IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 8 September 2024, accepted 20 September 2024, date of publication 24 September 2024, date of current version 8 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3467183

RESEARCH ARTICLE

Refined Software Defect Prediction Using Enhanced JAYA Optimization and Extreme Learning Machine

DEBASISH PRADHAN[1], DEBENDRA MUDULI[1], (Member, IEEE), ABU TAHA ZAMANI[2], (Member, IEEE), SYED IRFAN YAQOOB[3], SULTAN M. ALANAZI[2], RAKESH RANJAN KUMAR[1], (Member, IEEE), NIKHAT PARVEEN[4], AND MOHAMMAD SHAMEEM[5]

[1]Department of Computer Science and Engineering, C.V. Raman Global University, Bidya Nagar, Mahura, Janla, Bhubaneswar, Odisha 752054, India
[2]Department of Computer Science, Northern Border University, Arar 73211, Saudi Arabia
[3]Department of Computer Science and Engineering, AIT, Chandigarh University, Mohali, Punjab 140413, India
[4]Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation (Deemed to be University), Guntur, Andhra Pradesh 522302, India
[5]Interdisciplinary Research Center for Intelligent Secure Systems, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

Corresponding author: Debendra Muduli (muduli.debendra@gmail.com)

This work was supported by the Deanship of Scientific Research at Northern Border University, Arar, Saudi Arabia, under Project NBU-FFR-2024-1060-02.

ABSTRACT Ensuring the dependability of software before its public release is of utmost importance. Many software issues arise due to human errors made throughout the development process, highlighting the importance of addressing these errors early. It is crucial to incorporate testing resources at the beginning of development to minimize potential issues. Utilizing an approach that identifies modules susceptible to errors helps potential problems. With an understanding of the significance of precisely anticipating module failures, multiple automated solutions are already emerging. This work presents a refined software defect prediction model that utilizes a meta-heuristic optimization technique. The methodology integrates NASA's data collection procedure, which involves data cleansing, reducing the dimensionality of features, and predicting defects. Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are used to decrease the feature dimensionality, while an Extreme Learning Machine (ELM) is utilized for forecasting defects. The ELM parameters, such as weight and biases, are ideally chosen using the suggested improved JAYA optimization (IMJAYA) method. The model's validation involves assessing its accuracy, sensitivity, specificity, F1 score, and MCC metrics using a 10×5 cross-validation. The model is verified using NASA datasets that consist of several classes, such as CM1, KC2, KC3, MC1, PC1, and JM1. The PCA-LDA+IMJAYA-ELM model yields defect prediction accuracies of 95.73%, 98.08%, 94.87%, 96.23%, 97.10%, and 97.46% for the CM1, KC2, KC3, MC1, PC1, and JM1 datasets, respectively. The research outcomes show encouraging outcomes when using a meta-heuristic optimization technique with smaller feature sets for studies on predicting software defects.

INDEX TERMS Software defect prediction, PCA, LDA, JAYA optimization, ELM, NASA.

I. INTRODUCTION

Defect forecasting due to the constantly evolving nature of software engineering, challenges may emerge at any point during development, impacting the reliability and efficiency of the final product [1]. SDP employs predictive models and methods to anticipate and address these issues. Researchers use historical data and statistical analysis to forecast software defects by identifying trends and patterns indicating potential flaws in code segments. This proactive approach helps developers improve software quality, optimize resource usage, and anticipate challenges. Measures like code churn, developer

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Ni.

expertise, and code complexity are employed for prediction [2]. In the ever-changing field of software development, predicting and preventing errors is crucial for creating robust and dependable software, ultimately enhancing software reliability and customer satisfaction by reducing debugging time and optimizing development processes [3].

When it comes to discovering and repairing software defects, SDP using Machine Learning (ML) uses cutting-edge computational approaches. In the dynamic field of software development, vulnerabilities in code can cause systems to fail or become insecure [4]. This emerging area is meeting a need in the market by using ML algorithms to analyse massive datasets in search of patterns and possible software flaws; this is an area where traditional methods fall short [5]. Machine learning (ML) models can anticipate software problems by learning from past occurrences and analyzing code complexity, bug history data, and code modifications. This proactive method improves the efficiency and dependability of software systems while decreasing the costs associated with repairs. There is a growing need for dependable software, and ML in SDP can potentially revolutionize the development of first-rate solutions [6].

SDP using Machine Learning (ML) is critical for software quality assurance. SDP incorporating ML has emerged due to the increasing sophistication of software systems and the requirement for improved defect management throughout the development lifecycle. ML algorithms detect patterns in historical data to anticipate new code faults [7]. Researchers and practitioners are refining defect prediction models by employing deep learning. Integrating ML into software development processes is a proactive approach to designing more reliable and robust software systems [8].

Machine learning-driven SDP is pivotal in software engineering and quality assurance efforts, striving to identify and resolve defects promptly within the fast-paced software industry. ML empowers QA teams and developers to analyze vast datasets, identify trends, and forecast potential software issues. The integration of ML with SDP offers several advantages, enabling teams to identify and rectify development process issues before they escalate proactively [9]. This approach enhances software reliability and quality while reducing the time and costs of addressing errors later in the development cycle. SDP with ML also optimizes resource utilization by prioritizing testing efforts on critical code segments most likely to contain faults, thereby improving testing efficiency. ML-driven SDP contributes to system strengthening and stabilization by leveraging trends and historical errors to identify security vulnerabilities, helping developers avoid repeating past mistakes [10]. In today's software development environment, ML-driven SDP is essential for resource optimization, improved software quality, the construction of more reliable and secure software systems, and the early detection and correction of errors [11].

ML-based SDP has revolutionized software quality assurance and development practices. Traditional rule-based or manual methods for discovering and addressing software defects are labour-intensive and prone to errors. In contrast, ML algorithms leverage data to enhance fault detection and prediction accuracy. ML models excel at identifying trends and anomalies within large datasets, uncovering hidden patterns that aid in fault prediction. ML-based defect prediction systems exhibit exceptional versatility and adaptability [12]. As these models learn from data, their predictive capabilities improve, enabling software development teams to anticipate and address issues proactively. Each software project can benefit from a customized ML model tailored to predict specific software problems, enhancing the accuracy and usefulness of defect predictions. This data-driven, intelligent, and flexible approach to SDP using ML surpasses traditional methods [4]. ML is poised to quickly transform software defect prediction, repair, and prevention, resulting in more dependable and superior software systems.

Enhancing the model's efficiency is essential to reduce the time and financial resources spent on bug resolution, facilitating early bug detection before they progress to later stages of development [13]. Additionally, proactive actions taken during development enhance the program's overall security stance by mitigating potential risks. Optimizing the performance of the SDP model, reducing costs, and streamlining processes are critical objectives. These goals drive the development of a new hybrid intelligent model, striving to accomplish the subsequent goals:

- To develop a ML framework that boosts accuracy in detecting flawed software modules and potentially uncovers defects that were not previously identified.
- To enhance the precision of SDP through the reduction of feature dimension complexity via PCA-LDA
- To utilize an Extreme Learning Machine (ELM), an upgraded single hidden layer feed-forward network (SLFN), in combination with conventional learning methods to enhance overall generalization performance and expedite the learning process.
- To investigate the improved JAYA optimization method (IMJAYA) is to optimize the weights and biases of the ELM to enhance its generalization performance relative to prior approaches, achieved through a reduction in the number of hidden nodes.
- To assess its performance, test the proposed model using various datasets such as CM1, KC2, KC3, MC1, PC1, and JM1. Furthermore, its efficiency will be compared to existing methods as part of the evaluation process.

The key innovation of this study lies in the method employed for feature normalization and reduction. Integrating the PCA-LDA and IMJAYA-ELM algorithms, a hybrid methodology is developed to enhance the accuracy of software failure predictions. Here is an overview of the study: Section II summarizes the literature review, followed by a detailed presentation of the proposed approach in Section III. Section IVcomprehensively explores the experimental phase, Section V provides the overall discussion on the research work and Section VI concludes with details of the results and suggestions for further research.

II. RELATED WORK

Earlier studies have shown a growing interest in using ML algorithms for SDP model design. Researchers have applied ML techniques to forecast issues within software modules, and this area has been extensively researched. Understanding this prior work is crucial for appreciating the contributions of the current study. In their research, Kassaymeh et al. [14] utilized the salp swarm algorithm (SSA) along with a backpropagation neural network (BPNN) for SDP. This approach targets the prediction of software problems, a well-known issue in software engineering that aims to identify issues during or after a project. Using the SSA optimizer and BPNN, the authors improved prediction accuracy by optimizing BPNN parameters. The proposed technique was evaluated using software fault prediction datasets of various sizes and complexities, with performance metrics including AUC, Confusion Matrix, Sensitivity, Specificity, Accuracy, and Error Rate. The combination of SSA and BPNN outperformed other methods when applied to the Ar3 dataset, achieving a score of 0.89. Additionally, Qiao et al. [15] introduced a deep learning approach for software module defect prediction, which outperformed several other techniques on specific datasets. Their method significantly improved over existing techniques, reducing mean square error and increasing the squared correlation coefficient. In another study, Matloob et al. [16] surveyed SDP, focusing on ensemble or hybrid classification algorithms. They summarized trends in ensemble learning methods, highlighting popular techniques like random forest (RF), boosting, and bagging while noting that others, such as stacking, voting, and Extra Trees, were less commonly used. Various ensemble learning frameworks showed promising results in improving software defect prediction. Mehta and Patnaik [1] discovered that XGBoost and Stacking yielded highly effective results, achieving defect prediction accuracy exceeding 90.00% while also addressing data imbalance and high dimensionality issues within the defect dataset using SMOTE. Like PLS regression, SMOTE combines two methods to mitigate the impact of erroneous dataset dimensionality.

According to Li et al. [17], there is a rising interest in the use of unsupervised machine learning (ML) techniques in SDP, which offers advantages such as requiring less labelled training data. The primary objective is to evaluate the effectiveness of SDP using unsupervised learning methods. The researchers assessed prediction performance across various studies using confusion matrices and the Matthews Correlation Coefficient (MCC). A meta-analysis revealed unsupervised models demonstrate prediction capabilities comparable to supervised models within and across different software projects. The most effective unsupervised model families identified were FSOMs (Fuzzy Self-Organizing Maps) and Fuzzy CMeans (FCM). Furthermore, Prashanthi and Chandra Mohan [18] emphasized the critical importance of software quality due to the widespread use of software across various industries. SDP's primary objectives are to improve software quality and address software issues. The current state of SDP methods emphasizes dependability and robustness. The paper introduces the Optimized NN (Optimized Neural Network), a hybrid optimization-based neural network for software fault detection. Feature selection and SDP are integrated into this approach, where the relief algorithm is used for feature selection to identify deficient and non-deficient features. Subsequently, a neural network classifier is fine-tuned using a hybrid optimization strategy that combines the social spider algorithm (SSA) and the grey wolf optimizer (GWO). Through K-fold validation, the developed prediction model achieves an impressive accuracy of 0.9364 based on the evaluation.

Standard automated software fault prediction methods utilize machine learning (ML). However, current ML-based approaches necessitate manual feature extraction by humans, which is time-consuming and cannot often capture semantic information from bug-reporting tools. In contrast, deep learning (DL) methods enable automatic extraction and learning from complex, high-dimensional data. Giray et al. [19] developed a DL-based SDP model that automatically recognizes, analyzes, summarizes and synthesizes SDP data. They identified Convolutional Neural Network (CNN) as the most utilized DL algorithm for software defect prediction. Azzeh et al. [20] introduced a Support Vector Machine (SVM) model for SDP, commonly employed in building SDP models. They investigated the impact of four kernel functions and feature selection on SVM performance for SDP, assessing their stability. Their study examined various experimental parameters, including feature subsets, data granularity, and dataset imbalance ratio, and compared linear and nonlinear kernel functions. A comprehensive study using four kernel functions, ten feature subset selection criteria based on information gain, thirty-eight public datasets, and one evaluation measure demonstrated that SVM with a Radial Basis Function (RBF) kernel enhances SDP accuracy.

SDP is crucial in identifying problematic classes or modules early in software development. Mafarja et al. [21] conducted a study comparing several machine learning algorithms, including KNN, NB, LR, DT, SVM, and RF. They found that RF excelled in eliminating unnecessary or duplicated features. They utilized the Binary Whale Optimization Algorithm (BWOA) to further enhance RF performance and eliminate irrelevant or redundant features. The BWOA performance was enhanced through SBEWOA, which combines exploration techniques from the Grey Wolf Optimizer (GWO) and Harris Hawks Optimization (HHO) algorithms. Evaluation using PROMISE datasets considered accuracy, feature count, and fitness function metrics. Detecting software bugs is crucial for ensuring quality and reliability in software development. Yang et al. [22] proposed an SDP model incorporating a hybrid fitness function that leverages Particle Swarm Optimization (PSO) and Social

Spider Algorithm (SSA) to expedite convergence before SSA updates. They introduced a fitness function based on maximum likelihood estimation of initialization parameters. Their hybrid approach (SSA-PSO) demonstrated superior convergence time, stability, and accuracy performance across multiple datasets. Das et al. [23] employed the Golden Jackal Optimization (GJO) algorithm, inspired by golden jackal hunting behaviours, to develop a novel feature selection (FS) method for SDP. They tested this method using KNN, DT, NB, and QDA classifiers on SDP datasets. Comparisons against other feature selection approaches, including FSDE, FSPSO, FSGA, and FSACO, indicated that FSGJO consistently improved categorization accuracy across observations. Statistical analysis using Friedman and Holm tests confirmed that the proposed strategy was effective in selecting optimal features compared to previous methods.

