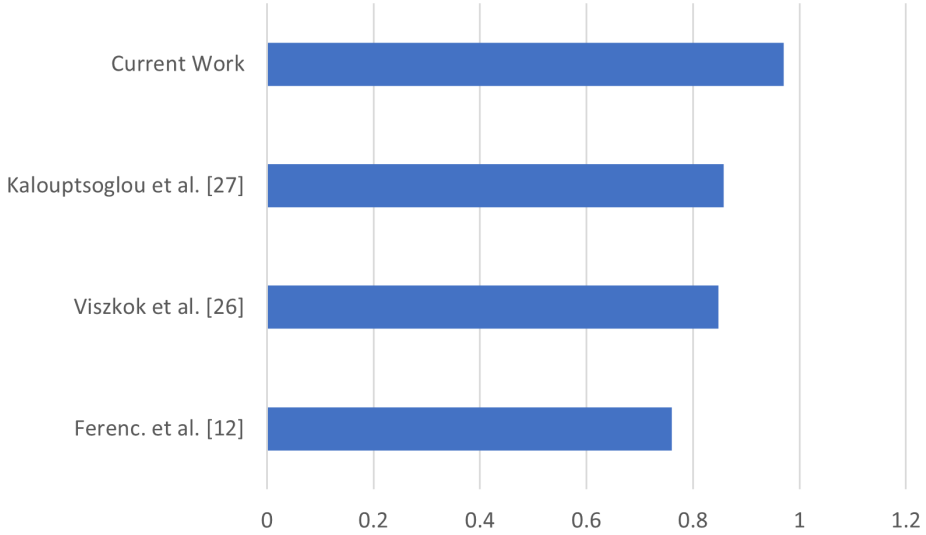


Figure 13. $F_1$-score performance comparison with existing studies for JavaScript dataset

### 6.3. Summary

Our study has unveiled the role of metaheuristic feature selection algorithms on the efficacy of VPMs. It has considered a wide variety of datasets, features, and machine-learning techniques and achieved high-performing VPMs with a maximum AUC of 0.9968 for metrics--based and 0.9998 for text-mining-based VPMs. Researchers can further use optimized hyperparameters and consider time and cost complexity issues for VPMs in the future.

## 7. Threats to validity

The current study covers the following threats to validity:

- Internal Validity: The selection of eight machine learning methods for the current study is based on previous research studies. Under-sampling techniques are not used due to loss of information. The paper has restricted its work to SMOTE data balancing techniques. Hybrid data balancing techniques are left for future scope.
- Construct Validity: This paper uses PHP-based open-source and JavaScript projects. The current study has included metrics-based datasets and text-mining-based datasets. The combination of the metrics and text-mining features is left for future scope. In addition to this, only default hyperparameters were considered and Bag of words was used for text mining. The authors are aware of the fact that optimized hyperparameters increase the performance of machine learning models [17] but are unaware about how would the combination of metaheuristic feature selection and optimized hyperparameters would perform thereby keeping it as a future aspect to be considered.
- Conclusion Validity: This paper uses AUC, precision, recall, and $F_1$-score for evaluating the performance of the prediction models. We have not used accuracy as they give biased results for imbalanced datasets. Also, considering the readability and clarity of the paper MCC and G-mean metrics are left for future scope.
- External Validity: The paper tries to perform the methodology on the PHP dataset and validate it be executing experiments on JavaScript. In addition, the granularity levels are also different, i.e., file for PHP and function for JavaScript. The work is confined to only two programming languages. In the future, more programming languages can be used and the results may vary.

## 8. Conclusions and future scope

The need for efficient VPMs has always been crucial and to achieve that, previous studies have done immense work, from balancing classes and appropriate hyperparameters selection to reducing the dimensionality through feature synthesis and feature selection methods. The present paper has worked on feature selection using nature-inspired and swarm intelligence-based algorithms. It has performed the empirical analysis on various combinations of eight machine learning techniques and six metaheuristic feature selection approaches on PHP and JavaScript datasets. The experiments are evaluated using six performance metrics, keeping the imbalanced nature of the datasets in mind. It has been used on metrics-based and text-token-based datasets. Further, the statistical comparison of metrics-based and text-mining-based VPMs is implemented by Wilcoxon signed rank test in the context of feature selection. The comparative analysis concludes that:

Metaheuristics feature selection methods improve the performance of VPMs (metrics and text-mining) in the range of 2.7%–25.58% in terms of AUC, 4.68%–96.56% in terms of precision, 10.11%–44.06% in terms of recall, and 8.03%–93.69% in terms of $F_1$-score.

The Wilcoxon signed rank test showed that overall 200 $p$-values are found significant out of 256 among all performance metrics. Therefore, overall it can be said that text-features-based VPMs are significantly better than metrics-based VPMs in 78.12% of the cases. But in the context of feature selection, 78 out of 128 cases performed significantly showing that text-features-based VPMs performed better than metrics-based VPMs for 60.9% of the instances when feature selection is applied.

The highest AUC values were obtained in metrics-based VPMs by KNN-HHO for Drupal, DT-SSA for Moodle, DT-GWO for PHPMyAdmin, and RF-GA for JavaScript. For text-mining based VPMs, the highest AUC values were obtained by MLP-GWO in Drupal, DT-GWO in Moodle, AB-GWO in PHPMyAdmin, and NB-HHO in JavaScript. The paper compares AUC values to find out the maximum-performing feature selection techniques for all machine learning algorithms. For metrics-based datasets, Drupal SSA; Moodle SSA, GWO, GA; PHPMyAdmin GWO; and JavaScript GA have performed the maximum times for all machine learning algorithms. For text-mining datasets, Drupal GWO, Moodle GWO, PHPMyAdmin GWO, and JavaScript WOA have performed maximum times. Overall, GWO has performed the maximum number of times. Furthermore, the present paper has outperformed the benchmark studies in terms of $F_1$-score.

In the Future, more deep learning methods like LSTM, GRU, etc., can be applied. Moreover, we have involved only default hyperparameters and in the future, analysis can be done using optimized hyperparameters. More metaheuristic algorithms can be applied and a combination of both metrics and text tokens can be used. Further, other programming languages can also be applied.

## CRediT authorship contribution statement

Dr. Deepali Bassi and Dr. Hardeep Singh contributed to the study's conception and design. Material preparation, data collection, and analysis were performed by Deepali Bassi. Hardeep Singh read and approved the final manuscript.

## Declaration of competing interest

The authors declare that they have no conflict of interest. We have no relevant financial or non-financial interests to disclose. We have no competing interests to declare that are relevant to the content of this article.

## Data availability

The Code Snippet is provided on the link https://drive.google.com/file/d/14qoY98ZPi-WalupRUquMALR_HHChuSaPB/view?usp=sharing

## Funding

No funding was received to assist with the preparation of this manuscript.

## References

[1] S. Matteson, "Report: Software failure caused $1.7 trillion in financial losses in 2017," *TechRepublic*, 2018. [Online]. https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017

[2] H. Hanif, M.H.N.M. Nasir, M.F. Ab Razak, A. Firdaus, and N.B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, Vol. 179, 2021, p. 103009.

[3] D.R. Kuhn, M.S. Raunak, and R. Kacker, "An analysis of vulnerability trends, 2008–2016," in *International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2017, pp. 587–588.

[4] S. Frei, "The known unknowns: Empirical analysis of publicly unknown security vulnerabilities," *NSS Labs*, 2013.

[5] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for Windows Vista," in *Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 421–428.

[6] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security and Privacy*, Vol. 3, No. 1, 2005, pp. 84–87.

[7] S.M. Ghaffarian and H.R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Computing Surveys (CSUR)*, Vol. 50, No. 4, 2017, pp. 1–36.

[8] A. Kaya, A.S. Keceli, C. Catal, and B. Tekinerdogan, "The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models," *Journal of Software: Evolution and Process*, Vol. 31, No. 9, 2019, p. e2164.

[9] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, Vol. 57, No. 3, 2011, pp. 294–313.

[10] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 23–33.

[11] K. Borowska and J. Stepaniuk, "Imbalanced data classification: A novel re-sampling approach combining versatile improved smote and rough sets," in *Computer Information Systems and Industrial Management: 15th IFIP TC8 International Conference*. Springer, 2016, pp. 31–42.

[12] R. Ferenc, P. Hegedűs, P. Gyimesi, G. Antal, D. Bán et al., "Challenging machine learning algorithms in predicting vulnerable JavaScript functions," in *IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. IEEE, 2019, pp. 8–14.

[13] P.K. Kudjo, S.B. Aformaley, S. Mensah, and J. Chen, "The significant effect of parameter tuning on software vulnerability prediction models," in *19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2019, pp. 526–527.

