A New Method for Software Defect Prediction Based on Optimized Machine Learning Techniques

SHAHO HASSEN [1], [1] Graduate School of Natural and Applied Sciences, Department of Computer Engineering, Atilim University, Ankara, Turkey. shaho.hassen@su.edu.krd

Prof. Dr. Ali YAZICI [2], [2] Department of Computer Engineering, Atilim University, Ankara, Turkey. ali.yazici@atilim.edu.tr

Prof. Dr. Alok MISHRA [3] [3] Department of Computer Engineering, Atilim University, Ankara, Turkey. profalokmishra@gmail.com

ABSTRACT

Objective: This thesis aimed to develop a robust neuro-computing model for software defect prediction, utilizing the Levenberg Marquardt Neural Network (LM-ANN) with a novel improved genetic algorithm as a heuristic model. The goal was to achieve higher accuracy and overcome local minima and convergence issues.

Theoretical framework: The LM-ANN was chosen for its adaptive learning capabilities in non-linear feature learning from defect data, but it faced challenges due to high weight estimation for 17 input features.

Method: To address local minima and convergence problems, an improved genetic algorithm heuristic model was developed to assist the LM-ANN in adaptive weight estimation and updates during learning.

Results and conclusion: The integration of the heuristic model with LM-ANN led to superior accuracy compared to classical neural networks on software fault datasets, making it a promising approach for defect prediction.

Implications of the research: The research presents a robust neuro-computing solution with improved adaptability, providing practical benefits for software defect prediction and addressing challenges in the field.

Originality/value: This work introduces a novel heuristic-driven neuro-computing model, combining the strengths of LM-ANN and the genetic algorithm. The focus on feature engineering further enhances its effectiveness in addressing class imbalance, overfitting, and convergence problems. The research contributes to advancing defect prediction methods in software engineering.

Keywords: Neuro-Computing, LM-ANN, Defect Prediction, Genetic Algorithm, ANN, Local Minima, HNC-SDP, and Levenberg Marquardt.

1. Introduction

Over the past few decades, software technologies have become an indispensable part of modern human society, driving constant innovation and advancement in the software industry [1, 2]. Global competitiveness and the need for efficient software solutions have prompted developers and enterprises to explore software reuse paradigms, such as Free-Open-Source Software (FOSS) components and function-reuse concepts, to reduce development cycle delays and costs [2, 3]. While software reuse offers cost-efficiency, it also introduces challenges such as aging, code smells, and faults due to excessive reliance on reused components [4-9]. These factors can lead to software malfunctions and impact overall reliability [2, 9].

In critical domains like finance, healthcare, defense, and industrial control, software reliability is paramount, and any compromise in this aspect could have severe consequences [9, 10]. Thus, achieving software reliability without compromising cost-effectiveness has become a pressing concern. Software defect detection and prediction have emerged as essential practices to address this demand [9]. However, the complexity of software designs and diverse development paradigms make manual fault detection in large and complex software systems infeasible. Manual testing not only consumes substantial resources and time but also carries the risk of human errors and misjudgments [6].

To tackle these challenges and optimize defect prediction, automated reusability prediction approaches have been proposed, utilizing software metrics and machine learning methods [11-13]. While some efforts have been made to predict defects in individual software classes or functions using software metrics, the optimality of the employed machine learning classifiers has remained questionable. Challenges like local minima, convergence limitations, and the optimal selection of software metrics for defect prediction hinder the development of computationally efficient solutions.

In the existing body of research, numerous software defect prediction (SDP) systems driven by machine learning tend to emphasize code complexity metrics like Lines of Code (LOC) and Depth of Inheritance Tree (DIT). However, they often neglect the inherent relationships between classes or components, such as cohesion and coupling. Overlooking these critical aspects can result in less accurate predictions since excessive reuse of different classes can also contribute to software faults [8, 9, 11, 12].

Furthermore, an unexplored issue in previous research is the challenge of class imbalance in defect prediction. The probability of a faulty class or defect occurrence is typically lower than that of normal classes, leading classical machine learning methods to exhibit false positives (favoring the majority class) under imbalanced data conditions, especially during local minima and premature convergence.

This research paper introduces a novel and robust machine learning model for software defect prediction, aiming to overcome the limitations and challenges of existing approaches. To achieve this, the model utilizes Object-Oriented Programming (OOP) metrics, particularly CKJM (Chidamber and Kemerer Java Metrics), to facilitate two-class classification. The proposed method involves data resampling through Synthetic Minority Over Sampling (SMOTE), followed by Min-Max normalization and heuristic-driven neuro-computing techniques. This comprehensive approach aims to improve the accuracy of defect prediction in software systems.

In this article, we present a detailed description of our proposed software defect prediction model, along with its performance evaluation using various NASA PROMISE datasets. The results demonstrate the model's superiority in terms of accuracy, precision, recall, and F-score, making it a promising solution for reliable and cost-effective software defect prediction. The subsequent sections delve into the architecture, methodology, and experimental findings of our heuristic-driven neuro-computing model for software defect prediction.

2. Literature Survey

Software defect prediction is a critical aspect of ensuring software reliability and quality. Over the years, researchers have explored various machine learning approaches for effective defect prediction. This literature survey provides an in-depth analysis of significant research papers in this domain, focusing on data mining-based methods, regression-based methods, neuro-computing-based approaches, genetic algorithms, decision tree-based techniques, association rule mining, and Bayesian neural networks.