Soft computing and ML are critical approaches for addressing the challenging process of SDP, which is essential for ensuring the fault-free performance of software components. Hassouneh et al. [25] introduced a Whale Optimization Algorithm (WOA) model utilizing a single-point crossover approach to enhance exploration and avoid local optima. Various selection methods were employed: Tournament, Roulette wheel, Linear rank, Stochastic universal sampling, and random-based selection. A comprehensive analysis of 17 PROMISE SDP datasets demonstrated that the proposed augmentation outperformed the original WOA and six other state-of-the-art approaches, leading to improved performance of the ML classifier. Lee et al. [26] proposed a cost-sensitive decision tree optimization method using harmony search (HS-CSDT) to enhance SDP model performance. In HS-CSDT, parameters such as the appropriate feature set, regularization method, class weight, and decision tree hyperparameters are optimized using harmony search. They compared HS-CSDT against similar methods across 28 open-source projects, evaluating metrics such as detection probability, false alarm rate, G-measure, and reduction in file inspection.

Dimensionality reduction techniques are crucial for addressing challenges posed by high-dimensional datasets in SDP. The abundance of features in software codebases makes it challenging to construct efficient and effective defect prediction models, a phenomenon often referred to as the curse of dimensionality. Dimensionality reduction methods such as PCA, LDA, and feature selection aim to decrease the number of features while retaining essential data. This process enhances model interpretability, reduces computational complexity, and mitigates overfitting. By extracting relevant patterns and eliminating noise or redundant information, dimensionality reduction facilitates faster identification and resolution of software defects, ultimately enabling the development of more effective defect prediction models.

In this study, an improved methodology of JAYA optimization (IMJAYA) has been incorporated into software defect prediction (SDP) to enhance the model's efficiency and accuracy. Swift bug identification and resolution are vital in software development. By leveraging JAYA optimization for feature selection, hyperparameter tuning, and training machine learning models, defect prediction systems can be tailored more effectively to specific datasets. The algorithm's capability to handle imbalanced datasets is particularly beneficial when addressing issues related to defective instance distribution. Researchers and practitioners can utilize JAYA optimization to analyze and potentially enhance defect prediction models, thereby advancing software development due to its straightforwardness and adaptability.

III. MATERIALS AND METHOD

This section describes the proposed methodology for software defect prediction and the data used to validate the proposed model.

A. PROPOSED METHODOLOGY

The methodology employed in the proposed model is outlined in this section. Figure 1 illustrates the general layout of the suggested model. The model is developed using the NASA dataset and progresses through three key steps: data preprocessing, dimensionality reduction, and performance evaluation. The system starts by acquiring input from the NASA dataset, where missing values are addressed through preprocessing. Subsequently, an appropriate dimensionality reduction technique is applied. The final step involves assessing the model's performance using various evaluation metrics and a hybrid approach that integrates IMJAYA (Improved JAYA) with an Extreme Learning Machine (ELM), IMJAYA-ELM.

B. DATASETS

The PROMISE repository, which is publicly accessible, provides access to datasets, including the NASA PROMISE dataset [1]. This dataset collection consists of software engineering datasets utilized in various NASA programs, encompassing bug reports, feature requests, and other essential metrics relevant to software engineering. Empirical studies and software flaw prediction in software engineering often rely on this common dataset. This study employed six specific NASA datasets (CM1, KC2, KC3, MC1, PC1, and JM1) [27], [28] to predict software problems. Table 1 presents comprehensive details regarding the datasets utilized.

The features table of all the above datasets are given below in Table 2 where *Yes* indicates the feature is present in that dataset and *No* indicates the feature is absent in that dataset.

C. DATA PRE-PROCESSING

The machine learning pre-processing phase aims to prepare the raw data for training machine learning models in a suitable format, which involves cleaning and standardizing the data to facilitate detailed analysis [29]. Tasks in this stage include addressing imbalanced data, partitioning, cleaning and transforming data, and selecting relevant features [3]. In this study, missing values in the dataset were estimated

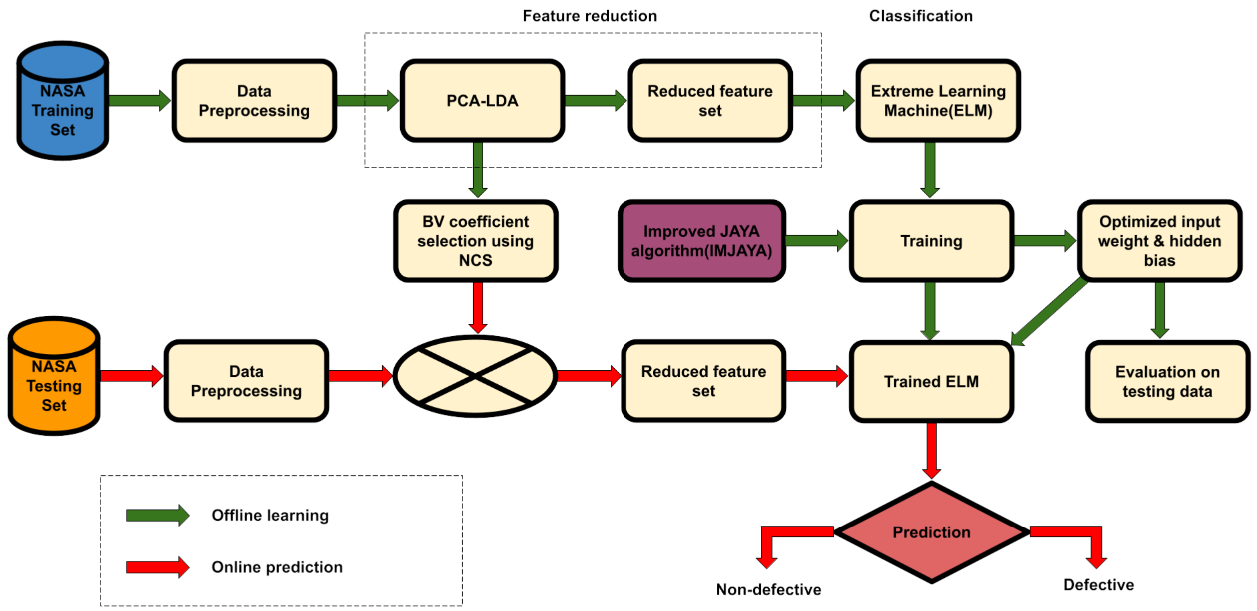


FIGURE 1. Schematic diagram for the suggested framework.

TABLE 1. Dataset used.

Dataset	Type	Language	Total instance	Non-defective instance	Defective instance
CM1	PROCEDURAL	C	327	285	42
KC2	OBJECT ORIENTED	C++	522	416	106
KC3	OBJECT ORIENTED	JAVA	194	159	35
MC1	OBJECT ORIENTED	C++	1988	1943	45
PC1	PROCEDURAL	C	705	645	60
JM1	PROCEDURAL	C	10885	2106	8779

during data preparation based on the dataset's features. The approach involved calculating the average value of a specific property across the dataset and using this average to fill in missing data entries.

D. FEATURE NORMALIZATION

An integral aspect of any machine learning pre-processing pipeline is feature normalization (FN), aimed at uniformly standardizing features across datasets [9]. This process involves adjusting the values of independent variables to facilitate quicker convergence of algorithms and prevent any single feature from exerting excessive influence on the learning process. Standard normalization techniques include z-score normalization and Min-Max scaling, which rescale data to fit within a specified range, typically between 0 and 1 [30]. FN enhances the model's performance and ability to generalize to new samples by making it less sensitive to variations in input data. Additionally, normalization improves the interpretability of model parameters, providing insights into how different features contribute to the learning process [31]. Overall, feature normalization is crucial for ensuring the robustness and effectiveness of machine learning algorithms across diverse scenarios [32].

Here, feature normalization is typically represented by

$$F_f = \frac{F - F_{min}}{F_{max} - F_{min}} \tag{1}$$

In this context, F_f denotes the normalization value, while F_{max} and F_{min} signify the maximum and minimum values of a particular feature, respectively.

E. FEATURE DIMENSIONALITY REDUCTION

Feature dimensionality reduction is crucial in predictive modelling and machine learning tasks [33]. As datasets become increasingly complex and high-dimensional, challenges such as heightened computational requirements, overfitting, and reduced model interpretability arise due to the abundance of features [34]. Feature dimensionality reduction techniques aim to address these challenges by identifying and modifying the most informative features while eliminating redundant or irrelevant ones [10]. The primary goals include improving prediction accuracy, reducing training time, and enhancing the model's generalizability across different scenarios. By reducing the number of features, these strategies simplify data representation, filter out irrelevant information, and highlight the most important patterns [35]. This streamlining process reduces computational complexity and enhances the model's interpretability, facilitating a deeper understanding

TABLE 2. Feature availability in each dataset.

Feature Name	CM1	KC2	KC3	MC1	PC1	JM1
LOC (Lines of Code)	Yes	Yes	Yes	Yes	Yes	Yes
ELOC (Executable LOC)	Yes	Yes	Yes	Yes	Yes	Yes
CYCLO (Cyclomatic Complexity)	Yes	Yes	Yes	Yes	Yes	Yes
NUM_OPERANDS	Yes	Yes	Yes	Yes	Yes	Yes
NUM_OPERATORS	Yes	Yes	Yes	Yes	Yes	Yes
HALSTEAD_DIFFICULTY	Yes	Yes	Yes	Yes	Yes	Yes
MAINTAINABILITY_INDEX	No	No	Yes	Yes	Yes	Yes
DEFECT	Yes	Yes	Yes	Yes	Yes	Yes
BRANCH_COUNT	Yes	Yes	Yes	Yes	Yes	Yes
CALL_PAIRS	No	No	No	Yes	Yes	Yes
PERCENT_COMMENTS	Yes	Yes	Yes	Yes	Yes	Yes
NUMBER_OF_CLASSES	Yes	Yes	Yes	Yes	Yes	Yes
NUMBER_OF_FILES	Yes	Yes	Yes	Yes	Yes	Yes
NUMBER_OF_FUNCTIONS	Yes	Yes	Yes	Yes	Yes	Yes
WEIGHTED_METHODS	Yes	Yes	Yes	Yes	Yes	Yes

of the relationships between features and the target variable. Feature dimensionality reduction is a critical area of research in handling high-dimensional data, which is essential for developing efficient machine learning models [5]. The process involves extracting relevant information from multiple features based on predefined criteria such as class distinguishability or classification performance. This curated approach improves model accuracy while mitigating overfitting issues and expediting the training process by reducing complexity and enhancing interpretability.

In this study, a hybrid approach called PCA-LDA has been implemented, combining PCA and LDA to reduce dimensionality. Further details about this methodology are discussed below:

1) PCA-LDA

Standard techniques for reducing dimensionality in machine learning and pattern recognition include PCA and LDA. PCA focuses on maximizing data variance, while LDA aims to identify feature combinations that better distinguish between different classes [36]. The intensive memory and computational requirements arise from large feature matrices, necessitating dimensionality reduction. PCA is commonly used to reduce dataset dimensions while preserving diversity, whereas LDA incorporates class information for dimensionality reduction in high-dimensional data [37]. However, traditional LDA faces challenges with high-dimensional, small-sample-size datasets due to issues with the within-class scatter matrix [38].

Combining PCA and LDA with specific machine learning classifiers improves outcomes by using smaller feature sets and enhancing accuracy. To ensure the uniqueness of the within-class scatter matrix, the total number of features (M) and the number of target classes (t) must be at least '$M + t$'. This study adopts a hybrid PCA and LDA approach to address this requirement. Initially, PCA is used to reduce the features from M dimensions to m dimensions, followed by LDA to reduce further the 'm' dimensions to 'c' dimensions, where c is much less than m and m is much less than M. The significance of the feature matrix is enhanced by sorting the eigenvalues of the features in descending order [33]. Next, the normalized cumulative sum of variance (NCV) for each feature is calculated [39]. The procedure to determine the NCV of the x^{th} feature is as follows:

$$NCV\left(x\right) = \frac{\sum_{u=1}^{i} \beta\left(u\right)}{\sum_{u=1}^{A} \beta\left(u\right)}, \quad 1 \leq x \leq V \tag{2}$$

In this scenario, 'V' denotes the dimension of the feature vector, and $\beta\left(u\right)$ represents the eigenvalue corresponding to the u^{th} feature. A manual threshold is set to conclude the selection process, and features are considered suitable if their NCV value surpasses the predetermined threshold. The most effective feature selection is determined through practical experimentation [40]. It is important to emphasize that only the coefficients of the first 'e' eigenvectors meeting the threshold are preserved for feature reduction during online prediction. In this context, these chosen eigenvectors are called "Basis Vectors" (B_v).

The combined dimensionality reduction technique, PCA-LDA, has been applied to select essential features from the dataset. The NCV metric helped identify relevant features from the high-dimensional dataset. NCV values for different features were calculated individually using both PCA and PCA-LDA methods [12]. The experiments revealed that PCA-LDA, with 16 features, retains more information than PCA, which requires 22 features. A manual NCV threshold of 0.97 has been established. The findings demonstrate that the PCA-LDA approach, especially with 16 features, achieves higher accuracy than PCA when applied to the KC2 dataset. Considering similar features, this methodology has been extended to other datasets such as CM1, KC3, MC1, PC1, and JM1 [41].

F. IMJAYA-ELM

This section briefly overviews Extreme Learning Machine (ELM) and JAYA Optimization, then introduces the proposed IMJAYA-ELM algorithm.

1) EXTREME LEARNING MACHINE(ELM)

The capacity of single-hidden layer feedforward neural networks (SLFNs) to accurately imitate continuous functions and categorize discontinuous portions has led to extensive use over the past two decades [42]. The Levenberg-Marquardt (LM) and the Backpropagation (BP) algorithms commonly train SLFNs using gradients [43]. Although widely utilized, traditional training algorithms suffer from several drawbacks, such as slow learning rates from inefficient steps, susceptibility to overfitting and getting stuck in local minima, necessitating multiple iterations for performance improvement. In contrast, the newly introduced ELM addresses these challenges posed by gradient-based methods [44]. The ELM exhibits strong capabilities in classification, regression, and prediction tasks. It learns more quickly and shows improved generalization performance compared to traditional learning algorithms such as BP, SVM, and Least Squares Support Vector Machines (LS-SVM). Unlike SLFNs, which calculate output weights by inverting the hidden layer output matrix [45], ELM randomly assigns parameters to hidden nodes, including input weights and hidden biases [46]. The mathematical intricacies of ELM are detailed below.