[14] E. Sara, C. Laila, and I. Ali, "The impact of smote and grid search on maintainability prediction models," in *IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2019, pp. 1–8.

[15] H. Osman, M. Ghafari, and O. Nierstrasz, "Hyperparameter optimization to improve bug prediction accuracy," in *Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*. IEEE, 2017, pp. 33–38.
[16] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," *arXiv preprint arXiv:1905.06872*, 2019.
[17] D. Bassi and H. Singh, "Optimizing hyperparameters for improvement in software vulnerability prediction models," in *Advances in Distributed Computing and Machine Learning*. Springer, 2022, pp. 533–544.
[18] H. Wang, Y. Zhang, J. Zhang, T. Li, and L. Peng, "A factor graph model for unsupervised feature selection," *Information Sciences*, Vol. 480, 2019, pp. 144–159.
[19] X. Tang, Y. Dai, and Y. Xiang, "Feature selection based on feature interactions with application to text categorization," *Expert Systems with Applications*, Vol. 120, 2019, pp. 207–216.
[20] J. Stuckman, J. Walden, and R. Scandariato, "The effect of dimensionality reduction on software vulnerability prediction models," *IEEE Transactions on Reliability*, Vol. 66, No. 1, 2016, pp. 17–37.
[21] X. Chen, Z. Yuan, Z. Cui, D. Zhang, and X. Ju, "Empirical studies on the impact of filter-based ranking feature selection on security vulnerability prediction," *IET Software*, Vol. 15, No. 1, 2021, pp. 75–89.
[22] M. Rostami, K. Berahmand, E. Nasiri, and S. Forouzandeh, "Review of swarm intelligence-based feature selection methods," *Engineering Applications of Artificial Intelligence*, Vol. 100, 2021, p. 104210.
[23] D. Bassi and H. Singh, "The effect of dual hyperparameter optimization on software vulnerability prediction models," *e-Informatica Software Engineering Journal*, Vol. 17, No. 1, 2023, p. 230102.
[24] W. Rhmann, "Software vulnerability prediction using grey wolf-optimized random forest on the unbalanced data sets," *International Journal of Applied Metaheuristic Computing (IJAMC)*, Vol. 13, No. 1, 2022, pp. 1–15.
[25] C.B. Şahin, Ö.B. Dinler, and L. Abualigah, "Prediction of software vulnerability based deep symbiotic genetic algorithms: Phenotyping of dominant-features," *Applied Intelligence*, Vol. 51, 2021, pp. 8271–8287.
[26] T. Viszkok, P. Hegedűs, and R. Ferenc, "Improving vulnerability prediction of JavaScript functions using process metrics," *arXiv preprint arXiv:2105.07527*, 2021.
[27] I. Kalouptsoglou, M. Siavvas, D. Kehagias, A. Chatzigeorgiou, and A. Ampatzoglou, "Examining the capacity of text mining and software metrics in vulnerability prediction," *Entropy*, Vol. 24, No. 5, 2022, p. 651.
[28] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, Vol. 62, No. 2, 2013, pp. 434–443.
[29] Y. Shin, A. Meneely, L. Williams, and J.A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, Vol. 37, No. 6, 2010, pp. 772–787.
[30] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Software Engineering*, Vol. 18, 2013, pp. 25–59.
[31] R. Lagerström, C. Baldwin, A. MacCormack, D. Sturtevant, and L. Doolan, "Exploring the relationship between architecture coupling and software vulnerabilities," in *Engineering Secure Software and Systems*. Springer, 2017, pp. 53–69.
[32] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun et al., "Combining software metrics and text features for vulnerable file prediction," in *20th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2015, pp. 40–49.
[33] I. Abunadi and M. Alenezi, "An empirical investigation of security vulnerabilities within web applications," *J. Univers. Comput. Sci.*, Vol. 22, No. 4, 2016, pp. 537–551.
[34] M.N. Khalid, H. Farooq, M. Iqbal, M.T. Alam, and K. Rasheed, "Predicting web vulnerabilities in web applications based on machine learning," in *Intelligent Technologies and Applications: First International Conference, INTAP*. Springer, 2019, pp. 473–484.

[35] C. Catal, A. Akbulut, E. Ekenoglu, and M. Alemdaroglu, "Development of a software vulnerability prediction web service based on artificial neural networks," in *Trends and Applications in Knowledge Discovery and Data Mining*. Springer, 2017, pp. 59–67.
[36] Z. Li and Y. Shao, "A survey of feature selection for vulnerability prediction using feature-based machine learning," in *Proceedings of the 11th International Conference on Machine Learning and Computing*, 2019, pp. 36–42.
[37] T. Sonnekalb, T.S. Heinze, and P. Mäder, "Deep security analysis of program code: A systematic literature review," *Empirical Software Engineering*, Vol. 27, No. 1, 2022, p. 2.
[38] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin et al., "Vuldeepecker: A deep learning-based system for vulnerability detection," *arXiv preprint arXiv:1801.01681*, 2018.
[39] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *Advances in neural information processing systems*, Vol. 32, 2019.
[40] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu et al., "Vuldeelocator: A deep learning-based fine-grained vulnerability detector," *IEEE Transactions on Dependable and Secure Computing*, Vol. 19, No. 4, 2021, pp. 2821–2837.
[41] I. Kalouptsoglou, M. Siavvas, D. Tsoukalas, and D. Kehagias, "Cross-project vulnerability prediction based on software metrics and deep learning," in *Computational Science and Its Applications – ICCSA*. Springer, 2020, pp. 877–893.
[42] H.H. Patel and P. Prajapati, "Study and analysis of decision tree based classification algorithms," *International Journal of Computer Sciences and Engineering*, Vol. 6, No. 10, 2018, pp. 74–78.
[43] Z. Jin, J. Shang, Q. Zhu, C. Ling, W. Xie et al., "RFRSF: Employee turnover prediction based on random forests and survival analysis," in *Web Information Systems Engineering – WISE*. Springer, 2020, pp. 503–515.
[44] M. Martinez-Arroyo and L.E. Sucar, "Learning an optimal naive Bayes classifier," in *18th International Conference on Pattern Recognition (ICPR'06)*, Vol. 3. IEEE, 2006, pp. 1236–1239.
[45] R.E. Schapire, "The boosting approach to machine learning: An overview," *Nonlinear estimation and classification*, 2003, pp. 149–171.
[46] C.C. Chang and C.J. Lin, "LIBSVM: A library for support vector machines," Vol. 2, No. 3, 2011. [Online]. https://doi.org/10.1145/1961189.1961199
[47] J.M. Keller, M.R. Gray, and J.A. Givens, "A fuzzy $k$-nearest neighbor algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, No. 4, 1985, pp. 580–585.
[48] H.F. Yu, F.L. Huang, and C.J. Lin, "Dual coordinate descent methods for logistic regression and maximum entropy models," *Machine Learning*, Vol. 85, 2011, pp. 41–75.
[49] M.C. Popescu, V.E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, Vol. 8, No. 7, 2009, pp. 579–588.
[50] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, Vol. 16, 2002, pp. 321–357.
[51] G. Douzas, F. Bacao, and F. Last, "Improving imbalanced learning through a heuristic oversampling method based on $k$-means and SMOTE," *Information Sciences*, Vol. 465, 2018, pp. 1–20.
[52] A. Zeb, F. Din, M. Fayaz, G. Mehmood, K.Z. Zamli et al., "A systematic literature review on robust swarm intelligence algorithms in search-based software engineering," *Complexity*, Vol. 2023, 2023.
[53] S. Kassaymeh, S. Abdullah, M.A. Al-Betar, and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem," *Journal of King Saud University – Computer and Information Sciences*, Vol. 34, No. 6, 2022, pp. 3365–3378.
[54] F.S. Gharehchopogh and H. Gholizadeh, "A comprehensive survey: Whale optimization algorithm and its applications," *Swarm and Evolutionary Computation*, Vol. 48, 2019, pp. 1–24.
[55] S. Mirjalili, A.H. Gandomi, S.Z. Mirjalili, S. Saremi, H. Faris et al., "Salp swarm algorithm: A bio-inspired optimizer for engineering design problems," *Advances in engineering software*, Vol. 114, 2017, pp. 163–191.

[56] J. Huang and C.X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, Vol. 17, No. 3, 2005, pp. 299–310.

[57] K.Z. Sultana, B.J. Williams, and A. Bosu, "A comparison of nano-patterns vs. software metrics in vulnerability prediction," in *25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 355–364.

[58] D. Wijayasekara, M. Manic, and M. McQueen, "Vulnerability identification and classification via text mining bug databases," in *IECON 40th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2014, pp. 3612–3618.