Liu et al. [14] proposed a generic multi-data training and validation model for fault classification, utilizing historical software metrics to improve defect prediction. Song et al. [15] employed association rule mining to enhance defect prediction accuracy in over 200 projects compared to conventional techniques.

Lessmann et al. [16] conducted an evaluation of 22 classifiers using NASA Metric datasets to determine their performance.

Munson et al. [17] explored the efficiency of discriminating analysis using PCA to minimize complexity metrics for defect prediction. Tom [18] utilized the Naïve Bayesian algorithm for fault classification, achieving better results based on conditional independence hypothesis.

Ohlsson et al. [19] employed a genetic algorithm for fault detection in telecommunication software modules. Riquelme et al. [20] used a genetic algorithm to predict defects, while Catal et al. [21] proposed an SDP system based on Artificial Immune System.

Drown et al. [32] employed evolutionary sampling for enhanced software quality assurance and reliability. Chen et al. [22] designed a defect prediction system using data mining techniques, specifically employing Bayesian Network models. Wang et al. [23] investigated defect prediction by employing the C4.5 mining algorithm and leveraging Spearman's rank correlation coefficient in their study.

Qinbao et al. [15] used association rule mining for defect prediction and correction with higher accuracy. Biwen et al. [24] proposed a C4.5 decision tree algorithm-based system with k-medoids clustering for improved fault prediction. Marwala [25] utilized Bayesian neural networks for fault detection in structures.

Various studies emphasized the significance of CK-Metrics, object-oriented metrics, and UML diagrams for defect prediction [26-32]. These metrics provide valuable insights into software features and help in identifying defects effectively.

In conclusion, This literature survey provides an overview of significant research in the field of software defect prediction. Various machine learning methods, such as decision trees, regression, and neuro-computing, were thoroughly examined for their effectiveness in defect prediction. Among these approaches, neuro-computing demonstrated higher efficacy; however, none of the existing methods adequately addressed issues related to class-imbalance, convergence, and local minima in classical machine learning.

The survey also highlighted the potential benefits of employing evolutionary computing algorithms like genetic algorithms and AIS to enhance data in defect prediction. Interestingly, while these algorithms have not yet been applied to boost the performance of machine learning methods for defect prediction, they hold promise for future improvements.

Based on these findings, the dissertation proposes a novel heuristic-driven neuro-computing approach for software defect prediction, utilizing OOP-CK metrics as benchmark datasets.

3. Material And Methods

In this article, we present an innovative approach known as the uristic-driven neurocomputing model for software defect prediction. The model is designed to effectively predict software defects by incorporating various stages, including data preparation, resampling techniques, and min-max normalization. By combining these essential steps, we create a robust framework that leverages the power of neurocomputing to enhance software defect prediction accuracy. Throughout this article, we will delve into the details of each stage, illustrating how they contribute to the overall effectiveness of our proposed model in software defect prediction.

3.1 Heuristic Driven Neuro-Computing Model for Software Defect Prediction

we present a comprehensive exploration of the processes involved in software defect prediction (SDP) tasks. We will delve into key procedures, such as data acquisition and processing, feature selection, resampling, and normalization. These fundamental steps set the foundation for our proposed heuristic-driven neuro-computing (HNC) algorithm, which we will thoroughly discuss and demonstrate its significance in the context of software defect prediction. Through this article, readers will gain valuable insights into the intricacies of SDP and the novel approach we have developed to enhance its accuracy and effectiveness.

3.1.1 Data Acquisition and Pre-processing

In our study, we evaluated the performance of our software defect prediction model by using standard benchmark datasets sourced from the NASA PROMISE archive. These datasets, including Ant1.7, Camel1.6, IVY, and JEdit, were collected from different software components using advanced mining techniques like Chidamber and Kamerer Java Virtual Machine (CKJM). The CKJM tool allowed us to extract 22 software metrics from the software, which were based on Object-Oriented Programming (OOP) principles.

However, we recognized that not all software metrics have equal importance in software defect prediction. To tackle this concern, we employed various feature engineering techniques. These techniques included univariate logistic regression-driven feature selection, resampling, and min-max normalization. By using these methods, we optimized the feature set and prepared the data for our analysis. In the following sections, we will provide a detailed explanation of these feature engineering processes and their contributions to the overall effectiveness of our software defect prediction model.

3.1.2 Univariate Logistic Regression based Feature Selection

In essence, the Univariate Logistic Regression (ULR) method is a statistical analysis technique that involves both dependent and independent variables. In our case, the dependent variable is the per-class software reusability, and the independent variables are the software metrics. As we are dealing with a two-class problem (Normal or Defect/Fault), the dependent variable takes on two labels: 1 for Normal and 0 for Defect/Fault. This allows us to evaluate the significance of each OOP CK metric in predicting software reusability. Mathematically, we utilize equation (1) to estimate the logistic regression value.

$$\pi(x) = \frac{e^{\alpha_0+\alpha_1 X}}{1+e^{\alpha_0+\alpha_1 X}} \quad (1)$$

In equation (1), the dependent variable is represented as logit[π(x)], and the independent variable is denoted by X. The parameter π denotes the likelihood factor, which signifies the importance of each metric in the analysis. We estimate the value of π(x) using equation (2).

$$logit[\pi(x)] = \alpha_0 + \alpha_1 X \quad (2)$$

Let's consider a dataset X with N rows and M+1 columns. In this context, M represents the number of independent variables for each row (in this case, 17 software metrics), and the additional column is reserved for the dependent variable. Now, let β be a column vector of length K+1, where K represents the number of parameters associated with the M columns of the independent variable. Each parameter in the vector β corresponds to one of the M columns of the independent variable.