In the context provided, S represents a list of randomly selected training samples (x_i, t_i), where $x_i = [x_{i1}, x_{i2}, \ldots, x_{il}]^T \in R^l$ and $t_i = [t_{i1}, t_{i2}, \ldots, t_{iC}]^T \in R^C$. The SLFNs comprise n_h hidden nodes, and the activation function $\theta(.)$ is defined as follows:

$$\sum_{i=1}^{n_h} w_i^o \theta(x_j) = \sum_{i=1}^{n_h} w_i^o \theta(w_i^h \cdot x_j + \hat{b}_i) = o_j, \quad j = 1, 2, \ldots, S \tag{3}$$

Here, the weight vector is $w_i^h = [w_{i1}^h, w_{i2}^h, \ldots, w_{il}^h]^T$ which specifies a connection in a hidden neuron (i^{th}) and the input neurons. Most likely, the hidden neuron's weight vector is interpreted as $w_i^o = [w_{i1}^o, w_{i2}^o, \ldots, w_{iC}^o]^T$. At last, the hidden neuron of bias is i^{th} specified by $\hat{b}_i$. The estimated value of SLFNs for S has zero error, i.e., $\exists\ w_i^h,\ w_i^o,\ \hat{b}_i$ such that

$$\sum_{i=1}^{n_h} w_i^o \theta(w_i^h \cdot x_j + \hat{b}_i) = t_j, \quad j = 1, 2, \ldots, S \tag{4}$$

The vector format of Equation(4) is explained below:

$$Zw^o = F \tag{5}$$

Here,

$$Z(w_1^h, w_2^h, \ldots, w_{n_h}^h, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{n_h}, x_1, x_2, \ldots, x_N) = \begin{bmatrix} \theta(w_1^h \cdot x_1 + \hat{b}_1) & \ldots & \theta(w_{n_h}^h \cdot x_1 + \hat{b}_{n_h}) \\ \vdots & \ldots & \vdots \\ \theta(w_1^h \cdot x_N + \hat{b}_1) & \ldots & \theta(w_{n_h}^h \cdot x_N + \hat{b}_{n_h}) \end{bmatrix}_{\mathbb{N} \times n_h},$$

$$w^o = \begin{bmatrix} w_1^{oT} \\ \vdots \\ w_{n_h}^{o\ T} \end{bmatrix}_{n_h \times C}$$

and

$$F = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{\mathbb{N} \times C}$$

where $\mathbf{Z}$ represent the hidden layer output matrix. Now, the output weight w^o can be calculated by finding the smallest least norm square (LS) solution of the above linear system equation (3) as

$$\hat{w^o} = Z^\dagger F \tag{6}$$

Since $\mathbf{Z}^\dagger$ denotes the Moore-Penrose(MP) generalized inverse of matrix Z, this technique aids ELM in improving its generalization efficiency. The LS solution with the least norm is the only one. ELM converges faster than conventional learning algorithms since its answer is analytically obtained without iteratively tuning parameters [47].

2) JAYA ALGORITHM

The JAYA algorithm, an innovative optimization technique devised by Rao, has gained significant popularity in research due to its simplicity and robustness. Results obtained with the JAYA algorithm consistently surpass those achieved by other optimization methods [48]. What sets this algorithm apart is its reliance on basic control parameters such as population size and generation number. The core concept behind the JAYA algorithm is to steer the solution directly towards the optimal outcome while avoiding less desirable paths [7].

Let us consider the function $f(s)$ as the objective function to maximize or minimize. There are n potential solutions at each iteration k, where j varies from 1 to n, and each solution has a dimension d, where d ranges from 1 to m.

At iteration k, the value of $s_{j,d}(k)$, which denotes the j^{th} solution in the d^{th} dimension, can be adjusted and its altered value is observed [49].

$$s'_{jd}(k) = s_{jd}(k) + r_{1d}(k)(s_{MOd}(k) - |s_{jd}(k)|) - r_{2d}(k)(s_{LOd}(k) - |s_{jd}(k)|) \quad (7)$$

where $s_{MO}(k)$ represents the value for the most optimal solution in d^{th}dimension and $s_{LO}(k)$ represents the value for the less optimal solution in d^{th} dimension during iteration k. It is worth mentioning that the candidate's most optimal and least optimal are the best and worst solutions [50], with the best and worst fitness values in the entire population of an iteration. $r_{1d}(k)$ and $r_{2d}(k)$ indicate two random numbers in dimension d during k^{th} iteration which lie in the interval $[0, 1]$. $s'_{jd}(k)$ denotes the updated value of $s_{jd}(k)$. The term "$r_{1d}(k)(s_{MOd}(k) - |s_{jd}(k)|)$" defines that the solution tries to move toward the best solution and the term "$r_{2d}(k)(s_{LOd}(k) - |s_{jd}(k)|)$" indicates that the solution tries avoid the worst solution. The modified $s'_{jd}(k)$ value is accepted if the functional value generated by it is better. The overall steps involved in the JAYA algorithm are shown in Figure.2.

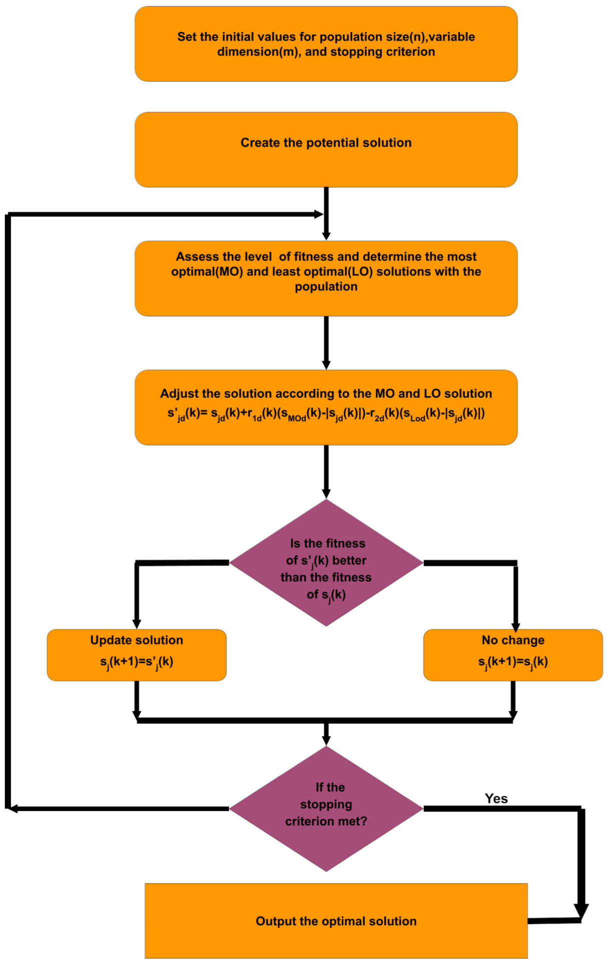

Set the initial values for population size(n),variable dimension(m), and stopping criterion
Create the potential solution
Assess the level of fitness and determine the most optimal(MO) and least optimal(LO) solutions with the population
Adjust the solution according to the MO and LO solution
s'jd(k)= sjd(k)+r1d(k)(sMOd(k)-|sjd(k)|)-r2d(k)(sLod(k)-|sjd(k)|)
Is the fitness of s'j(k) better than the fitness of sj(k)
Update solution sj(k+1)=s'j(k)
No change sj(k+1)=sj(k)
If the stopping criterion met?
Yes
Output the optimal solution

FIGURE 2. Flowchart for JAYA Algorithm.

3) PROPOSED IMPROVED METHODOLOGY FOR JAYA BASED EXTREME LEARNING MACHINE(IMJAYA-ELM)

ELM faces two critical challenges due to its method of arbitrarily selecting input weights and hidden biases. Firstly, this approach necessitates more hidden neurons than traditional gradient-based methods, decreasing responsiveness to unknown test data. Secondly, the increased number of hidden neurons contributes to an ill-conditioned output matrix M within the hidden layer, leading to poorer generalization performance [50]. It's important to highlight the condition number as a useful qualitative measure for evaluating matrix conditioning. This number quantifies how close a system is to becoming ill-conditioned, with a low condition number indicating good conditioning and a high number indicating the opposite. The 2-norm condition number of matrix M can be computed as follows:

$$\kappa_2(M) = \sqrt{\frac{\lambda_{max}(M^T M)}{\lambda_{min}(M^T M)}} \quad (8)$$

Here, $\lambda_{max}(M^T M)$ denotes the largest eigenvalue and $\lambda_{min}(M^T M)$ specified as smallest eigenvalue of the matrix $M^T M$.

To address these challenges, researchers have explored the use of evolutionary algorithms (EAs) and swarm intelligence-based algorithms over the past decade, leveraging their capabilities in global optimization [7]. Some researchers proposed a hybrid approach known as evolutionary ELM (E-ELM), where a modified differential evolution (DE) algorithm optimizes hidden node parameters and a Moore-Penrose (MP) generalized inverse is employed to find solutions [49]. They demonstrated that E-ELM offers faster learning and improved generalization compared to traditional methods, with a more streamlined network structure than ELM. However, E-ELM requires tuning two additional parameters: mutation and crossover factors. Another approach utilizes a PSO-based evolutionary ELM (PSO-ELM) to select hidden node parameters, necessitating only one parameter for tuning. Researchers have integrated boundary conditions into conventional Particle Swarm Optimization (PSO) to enhance the performance of Extreme Learning Machine (ELM) [51]. Subsequently, an enhanced PSO-based ELM (IPSO-ELM) was proposed to optimize Single Hidden Layer Feedforward Networks (SLFNs). IPSO incorporates the root mean squared error (RMSE) and the norm of output weights from a validation set to achieve improved convergence performance. A hybrid learning algorithm has been introduced using a real-coded genetic algorithm and ELM (RCGA-ELM) for no-reference image quality assessment. However, RCGA requires two genetic parameters—crossover and mutation [52]. Meanwhile, an input weight selection technique has been introduced to enhance ELM's conditioning using linear hidden neurons. This approach achieved numerical stability without compromising accuracy.

Various researchers have employed optimization algorithms such as Genetic Algorithms (GA) and its variants, Particle Swarm Optimization (PSO) and its derivatives, Differential Evolution (DE), and others to discover optimal hidden node parameters. While these methods offer distinct advantages, they require careful tuning of algorithm-specific parameters, significantly impacting their performance [53]. To tackle the issues related to parameter tuning mentioned above, this research utilizes a parameter-free method known as the JAYA algorithm. A new approach named IMJAYA-ELM is introduced by integrating the Improved JAYA (IMJAYA) algorithm with ELM. IMJAYA-ELM mitigates issues identified in recent literature by leveraging IMJAYA to optimize hidden node parameters and using the Moore-Penrose (MP) generalized inverse for analytical solutions [35].

The improved JAYA algorithm targets global optima by considering both the root mean squared error (RMSE) and the norm of output weights in Single Hidden Layer Feedforward Networks (SLFNs), which enhances generalization performance and SLFN conditioning. IMJAYA aims to minimize the norm of output weights and restrict hidden node parameters within specific ranges to improve the convergence performance of ELM. The proposed IMJAYA-ELM approach is outlined as follows:

(1)Initially, all candidate solutions within the population will be randomly initialized, and each candidate solution will include a set of input weights and hidden biases.

$$s_j = \Big[w^h_{11}, w^h_{12}, \ldots, w^h_{1l}, w^h_{21}, w^h_{22}, \ldots, w^h_{2l}, w^h_{n_h1}, w^h_{n_h2}, \ldots, w^h_{n_hl}, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{n_h}\Big] \tag{9}$$

It's important to note that all input weights and hidden biases are initialized randomly within a range of $[-1, 1]$.

(2) For each solution, assess the output weights and fitness. In this context, fitness is defined as the root-mean-squared error (RMSE) calculated on the validation set rather than the entire training set to mitigate overfitting. The fitness is determined by:

$$f_y() = \sqrt{\frac{\sum_{j=1}^{N_v} || \sum_{i=1}^{n_h} w^o_i \theta(w^h_i \cdot x_j + \hat{b}_i) - t_j||^2_2}{N_v}} \tag{10}$$

Here, N_v is specified as the list of validation samples.

(3) Identify the best (s_{MO}) and worst (s_{LO}) solutions from all the solutions in the population and then adjust the solutions using equation (7).

(4) Update the solutions based on the fitness value and the norm of the output weights, then generate a new population using the following procedure:

$$s_j(k+1) = \begin{cases} s'_j(k)\text{if}\left(s_j(k)\right) - f \\ (s'_j(k)) > \epsilon f(s_j(k)) or (|f(s_j(k)) \\ -f(s'_j(k))| < \epsilon f(s_j(k)) and ||\omega^o_{s'_j}|| \\ < ||\omega^o_{s_j}|| \\ s_j(k)\text{otherwise} \end{cases} \tag{11}$$

(5) According to the literature, all input weights and biases are recommended to fall within the range of $[-1, 1]$. Therefore, the IMJAYA-ELM uses the following equation to address the issue of solutions going out of this range.

$$s_{jd}(k+1) = \begin{cases} -1 & \text{if } s_{jd}(k+1) < -1 \\ 1 & \text{if } s_{jd}(k+1) > 1, \end{cases} \quad 1 \le j \le N_p, 1 \le d \le D \tag{12}$$

(6) Continuously perform steps (3) to (5) until the maximum number of iterations is reached. Eventually, the best input weights and hidden biases are determined and used with the testing data to evaluate the system's performance.