[59] D.H. Wolpert, "The supervised learning no-free-lunch theorems," *Soft computing and industry: Recent applications*, 2002, pp. 25–42.

[60] D. Tomar and S. Agarwal, "Prediction of defective software modules using class imbalance learning," *Applied Computational Intelligence and Soft Computing*, Vol. 2016, 2016, pp. 6–6.

## Appendix A

Table A1. Static source code metrics

| Dataset | Metrics | Description |
|---|---|---|
| PHP Dataset | nonecholoc | Non-HTML lines of code |
| | loc | Total lines of code in a PHP file |
| | nmethods | No. of functions in a file |
| | ccomdeep, ccom | Cyclomatic complexity |
| | nest | Maximum depth of nested loops |
| | hvol | Halstead's volume |
| | nIncomingCalls | Fan-in |
| | nIncomingCallsUniq | Internal functions Called |
| | nOutgoingInternCalls | Fan-out |
| | nOutgoingExternFlsCalled | Total external calls |
| | nOutgoingExternFlsCalledUniq | External methods called |
| | nOutgoingExternCalls | External calls to methods |
| JavaScript Dataset | CC | Clone Coverage |
| | CCL | Clone Classes |
| | CCO | Clone Complexity |
| | CI | Clone Instances |
| | CLC | Clone Line Coverage |
| | LDC | Lines of Duplicated Code |
| | McCC, CCYL | Cyclomatic Complexity |
| | NL | Nesting Level |
| | NLE | Nesting Level without else-if |
| | CD, TCD | Comment Density |
| | CLOC, TCLOC | Comment Lines of Code |
| | DLOC | Documentation Lines of Code |
| | LLOC, TLLOC | Logical Lines of Code |
| | LOC, TLOC | Lines of Code |
| | NOS, TNOS | Number of Statements |
| | NUMPAR, PARAMS | Number of Parameters |
| | HOR_D | No. of Distinct Halstead Operators |
| | HOR_T | No. of Total Halstead Operators |
| | HON_D | No. of Distinct Halstead Operands |
| | HON_T | No. of Total Halstead Operands |
| | HLEN | Halstead Length |
| | HVOC | Halstead Vocabulary Size |
| | HDIFF | Halstead Difficulty |
| | HVOL | Halstead Volume |
| | HEFF | Halstead Effort |
| | HBUGS | Halstead Bugs |
| | HTIME | Halstead Time |
| | CYCL DENS | Cyclomatic Density |

Table A2. Performance results of metrics-based Drupal dataset

| Machine Learning Algorithms | Feature Selection Techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.8214 | 0.9090 | 0.7143 | 0.80 | 5 [0, 2, 3, 5, 12] |
| | SSA | **0.9643** | **0.991** | 0.9285 | **0.9587** | 5 [1, 4, 6, 10, 11] |
| | GA | 0.8928 | 0.8666 | 0.9286 | 0.8966 | 3 [3, 5, 6] |
| | GWO | 0.9286 | 0.875 | **0.9912** | 0.9295 | 3 [2, 4, 9] |
| | HHO | 0.8929 | 0.8235 | 0.9911 | 0.8995 | 6 [2, 3, 7, 9, 11] |
| | WOA | 0.8571 | 0.7777 | 0.9910 | 0.8714 | 5 [4, 6, 8, 10, 11] |
| SVM | PSO | 0.8215 | 0.7647 | 0.9286 | 0.8387 | 2 [2, 8] |
| | SSA | **0.8928** | 0.8235 | **0.9925** | **0.9001** | 3 [4, 6, 11] |
| | GA | 0.8215 | 0.8462 | 0.7857 | 0.8148 | 5 [ 4, 6, 7, 9, 11] |
| | GWO | 0.8571 | **0.9166** | 0.7857 | 0.8461 | 1 [4] |
| | HHO | 0.8219 | 0.7647 | 0.9285 | 0.8387 | 1 [3] |
| | WOA | 0.8215 | 0.80 | 0.8571 | 0.8276 | 2 [6, 8] |
| KNN | PSO | 0.8254 | 0.7368 | **0.9947** | 0.8465 | 3 [1, 4, 6] |
| | SSA | 0.8576 | 0.7777 | 0.9911 | 0.8715 | 4 [2, 3, 8, 10] |
| | GA | 0.9286 | 0.875 | 0.9915 | 0.9296 | 3 [2, 7, 10] |
| | GWO | 0.8936 | 0.8666 | 0.9285 | 0.8965 | 2 [1, 10] |
| | HHO | **0.9658** | **0.9941** | 0.9572 | **0.9753** | 4 [1, 3, 7, 9] |
| | WOA | 0.8926 | 0.8235 | 0.9912 | 0.8996 | 9 [1, 2, 4, 5, 7, 8, 9, 10, 11] |
| DT | PSO | 0.9286 | 0.875 | **0.9925** | 0.9301 | 6 [5, 6, 7, 8, 9, 10] |
| | SSA | **0.9652** | **0.9898** | 0.9286 | **0.9582** | 7 [1, 3, 4, 5, 8, 10, 11] |
| | GA | 0.8925 | 0.9231 | 0.8572 | 0.8889 | 4 [1, 3, 5, 7] |
| | GWO | 0.8965 | 0.8235 | 0.9924 | 0.9001 | 4 [1, 6, 7, 9] |
| | HHO | 0.8573 | 0.9166 | 0.7857 | 0.8461 | 6 [4, 5, 6, 7, 10, 12] |
| | WOA | 0.8254 | 0.80 | 0.8571 | 0.8276 | 5 [0, 2, 5, 6, 7] |
| AB | PSO | 0.8589 | 0.8125 | 0.9286 | 0.8666 | 6 [2, 3, 5, 6, 9, 11] |
| | SSA | 0.8962 | 0.8666 | 0.9285 | 0.8965 | 5 [3, 5, 6, 8, 9] |
| | GA | 0.9286 | 0.875 | 0.9942 | 0.9307 | 7 [2, 4, 5, 6, 8, 10, 12] |
| | GWO | **0.9656** | **0.9333** | **0.9957** | **0.9634** | 5 [1, 4, 8, 9, 11] |
| | HHO | 0.8225 | 0.80 | 0.8571 | 0.8275 | 2 [1, 5] |
| | WOA | 0.8572 | 0.7777 | 0.9854 | 0.8693 | 6 [3, 5, 8, 9, 10, 11] |
| NB | PSO | 0.8929 | **0.9948** | 0.7857 | 0.8779 | 3 [1, 10, 11] |
| | SSA | 0.8254 | 0.9090 | 0.7142 | 0.7999 | 2 [1, 5] |
| | GA | 0.8216 | 0.8461 | 0.7857 | 0.8148 | 1 [11] |
| | GWO | **0.8939** | 0.9231 | **0.8571** | **0.8888** | 2 [2, 5] |
| | HHO | 0.8214 | 0.9789 | 0.6428 | 0.7761 | 1 [11] |
| | WOA | 0.8575 | 0.9788 | 0.7142 | 0.8258 | 1 [11] |
| LR | PSO | 0.8926 | 0.8235 | **0.9974** | 0.9021 | 7 [0, 3, 5, 6, 8, 11, 12] |
| | SSA | 0.8936 | 0.8666 | 0.9286 | 0.8965 | 5 [2, 3, 6, 8, 9] |
| | GA | **0.9286** | **0.9915** | 0.8571 | **0.9194** | 4 [0, 1, 5, 11] |
| | GWO | 0.8956 | 0.9231 | 0.8571 | 0.8888 | 2 [2, 5] |
| | HHO | 0.8965 | 0.9231 | 0.7857 | 0.8752 | 10 [0, 1, 2, 3, 4, 5, 7, 9, 10, 12] |
| | WOA | 0.8573 | 0.8125 | 0.9286 | 0.8666 | 3 [1, 5, 11] |
| MLP | PSO | 0.76 | 0.8181 | 0.6428 | 0.7199 | 7 [1, 3, 4, 6, 8, 11, 12] |
| | SSA | 0.76 | 0.7059 | 0.8571 | 0.7742 | 5 [3, 5, 7, 8, 9] |
| | GA | 0.8571 | 0.8125 | 0.9285 | 0.8666 | 6 [2, 3, 4, 5, 8, 9] |
| | GWO | 0.8215 | **0.9090** | 0.7143 | 0.7999 | 4 [2, 5, 8, 9] |
| | HHO | 0.7858 | 0.75 | 0.8571 | 0.7998 | 7 [1, 2, 3, 4, 5, 11, 12] |
| | WOA | **0.8927** | 0.8666 | **0.9286** | **0.8965** | 8 [0, 1, 3, 5, 6, 8, 10, 12] |