When we utilize the logistic regression function, also known as the Logit function, we calculate the log-odds of the likelihood of success in relation to the linear component. Mathematically, this can be expressed as follows:

$$Logit\left(\frac{\theta_i}{1-\theta_i}\right) = \sum_{m=0}^{M} x_{im}\beta_m \quad i = 1,2,\dots,N \quad (3)$$

In equation (3), the term (θ_i / (1-θ_i)) represents a factor commonly known as the odds of an event. Now, let's consider the variable y, takes a value of 1 for the Normal or Defect/Fault class. This variable follows a Bernoulli distribution with a probability parameter denoted as p. For each instance, we calculate the probability parameter (p-value), and the model selects instances where p≥0. 05..

Despite the CKJM model extracting a total of 22 distinct features, the Univariate Logistic Regression (ULR) feature selection method identified six specific OOP metrics that hold significant importance in software defect prediction. These metrics are as follows:

1. WMC - Weighted Methods per Class
2. DIT - Depth of Inheritance Tree
3. NOC - Number of Children
4. CBO - Coupling between Object Classes
5. RFC - Response for a Class
6. LCOM - Lack of Cohesion in Methods

Additionally, the following metrics were considered for further computing, making a total of 17 different characteristics:

1. Ca - Afferent Couplings
2. Ce - Efferent Couplings
3. NPM - Number of Public Methods
4. DAM - Data Access Metric
5. MOA - Measure of Aggregation
6. MFA - Measure of Functional Abstraction
7. CAM - Cohesion Among Methods of Class
8. CC - Cyclomatic Complexity
9. LOC - Lines of Code
10. IC - Inheritance Coupling
11. CBM - Coupling Between Methods
12. AMC - Average Method Complexity

These selected features form the basis for further computations and analysis in the software defect prediction model.

3.1.3 Feature Re-sampling:

This study acknowledged the problem of class-imbalance in the IVY, JEdit, Camel, and Ant datasets, where the number of defect classes is considerably smaller compared to the normal classes. To tackle this issue, the researchers used a resampling technique on the input data. Specifically, they employed the synthetic minority oversampling method to increase the number of minority samples, ensuring the training process's effectiveness. This approach aimed to mitigate the skewed performance and high false-positive outputs that can arise due to class-imbalance in machine learning models. By addressing this concern, the study sought to enhance the overall performance and reliability of their software defect prediction model.

To handle class-imbalance, the study used a confidence level of 95% for both up-sampling and down-sampling techniques. Simple over-sampling or under-sampling, or random resampling, was deemed inadequate in fully resolving class-imbalance issues as it could introduce bias in favor of the majority class. This bias may cause new samples to be predicted as the majority class, resulting in false predictions and reduced model accuracy. Therefore, the researchers opted for a more robust approach to ensure the effectiveness and reliability of their software defect prediction model.

To address the aforementioned limitations, the proposed model implemented the synthetic minority oversampling technique. This involved generating synthetic positive samples using the K-nearest neighbor (k-NN) algorithm, with a specific choice of 5-Nearest Neighborhood for the minority "Defect or Fault" class. The next step was to balance the dataset by equalizing the number of samples, ensuring that the majority class had the same number of samples as the minority class. This approach aimed to create a more balanced dataset, leading to improved machine learning performance and higher prediction accuracy in software defect prediction.

3.1.4 Min-Max Normalization

It is well-known that data imbalance and convergence are significant challenges in classification or prediction systems, particularly in models with large sets of features. After performing feature extraction and selection, the retrieved data may vary in size and range, leading to computational issues such as premature convergence and overfitting of the learning model. This can negatively impact the overall computational efficiency, accuracy, and reliability of the system.

To tackle this issue, the proposed model employs Min-Max normalization on the input data. The Min-Max normalization algorithm, denoted by equation (3.4), scales the feature values to lie within the range of 0 to 1. Through linear conversion and mapping of each data element x_i' from the selected features X, the proposed model ensures normalization within the range [0, 1]. This normalization process is crucial for handling unstructured data effectively, addressing convergence problems, and improving the computational efficiency and reliability of the learning model. Equation (4) is utilized in the proposed model to estimate the normalized value(s) of the input data x_i.

$$Norm(x_i) = x_i' = \frac{x_i - min\,(X)}{max(X) - min\,(X)} \quad (4)$$

In equation (4), the terms min(X) and max(X) represent the minimum and maximum values

of the dataset X, respectively. These values define the lower and upper bounds of the normalization process.

Data normalization was applied to all input benchmark datasets in the suggested model, resulting in the following normalized:

$$[D_i] = Norm(i \subset JEdit, IVY, ANT, CAMEL) \quad (5)$$

3.1.5 Heuristic Driven Neuro-Computing Model for SDP

Once the input data or selected features from each dataset were normalized, the proposed model then proceeded with the two-class classification using the heuristic-driven neuro-computing model (HNC). The HNC model is a key component of the study, and its derivation and functioning are described in detail below:

3.1.5.1 Independent and Dependent Variable Definition

The primary aim of this study is to explore the association between various metrics and the likelihood of software faults. It is essential to recognize that the relationship between measurements and fault proneness at the class level is not linear. To tackle these challenges, the study used defects as the dependent variable while considering specific CK metrics as independent variables. The objective is to establish a correlation between the occurrence of faults in a class and the CK metrics. The analysis in this thesis examines the impact of 17 CK metrics on the occurrence of faults, as illustrated below:

$$\ldots T, NOC, CBO, RFC, LCOM, Ca, Ce, NPM, \ldots MFA, CAM, CC, LOC, CBM, AMC,) \quad (6)$$

3.1.5.2 Neu-Computing Model: Definition

Over the years, artificial intelligence systems have made significant advancements, drawing inspiration from biological neural networks and their functioning. Researchers from diverse fields have utilized Artificial Neural Networks (ANNs) to solve various computational problems. Traditional mechanisms for computational problem-solving have shown limitations in performance, leading to the emergence of ANNs as a promising technique. ANNs have become a preferred alternative for addressing major computational and decision-oriented issues, including software defect prediction and classification in this thesis.