The suggested approach described in equation(11) is utilized to determine the optimal input weights and hidden biases, aiming to achieve smaller norm values for the output weights of SLFNs. Consequently, this results in a reduced condition value for the output hidden matrix. The proposed IMJAYA-ELM offers several advantages: it operates without algorithm-specific parameters, enhances conditioning, and demonstrates improved generalization performance with a more compact network structure. In contrast to other gradient-based techniques and traditional ELM methods, this approach does not require activation functions to be differentiable. The algorithm for the implementation of the proposed model for both offline learning and online prediction is given in algorithm 1 and algorithm 2.

Algorithm 1 Steps for Implementing the Suggested PCA-LDA+IMJAYA-ELM Method for Offline Learning

```
0: for each data sample do
0: Prepare the data by eliminating null values and applying
       min-max normalization to scale the features within
       the range of [0,1].
0: Acquire the normalized set of feature vectors with
       dimension $V$.
0: end for
0: Use PCA-LDA approach to reduce the dimension of
       feature vector from $V$ to $l$, where $l$ is calculated
       from NCV measure. Retain the corresponding $l$ basis
       vector coefficients.
0: Conduct k-fold cross-validation on the entire dataset to
       create training, validation, and testing data sets.
0: Train the ELM algorithm using the IMJAYA algorithm
       to determine the optimized input weights and hidden
       biases. In the IMJAYA process, assess the fitness
       RMSE on the validation set.
0: Determine the output weights using the optimized input
       weights and hidden biases.
0: Assess the prediction performance on the test set.=0
```

IV. EXPERIMENTAL SETUP AND PERFORMANCE METRICS USED

The IMJAYA-ELM model has been evaluated on a Windows 11 operating system with a core-i7 CPU clocked at 3.4GHz

Algorithm 2 Steps for Implementing the Suggested PCA-LDA+IMJAYA-ELM Method for Online Prediction

```
0: Input the unknown data into the proposed system.
0: Normalize the features to preprocess the query data.
0: Acquire the normalized feature and store it in a feature
     vector.
0: Determine the reduced set of features by multiplying
     the feature vector with the retained basis vector
     coefficients.
0: Input the reduced feature set into the SLFN classifier
     trained using IMJAYA-ELM and forecast the output
     label as either defective or non-defective.=0
```

and 16GB of RAM, using MATLAB version R2017B and the required tools. Various NASA datasets, including CM1, KC2, KC3, MC1, PC1, and JM1 were utilized for SDP. Details of the parameter list are provided below:

- True Positive Rate (Sensitivity): The sensitivity of a model refers to the proportion of true positive cases it correctly identifies.

$$Sensitivity = \frac{X}{X+Y} \tag{13}$$

- True Negative Rate (Specificity): Model specificity refers to correctly identifying negative cases among all potential negative instances.

$$Specificity = \frac{Z}{Z+T} \tag{14}$$

- Accuracy(Acc): The accuracy of a model is defined as the percentage of correctly classified cases out of all the instances evaluated, encompassing both positive and negative classifications.

$$Accuracy = \frac{X+Z}{X+Y+Z+T} \tag{15}$$

- Precision: One way to measure a classification model's effectiveness is by accurately predicting positive cases. The ratio of accurate optimistic predictions to total positive cases projected measures precision.

$$Precision = \frac{X}{X+T} \tag{16}$$

- F1 Score: The F1 Score assesses the effectiveness of a binary classification model by blending precision and recall. It achieves this balance by calculating the harmonic mean of precision and recall metrics.

$$F1Score = \frac{2X}{2X+T+Y} \tag{17}$$

- Matthews Correlation Coefficients(MCC): MCC is employed to assess the performance of binary classification models by considering both correct and incorrect predictions. Its value ranges from -1 to 1, where 1 signifies perfect prediction, 0 indicates no improvement over random prediction, and -1 indicates complete disagreement between prediction and observation.

$$MCC = \frac{(X*Z - T*Y)}{\sqrt{((X+T)*(X+Y)*(Z+T)*(Z+Y))}} \tag{18}$$

X, Y, Z, and T represent true positive, false negative, true negative, and false positive, respectively.

In machine learning, the evaluation of the gap between the anticipated outcomes of a model and the actual values in the training data is performed using the cost function, also known as a loss function or objective function. This component is crucial in machine learning activities. The main objective during training is to reduce this cost function to improve the accuracy of the forecasts generated by the model. The choice of cost functions for regression, classification, or prediction tasks is contingent upon the particular goals of the machine learning activity. Mean Squared Error (MSE) and Mean Absolute Error (MAE) are often used for regression tasks. Subsequently, optimization methods such as gradient descent are employed to fine-tune the model's parameters, minimizing the cost function and enhancing the model's overall performance.

The designated cost functions are MSE and MAD. When evaluating the efficacy of a prediction model, these measures are commonly used in regression analysis and time series prediction. They provide a numerical assessment of predictive accuracy by quantifying the average discrepancies between observed and expected values. The MSE is calculated by averaging the squared differences between predicted and actual values. The equation is as follows:

$$MSE = \frac{1}{X}\sum_{j=0}^{X}\left(e_j - \overline{e}_j\right)^2 \tag{19}$$

MAD is computed by averaging the absolute differences between predicted and actual values, assigning equal importance to errors regardless of magnitude. The MAD formula is described as follows:

$$MAD = Median\left(|e_j - \overline{e}_j|\right) \tag{20}$$

In the equation referenced as (19) and (20), X represents the overall count of samples within the dataset. Here, e_j denotes the actual label assigned to the j^{th} sample, while $\overline{e}_j$ stands for the predicted output associated with the j^{th} sample.

The model PCA-LDA+IMJAYA-ELM has been evaluated using multiple datasets, including CM1, KC2, KC3, MC1, PC1, and JM1. Cross-validation (CV) was employed for statistical analysis to enhance the classifier's generalization across different datasets and prevent overfitting. CV ensured consistency in class distributions by dividing the data into testing, validation, and training folds within a 5-fold configuration. The validation fold determined when training should stop and assisted in parameter tuning for IMJAYA-ELM. Performance assessment included sensitivity, specificity, accuracy, F1 score, MCC, and precision. Model performance

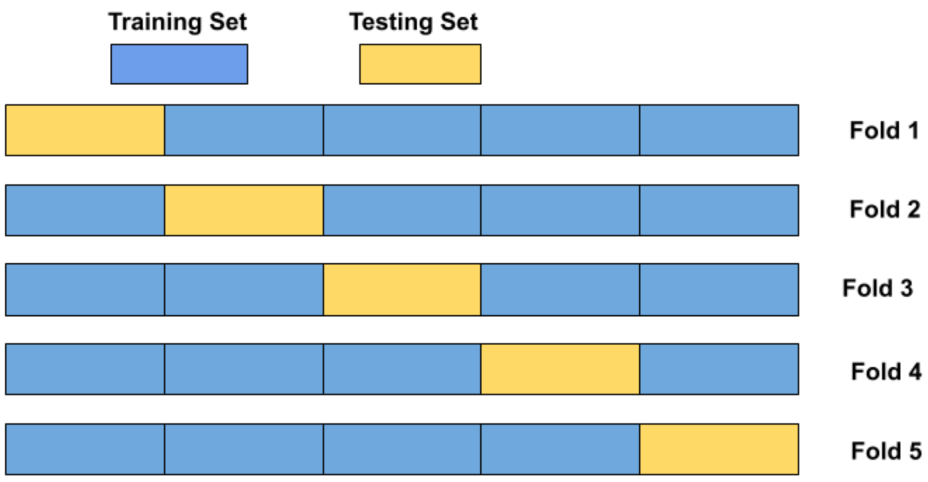


FIGURE 3. Five fold cross-validation for KC2 dataset.

was evaluated using loss functions MAD and MSE. Initially, the complete feature set has been utilized, followed by testing after applying two well-known feature reduction methods: PCA and PCA-LDA. The entire experiment was repeated ten times across five datasets to ensure result reliability and reduce randomness.

The methods have been examined using data obtained from NASA datasets. Figure 1 showcases the considerable endeavour dedicated to this paper via a block diagram. To address the potential overfitting issue, a commonly utilized technique in experimental studies is five-fold cross-validation (CV), as illustrated in Figure 3.

This study utilizes NASA datasets to assess the suggested methodology described in Table 1. The paper comprehensively explains the techniques graphically depicted in a block diagram in Figure 1. Within this particular framework, the model has been subjected to two experiments: one in which feature reduction has not been employed and another in which feature reduction has been employed. Cross-validation (CV) has been utilized to limit the possibility of overfitting and improve the model's accuracy by randomly dividing the data samples into training and testing sets. Model performance evaluation involves using many metrics, such as sensitivity, specificity, precision, F1 score, Matthews correlation coefficient (MCC), and accuracy. A range of conventional ML techniques, such as MLP, AB, J48, RF, NB, DT, and ELM, are evaluated using the NASA dataset. The studies utilize a hybrid classifier known as Improved JAYA Optimization with an Extreme Learning Machine (IMJAYA-ELM). Both algorithms have been set up with an initial population size of 20 and a maximum iteration limit of 200.

Figure 4 shows the learning curve by comparing the loss function of ELM, PCA+IMJAYA-ELM, and PCA-LDA+IMJAYA-ELM. The figure shows that

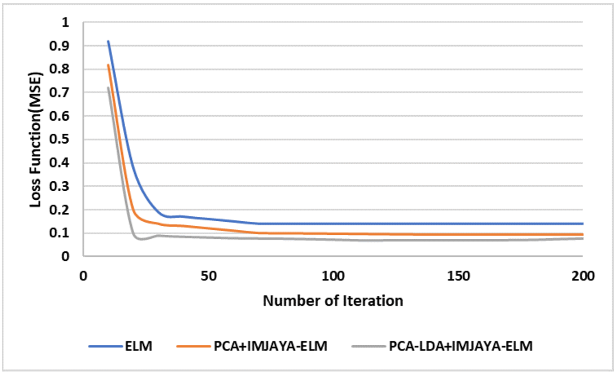


FIGURE 4. Learning curve outcomes comparison of suggested models.

PCA-LDA+IMJAYA-ELM minimizes the loss function more than ELM and PCA+IMJAYA-ELM alone.

This study includes three phases: without feature reduction, feature reduction with PCA, and feature reduction with PCA-LDA. These phases involve using seven different machine learning classifiers and five distinct datasets from NASA. In the initial phase, performance analysis has been conducted without feature reduction. When ELM was combined with IMJAYA, an accuracy of 96.15% was attained for the KC2 dataset without the use of any features shown in Table 3. The second phase evaluated the performance using PCA with 22 features (detailed in Table 4). For the KC2 dataset, PCA combined with IMJAYA-ELM resulted in an accuracy of 97.12%. The above procedure has been repeated in the third phase using PCA-LDA, which reduced the number of features to 16 (as shown in Table 5). The optimal accuracy of 98.08% for this problem was achieved by combining PCA-LDA with IMJAYA-ELM.

The outcomes of performing 10 × 5 fold CV using PCA-LDA+IMJAYA-ELM on the CM1, KC2, KC3, MC1, PC1, and JM1 datasets are presented in Table 6,

TABLE 3. Defect prediction accuracy without feature reduction.

Classifier	Dataset	Fold	Sensitivity(%)	Specificity(%)	Precision(%)	F1 Score(%)	MCC(%)	Acc(%)
MLP	CM1	5	93.33	92.31	91.80	92.56	85.60	92.80
AB	CM1	5	93.48	92.45	91.91	92.71	85.72	92.94
J48	CM1	5	91.67	95.38	94.83	93.22	87.21	93.60
RF	CM1	5	90.00	93.85	93.10	91.53	84.00	92.00
NB	CM1	5	91.66	95.31	94.74	93.17	87.16	93.20
DT	CM1	5	91.76	95.44	94.91	93.33	87.35	93.68
ELM	CM1	5	90.31	96.92	96.43	93.10	87.33	93.91
IMJAYA-ELM	CM1	5	91.67	96.92	96.49	94.02	88.86	94.40
MLP	KC2	5	96.39	90.48	97.56	96.97	85.38	95.19
AB	KC2	5	93.98	95.24	98.73	96.30	83.81	94.23
J48	KC2	5	96.88	90.95	97.82	97.15	85.71	95.45
RF	KC2	5	95.58	90.93	97.53	96.64	82.93	94.66
NB	KC2	5	96.09	90.18	97.26	96.67	85.08	95.42
DT	KC2	5	94.98	95.04	98.55	96.73	86.05	94.95
ELM	KC2	5	96.79	90.88	97.96	97.37	85.78	95.59
IMJAYA-ELM	**KC2**	**5**	**97.59**	**90.48**	**97.59**	**97.59**	**88.07**	**96.15**
MLP	KC3	5	96.88	71.43	93.94	95.38	72.65	92.31
AB	KC3	5	93.75	71.43	93.75	93.75	65.18	89.74
J48	KC3	5	94.66	72.58	94.81	94.66	66.32	90.58
RF	KC3	5	95.15	73.26	95.32	95.15	66.86	91.08
NB	KC3	5	94.83	72.78	95.01	94.81	66.57	90.78
DT	KC3	5	96.38	71.03	93.44	95.06	72.21	91.80
ELM	KC3	5	96.94	71.49	94.01	95.44	71.71	92.37
IMJAYA-ELM	KC3	5	97.38	71.93	94.44	95.88	73.15	92.81
MLP	MC1	5	94.09	57.14	99.19	96.57	26.78	93.43
AB	MC1	5	94.60	57.14	99.19	96.84	28.04	93.94
J48	MC1	5	93.83	42.86	98.92	96.31	19.18	92.93
RF	MC1	5	93.57	28.57	98.64	96.04	11.58	92.42
NB	MC1	5	93.94	42.92	98.98	96.37	19.24	92.99
DT	MC1	5	94.03	57.34	99.38	96.63	26.41	93.38
ELM	MC1	5	94.60	71.43	99.46	96.97	35.13	94.19
IMJAYA-ELM	MC1	5	95.12	71.43	99.46	97.24	36.75	94.70
MLP	PC1	5	95.22	74.81	97.46	96.17	63.38	93.49
AB	PC1	5	96.12	58.33	96.12	96.12	54.46	92.91
J48	PC1	5	95.51	75.12	97.77	96.59	63.77	93.77
RF	PC1	5	95.35	75.00	97.62	96.47	63.67	93.62
NB	PC1	5	95.35	58.33	96.09	95.72	51.78	92.20
DT	PC1	5	96.90	66.67	96.90	96.90	63.57	94.33
ELM	PC1	5	95.35	91.67	99.19	97.23	74.57	95.04
IMJAYA-ELM	PC1	5	96.90	83.33	98.43	97.66	74.86	95.74
MLP	JM1	5	93.97	79.73	99.45	96.63	42.85	93.62
AB	JM1	5	93.48	75.23	99.32	96.31	39.13	93.02
J48	JM1	5	94.23	79.73	99.45	96.77	43.67	93.88
RF	JM1	5	94.08	79.73	99.45	96.69	43.2	93.71
NB	JM1	5	92.95	72.97	99.25	96.00	36.65	92.46
DT	JM1	5	94.90	81.98	99.51	97.15	47.11	94.57
ELM	JM1	5	95.46	85.59	99.61	97.49	51.22	95.21
IMJAYA-ELM	JM1	5	96.16	87.84	99.67	97.88	55.61	95.96