Table A3. Performance results of metrics-based Moodle dataset

| Machine Learning Algorithms | Feature Selection Techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.9452 | 0.9090 | 0.7143 | 0.80 | 5 [0, 2, 3, 5, 12] |
| | SSA | **0.9573** | **0.9406** | 0.9761 | **0.9581** | 5 [1, 5, 6, 9, 11] |
| | GA | 0.9407 | 0.9028 | 0.9863 | 0.9427 | 6 [2, 3, 5, 6, 10, 11] |
| | GWO | 0.9366 | 0.8899 | **0.9966** | 0.9402 | 5 [2, 4, 5, 8, 10] |
| | HHO | 0.9435 | 0.9085 | 0.9863 | 0.9458 | 7 [0, 1, 3, 5, 6, 8, 11] |
| | WOA | 0.9212 | 0.8639 | 0.9847 | 0.9203 | 8 [0, 1, 2, 4, 5, 8, 10, 12] |
| SVM | PSO | 0.8239 | 0.8593 | 0.7739 | 0.8144 | 7 [1, 2, 3, 5, 8, 10, 11] |
| | SSA | 0.8596 | **0.8977** | 0.8116 | 0.8525 | 6 [2, 4, 8, 10, 11, 12] |
| | GA | 0.8496 | 0.8592 | 0.8356 | 0.8472 | 8 [1, 2, 5, 7, 8, 10, 11, 12] |
| | GWO | **0.8659** | 0.8844 | 0.8391 | **0.8611** | 7 [1, 3, 5, 8, 10, 11, 12] |
| | HHO | 0.8425 | 0.8401 | **0.8459** | 0.8429 | 3 [2, 3, 8] |
| | WOA | 0.8335 | 0.8679 | 0.7876 | 0.8258 | 8 [0, 2, 3, 6, 8, 9, 11, 12] |
| KNN | PSO | **0.9616** | **0.9269** | **0.9968** | **0.9605** | 7 [1, 2, 3, 4, 8, 10, 12] |
| | SSA | 0.9539 | 0.9179 | 0.9965 | 0.9556 | 8 [0, 1, 3, 5, 8, 9, 11, 12] |
| | GA | 0.9558 | 0.9235 | 0.9931 | 0.9571 | 6 [0, 4, 8, 9, 11, 12] |
| | GWO | 0.9572 | 0.9265 | 0.9848 | 0.9547 | 7 [0, 2, 3, 7, 9, 11, 12] |
| | HHO | 0.9588 | 0.9241 | 0.9878 | 0.9549 | 8 [0, 3, 4, 5, 7, 8, 10, 11] |
| | WOA | 0.9578 | 0.9238 | 0.9966 | 0.9588 | 7 [0, 3, 4, 5, 7, 8, 11] |
| DT | PSO | 0.9865 | 0.9765 | 0.9965 | 0.9864 | 6 [0, 3, 5, 6, 10, 11] |
| | SSA | **0.9968** | **0.9962** | 0.9963 | **0.9962** | 5 [0, 4, 5, 6, 10] |
| | GA | 0.9949 | 0.9932 | **0.9966** | 0.9948 | 5 [0, 4, 5, 7, 10] |
| | GWO | 0.9869 | 0.9797 | 0.9932 | 0.9864 | 5 [2, 3, 6, 8, 11] |
| | HHO | 0.9897 | 0.9831 | 0.9965 | 0.9897 | 7 [1, 2, 3, 4, 5, 6, 10] |
| | WOA | 0.9885 | 0.9863 | 0.9897 | 0.9881 | 7 [0, 3, 5, 8, 9, 10, 12] |
| AB | PSO | 0.9708 | 0.9661 | 0.9762 | 0.9711 | 7 [2, 5, 6, 8, 10, 11, 12] |
| | SSA | 0.9674 | **0.9823** | 0.9521 | 0.9669 | 4 [5, 8, 9, 12] |
| | GA | 0.9556 | 0.9291 | **0.9863** | 0.9568 | 5 [1, 3, 5, 9, 11] |
| | GWO | **0.9745** | 0.9792 | 0.9692 | **0.9742** | 7 [1, 2, 4, 5, 6, 10, 11] |
| | HHO | 0.9578 | 0.9435 | 0.9726 | 0.9578 | 13 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] |
| | WOA | 0.9591 | 0.9437 | 0.9761 | 0.9595 | 7 [1, 2, 5, 6, 8, 9, 11] |
| NB | PSO | 0.7398 | 0.8017 | 0.6369 | 0.7099 | 2 [5, 12] |
| | SSA | 0.7448 | 0.8235 | 0.6233 | 0.7095 | 4 [1, 4, 5, 12] |
| | GA | **0.8031** | 0.8525 | **0.7329** | **0.7882** | 3 [5, 10, 12] |
| | GWO | 0.7456 | 0.7582 | 0.7089 | 0.7327 | 3 [5, 11, 12] |
| | HHO | 0.7486 | 0.8169 | 0.6404 | 0.7179 | 2 [5, 12] |
| | WOA | 0.7696 | **0.8969** | 0.5959 | 0.7161 | 1 [5] |
| LR | PSO | 0.8356 | 0.7734 | **0.9589** | 0.8563 | 9 [1, 3, 5, 6, 7, 8, 9, 11, 12] |
| | SSA | 0.8589 | 0.8046 | 0.9452 | 0.8691 | 9 [0, 1, 3, 5, 6, 7, 8, 9, 11] |
| | GA | 0.8169 | 0.7621 | 0.9212 | 0.8341 | 7 [4, 6, 7, 8, 9, 10, 12] |
| | GWO | 0.8659 | **0.8292** | 0.9143 | 0.8697 | 7 [1, 4, 5, 6, 7, 9, 10] |
| | HHO | 0.8069 | 0.7687 | 0.8767 | 0.8192 | 9 [1, 2, 3, 4, 6, 7, 9, 11, 12] |
| | WOA | **0.8756** | 0.8073 | 0.976 | **0.8837** | 11 [0, 1, 2, 3, 5, 6, 7, 8, 9, 11, 12] |
| MLP | PSO | 0.7828 | 0.8113 | 0.7363 | 0.7719 | 7 [1, 3, 4, 5, 8, 9, 10] |
| | SSA | 0.9297 | 0.9114 | 0.9521 | 0.9313 | 10 [1, 2, 4, 3, 5, 8, 9, 10, 11, 12] |
| | GA | **0.9439** | **0.9218** | 0.9692 | **0.9449** | 9 [0, 2, 3, 4, 5, 8, 10, 11, 12] |
| | GWO | 0.8956 | 0.8434 | 0.9589 | 0.8974 | 5 [3, 5, 8, 9, 11] |
| | HHO | 0.9023 | 0.8983 | 0.9075 | 0.9029 | 8 [0, 3, 5, 8, 9, 10, 11, 12] |
| | WOA | 0.9356 | 0.8944 | **0.9863** | 0.9381 | 5 [3, 4, 5, 8, 9] |