The history of ANN's development can be divided into three phases. The first phase, during the 1940s, saw significant contributions from researchers like McCulloch and Pitts. The second phase, in the 1960s, involved researchers such as Rosenblatt and Minsky et al., who proposed theories like perceptron convergence and highlighted the limitations of simple perceptron-based NN. This phase motivated researchers to work on optimizing ANN networks for efficient applications in computer science, lasting for about 20 years. The third phase emerged in the 1980s when ANNs achieved significant breakthroughs. During this time, Hopfield's energy approach was introduced, and the back-propagation algorithm became a revolutionary development, especially for multilayer perceptrons. It underwent several optimizations and improvements. Various researchers have contributed to the continuous development and refinement of ANN techniques throughout the years.

3.1.5.3 Computational Models of Neurons

The neuron formula computes the weighted sum of its n input signals, represented as x_j = 1, 2, ..., n. If the total sum exceeds a pre-defined threshold U, the neuron generates an output of 1; otherwise, the output is 0.

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right) \quad (7)$$

The neuron's formula incorporates a periodic function θ () with a unit step at 0, and the relationship between the synapse weight and the jth input is represented by w_j. The threshold U is considered as another weight w_0 = -U, attached to the neuron, with a constant input x0 = 1. Positive weights are associated with excitatory synapses, while negative weights are linked to inhibitory synapses. McCulloch and Pitts demonstrated that when the weights are appropriately chosen, a synchronous arrangement of such neurons can perform universal computations.

The McCulloch-Pitts neuron is biologically inspired, where axons and dendrites are likened to wires and linkages, synapses are represented by connection weights, and neuronal activity is approximated by the activation function. However, the model has certain assumptions that don't completely mirror actual neuron behavior. To enhance the model's capabilities, various activation functions have been introduced, such as piecewise linear, sigmoid, or Gaussian functions. Among these, the sigmoid function, particularly the logistic function, is commonly used in Artificial Neural Networks (ANNs) due to its smoothness and favorable asymptotic properties.

$$g(x) = \frac{1}{(1 + exp^{-\beta x})} \quad (8)$$

3.1.5.4 ANN Architectures

Artificial Neural Networks (ANNs) can be conceptualized as weighted directed graphs, where artificial neurons are depicted as nodes, and directed edges symbolize connections between neuron outputs and inputs. ANNs can be classified into two main types based on their connection patterns:

- Feed-forward networks: These networks do not have loops in their graphs, resulting in unidirectional connections between neurons organized into layers. The most well-known example is the multi-layer perceptron. Feed-forward networks produce a single set of output values from a given input, making them static and memory-less, as their response to an input is independent of the previous network state.
- Recurrent (or feedback) networks: These networks feature loops created by feedback connections, making them dynamic systems. Upon receiving a new input pattern, the neuron outputs are computed, and the feedback paths modify the inputs to each neuron, causing the network to transition to a new state. Due to their network architectures, recurrent networks necessitate distinct learning algorithms for training.

The subsequent section offers a comprehensive overview of the learning processes for the various types of networks mentioned earlier.

3.1.5.5 ANN Learning

Learning plays a crucial role in intelligence, and within the domain of Artificial Neural Networks (ANNs), it pertains to updating the network's architecture and connection weights to effectively accomplish a particular task. ANNs possess the ability to learn automatically from examples, distinguishing them from conventional expert systems that depend on pre-defined rules.

The field of learning in neural networks consists of three primary paradigms: supervised, unsupervised, and hybrid. In supervised learning, the network is provided with correct answers for each input pattern, enabling it to adjust the weights accordingly. Unsupervised learning, on the other hand, focuses on exploring the underlying structure and correlations in the data without explicit correct answers. Hybrid learning combines elements of both supervised and unsupervised approaches to leverage their respective advantages.

In learning theory, three important aspects are considered: capacity, sample complexity, and computational complexity. Capacity relates to the network's ability to store patterns and establish decision boundaries. Sample complexity determines the minimum number of training patterns required for the network to generalize effectively. Computational complexity, on the other hand, refers to the time taken by learning algorithms to process and adjust the network's weights during training.

There are four primary types of learning rules: error correction, Boltzmann, Hebbian, and competitive learning. In this thesis, the error correction model has been chosen for software defect prediction (SDP). The error correction rules play a vital role in enhancing the learning process for SDP, aiming to optimize the prediction accuracy and overall performance of the model.

3.1.5.6 Error-Correction Rules Based Learning

In the supervised learning paradigm, the network is given the desired outputs for each input pattern. While learning, the actual output y produced by the network might deviate from the desired output d. Error-correction learning rules function by utilizing the error signal (d - y) to adjust the connection weights gradually, with the objective of minimizing this error and improving the network's accuracy in generating the desired outputs.