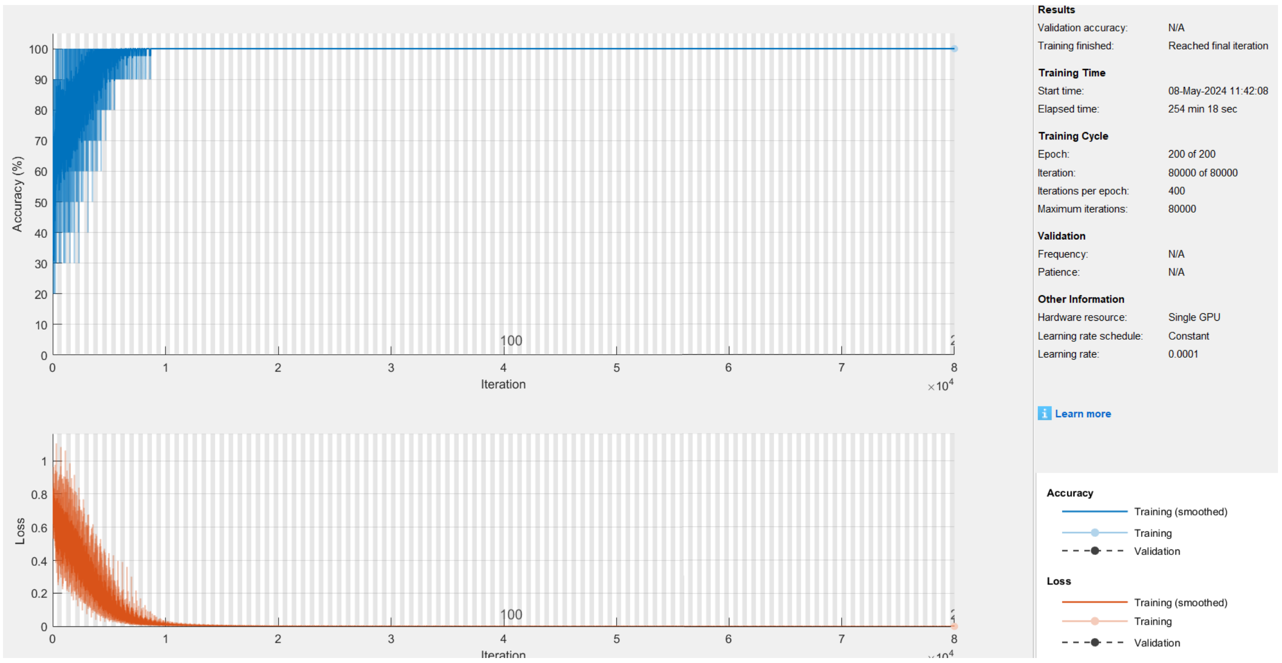
TABLE 4. Defect prediction accuracy with PCA.

Classifier	Dataset	Fold	No of Feature	Sensitivity(%)	Specificity(&)	Precision(%)	F1 Score(%)	MCC(%)	Acc(%)
PCA+MLP	CM1	5	22	96.49	71.43	96.49	96.49	67.92	93.75
PCA+AB	CM1	5	22	91.66	95.31	94.74	93.17	87.16	93.29
PCA+J48	CM1	5	22	93.51	92.48	91.94	92.74	85.76	93.97
PCA+RF	CM1	5	22	93.38	92.35	91.81	92.61	85.62	92.83
PCA+NB	CM1	5	22	93.34	92.31	91.76	92.57	85.59	92.79
PCA+DT	CM1	5	22	94.72	71.55	99.58	96.97	35.24	94.32
PCA+ELM	CM1	5	22	96.09	90.18	97.26	96.67	85.08	94.45
PCA+IMJAYA-ELM	CM1	5	22	96.49	87.50	98.21	97.35	79.89	95.38
PCA+MLP	KC2	5	22	97.68	90.57	97.68	97.68	88.16	96.06
PCA+AB	KC2	5	22	96.67	90.74	97.61	96.74	85.50	95.24
PCA+J48	KC2	5	22	97.58	90.47	97.58	97.58	88.06	96.14
PCA+RF	KC2	5	22	96.58	90.65	97.52	96.85	85.41	95.15
PCA+NB	KC2	5	22	96.94	91.01	97.88	97.21	85.05	95.51
PCA+DT	KC2	5	22	97.90	90.79	97.90	97.16	88.38	96.65
PCA+ELM	KC2	5	22	98.09	90.98	98.09	98.09	88.57	96.84
PCA+IMJAYA-ELM	**KC2**	**5**	**22**	**97.59**	**95.24**	**98.78**	**98.18**	**91.25**	**97.12**
PCA+MLP	KC3	5	22	97.12	71.67	94.19	95.62	71.89	92.54
PCA+AB	KC3	5	22	96.65	71.20	93.72	95.15	71.42	92.08
PCA+J48	KC3	5	22	95.55	70.19	92.61	94.23	71.38	90.97
PCA+RF	KC3	5	22	96.33	70.97	93.39	95.01	72.16	91.75
PCA+NB	KC3	5	22	96.12	70.76	93.18	94.80	71.95	91.54
PCA+DT	KC3	5	22	96.76	71.31	93.83	95.26	71.53	92.19
PCA+ELM	KC3	5	22	97.46	71.95	94.53	96.01	72.23	92.88
PCA+IMJAYA-ELM	KC3	5	22	98.22	72.77	95.28	96.72	73.99	93.65
PCA+MLP	MC1	5	22	94.60	66.67	99.19	96.84	36.22	93.97
PCA+AB	MC1	5	22	94.60	77.78	99.46	96.97	42.08	94.22
PCA+J48	MC1	5	22	93.83	66.67	99.18	96.43	34.07	93.32
PCA+RF	MC1	5	22	93.44	66.54	99.05	96.17	33.29	92.83
PCA+NB	MC1	5	22	93.57	66.67	99.18	96.30	33.42	92.96
PCA+DT	MC1	5	22	94.60	55.56	98.92	96.71	30.18	93.72
PCA+ELM	MC1	5	22	95.37	66.67	99.20	97.25	38.75	94.72
PCA+IMJAYA-ELM	MC1	5	22	95.89	77.78	99.47	97.64	46.93	95.48
PCA+MLP	PC1	5	22	95.47	75.12	97.74	96.59	63.79	93.74
PCA+AB	PC1	5	22	95.35	75.00	97.62	96.47	63.67	93.62
PCA+J48	PC1	5	22	96.12	66.67	97.64	96.88	57.24	94.20
PCA+RF	PC1	5	22	96.12	75.00	97.64	96.88	66.36	94.33
PCA+NB	PC1	5	22	96.12	55.56	96.88	96.50	49.22	93.48
PCA+DT	PC1	5	22	96.90	75.00	97.66	97.28	69.35	95.04
PCA+ELM	PC1	5	22	96.90	83.33	98.43	97.66	74.86	95.74
PCA+IMJAYA-ELM	PC1	5	22	97.67	98.05	98.44	98.05	78.13	96.45
PCA+MLP	JM1	5	22	94.44	72.97	99.26	96.79	40.77	93.91
PCA+AB	JM1	5	22	94.27	72.97	99.26	96.7	40.23	93.73
PCA+J48	JM1	5	22	94.78	75.68	99.34	97.00	43.33	94.29
PCA+RF	JM1	5	22	94.62	75.68	99.34	96.92	42.79	94.14
PCA+NB	JM1	5	22	94.17	72.97	99.26	96.65	39.95	93.64
PCA+DT	JM1	5	22	95.49	86.49	99.63	97.52	51.9	95.27
PCA+ELM	JM1	5	22	96.26	85.59	99.61	97.91	54.93	95.99
PCA+IMJAYA-ELM	JM1	5	22	96.96	87.84	99.68	98.3	60.06	96.74

Table 7, Table 8, Table 9, Table 10, and Table 11 respectively where fd represents folds and avg stands for average. Notably, the KC2 dataset demonstrated the highest average accuracy, reaching 98.08%.

TABLE 5. Defect prediction accuracy with PCA-LDA.

Classifier	Dataset	Fold	No of Feature	Sensitivity(%)	Specificity(%)	Precision(%)	F1 Score(%)	MCC(%)	Acc(%)
PCA-LDA+MLP	CM1	5	16	94.86	87.62	98.31	96.55	75.03	93.97
PCA-LDA+AB	CM1	5	16	94.74	87.50	98.18	96.43	74.88	93.85
PCA-LDA+J48	CM1	5	16	96.22	90.31	97.39	96.81	85.21	94.58
PCA-LDA+RF	CM1	5	16	93.91	86.67	97.35	95.61	74.05	93.02
PCA-LDA+NB	CM1	5	16	94.35	87.11	96.11	96.05	74.49	93.46
PCA-LDA+DT	CM1	5	16	96.48	90.57	97.65	97.06	85.47	94.84
PCA-LDA+ELM	CM1	5	16	96.32	87.31	98.02	97.16	79.70	95.19
PCA-LDA+IMJAYA-ELM	CM1	5	16	96.14	87.15	97.86	97.01	97.54	95.73
PCA-LDA+MLP	KC2	5	16	97.09	94.77	98.27	97.68	90.74	96.33
PCA-LDA+AB	KC2	5	16	97.16	91.23	98.11	97.43	85.27	95.73
PCA-LDA+J48	KC2	5	16	96.91	94.59	98.09	97.50	90.58	96.44
PCA-LDA+RF	KC2	5	16	97.59	90.48	97.59	97.59	88.07	96.15
PCA-LDA+NB	KC2	5	16	97.37	91.44	98.33	97.64	85.38	95.92
PCA-LDA+DT	KC2	5	16	97.31	94.98	98.50	97.89	90.97	96.84
PCA-LDA+ELM	KC2	5	16	97.59	95.24	98.78	98.18	91.25	97.12
PCA-LDA+IMJAYA-ELM	**KC2**	**5**	**16**	**98.80**	**95.24**	**98.80**	**98.80**	**94.03**	**98.08**
PCA-LDA+MLP	KC3	5	16	94.47	86.36	97.44	95.75	76.10	93.12
PCA-LDA+AB	KC3	5	16	93.91	85.87	96.93	95.40	75.67	92.47
PCA-LDA+J48	KC3	5	16	93.41	85.46	96.47	94.91	75.03	91.79
PCA-LDA+RF	KC3	5	16	93.75	85.71	96.77	95.24	75.52	92.31
PCA-LDA+NB	KC3	5	16	93.34	85.31	96.36	94.83	75.11	91.91
PCA-LDA+DT	KC3	5	16	93.92	85.88	96.94	95.41	75.69	92.48
PCA-LDA+ELM	KC3	5	16	94.84	86.73	97.81	96.12	76.47	93.49
PCA-LDA+IMJAYA-ELM	KC3	5	16	96.88	85.71	96.88	96.88	82.59	94.87
PCA-LDA+MLP	MC1	5	16	95.89	66.67	99.20	97.52	40.70	95.23
PCA-LDA+AB	MC1	5	16	95.37	66.67	99.20	97.25	38.75	94.72
PCA-LDA+J48	MC1	5	16	94.86	55.56	98.93	96.85	30.89	93.97
PCA-LDA+RF	MC1	5	16	94.34	55.56	98.92	96.58	29.50	93.47
PCA-LDA+NB	MC1	5	16	94.60	60.00	98.92	96.71	33.98	93.73
PCA-LDA+DT	MC1	5	16	95.12	55.56	98.93	96.99	31.65	94.22
PCA-LDA+ELM	MC1	5	16	95.89	77.78	99.47	97.64	46.93	95.48
PCA-LDA+IMJAYA-ELM	MC1	5	16	96.40	88.89	99.73	98.04	55.49	96.23
PCA-LDA+MLP	PC1	5	16	96.58	77.45	98.13	97.33	67.71	95.34
PCA-LDA+AB	PC1	5	16	95.96	77.62	98.16	97.05	64.56	94.73
PCA-LDA+J48	PC1	5	16	96.12	77.78	98.41	97.25	64.76	94.93
PCA-LDA+RF	PC1	5	16	95.59	77.96	99.08	97.09	62.05	94.64
PCA-LDA+NB	PC1	5	16	95.35	77.78	98.40	96.85	61.81	94.20
PCA-LDA+DT	PC1	5	16	96.90	77.78	98.43	97.66	68.08	95.65
PCA-LDA+ELM	PC1	5	16	96.90	88.89	99.21	98.04	75.17	96.38
PCA-LDA+IMJAYA-ELM	PC1	5	16	97.67	88.89	99.21	98.44	78.91	97.10
PCA-LDA+MLP	JM1	5	16	95.29	76.58	99.37	97.29	45.68	94.82
PCA-LDA+AB	JM1	5	16	94.9	75.68	99.34	97.07	43.78	94.42
PCA-LDA+J48	JM1	5	16	95.61	77.58	99.39	97.46	47.56	95.08
PCA-LDA+RF	JM1	5	16	95.42	73.09	99.27	97.31	44.3	94.85
PCA-LDA+NB	JM1	5	16	94.71	75.68	99.34	96.97	43.1	94.22
PCA-LDA+DT	JM1	5	16	96.13	80.72	99.48	97.78	51.64	95.76
PCA-LDA+ELM	JM1	5	16	96.41	85.2	99.6	97.98	55.57	96.13
PCA-LDA+IMJAYA-ELM	JM1	5	16	97.73	87.00	99.65	98.68	64.86	97.46



Results
Validation accuracy: N/A
Training finished: Reached final iteration

Training Time
Start time: 08-May-2024 11:42:08
Elapsed time: 254 min 18 sec

Training Cycle
Epoch: 200 of 200
Iteration: 80000 of 80000
Iterations per epoch: 400
Maximum iterations: 80000

Validation
Frequency: N/A
Patience: N/A

Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.0001

FIGURE 5. The performance graph during training of the proposed model across multiple iterations using the KC2 dataset.