Table A4. Performance results of metrics-based PHPMyAdmin dataset

| Machine Learning Algorithms | Feature Selection Techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.8965 | 0.8333 | 0.8856 | 0.8586 | 3 [3, 9, 11] |
| | SSA | 0.9327 | 0.9643 | 0.90 | 0.9310 | 6 [1, 3, 4, 6, 8, 9] |
| | GA | 0.8643 | 0.8621 | 0.8625 | 0.8623 | 4 [3, 7, 10, 11] |
| | GWO | 0.8994 | 0.9615 | 0.8333 | 0.8929 | 3 [2, 10, 12] |
| | HHO | **0.9661** | **0.9655** | **0.9489** | **0.9658** | 5 [1, 5, 8, 9, 12] |
| | WOA | 0.8304 | 0.8333 | 0.8453 | 0.8392 | 6 [2, 4, 5, 8, 10, 12] |
| SVM | PSO | 0.7802 | 0.8148 | 0.7333 | 0.7719 | 3 [0, 3, 10] |
| | SSA | 0.7799 | 0.7666 | 0.7931 | 0.7797 | 3 [3, 5, 10] |
| | GA | **0.8306** | **0.8276** | **0.8279** | **0.8277** | 4 [1, 2, 8, 10] |
| | GWO | 0.7629 | 0.7666 | 0.7698 | 0.7688 | 5 [1, 4, 5, 7, 10] |
| | HHO | 0.7965 | 0.80 | 0.8154 | 0.8076 | 3 [3, 8, 9] |
| | WOA | 0.7305 | 0.7916 | 0.6333 | 0.7037 | 1 [9] |
| KNN | PSO | 0.85 | 0.7631 | 0.8496 | 0.8041 | 6 [2, 3, 4, 5, 8, 12] |
| | SSA | 0.8666 | 0.7838 | 0.8989 | 0.8374 | 6 [3, 5, 7, 8, 11, 12] |
| | GA | 0.8655 | 0.8181 | 0.9310 | 0.8709 | 4 [3, 7, 10, 12] |
| | GWO | **0.9161** | **0.875** | **0.9655** | **0.9181** | 3 [2, 11, 12] |
| | HHO | 0.8811 | 0.8709 | 0.90 | 0.8852 | 4 [2, 4, 5, 10] |
| | WOA | 0.8626 | 0.8055 | 0.8655 | 0.8344 | 8 [2, 4, 5, 6, 7, 9, 10, 12] |
| DT | PSO | 0.9488 | 0.9643 | 0.9311 | 0.9473 | 4 [1, 3, 5, 10] |
| | SSA | 0.9327 | 0.9032 | 0.9655 | 0.9333 | 8 [1, 3, 5, 7, 9, 10, 11, 12] |
| | GA | 0.9827 | **0.9777** | 0.9655 | 0.9715 | 6 [3, 4, 6, 8, 10, 12] |
| | GWO | **0.9833** | 0.9666 | **0.9897** | **0.9831** | 4 [1, 3, 6, 9] |
| | HHO | 0.9494 | 0.9655 | 0.9333 | 0.9491 | 9 [2, 3, 4, 5, 6, 8, 9, 10, 12] |
| | WOA | 0.9488 | 0.9643 | 0.9310 | 0.9474 | 3 [1, 3, 5] |
| AB | PSO | 0.95 | 0.9063 | **0.9782** | 0.9408 | 8 [1, 2, 5, 6, 8, 9, 10, 12] |
| | SSA | **0.9609** | **0.9666** | **0.9665** | 0.9665 | 6 [1, 4, 5, 7, 9, 11] |
| | GA | 0.9494 | 0.9333 | 0.9655 | 0.9492 | 6 [0, 2, 4, 5, 9, 11] |
| | GWO | 0.9488 | 0.9355 | 0.9666 | 0.9507 | 5 [5, 6, 8, 10, 11] |
| | HHO | 0.9321 | 0.9333 | 0.9215 | 0.9273 | 10 [1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12] |
| | WOA | 0.9327 | 0.9032 | 0.9655 | 0.9333 | 7 [0, 5, 6, 7, 9, 10, 11] |
| NB | PSO | 0.7437 | 0.7027 | **0.8666** | **0.7762** | 6 [0, 1, 5, 9, 10, 11] |
| | SSA | **0.7638** | 0.7272 | 0.8276 | 0.7742 | 5 [1, 4, 5, 10, 11] |
| | GA | 0.7626 | 0.7666 | 0.7661 | 0.7665 | 6 [0, 1, 5, 6, 9, 11] |
| | GWO | 0.7438 | 0.8181 | 0.6208 | 0.7059 | 3 [5, 10, 11] |
| | HHO | 0.7311 | **0.8184** | 0.60 | 0.6923 | 5 [0, 1, 5, 8, 11] |
| | WOA | 0.7454 | 0.75 | 0.7241 | 0.7368 | 2 [0, 8] |
| LR | PSO | 0.8638 | 0.8437 | **0.90** | **0.8709** | 6 [1, 2, 3, 5, 7, 9] |
| | SSA | **0.8649** | 0.8387 | 0.8966 | 0.8666 | 7 [0, 2, 3, 9, 10, 11, 12] |
| | GA | 0.8465 | **0.8846** | 0.7931 | 0.8363 | 7 [2, 4, 6, 8, 9, 10, 12] |
| | GWO | 0.8298 | 0.8519 | 0.7933 | 0.8214 | 5 [1, 2, 7, 8, 9] |
| | HHO | 0.7971 | 0.8214 | 0.7666 | 0.7931 | 8 [1, 2, 4, 5, 7, 8, 9, 12] |
| | WOA | 0.7609 | 0.8261 | 0.6552 | 0.7308 | 6 [1, 2, 4, 6, 9, 10] |
| MLP | PSO | 0.7477 | 0.8261 | 0.6333 | 0.7169 | 6 [0, 2, 5, 7, 9, 12] |
| | SSA | 0.7465 | 0.7187 | 0.7931 | 0.7541 | 9 [0, 2, 3, 5, 8, 9, 10, 12] |
| | GA | 0.8393 | 0.88 | 0.7586 | 0.8148 | 6 [2, 4, 5, 8, 10, 12] |
| | GWO | **0.9149** | **0.9285** | 0.8965 | **0.9122** | 5 [1, 5, 8, 10, 12] |
| | HHO | 0.8649 | 0.8387 | 0.8966 | 0.8666 | 10 [1, 2, 4, 5, 7, 8, 9, 10, 11, 12] |
| | WOA | 0.8293 | 0.7941 | **0.90** | 0.8437 | 9 [0, 2, 3, 5, 7, 8, 10, 11, 12] |

Table A5: Performance results of metrics-based JavaScript dataset

| Machine learning algorithms | Feature selection techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.9544 | 0.96 | 0.9583 | 0.9592 | 18 [1, 3, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16, 17, 20, 22, 23, 27, 34] |
| | SSA | 0.9653 | 0.9732 | 0.9567 | 0.9649 | 12 [1, 5, 6, 7, 9, 15, 20, 22, 24, 25, 27, 30] |
| | GA | **0.9705** | **0.9798** | 0.9605 | **0.9701** | 13 [1, 3, 4, 5, 6, 11, 13, 15, 16, 19, 23, 28, 30] |
| | GWO | 0.9699 | 0.9708 | **0.9689** | 0.9698 | 10 [1, 2, 4, 7, 8, 13, 25, 29, 30, 34] |
| | HHO | 0.9578 | 0.9673 | 0.9473 | 0.9572 | 23 [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 14, 15, 18, 20, 21, 22, 24, 26, 29, 30, 31, 33] |
| | WOA | 0.9563 | 0.9691 | 0.9426 | 0.9557 | 13 [1, 2, 3, 5, 7, 8, 9, 16, 17, 21, 22, 24] |
| SVM | PSO | 0.7423 | 0.7826 | 0.6707 | 0.7223 | 7 [0, 1, 2, 4, 8, 20, 26] |
| | SSA | 0.7456 | 0.8707 | 0.5766 | 0.6938 | 18 [1, 4, 5, 6, 9, 12, 14, 15, 16, 17, 19, 20, 22, 25, 27, 30, 32, 34] |
| | GA | **0.8035** | 0.8554 | 0.7291 | **0.7872** | 13 [2, 4, 6, 7, 8, 10, 15, 16, 20, 21, 26, 27, 32] |
| | GWO | 0.7569 | 0.7845 | 0.7055 | 0.7429 | 12 [2, 7, 8, 9, 10, 11, 14, 15, 16, 20, 26, 33] |
| | HHO | 0.5652 | 0.5374 | **0.9322** | 0.6818 | 4 [7, 14, 18, 20] |
| | WOA | 0.7516 | **0.8721** | 0.5898 | 0.7037 | 12 [0, 2, 4, 7, 9, 11, 13, 17, 24, 26, 30, 33] |
| KNN | PSO | 0.9318 | **0.9105** | 0.9577 | 0.9335 | 15 [1, 2, 6, 9, 12, 14, 15, 16, 20, 21, 22, 24, 31, 33, 34] |
| | SSA | **0.9322** | 0.9041 | **0.9671** | **0.9345** | 16 [4, 5, 7, 11, 12, 13, 14, 15, 19, 20, 21, 23, 24, 25, 26, 31] |
| | GA | 0.9203 | 0.9011 | 0.9435 | 0.9218 | 15 [2, 4, 5, 6, 7, 10, 11, 13, 15, 18, 19, 25, 27, 31, 34] |
| | GWO | 0.9312 | 0.9032 | 0.9661 | 0.9336 | 13 [1, 7, 11, 14, 15, 17, 20, 24, 25, 27, 31, 33, 34] |
| | HHO | 0.9186 | 0.8952 | 0.9483 | 0.9209 | 16 [1, 2, 5, 7, 10, 11, 13, 15, 16, 18, 21, 28, 29, 31, 32, 34] |
| | WOA | 0.9295 | 0.9043 | 0.9604 | 0.9316 | 23 [2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 23, 25, 27, 30, 31, 32] |
| DT | PSO | **0.9605** | 0.9579 | **0.9633** | **0.9606** | 16 [3, 5, 6, 7, 8, 11, 12, 15, 16, 17, 22, 28, 29, 31, 32, 34] |
| | SSA | 0.9532 | 0.9538 | 0.9529 | 0.9534 | 11 [4, 7, 9, 10, 11, 14, 15, 20, 22, 23, 33] |
| | GA | 0.9592 | **0.9595** | 0.9586 | 0.9591 | 13 [3, 6, 8, 9, 13, 14, 16, 18, 20, 23, 25, 27, 32] |
| | GWO | 0.9551 | 0.9523 | 0.9586 | 0.9554 | 11 [9, 10, 11, 15, 17, 20, 22, 25, 28, 29, 32] |
| | HHO | 0.9584 | 0.9479 | 0.9595 | 0.9537 | 13 [0, 8, 9, 11, 13, 16, 17, 18, 19, 22,,26, 31, 34] |
| | WOA | 0.9567 | 0.9525 | 0.9614 | 0.9569 | 11 [6, 7, 13, 14, 15, 19, 21, 23, 24, 25, 30] |
| AB | PSO | **0.8923** | **0.9273** | 0.8513 | **0.8877** | 20 [0, 2, 5, 8, 9, 11, 12, 13, 15, 17, 18, 19, 21, 23, 24, 25, 27, 30, 32, 33] |
| | SSA | 0.8749 | 0.9037 | 0.8392 | 0.8702 | 21 [1, 3, 4, 5, 6, 8, 10, 12, 15, 17, 20, 21, 23, 24, 25, 26, 28, 30, 31, 32, 34] |
| | GA | 0.8797 | 0.9147 | 0.8373 | 0.8743 | 12 [4, 7, 8, 9, 12, 15, 17, 18, 20, 21, 25, 33] |
| | GWO | 0.8892 | 0.9183 | **0.8561** | 0.8861 | 8 [8, 12, 13, 15, 17, 18, 21, 22] |
| | HHO | 0.8858 | 0.9108 | 0.8551 | 0.8821 | 19 [1, 3, 4, 5, 6, 7, 9, 10, 12, 13, 17, 20, 22, 25, 28, 30, 31, 33, 34] |
| | WOA | 0.8721 | 0.9211 | 0.8137 | 0.8641 | 13 [3, 4, 6, 10, 11, 13, 16, 17, 19, 21, 26, 27, 31] |
| NB | PSO | **0.6929** | **0.6414** | **0.8749** | **0.7401** | 13 [1, 2, 3, 4, 8, 14, 15, 18, 21, 24, 25, 30, 34] |