The perceptron learning rule is based on this error-correction principle and is used for perceptrons, which are single neurons with adjustable weights (w_j, j = 1, 2, ..., n) and a threshold U. Given an input vector x = (x_1, x_2, ..., x_n), the net input to the neuron is calculated as follows:

$$\boldsymbol{v} = \sum_{j=1}^{n} w_j x_j - u \quad (9)$$

In a two-class classification scenario, the perceptron functions as a binary classifier. If the net input v is positive, the perceptron's output y is +1, indicating one class. On the other hand, if v is non-positive, the output y is 0, representing the other class. The decision boundary, defined by a linear equation, separates the input space into two regions, categorizing input patterns into their respective classes based on the sign of the net input.

Rosenblatt [33] devised a learning procedure to calculate the weights and threshold in a perceptron using a set of training patterns. The perceptron learning procedure achieves convergence after a finite number of iterations when training patterns are drawn from two classes that are linearly separable, as confirmed by the perceptron convergence theorem. However, in real-world scenarios, it is often uncertain whether the patterns are linearly separable, leading to challenges in applying the standard perceptron learning algorithm. To address this, variations of the learning algorithm have been proposed in the literature to handle non-linearly separable data and enhance the perceptron's performance in practical applications [34].

3.1.5.7 LM-ANN Neuro-Computing for SDP Learning

LM-ANN, regarded as one of the top neuro-computing models, was independently developed by Kenneth Levenberg and Donald Marquardt. This model offers a numerical solution for minimizing nonlinear functions and is known for its fast and stable convergence. In the domain of artificial neural networks, LM-ANN is particularly well-suited for training small- and medium-sized problems.

The error backpropagation (EBP) algorithm, also known as the steepest descent algorithm, is another extensively employed method for training neural networks. Although EBP brought considerable advancements, its slow convergence remains a challenge, mainly due to the requirement for suitable step sizes and the presence of "error valleys" caused by varying curvature in different directions during the optimization process.

To overcome the slow convergence issue, the Gauss-Newton algorithm utilizes second-order derivatives to assess the curvature of the error surface. This enables the algorithm to find appropriate step sizes and achieve faster convergence. However, this improvement is contingent on having a reasonable quadratic approximation of the error function; otherwise, the algorithm may diverge and lead to undesirable results.

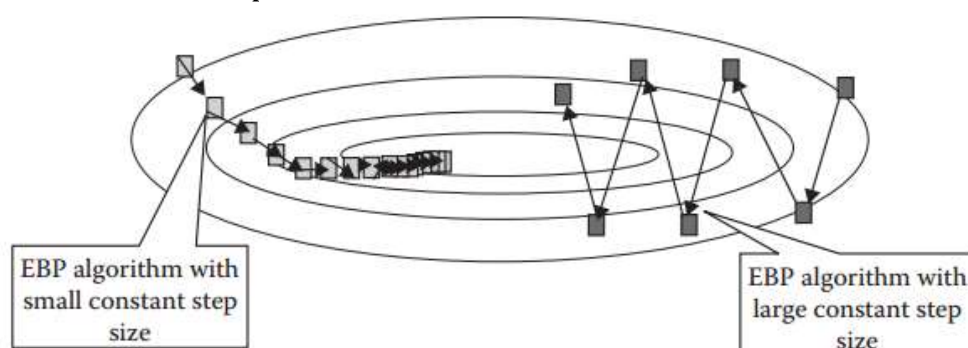

Fig.1 Searching process of the steepest descent method

3.1.5.8 Heuristic Driven LM-ANN Neuro-Computing for Software Defect Prediction

Artificial Neural Networks (ANNs) emulate the functional aspects of the human brain, allowing them to learn from input data or patterns and categorize unknown inputs into target categories. The ANN model comprises three layers: the input layer, hidden layer, and output layer (as shown in Figure 3.2). This architecture incorporates multiple neurons that process input data at the hidden layers, leading to classification at the output layer. During the learning process, the ANN employs error-reduction methods to calculate the discrepancy between expected and observed outputs, with the ultimate goal of achieving zero-error or minimizing the error to improve accuracy in the classification tasks.

The proposed ANN algorithm in this study focuses on two-class classification, where each class is classified as either Normal Class (labeled as "1") or Faulty Class (labeled as "0"). The input features for each class of the software are used as input to the ANN, and the number of hidden layers can vary depending on the specific configuration. In the input layer, a linear activation function is applied, resulting in an output that is identical to the input itself. The output from the hidden layer is then fed to the input of the output layer. The output layer of the ANN utilizes the Sigmoid function to produce the final output, providing the classification result for the input data.