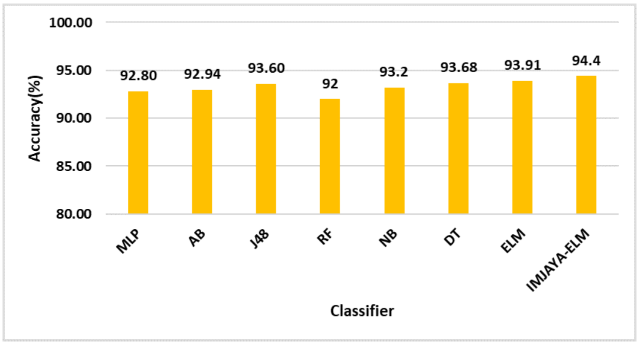


FIGURE 6. Defect prediction accuracy for CM1 dataset without feature reduction.

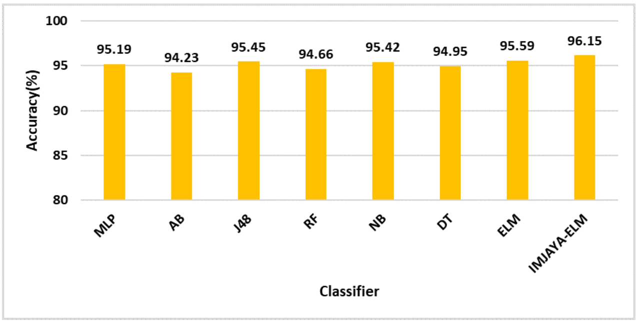


FIGURE 8. Defect prediction accuracy for KC2 dataset without feature reduction.

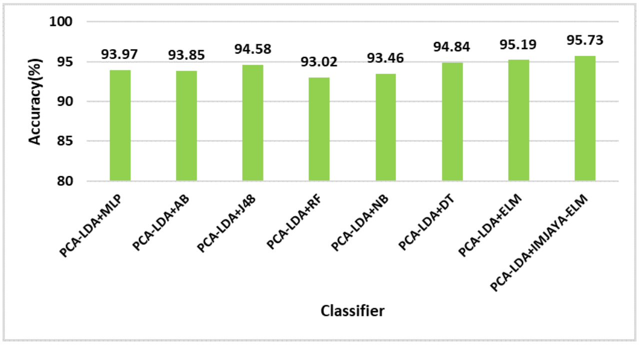


FIGURE 7. Defect prediction accuracy for CM1 dataset with feature reduction.

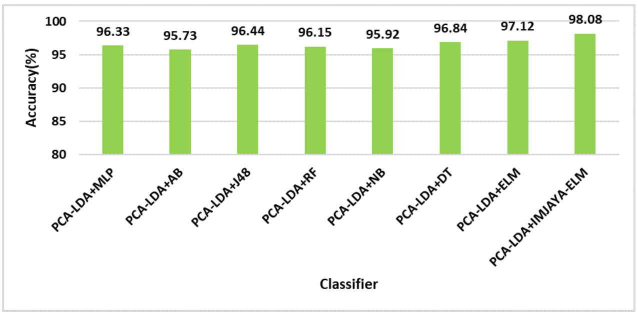


FIGURE 9. Defect prediction accuracy for KC2 dataset with feature reduction.

Figure 5 illustrates the training performance graph of the proposed model across multiple iterations for the KC2 dataset.

Graphs illustrating the performance of multiple ML classifiers, both with and without feature reduction, have been generated for each of the six datasets analyzed in this study. Specifically, the accuracy of ML classifiers without feature reduction for the datasets CM1, KC2, KC3, MC1, PC1, and JM1 is depicted in Figure 6, Figure 8, Figure 10, Figure 12, Figure 14, and Figure 16 respectively. Conversely, Figure 7, Figure 9, Figure 11, Figure 13, Figure 15, and Figure 17

TABLE 6. Average accuracy(%) of employed framework for CM1 dataset using 5-fold 10 times.

Run	Fd1	Fd2	Fd3	Fd4	Fd5	Avg(%)
1	98.47	95.39	95.39	98.47	95.39	96.62
2	96.92	93.85	93.85	96.92	93.85	95.08
3	95.39	95.39	95.39	95.39	95.39	95.39
4	93.85	93.85	93.85	93.85	93.85	93.85
5	98.47	95.39	96.92	98.47	98.47	97.54
6	96.92	98.47	95.39	96.92	98.47	97.23
7	95.39	96.92	95.39	95.39	96.92	96.00
8	93.85	95.39	93.85	93.85	95.39	94.47
9	95.39	93.85	95.39	9539	93.85	94.77
10	93.85	96.92	93.85	98.47	98.47	96.31
Avg(%)						95.73

TABLE 7. Average accuracy(%) of employed framework for KC2 dataset using 5-fold 10 times.

Run	Fd1	Fd2	Fd3	Fd4	Fd5	Avg(%)
1	97.12	96.15	98.08	99.04	99.04	97.89
2	98.08	99.04	98.08	99.04	99.04	98.66
3	96.15	99.04	97.12	98.08	99.04	97.89
4	97.12	98.08	99.04	98.08	97.12	97.89
5	96.15	99.04	99.04	99.04	99.04	98.46
6	99.04	99.04	97.12	99.04	96.15	98.08
7	97.12	99.04	98.08	98.08	98.08	98.08
8	97.12	98.08	99.04	97.12	99.04	98.08
9	98.08	97.12	99.04	96.15	96.15	97.31
10	99.04	96.15	99.04	99.04	99.04	98.46
Avg(%)						98.08

TABLE 8. Average accuracy(%) of employed framework for KC3 dataset using 5-fold 10 times.

Run	Fd1	Fd2	Fd3	Fd4	Fd5	Avg(%)
1	89.74	94.87	94.87	97.44	89.74	93.33
2	94.87	94.87	94.87	97.44	94.87	95.38
3	97.44	92.31	92.31	97.44	97.44	95.39
4	94.87	92.31	94.87	97.44	94.87	94.87
5	97.44	92.31	97.44	94.87	97.44	95.90
6	94.87	89.74	94.87	97.44	94.87	94.36
7	94.87	89.74	97.44	94.87	92.31	93.85
8	94.87	94.87	89.74	89.74	94.87	92.82
9	97.44	94.87	97.44	97.44	97.44	96.93
10	97.44	89.74	97.44	97.44	97.44	95.90
Avg(%)						94.87

show the accuracy with reduced features for these datasets. Figure 18 presents a comparative analysis of accuracy across the different datasets examined in this study.

TABLE 9. Average accuracy(%) of employed framework for MC1 dataset using 5-fold 10 times.

Run	Fd1	Fd2	Fd3	Fd4	Fd5	Avg(%)
1	96.48	96.23	96.23	96.48	96.48	96.38
2	96.23	96.48	96.23	96.23	96.48	96.33
3	96.48	96.23	96.48	96.48	96.23	96.38
4	96.48	95.98	96.23	95.73	96.48	96.18
5	96.48	95.73	96.48	96.23	95.73	96.13
6	96.48	96.23	96.48	95.98	96.23	96.28
7	96.23	96.48	96.23	96.48	96.48	96.38
8	95.98	96.23	96.48	96.23	96.23	96.23
9	95.73	95.98	95.73	95.98	95.98	95.88
10	96.23	95.73	96.23	95.73	96.48	96.08
Avg(%)						96.23

TABLE 10. Average accuracy(%) of employed framework for PC1 dataset using 5-fold 10 times.

Run	Fd1	Fd2	Fd3	Fd4	Fd5	Avg(%)
1	96.45	97.16	97.16	97.87	97.87	97.30
2	97.16	97.87	97.16	97.87	96.45	97.30
3	97.87	97.16	97.87	97.87	95.74	97.30
4	97.87	96.45	97.87	97.87	97.87	97.59
5	96.45	95.74	96.45	97.16	97.87	96.73
6	96.45	97.16	95.74	96.45	97.16	96.59
7	95.74	96.45	97.16	97.87	96.45	96.73
8	97.16	95.74	97.87	97.16	97.87	97.16
9	97.87	97.16	95.74	97.87	97.87	97.30
10	97.87	96.45	97.87	95.74	97.16	97.02
Avg(%)						97.10

TABLE 11. Average accuracy(%) of employed framework for JM1 dataset using 5-fold 10 times.

Run	Fd1	Fd2	Fd3	Fd4	Fd5	Avg(%)
1	96.77	97.54	97.54	97.99	97.99	97.57
2	97.54	97.99	97.54	97.99	97.99	97.81
3	97.54	97.54	97.99	97.54	95.74	97.27
4	97.99	96.77	97.99	97.99	97.99	97.75
5	97.54	95.74	96.77	97.54	97.99	97.12
6	97.99	97.54	95.74	96.77	97.54	97.12
7	95.74	97.99	97.54	97.99	96.77	97.21
8	97.54	95.74	97.99	97.54	97.99	97.36
9	97.99	97.54	95.74	97.99	97.99	97.45
10	97.99	96.77	97.99	95.74	97.54	97.99
Avg(%)						97.46

The Figure 19, Figure 20, Figure 21, Figure 22, Figure 23, and Figure 24 illustrates the confusion matrices for CM1, KC2, KC3, MC1, PC1, and JM1 respectively.

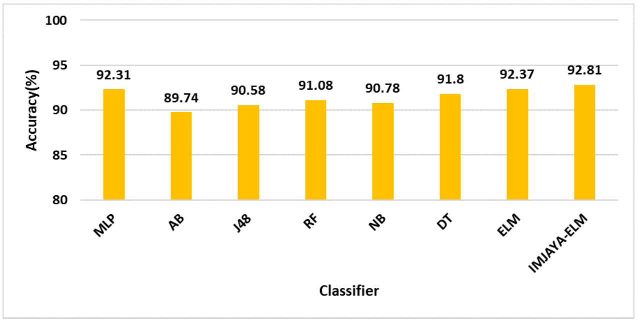


FIGURE 10. Defect prediction accuracy for KC3 dataset without feature reduction.

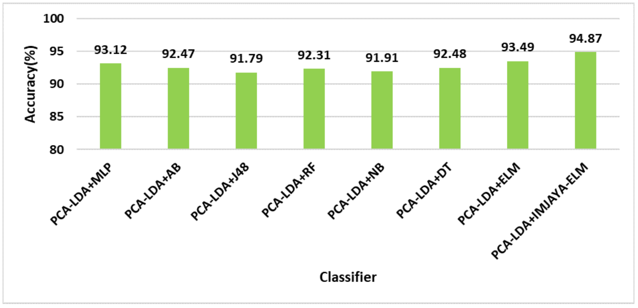


FIGURE 11. Defect prediction accuracy for KC3 dataset with feature reduction.

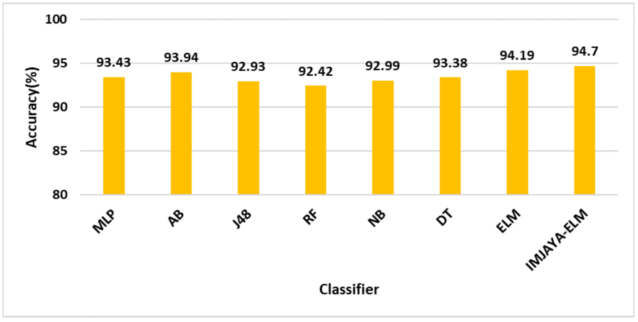


FIGURE 12. Defect prediction accuracy for MC1 dataset without feature reduction.

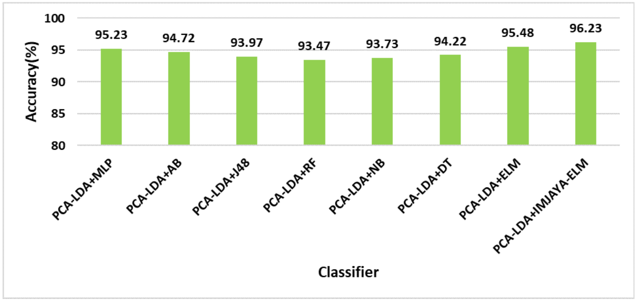


FIGURE 13. Defect prediction accuracy for MC1 dataset with feature reduction.

Figure 25 presents a chart showing the relationship between the number of features and accuracy using PCA and PCA-LDA.