Table A5 continued

| Machine learning algorithms | Feature selection techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| NB | SSA | 0.6036 | 0.5802 | 0.7488 | 0.6538 | 22 [1, 2, 3, 4, 6, 8, 9, 10, 11, 13, 15, 16, 17, 19, 21, 23, 26, 27, 29, 31, 33, 34] |
| | GA | 0.6553 | 0.6071 | 0.7796 | 0.6826 | 8 [8, 9, 14, 16, 21, 27, 29, 30] |
| | GWO | 0.6396 | 0.5975 | 0.8561 | 0.7038 | 8 [5, 10, 12, 14, 15, 17, 26, 33] |
| | HHO | 0.6271 | 0.5978 | 0.7761 | 0.6754 | 17 [3, 4, 5, 6, 7, 8, 9, 12, 16, 17, 19, 21, 23, 26, 29, 31, 34] |
| | WOA | 0.5654 | 0.5363 | 0.5671 | 0.5513 | 6 [2, 10, 17, 25, 31, 33] |
| LR | PSO | 0.7226 | 0.7095 | 0.7535 | 0.7308 | 13 [4, 6, 8, 9, 11, 12, 14, 15, 18, 20, 23, 24, 25, 30] |
| | SSA | 0.6965 | 0.6618 | 0.8043 | 0.7261 | 15 [5, 7, 8, 10, 11, 13, 19, 20, 22, 23, 24, 25, 28, 32, 33] |
| | GA | **0.7362** | **0.7307** | 0.7478 | 0.7392 | 15 [8, 12, 13, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 31, 33] |
| | GWO | 0.7238 | 0.6904 | **0.8118** | **0.7462** | 16 [0, 1, 3, 6, 11, 12, 14, 16, 19, 24, 25, 28, 29, 30, 31, 33] |
| | HHO | 0.6942 | 0.6648 | 0.7855 | 0.7203 | 24 [1, 2, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17, 18, 19, 20, 22, 23, 24, 25, 26, 29, 31, 32, 33] |
| | WOA | 0.7063 | 0.6931 | 0.7394 | 0.7155 | 27 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 18, 19, 20, 23, 24, 25, 26, 27, 28, 29, 30] |
| MLP | PSO | 0.6603 | 0.7143 | 0.5551 | 0.6247 | 22 [2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 22, 24, 26, 28, 30, 32, 33, 34] |
| | SSA | 0.8573 | **0.8916** | 0.8127 | 0.8504 | 18 [1, 2, 3, 6, 7, 11, 12, 13, 15, 16, 22, 23, 24, 26, 30, 32, 33, 34] |
| | GA | 0.8546 | 0.8632 | 0.8429 | **0.8526** | 21 [0, 1, 2, 3, 4, 5, 6, 7, 11, 13, 15, 16, 17, 19, 20, 21, 23, 24, 27, 33] |
| | GWO | 0.8345 | 0.8099 | 0.8739 | 0.8407 | 16 [2, 3, 4, 8, 9, 16, 18, 21, 22, 23, 26, 27, 30, 32, 33, 34] |
| | HHO | **0.8745** | 0.8783 | 0.8692 | 0.8737 | 19 [0, 2, 45, 6, 8, 9, 10, 11, 12, 13, 14, 16, 21, 22, 25, 26, 27, 33] |
| | WOA | 0.8467 | 0.8275 | **0.8758** | 0.8509 | 23 [0, 1, 2, 3, 4, 6, 7, 8, 9, 13, 14, 15, 16, 19, 20, 21, 23, 24, 25, 27, 32, 33, 34] |

Table A6. Performance results of text-features-based Drupal dataset

| Machine learning algorithms | Feature selection sechniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.8929 | 0.8235 | 0.8878 | 0.8544 | 1740 |
| | SSA | 0.9289 | 0.875 | 0.9156 | 0.8948 | 1814 |
| | GA | 0.9658 | 0.9356 | 0.9286 | 0.9321 | 1614 |
| | GWO | **0.9666** | **0.9587** | **0.9289** | **0.9435** | 442 |
| | HHO | 0.8928 | 0.8666 | 0.9286 | 0.8965 | 956 |
| | WOA 0 | 0.9289 | 0.9283 | 0.9254 | 0.9268 | 177 |
| SVM | PSO | 0.8929 | 0.9231 | 0.8571 | 0.8888 | 1800 |
| | SSA | 0.8578 | 0.8125 | **0.9286** | 0.8666 | 1862 |
| | GA | **0.9289** | **0.9145** | 0.8571 | **0.8848** | 1519 |
| | GWO | 0.8573 | 0.9166 | 0.7857 | 0.8461 | 513 |
| | HHO | 0.8571 | 0.7898 | 0.7143 | 0.7501 | 158 |
| | WOA | 0.8215 | 0.8462 | 0.7857 | 0.8148 | 362 |
| KNN | PSO | 0.8929 | 0.8666 | 0.9286 | 0.8965 | 1739 |
| | SSA | 0.9641 | 0.9333 | 0.9789 | 0.9555 | 1855 |
| | GA | 0.9642 | 0.9845 | 0.9285 | 0.9556 | 1576 |
| | GWO | **0.9929** | **0.9912** | **0.9899** | **0.9903** | 597 |
| | HHO | 0.8954 | 0.8235 | 0.9087 | 0.8641 | 804 |
| | WOA | 0.9643 | 0.9356 | 0.9286 | 0.9321 | 122 |
| DT | PSO | 0.8216 | 0.7647 | 0.9286 | 0.8387 | 1808 |
| | SSA | 0.8928 | 0.8235 | 0.8812 | 0.8513 | 1876 |
| | GA | **0.9648** | **0.9333** | **0.9485** | **0.9608** | 1736 |
| | GWO | 0.9285 | 0.875 | 0.9148 | 0.8945 | 528 |
| | HHO | 0.8573 | 0.7777 | 0.8821 | 0.8266 | 1836 |
| | WOA | 0.8929 | 0.9974 | 0.7857 | 0.8799 | 1834 |
| AB | PSO | 0.8929 | 0.8666 | 0.9113 | 0.8883 | 1831 |
| | SSA | 0.9286 | 0.875 | 0.9142 | 0.8942 | 1825 |
| | GA | **0.9648** | **0.9745** | **0.9486** | **0.9614** | 1614 |
| | GWO | 0.9285 | 0.875 | 0.9227 | 0.8982 | 1181 |
| | HHO | 0.9642 | 0.9333 | 0.9318 | 0.9325 | 1509 |
| | WOA | 0.8931 | 0.8235 | 0.8988 | 0.8595 | 1921 |
| NB | PSO | **0.9289** | 0.8847 | 0.8572 | 0.8707 | 1738 |
| | SSA | 0.8572 | 0.8452 | 0.7143 | 0.7743 | 1899 |
| | GA | 0.8929 | 0.8235 | **0.8788** | 0.8506 | 1532 |
| | GWO | 0.8927 | 0.8154 | 0.7857 | 0.8003 | 302 |
| | HHO | 0.8214 | 0.9090 | 0.7143 | 0.7999 | 1313 |
| | WOA | 0.8927 | **0.9231** | 0.8571 | **0.8888** | 1499 |
| LR | PSO | 0.9642 | 0.9333 | 0.9415 | 0.9374 | 1805 |
| | SSA | 0.9288 | 0.875 | 0.9012 | 0.8879 | 1870 |
| | GA | 0.9542 | 0.9233 | 0.9892 | 0.9552 | 1584 |
| | GWO | **0.9982** | **0.9911** | **0.9988** | **0.9949** | 365 |
| | HHO | 0.9641 | 0.9312 | 0.9325 | 0.9318 | 2007 |
| | WOA | 0.9682 | 0.9433 | 0.9512 | 0.9472 | 360 |
| MLP | PSO | 0.8929 | 0.8666 | 0.9286 | 0.8966 | 1798 |
| | SSA | 0.8234 | 0.7647 | 0.9287 | 0.8387 | 1915 |
| | GA | 0.9613 | 0.9224 | 0.9825 | 0.9515 | 1514 |
| | GWO | **0.9986** | **0.9901** | **0.9928** | **0.9914** | 642 |
| | HHO | 0.9642 | 0.9333 | 0.9242 | 0.9246 | 815 |
| | WOA | 0.9788 | 0.9333 | 0.9415 | 0.9378 | 1813 |