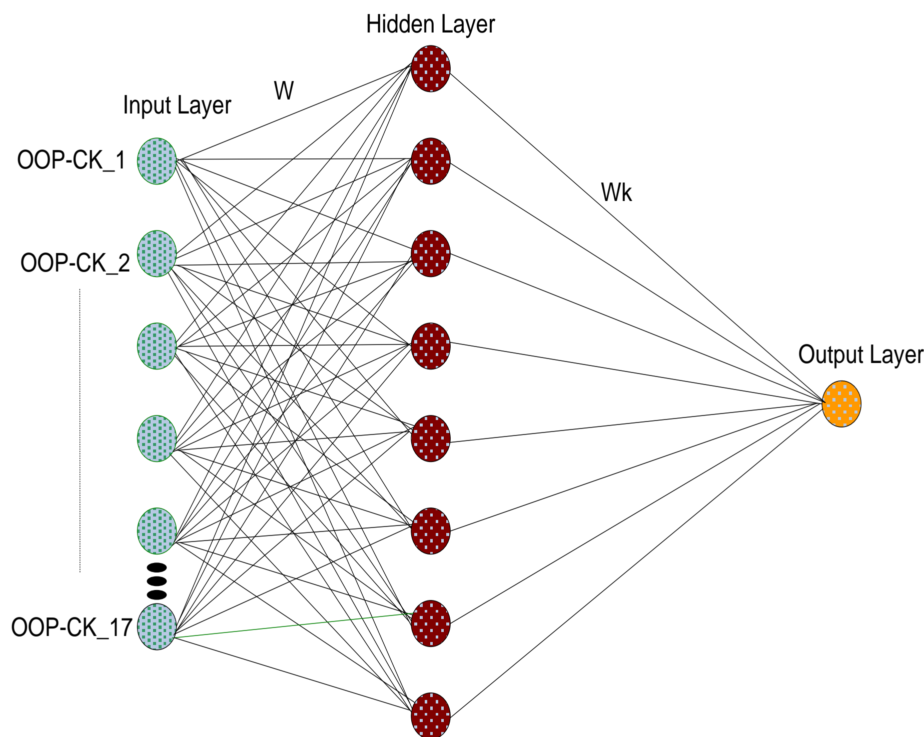

Fig.2 An illustration of ANN architecture with single hidden layer with one output node

$$O_h = \frac{1}{1 + e^{-I_h}} \tag{10}$$

In equation (10), I_h represents the input at the hidden layer. ANN is commonly defined as Y' = f(W, X), where Y' denotes the output vector, and X and W represent the input and weight values, respectively. The ANN aims to minimize a certain error function, such as mean square error (MSE), to achieve higher accuracy, which is estimated using equation (11).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i' - y_i)^2 \tag{11}$$

In equation (11), y represents the observed value, while y_i' is the expected value.

The neuro-computing architecture utilized in this study (Figure 2) comprises 17 input nodes, each representing one of the 17 CK metrics from multiple classes as individual inputs. As the expected outputs are binary, either FAULTY or NO-FAULTY, only one output node is necessary. The defined ANN architecture includes 19 hidden layers to strike a balance between performance and computational complexity. Thus, a total of 342 weights (17 input nodes + 1 output node) * 19 hidden nodes) need to be estimated for fault prediction and classification purposes.

To facilitate learning in the targeted neuro-model (Figure 2), the proposed neuro-computing approach involves the continuous estimation of 342 weight parameters in each iteration. However, this process can lead to challenges related to convergence and potential local minima issues, which may affect overall performance. To address these concerns, the researchers incorporate a heuristic model, specifically the genetic algorithm, to continuously fine-tune the 342 weight parameters. This strategy aims to overcome local minima problems and enhance the overall performance of the neuro-computing model. The use of the genetic algorithm is hypothesized to yield superior results in the optimization process.

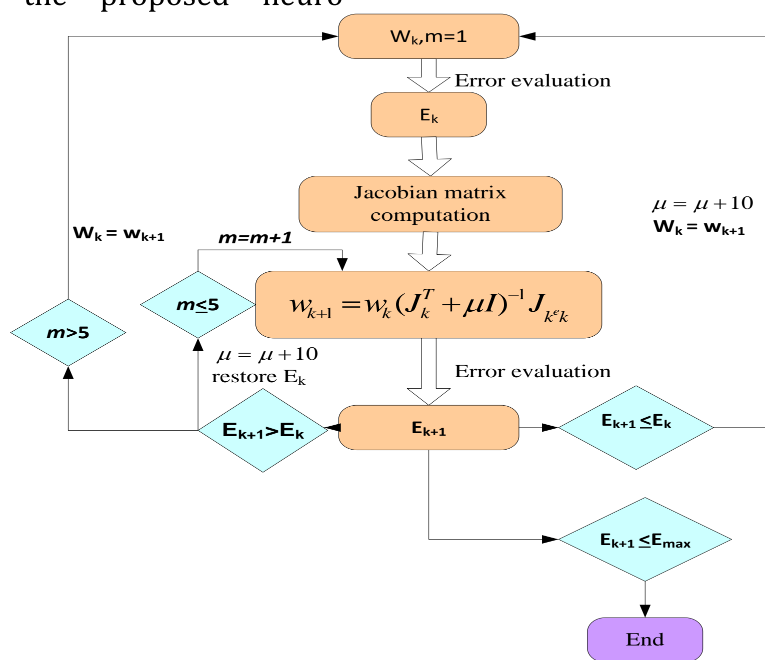

Fig.3 HCN Model: $\boldsymbol{W_k}$ is the current weight, $\boldsymbol{W_{k+1}}$is the next weight, $\boldsymbol{E_{k+1}}$is the current total error, and $\boldsymbol{E_k}$ is the final error

The overall training function of the targeted LM-ANN neuro-computing model follows the mechanism depicted in Figure 3. The core of this model lies in equation (w_(k+1)=w_k-(J_k^T J_k+μI)^(-1) J_k e_k), which is used to update the weight parameters for continuous feature learning. However, estimating 342 weight parameters over each iteration can lead to issues like local minima and convergence. To address this, the study applies a heuristic model called genetic algorithm.

In the proposed model, the genetic algorithm dynamically estimates the optimal set of weight parameters to tune the weight parameters in equation (w_(k+1) =w_k-(J_k^T J_k+μI) ^ (-1) J_k e_k), which facilitates efficient learning and yields superior results. This approach helps the LM-ANN (Figure 3) to overcome convergence and local minima problems during training, enhancing the overall performance of the model.