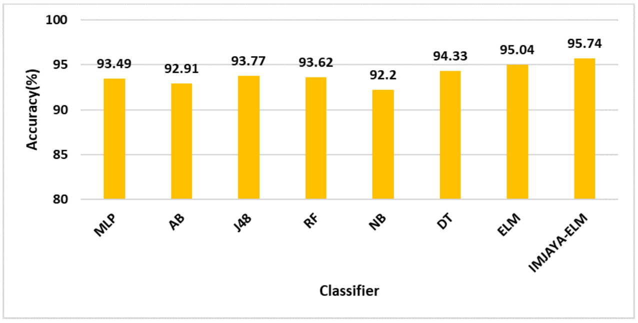


FIGURE 14. Defect prediction accuracy for PC1 dataset without feature reduction.

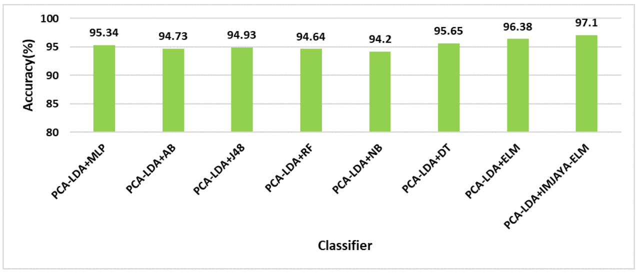


FIGURE 15. Defect prediction accuracy for PC1 dataset with feature reduction.

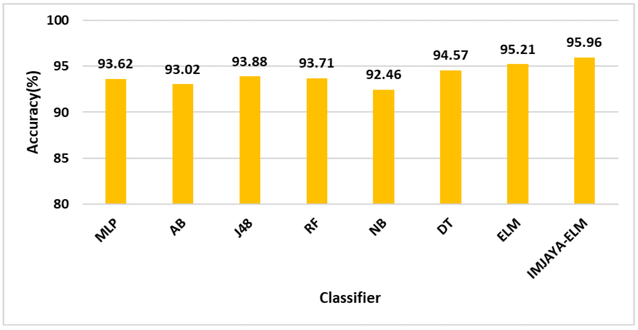


FIGURE 16. Defect prediction accuracy for JM1 dataset without feature reduction.

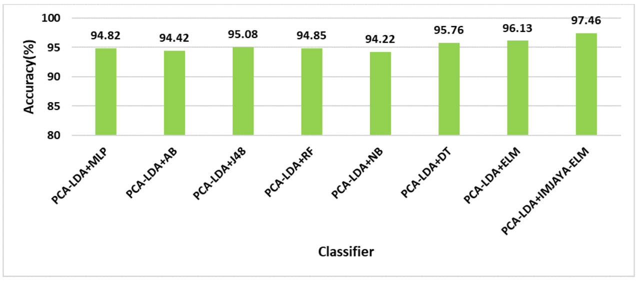


FIGURE 17. Defect prediction accuracy for JM1 dataset with feature reduction.

Table 12 represents the comparison of accuracy for previously existing methodologies with the proposed methodology in which the proposed methodology outperformed with an accuracy of 98.08% for PCA-LDA+IMJAYA-ELM.

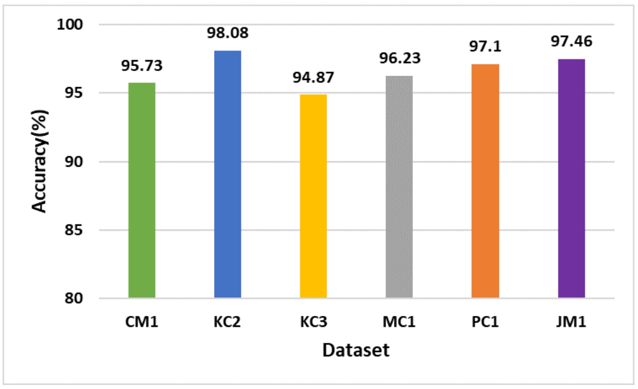


FIGURE 18. Comparision of accuracy for datasets with feature reduction.

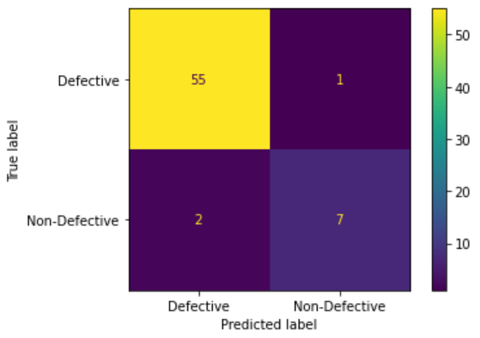


FIGURE 19. Confusion matrices for CM1 dataset with PCA-LDA.

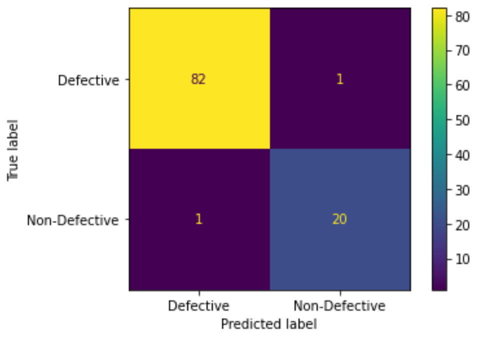


FIGURE 20. Confusion matrices for KC2 dataset with PCA-LDA.

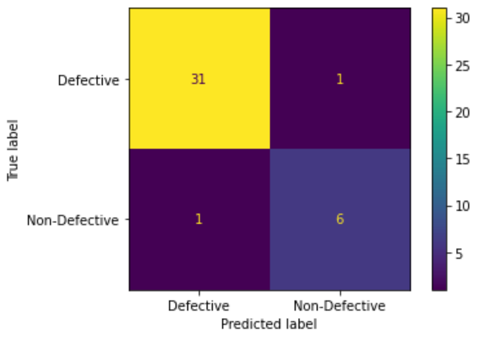


FIGURE 21. Confusion matrices for KC3 dataset with PCA-LDA.

V. DISCUSSION

This work presents and assesses the enhanced Jaya method for software defect prediction alongside an Extreme Learning Machine (ELM) using the CM1, KC2, KC3, MC1, PC1, and JM1 datasets. The approach, incorporating PCA-LDA for feature dimensionality reduction, significantly improved prediction accuracy, reaching the highest accuracy of 98.08% on the KC2 dataset. Notably, the PCA-LDA reduced the feature

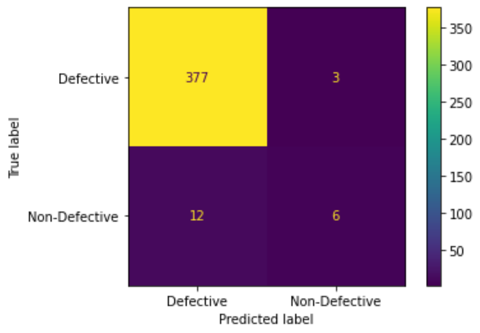


FIGURE 22. Confusion matrices for MC1 dataset with PCA-LDA.

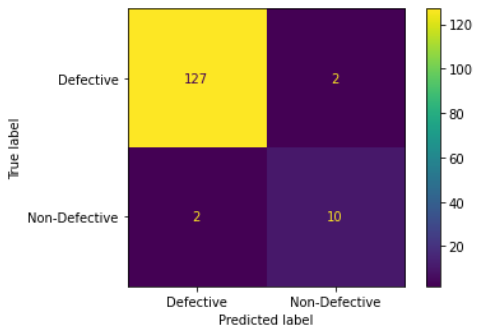


FIGURE 23. Confusion matrices for PC1 dataset with PCA-LDA.

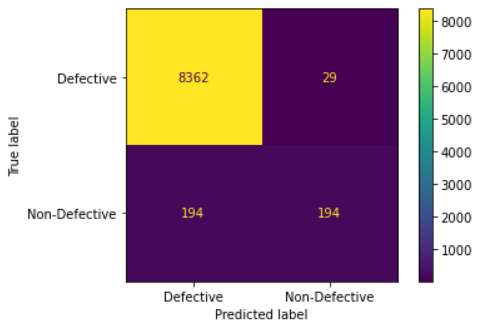


FIGURE 24. Confusion matrices for JM1 dataset with PCA-LDA.

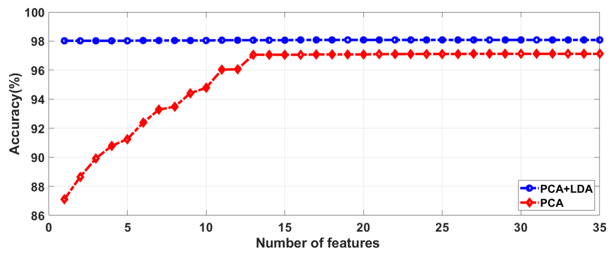


FIGURE 25. No of Features Vs Accuracy.

TABLE 12. Comparison of the utilized model with alternative models.

References	Methodologies	Acc(%)
Mehta et al. [1]	XGBoost + Stacking	90.00
Kassaymeh et al. [14]	SSA+BPNN	97.03
Mustaqeem et al. [54]	SVM+PCA	95.20
Eldho et al. [55]	SVM+ELM	89.00
Wahono et al. [11]	PSO+Bagging	91.52
Proposed Methodology	**PCA-LDA+IMJAYA-ELM**	**98.08**

set from 22 to 16, highlighting its effectiveness in minimizing data dimensionality while preserving predictive performance. This approach's primary advantage is its ability to achieve high accuracy with a reduced feature set, resulting in more efficient models with lower computational costs. However, a potential disadvantage is the dependency on specific datasets, which may limit the generalizability of the model to other datasets not included in this study. Future work could explore the application of the improved Jaya algorithm with other feature reduction techniques and datasets to further validate its robustness and adaptability across diverse software defect prediction scenarios. Additionally, integrating ensemble methods could enhance predictive performance further.

VI. CONCLUSION

To make software systems more accurate and efficient, researchers are always looking for ways to improve defect prediction algorithms. This research will help software projects save time and money by identifying potential problems before formal testing begins. Feature normalization, dimensional reduction, and classification are all part of the strategy used in this study. Dimensionality reduction of features is achieved through PCA-LDA in this model. For optimizing the parameters of the ELM, which include the input weights and biases, an improved JAYA optimization (IMJAYA) strategy is utilized during the forecasting phase. This hybrid approach is utilized to optimize these parameters. The comprehensive model PCA-LDA+IMJAYA-ELM demonstrates superior performance compared to earlier models using NASA's KC2 dataset, particularly with fewer features and achieves higher accuracy levels. Future studies could investigate other hybrid models to enhance the performance and eliminate software system defect prediction errors. Developing ensemble approaches that use deep-learning techniques to improve the model's prediction accuracy is another potential area of focus.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA for funding this research work through the project number "NBU-FFR-2024-1060-02."

REFERENCES

[1] S. Mehta and K. S. Patnaik, "Improved prediction of software defects using ensemble machine learning techniques," *Neural Comput. Appl.*, vol. 33, no. 16, pp. 10551–10562, Aug. 2021.
[2] S. Santhosh, K. Khatter, and D. Relan, "Software fault prediction using particle swarm optimization and random forest," in *Proc. Int. Conf. Data Sci. Appl. (ICDSA)*, vol. 1. Singapore: Springer, Feb. 2023, pp. 843–851.
[3] D. R. Nayak, R. Dash, and B. Majhi, "Development of pathological brain detection system using Jaya optimized improved extreme learning machine and orthogonal ripplet-II transform," *Multimedia Tools Appl.*, vol. 77, no. 17, pp. 22705–22733, Sep. 2018.
[4] M. F. I. Khan and A. K. M. Masum, "Predictive analytics and machine learning for real-time detection of software defects and agile test management," *Educ. Admin., Theory Pract.*, vol. 30, no. 4, pp. 1051–1057, 2024.
[5] M. S. Saeed, "Role of feature selection in cross project software defect prediction—A review," *Int. J. Comput., Inf. Manuf.*, vol. 3, no. 2, pp. 37–56, 2023.
[6] H. C. Sharma, A. Kero, M. Arora, V. Sharma, G. Makkar, and P. Semwal, "An isolation forest algorithm for the detection of effective software defects estimation," *Power Syst. Technol.*, vol. 48, no. 1, pp. 646–658, 2024.
[7] K. Holzinger, V. Palade, R. Rabadan, and A. Holzinger, "Darwin or Lamarck? Future challenges in evolutionary algorithms for knowledge discovery and data mining," in *Interactive Knowledge Discovery and Data Mining in Biomedical Informatics: State-of-the-Art and Future Challenges*. Berlin, Germany: Springer, 2014, pp. 35–56.
[8] K. Alrashedy, V. J. Hellendoorn, and A. Orso, "Learning defect prediction from unrealistic data," in *Proc. IEEE Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2024, pp. 556–567.
[9] B. Salve, P. Pokharkar, S. Pillai, and R. Kapse, "A novel approach for software defect prediction based on dimensionality reduction," *Int. Res. J. Eng. Technol.*, vol. 7, no. 3, pp. 4392–4400, Mar. 2020.
[10] A. Abdu, Z. Zhai, H. A. Abdo, R. Algabri, M. A. Al-Masni, M. S. Muhammad, and Y. H. Gu, "Semantic and traditional feature fusion for software defect prediction using hybrid deep learning model," *Sci. Rep.*, vol. 14, no. 1, p. 14771, Jul. 2024.
[11] R. S. Wahono and N. Suryana, "Combining particle swarm optimization based feature selection and bagging technique for software defect prediction," *Int. J. Softw. Eng. Appl.*, vol. 7, no. 5, pp. 153–166, Sep. 2013.
[12] P. R. Bal and S. Kumar, "WR-ELM: Weighted regularization extreme learning machine for imbalance learning in software fault prediction," *IEEE Trans. Rel.*, vol. 69, no. 4, pp. 1355–1375, Dec. 2020.
[13] X. Yu, J. Rao, L. Liu, G. Lin, W. Hu, J. W. Keung, J. Zhou, and J. Xiang, "Improving effort-aware defect prediction by directly learning to rank software modules," *Inf. Softw. Technol.*, vol. 165, Jan. 2024, Art. no. 107250.
[14] S. Kassaymeh, S. Abdullah, M. A. Al-Betar, and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem," *J. King Saud Univ., Comput. Inf. Sci.*, vol. 34, no. 6, pp. 3365–3378, Jun. 2022.
[15] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, Apr. 2020.
[16] F. Matloob, T. M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M. A. Khan, S. Abbas, and T. R. Soomro, "Software defect prediction using ensemble learning: A systematic literature review," *IEEE Access*, vol. 9, pp. 98754–98771, 2021.
[17] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Inf. Softw. Technol.*, vol. 122, Jun. 2020, Art. no. 106287.
[18] M. Prashanthi and M. Chandra Mohan, "Hybrid optimization-based neural network classifier for software defect prediction," *Int. J. Image Graph.*, vol. 24, no. 4, Jul. 2024, Art. no. 2450045.
[19] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *J. Syst. Softw.*, vol. 195, Jan. 2023, Art. no. 111537.
[20] M. Azzeh, Y. Elsheikh, A. B. Nassif, and L. Angelis, "Examining the performance of kernel methods for software defect prediction based on support vector machine," *Sci. Comput. Program.*, vol. 226, Mar. 2023, Art. no. 102916.
[21] M. Mafarja, T. Thaher, M. A. Al-Betar, J. Too, M. A. Awadallah, I. A. Doush, and H. Turabieh, "Classification framework for faulty-software using enhanced exploratory whale optimizer-based feature selection scheme and random forest ensemble learning," *Appl. Intell.*, vol. 53, no. 15, pp. 18715–18757, Feb. 2023.