Table A7. Performance results of text-features-based Moodle dataset

| Machine learning algorithms | Feature selection techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.9263 | 0.9029 | 0.9554 | 0.9284 | 7938 |
| | SSA | 0.9434 | 0.9164 | 0.9761 | 0.9453 | 8097 |
| | GA | 0.9383 | 0.9324 | 0.9452 | 0.9387 | 7836 |
| | GWO | 0.9315 | 0.8937 | **0.9794** | 0.9346 | 3508 |
| | HHO | **0.9469** | **0.9364** | 0.9589 | **0.9475** | 4898 |
| | WOA | 0.9263 | 0.9055 | 0.9521 | 0.9282 | 4468 |
| SVM | PSO | 0.7842 | 0.9415 | 0.6062 | 0.7375 | 7843 |
| | SSA | **0.8031** | 0.9784 | **0.6198** | **0.7589** | 8048 |
| | GA | 0.7774 | 0.9133 | 0.6131 | 0.7336 | 8148 |
| | GWO | 0.7723 | 0.9035 | 0.6096 | 0.7281 | 7081 |
| | HHO | 0.7739 | 0.9348 | 0.5891 | 0.7227 | 8110 |
| | WOA | 0.7825 | **0.9941** | 0.5684 | 0.7233 | 5425 |
| KNN | PSO | **0.9421** | 0.8997 | **0.9829** | **0.9394** | 8049 |
| | SSA | 0.9001 | 0.8498 | 0.9692 | 0.9056 | 8146 |
| | GA | 0.9221 | 0.8742 | 0.9761 | 0.9223 | 8165 |
| | GWO | 0.9224 | **0.9126** | 0.9657 | 0.9384 | 5653 |
| | HHO | 0.9386 | 0.8816 | 0.9692 | 0.9233 | 8273 |
| | WOA | 0.9365 | 0.8974 | 0.9589 | 0.9272 | 9683 |
| DT | PSO | 0.9232 | 0.8782 | **0.9623** | 0.9183 | 8032 |
| | SSA | 0.9359 | 0.8984 | 0.9384 | 0.9179 | 8182 |
| | GA | 0.9212 | 0.8907 | 0.9486 | 0.9187 | 7777 |
| | GWO | **0.9561** | **0.9302** | 0.9589 | **0.9443** | 4126 |
| | HHO | 0.9242 | 0.8846 | 0.9452 | 0.9139 | 7445 |
| | WOA | 0.9221 | 0.9085 | 0.9178 | 0.9131 | 13546 |
| AB | PSO | 0.8921 | 0.8225 | 0.9758 | 0.9026 | 8105 |
| | SSA | 0.9195 | 0.8769 | 0.9761 | 0.9238 | 8139 |
| | GA | 0.9161 | 0.8627 | **0.9897** | 0.9218 | 7558 |
| | GWO | **0.9315** | **0.8937** | 0.9795 | **0.9346** | 5260 |
| | HHO | 0.9195 | 0.8816 | 0.9692 | 0.9233 | 8304 |
| | WOA | 0.9092 | 0.8699 | 0.9623 | 0.9138 | 8374 |
| NB | PSO | 0.8442 | 0.7638 | **0.9966** | 0.8648 | 8060 |
| | SSA | 0.8168 | 0.7318 | 0.9818 | 0.8385 | 8229 |
| | GA | **0.8631** | 0**0.7849** | 0.9778 | **0.8707** | 7596 |
| | GWO | 0.8356 | 0.7526 | 0.9818 | 0.8521 | 3992 |
| | HHO | 0.8185 | 0.7337 | 0.9878 | 0.8419 | 11754 |
| | WOA | 0.8322 | 0.7487 | 0.9888 | 0.8521 | 14599 |
| LR | PSO | 0.8938 | 0.8267 | 0.9966 | 0.9037 | 8056 |
| | SSA | 0.9024 | 0.8366 | **0.9978** | 0.9111 | 8259 |
| | GA | 0.9041 | 0.8391 | 0.9918 | 0.9091 | 7583 |
| | GWO | 0.8904 | 0.8202 | 0.9789 | 0.8925 | 3107 |
| | HHO | **0.9144** | **0.8601** | 0.9897 | **0.9204** | 7383 |
| | WOA | 0.8989 | 0.8319 | 0.9978 | 0.9082 | 7005 |
| MLP | PSO | 0.9023 | 0.8366 | 0.9789 | 0.9021 | 7847 |
| | SSA | 0.8938 | **0.8965** | 0.8904 | 0.8934 | 8247 |
| | GA | 0.8767 | 0.8437 | 0.9246 | 0.8822 | 8111 |
| | GWO | **0.9144** | 0.8757 | 0.9657 | **0.9186** | 7920 |
| | HHO | 0.8972 | 0.8295 | 0.9856 | 0.9008 | 6583 |
| | WOA | 0.8904 | 0.8221 | **0.9966** | 0.9009 | 9036 |