3.1.5.9 Heuristic Driven LM-ANN Weight Tuning

In this study, to tackle the challenges associated with estimating a large number of weight parameters and to address issues like

local minima and convergence, the researchers employ a heuristic concept called Genetic Algorithm (GA). GA draws inspiration from Darwin's principle of selection and operates as an evolutionary computing technique, intended to identify optimal solutions from a pool of potential sub-solutions. In the context of optimizing the targeted LM-ANN, GA seeks to enhance the learning process of the ANN by discovering an optimal set of weight parameters that can lead to higher accuracy in the classification tasks.

The Genetic Algorithm (GA) begins by initializing a random set of weights as an initial population, where each population is represented as binary strings encoding a potential solution. Each candidate solution is then assessed based on a fitness value, which indicates its suitability as a solution. The candidates with higher fitness values are retained, while those with lower fitness are eliminated. The next generation of candidate solutions is created through a process of crossover or reproduction, which occurs based on predetermined probabilities for crossover and mutation. This selection and breeding process allows GA to iteratively improve the population of solutions, leading to better solutions over successive generations.

In the neuro-computing model with an i-h-o network configuration (i input layer, h hidden layer, and o output layer), each of the 17 software metrics is input to each input neuron. The resulting architecture becomes 17-19-1, with 19 hidden layers utilized to handle the model's complexity effectively. For this specific architecture, the total number of weights required is N (12).

$$N = (i + O) * h \quad (12)$$

The process of creating appropriate populations and evolving them through successive generations continues until the optimal target value, known as the stopping criteria, is reached. This stopping criteria is achieved when the set of weight parameters results in an error of zero or near-zero during ANN learning. In other words, the genetic algorithm iteratively refines the populations until the ANN's learning process achieves a highly accurate solution.

To improve learning and fine-tune the weights, the proposed model considers each weight value as a gene in the chromosomes. Hence, for a total gene length of l, the length of the chromosome, L_{Chrom}, is determined using equation (13).

$$L_{Chrom} = N * l = (i + O) * h * l \quad (13)$$

In the proposed Genetic Algorithm (GA) model, all chromosomes are used as input weights and form the population. The fitness of each generation is evaluated to determine their suitability in achieving the objective of minimizing the Mean Squared Error (MSE). Through weight updates or tuning in each generation, the GA model aims to find the optimal set of weight parameters that result in the minimum Root Mean Squared Error (RMSE). The weights (W_k) are updated or tuned in accordance with equation (14) in the proposed model.

$$W_k = \begin{cases} if\ 0 \leq x_{kl+1} < 5 \\ -\dfrac{x_{kl+2*}10^{l-2} + x_{kl+3*}10^{l-3} + \cdots + x_{(k+1)l}}{10^{l-2}} \\ if\ 5 <= x_{kl+l} <= 9 \\ +\dfrac{x_{kl+2*}10^{l-2} + x_{kl+3*}10^{l-3} + \cdots + x_{(k+1)l}}{10^{l-2}} \end{cases} \quad (14)$$

The pseudo-code for the proposed heuristic-driven neuro-computing (HNN) model is presented in Figure 4.

Algorithm for Fitness Estimation

Input: $\bar{I_i} = (I_{1i}, I_{2i}, I_{3i}, \cdots, I_{li})$

Output: $\bar{T_i} = (T_{1i}, T_{2i}, T_{3i}, \cdots, T_{ni})$

Where $\bar{I_i}, \bar{T_i}$ state the input and output pairs of the $i-h-o$ configuration of neural network.

Phase1: Determine weights $\bar{W_i}$ for C_i by equation (4.3)

Phase2: Expecting $\bar{W_i}$ be a constant weight, perform training of N input as well as compute output O_i

Phase3: Determine MSE E_j for all input instance j , $E_j = (T_{ji} - O_{ji})$

Phase4: Determine RMSE of chromosome $C_i E_i = \sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}$

Where N indicates the total number of training records (data)

Phase5: Determine the value of fitness for chromosome $C_i F_i = \frac{1}{E_i} = \frac{1}{\sqrt{\frac{\sum_{j=1}^{j=N} E_j}{N}}}$

Fig.4 Pseudo code for HNC

4. Results And Discussion

In this section, we present the results of our simulations and statistical analyses to assess the performance of the proposed heuristic-driven neuro-computing (HNC-SDP) model compared to existing state-of-the-art models. For the evaluation, we utilized four benchmark datasets: IVY, CAMEL, ANT, and JEDIT, which were obtained from the NASA PROMISE repository for software defect prediction. The HNC-SDP model was developed using MATLAB2015b software and executed on a computer with 8GB RAM and an Intel i5 processor running Microsoft Windows 10. A detailed examination of the proposed model and the statistical analyses are discussed in the subsequent sections.

4.1 Characterization of Performance

The performance evaluation of the proposed HNC-SDP model was conducted individually for each dataset, and the corresponding confusion metrics were derived to assess its performance. Confusion metrics offer crucial statistical measures, including accuracy, precision, recall, and F-Measure, to evaluate the effectiveness of the model. Table 1 provides a comprehensive summary of the confusion metrics and the corresponding statistical parameters for each test case.

In this research, for each simulation case involving the JEDIT, Ant, Camel, and IVY datasets, the confusion matrix was computed to determine the counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Utilizing these matrix values, the model's performance was assessed in terms of accuracy, precision, recall, and F-Measure, as detailed in Table 1. These metrics provide valuable insights into the effectiveness of the proposed model across different datasets.