[22] L. Yang, Z. Li, D. Wang, H. Miao, and Z. Wang, "Software defects prediction based on hybrid particle swarm optimization and sparrow search algorithm," *IEEE Access*, vol. 9, pp. 60865–60879, 2021.

[23] H. Das, S. Prajapati, M. K. Gourisaria, R. M. Pattanayak, A. Alameen, and M. Kolhar, "Feature selection using golden jackal optimization for software fault prediction," *Mathematics*, vol. 11, no. 11, p. 2438, May 2023.

[24] A. Ghaedi, A. K. Bardsiri, and M. J. Shahbazzadeh, "A software defect prediction approach by cat hunting optimization (CHO) algorithm and deep learning based on the GAN-VGG16 networks," SSRN Electron. Library, Rep., Amsterdam, The Netherlands, Rep., 2023.

[25] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, "Boosted whale optimization algorithm with natural selection operators for software fault prediction," *IEEE Access*, vol. 9, pp. 14239–14258, 2021, doi: 10.1109/ACCESS.2021.3052149.

[26] J. Lee, J. Choi, D. Ryu, and S. Kim, "Holistic parameter optimization for software defect prediction," *IEEE Access*, vol. 10, pp. 106781–106797, 2022.

[27] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[28] T. Siddiqui and M. Mustaqeem, "Performance evaluation of software defect prediction with NASA dataset using machine learning techniques," *Int. J. Inf. Technol.*, vol. 15, no. 8, pp. 4131–4139, Dec. 2023.

[29] N. A. A. Khleel and K. Nehéz, "Software defect prediction using a bidirectional LSTM network combined with oversampling techniques," *Cluster Comput.*, vol. 27, no. 3, pp. 3615–3638, Jun. 2024.

[30] A. M. Ismail, S. H. A. Hamid, A. A. Sani, and N. N. M. Daud, "Toward reduction in false positives just-in-time software defect prediction using deep reinforcement learning," *IEEE Access*, vol. 12, pp. 47568–47580, 2024.

[31] R. Mao, L. Zhang, and X. Zhang, "Mutation-based data augmentation for software defect prediction," *J. Softw., Evol. Process*, vol. 36, no. 6, p. 2634, Jun. 2024.

[32] B. P. Rai, C. Raghuvanshi, and A. K. Singh, "Prediction of software defect using feature extraction technique: A study," *NeuroQuantology*, vol. 20, no. 14, p. 2479, 2022.

[33] Rajvardhan, K. Anand, and A. K. Jena, "Forecasting software modules with known defects through the quadratic discriminant analysis feature reduction technique," in *Proc. Int. Conf. Emerg. Syst. Intell. Comput. (ESIC)*, Bhubaneswar, India, 2024, pp. 713–717, doi: 10.1109/ESIC60604.2024.10481542.

[34] S. Feng, J. Keung, Y. Xiao, P. Zhang, X. Yu, and X. Cao, "Improving the undersampling technique by optimizing the termination condition for software defect prediction," *Expert Syst. Appl.*, vol. 235, Jan. 2024, Art. no. 121084.

[35] S. Mallik, D. Pradhan, D. Muduli, A. Rath, G. Panda, S. Dash, and H. Qin, "A novel approach to enhance software defect prediction using an improved grey wolf optimization based extreme learning machine technique," Sci. Rep., London, U.K., 2024, doi: 10.21203/rs.3.rs-4110665/v1.

[36] S. Ali, M. Hassan, J. Y. Kim, M. I. Farid, M. Sanaullah, and H. Mufti, "FF-PCA-LDA: Intelligent feature fusion based PCA-LDA classification system for plant leaf diseases," *Appl. Sci.*, vol. 12, no. 7, p. 3514, Mar. 2022.

[37] Q. Zhang, J. Zhang, T. Feng, J. Xue, X. Zhu, N. Zhu, and Z. Li, "Software defect prediction using deep Q-learning network-based feature extraction," *IET Softw.*, vol. 2024, pp. 1–34, May 2024.

[38] C. L. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," in *Proc. 4th Int. Conf. Trends Electron. Informat. (ICOEI)*, Jun. 2020, pp. 728–733.

[39] I. Arora and A. Saha, "ELM and KELM based software defect prediction using feature selection techniques," *J. Inf. Optim. Sci.*, vol. 40, no. 5, pp. 1025–1045, Jul. 2019.

[40] S. K. Pandey, D. Rathee, and A. K. Tripathi, "Software defect prediction using K-PCA and various kernel-based extreme learning machine: An empirical study," *IET Softw.*, vol. 14, no. 7, pp. 768–782, Dec. 2020.

[41] P. R. Bal, "Cross project software defect prediction using extreme learning machine: An ensemble based study," in *Proc. ICSOFT*, 2018, pp. 354–361.

[42] M. S. Saeed, "Cross project software defect prediction using ensemble learning, a comprehensive review," *Int. J. Comput. Innov. Sci.*, vol. 3, no. 2, pp. 34–42, 2024.

[43] S. Zheng, J. Gai, H. Yu, H. Zou, and S. Gao, "Software defect prediction based on fuzzy weighted extreme learning machine with relative density information," *Sci. Program.*, vol. 2020, pp. 1–18, Nov. 2020.

[44] K. Thirumoorthy and J. J. J. Britto, "A feature selection model for software defect prediction using binary Rao optimization algorithm," *Appl. Soft Comput.*, vol. 131, Dec. 2022, Art. no. 109737.

[45] M. Mustaqeem, S. Mustajab, and M. Alam, "A hybrid approach for optimizing software defect prediction using a grey wolf optimization and multilayer perceptron," *Int. J. Intell. Comput. Cybern.*, vol. 17, no. 2, pp. 436–464, May 2024.

[46] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Defect prediction as a multiobjective optimization problem," *Softw. Test., Verification Rel.*, vol. 25, no. 4, pp. 426–459, Jun. 2015.

[47] K. Thirumoorthy and J. J. J. Britto, "A clustering approach for software defect prediction using hybrid social mimic optimization algorithm," *Computing*, vol. 104, no. 12, pp. 2605–2633, Dec. 2022.

[48] R. Malhotra and K. Khan, "A novel software defect prediction model using two-phase grey wolf optimisation for feature selection," *Cluster Comput.*, vol. 27, pp. 1–23, 2024.

[49] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, "Evolutionary extreme learning machine," *Pattern Recognit.*, vol. 38, no. 10, pp. 1759–1763, Oct. 2005.

[50] G. Airlangga, "Integrating LightGBM and XGBoost for software defect classification problem," *Jurnal Media Informatika Budidarma*, vol. 8, no. 1, pp. 430–440, Jan. 2024.

[51] D. Pradhan and D. Muduli, "Software defect prediction model using AdaBoost based random forest technique," in *Proc. 14th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Jul. 2023, pp. 1–6.

[52] M. A. Mabayoje, A. O. Balogun, A. O. Bajeh, and B. A. Musa, "Software defect prediction: Effect of feature selection and ensemble methods," *FUW Trends Sci. Technol. J.*, vol. 3, no. 2, pp. 518–522, 2018.

[53] X. Cai, Y. Niu, S. Geng, J. Zhang, Z. Cui, J. Li, and J. Chen, "An undersampled software defect prediction method based on hybrid multi-objective cuckoo search," *Concurrency Comput., Pract. Exp.*, vol. 32, no. 5, Mar. 2020, Art. no. e5478.

[54] M. Mustaqeem and M. Saqib, "Principal component based support vector machine (PC-SVM): A hybrid technique for software defect detection," *Cluster Comput.*, vol. 24, no. 3, pp. 2581–2595, Sep. 2021.

[55] K. J. Eldho, "Impact of unbalanced classification on the performance of software defect prediction models," *Indian J. Sci. Technol.*, vol. 15, no. 6, pp. 237–242, Feb. 2022.

DEBASISH PRADHAN received the B.Tech. degree in information technology and the M.Tech. degree in computer science and engineering from the Biju Pattanaik University of Technology, Rourkela, Odisha, India, in 2010 and 2013, respectively. He is currently pursuing the Ph.D. degree in computer science and engineering with C. V. Raman Global University, Bhubaneswar, Odisha. He has more than ten years of teaching experience as a Senior Lecturer with the Department of Computer Science and Engineering, since 2021, in various private engineering colleges in Bhubaneswar. His research interests include software defect prediction, object slicing, and the cost estimation of software projects.

DEBENDRA MUDULI (Member, IEEE) received the M.Tech. degree in computer science and engineering and the Ph.D. degree in computer science from the National Institute of Technology, Rourkela, in 2016 and 2022, respectively. He is currently an Associate Professor with the Department of Computer Science and Engineering, C. V. Raman Global University, Bhubaneswar, India. His work has been published in numerous journals and international conferences. His research interests include cloud computing, pattern recognition, and medical image processing. He is a Life Member of the Indian Society for Technical Education (ISTE). He is actively involved in the Academic Community. He is serving as a reviewer for various reputable publications.

ABU TAHA ZAMANI (Member, IEEE) has been a Lecturer with the Department of Computer Science, Faculty of Science, Northern Border University, Arar, Saudi Arabia. He has several research articles in reputed international journals. His research interests include cloud computing, ad-hoc networks, cyber security, AI, the IoT, machine learning, and data science. He is a member of various international journals, like ACM. He is an editorial board member of some reputed international journals in computer sciences. He serves as a reviewer for various journals.

SYED IRFAN YAQOOB received the bachelor's degree in IT, and the M.Tech. and Ph.D. degree in computer science engineering. He focused on cloud computing and its development. He is an Associate Professor at Chandigarh University, Mohali, Punjab, Computer Science and Engineering Department (AIT), passionate about teaching, researching, and developing innovative solutions in the field of Computer Science Engineering. He has published multiple papers in reputed journals and conferences, and received grants and awards for research projects, also mentor students and guide them in their academic and professional pursuits. He has more than nine years of teaching experience in different colleges and Universities, He has acquired skills in cloud computing, machine learning, the IoT, software engineering, and programming languages such as C, C++, and Java.

SULTAN M. ALANAZI received the master's degree in IT and the Ph.D. degree in computer science from the University of Nottingham, U.K. He was a Teaching Assistant with the University of Nottingham. He is currently an Assistant Professor and the Head of the Department of Computer Science, Northern Border University, Saudi Arabia. With extensive experience in information technology, he has published several research papers in esteemed international journals and conferences. His research interests include AI, machine learning, natural language processing, cybersecurity, and recommender systems.

RAKESH RANJAN KUMAR (Member, IEEE) received the M.Tech. degree from MNNIT, Allahabad, India, and the Ph.D. degree from IIT (ISM), Dhanbad, India. He is currently an Assistant Professor with the Department of CSE, C. V. Raman Global University, Bhubaneswar, Odisha, India. He has published more than 25 papers in reputed journals and conferences. His current research interests include cloud computing, DevOps, and optimization techniques. He acted as a reviewers in many reputed journals and conferences.

NIKHAT PARVEEN received the B.Sc. and M.C.A. degrees in computer science from Andhra University, Andhra Pradesh, India, in 2000 and 2003, respectively, and the Ph.D. degree from the Department of Computer Application, Integral University, Lucknow, Uttar Pradesh, India. She is currently an Associate Professor with the Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Guntur, India. She has more than 12 years of teaching experience and six years of research experience. She has six national patents. Her research interests include artificial intelligence, machine learning, security software, security testing, software engineering, and requirement engineering. She is also working in the areas of soft computing, image analysis, big data analytics, and the IoT. Her research has been chronicled in over 30 journal publications and international conferences. She is a Life Time Member of CSI, ACM, IAENG, and IACSIT.

MOHAMMAD SHAMEEM received the Ph.D. degree from the Indian Institute of Technology (Indian School of Mines), Dhanbad. Currently, he is working as a Post-Doctoral Researcher with the Interdisciplinary Research Centre for Intelligent Secure Systems, King Fahd university of petroleum and Minerals, Dhahran, Saudi Arabia. His research interests include agile software development, empirical software engineering, and global software engineering. He has published various papers in well-reputed SCI and Scopus journals, such as *Journal of Software: Evolution and Process* (Wiley), *Applied Soft Computing* (Elsevier), and *Arabian Journal for Science and Engineering* (Springer). Moreover, he has presented his research articles.