Table A8. Performance results of text-features-based PHPMyAdmin dataset

| Machine learning algorithms | Feature selection techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.9831 | **0.9677** | 0.9712 | 0.9694 | 2268 |
| | SSA | 0.9661 | 0.9375 | 0.9145 | 0.9258 | 2421 |
| | GA | 0.9789 | 0.9442 | 0.9645 | 0.9542 | 2118 |
| | GWO | **0.9833** | 0.9666 | 0.9918 | **0.9791** | 822 |
| | HHO | 0.9742 | 0.9555 | **0.9945** | 0.9656 | 1237 |
| | WOA | 0.8983 | 0.8965 | 0.8978 | 0.8972 | 1721 |
| SVM | PSO | 0.8644 | 0.92 | 0.7931 | 0.8518 | 2388 |
| | SSA | 0.8475 | 0.8166 | 0.7586 | 0.7865 | 2519 |
| | GA | 0.8478 | 0.8565 | 0.7333 | 0.7903 | 2041 |
| | GWO | **0.9152** | **0.9311** | **0.90** | **0.9152** | 560 |
| | HHO | 0.7626 | 0.90 | 0.60 | 0.72 | 2021 |
| | WOA | 0.8644 | 0.8541 | 0.7333 | 0.7891 | 462 |
| KNN | PSO | 0.9322 | 0.8824 | 0.8978 | 0.8903 | 2381 |
| | SSA | 0.9661 | 0.9375 | 0.9415 | 0.9395 | 2500 |
| | GA | **0.9833** | **0.9666** | **0.9778** | **0.9722** | 2390 |
| | GWO | 0.9662 | 0.9465 | 0.9558 | 0.9511 | 894 |
| | HHO | 0.8983 | 0.8333 | 0.8812 | 0.8566 | 1353 |
| | WOA | 0.9322 | 0.8787 | 0.9015 | 0.8899 | 2736 |
| DT | PSO | 0.9831 | 0.9345 | 0.9289 | 0.9317 | 2330 |
| | SSA | 0.9661 | 0.9476 | 0.9554 | 0.9516 | 2465 |
| | GA | 0.9492 | 0.9063 | 0.9331 | 0.9196 | 2349 |
| | GWO | **0.9859** | **0.9677** | **0.9789** | **0.9736** | 738 |
| | HHO | 0.8983 | 0.8286 | 0.8844 | 0.8555 | 1859 |
| | WOA | 0.8827 | 0.8235 | 0.9655 | 0.8888 | 2856 |
| AB | PSO | 0.9827 | 0.9544 | 0.9614 | 0.9578 | 2410 |
| | SSA | 0.9877 | 0.9456 | 0.9541 | 0.9498 | 2378 |
| | GA | 0.9661 | 0.9375 | 0.9542 | 0.9457 | 2197 |
| | GWO | **0.9879** | **0.9667** | **0.9789** | **0.9727** | 1540 |
| | HHO | 0.9152 | 0.9286 | 0.8965 | 0.9123 | 2195 |
| | WOA | 0.9616 | 0.9412 | 0.9433 | 0.9422 | 3200 |
| NB | PSO | 0.8305 | 0.75 | 0.8245 | 0.7855 | 2375 |
| | SSA | **0.8474** | **0.7631** | **0.8355** | **0.7976** | 2463 |
| | GA | 0.7966 | 0.7073 | 0.7889 | 0.7458 | 2080 |
| | GWO | 0.7627 | 0.6905 | 0.7088 | 0.6995 | 647 |
| | HHO | 0.7333 | 0.6444 | 0.7225 | 0.6813 | 2271 |
| | WOA | 0.8135 | 0.7317 | 0.8145 | 0.7708 | 3206 |
| LR | PSO | **0.9666** | **0.9917** | **0.9333** | **0.9616** | 2335 |
| | SSA | 0.9491 | 0.9121 | 0.8965 | 0.9042 | 2539 |
| | GA | 0.9322 | 0.9643 | 0.90 | 0.9311 | 2081 |
| | GWO | 0.9661 | 0.9356 | 0.9123 | 0.9238 | 624 |
| | HHO | 0.9155 | 0.9311 | 0.90 | 0.9153 | 2678 |
| | WOA | 0.9492 | 0.9655 | 0.9331 | 0.9492 | 1143 |
| MLP | PSO | 0.9316 | 0.9629 | 0.8965 | 0.9286 | 2360 |
| | SSA | 0.9655 | 0.9375 | 0.9412 | 0.9393 | 2478 |
| | GA | 0.9778 | 0.9485 | 0.9389 | 0.9437 | 2203 |
| | GWO | **0.9878** | **0.9756** | 0.9614 | 0.9684 | 685 |
| | HHO | 0.9831 | 0.9666 | **0.9721** | **0.9693** | 2624 |
| | WOA | 0.9491 | 0.9333 | 0.9655 | 0.9492 | 2073 |

Table A9. Performance results of text-features-based JavaScript dataset

| Machine learning algorithms | Feature selection techniques | AUC | Precision | Recall | $F_1$-score | $N$_Features |
|---|---|---|---|---|---|---|
| RF | PSO | 0.9985 | 0.9981 | 0.9929 | 0.9985 | 3172 |
| | SSA | 0.9972 | 0.9984 | 0.9945 | 0.9972 | 3282 |
| | GA | 0.9995 | 0.9878 | 0.9981 | 0.9978 | 3159 |
| | GWO | 0.9981 | 0.9991 | 0.9972 | 0.9981 | 2727 |
| | HHO | 0.9946 | 0.9978 | 0.9758 | 0.9882 | 5729 |
| | WOA | **0.9995** | **0.9994** | **0.9991** | **0.9995** | 6552 |
| SVM | PSO | 0.9609 | 0.9321 | 0.9943 | 0.9622 | 3272 |
| | SSA | 0.9665 | 0.9404 | 0.9962 | 0.9675 | 3300 |
| | GA | 0.9665 | 0.9435 | 0.9925 | 0.9674 | 3267 |
| | GWO | 0.9586 | 0.9333 | 0.9878 | 0.9597 | 1462 |
| | HHO | **0.9788** | **0.9593** | **0.9991** | **0.9793** | 1625 |
| | WOA | 0.9519 | 0.9255 | 0.9831 | 0.9534 | 5687 |
| KNN | PSO | 0.9741 | 0.9623 | 0.9868 | 0.9744 | 3221 |
| | SSA | 0.9675 | 0.9437 | **0.9944** | 0.9684 | 3216 |
| | GA | 0.9788 | 0.9652 | 0.9934 | 0.9792 | 3304 |
| | GWO | 0.9755 | 0.9676 | 0.9839 | 0.9757 | 2829 |
| | HHO | 0.9892 | 0.9859 | 0.9925 | 0.9892 | 7920 |
| | WOA | **0.9896** | **0.9869** | 0.9928 | **0.9897** | 9766 |
| DT | PSO | 0.9915 | 0.9887 | 0.9789 | 0.9837 | 3099 |
| | SSA | 0.9978 | 0.9784 | 0.9578 | 0.9679 | 3176 |
| | GA | 0.9847 | 0.9812 | 0.9846 | 0.9828 | 2901 |
| | GWO | 0.9914 | 0.9963 | **0.9978** | **0.9972** | 1204 |
| | HHO | 0.9745 | 0.9625 | 0.9562 | 0.9593 | 3168 |
| | WOA | **0.9994** | **0.9988** | 0.9921 | 0.9954 | 3433 |
| AB | PSO | 0.9947 | 0.9978 | 0.9952 | **0.9964** | 3043 |
| | SSA | 0.9985 | 0.9956 | **0.9958** | 0.9916 | 3250 |
| | GA | 0.9942 | 0.9924 | 0.9845 | 0.9884 | 3133 |
| | GWO | **0.9995** | **0.9981** | 0.9854 | 0.9917 | 7719 |
| | HHO | 0.9932 | 0.9954 | 0.9876 | 0.9914 | 8188 |
| | WOA | 0.9914 | 0.9911 | 0.9863 | 0.9886 | 8197 |
| NB | PSO | 0.9985 | 0.9991 | 0.9981 | 0.9986 | 3100 |
| | SSA | 0.9957 | 0.9953 | 0.9963 | 0.9957 | 3208 |
| | GA | 0.9995 | 0.9991 | 0.9990 | **0.9996** | 2817 |
| | GWO | 0.9995 | **0.9994** | **0.9992** | 0.9993 | 6659 |
| | HHO | **0.9998** | 0.9945 | 0.9946 | 0.9942 | 4856 |
| | WOA | 0.9978 | 0.9947 | 0.9952 | 0.9949 | 4522 |
| LR | PSO | 0.9912 | 0.9445 | 0.9685 | 0.9563 | 3051 |
| | SSA | 0.9818 | 0.9525 | 0.9669 | 0.9596 | 3210 |
| | GA | 0.9771 | 0.9859 | 0.9554 | 0.9704 | 2770 |
| | GWO | **0.9912** | **0.9879** | 0.9715 | 0.9795 | 4643 |
| | HHO | 0.9698 | 0.9781 | 0.9772 | 0.9776 | 5757 |
| | WOA | 0.9745 | 0.9878 | **0.9884** | **0.9881** | 6319 |
| MLP | PSO | 0.9976 | 0.9972 | 0.9981 | 0.9974 | 3019 |
| | SSA | 0.9991 | 0.9992 | 0.9989 | 0.9989 | 4195 |
| | GA | 0.9945 | 0.9965 | 0.9978 | 0.9984 | 3212 |
| | GWO | **0.9995** | 0.9981 | 0.9925 | 0.9991 | 3094 |
| | HHO | 0.9964 | **0.9995** | 0.9987 | 0.9994 | 5880 |
| | WOA | 0.9915 | 0.9991 | **0.9996** | **0.9997** | 2289 |

## Authors and affiliations

Deepali Bassi
e-mail: deepalics.rsh@gndu.ac.in
ORCID: https://orcid.org/0000-0002-6744-1957
Department of Computer Science, Guru Nanak Dev University, India

Hardeep Singh
e-mail: hardeep.dcse@gndu.ac.in
Department of Computer Science, Guru Nanak Dev University, India