Table.1 Performance Parameters

Parameter	Mathematical Expression	Definition
Accuracy	$\frac{(TN+TP)}{(TN+FN+FP+TP)}$	Indicates the percentage of predicted fault prone modules that are examined

		out of all modules.
Precision	$\frac{TP}{(TP+FP)}$	Indicates the extent to which repeated observations under test conditions provide the same findings.
F-measure	$2.\frac{Recall.Precision}{Recall+Precision}$	It creates a single score by combining the precision and recall numeric values, which is specified as the harmonic mean of the recall and precision.
Recall	$\frac{TP}{(TP+FN)}$	It specifies the number of objects that must be listed.

In this study, the performance evaluation involved using multiple input datasets, and the overall assessment of performance was carried out using two methods: intra-model comparison and inter-model comparison. These approaches allowed for a comprehensive evaluation of the proposed model's performance compared to other models and its effectiveness across different datasets.

Intra-model comparison entailed a statistical evaluation of the proposed HNC-SDP model's performance using various input datasets, focusing on metrics such as accuracy, precision, recall, and F-Measure. This analysis aimed to gauge how well the model performed across different datasets and provided insights into its consistency and reliability.

On the other hand, inter-model comparison involved analyzing the relative performance of the proposed HNC-SDP model in comparison to different existing methods.

The detailed analysis and significance of the simulation results are discussed in the following sections.

4.1.1 Intra-Model Performance Characterization

In this evaluation, the goal is to analyze the performance of the proposed model using various input datasets. The objective is to assess the effectiveness of the proposed model across different inputs and identify its strengths and weaknesses. Additionally, this assessment helps determine the average performance of the proposed system, which can be used later for comparing its relative performance to other models (inter-model comparison).

4.1.2 Test Case-1 IVY Dataset

The confusion metrics for the IVY dataset, representing faulty and non-faulty (normal class) instances, are provided in Table 2 and Table 3. These metrics are obtained both before and after the execution of the proposed HNC-SDP model. The purpose of obtaining these metrics is to compare and contrast the results, which will help in understanding the classification performance of the proposed model (HNC-SDP).

Table.2 Confusion matrix for IVY dataset before HNC-SDP execution

	Normal	Fault/Defect
Normal	481	0
Fault/Defect	40	0

Table.3 Confusion matrix for IVY data after prediction

	Normal	Fault/Defect
Normal	311	1
Fault/Defect	36	4

Table 4 presents the statistical results obtained for accuracy, precision, recall, and F-Measure. The proposed HNC-SDP model achieved an accuracy of 88.35%, precision of 99.36%, F-Measure of 93.8%, and recall (sensitivity) of 88.83%. Notably, the higher F-Measure value of 0.93 indicates the effectiveness of the proposed model even under class-imbalanced conditions. The decrease in RMSE or MSE values over generations, as shown in Figure 5 for different datasets, demonstrates the efficient learning of the proposed neuro-computing model, leading to superior performance.

Table.4 Cumulative HNC-SDP Performance evaluation for IVY data

Accuracy	Precision	F-Measure	Recall
0.8835	0.9936	0.9380	0.8883

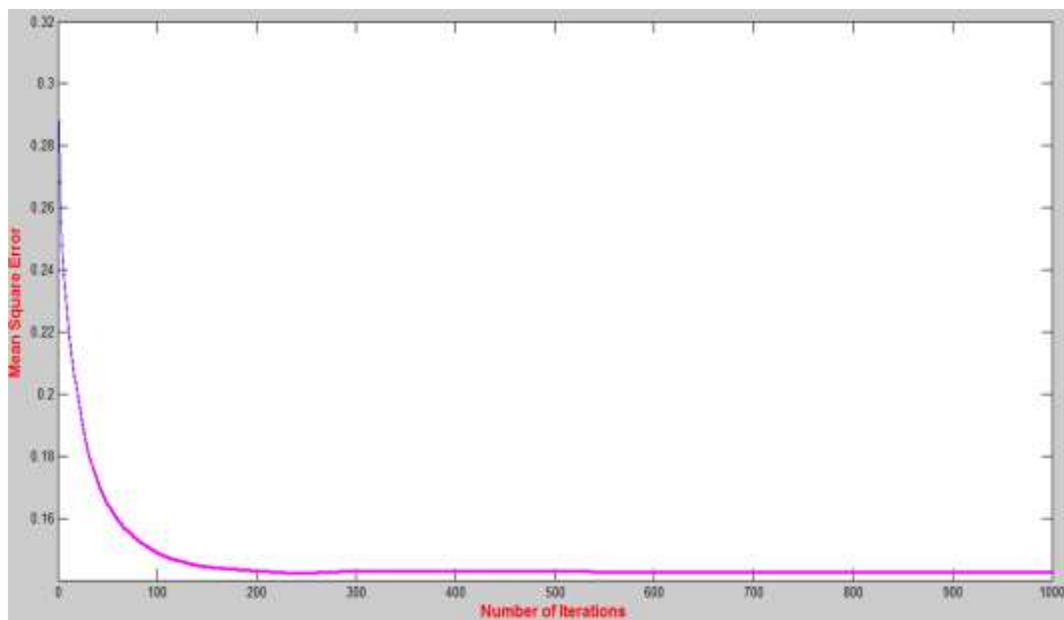

Fig.5 MSE variation for HNC-SDP model over IVY dataset

In this study, the Mean Square Error (MSE) was utilized as the fitness value during the learning process using the proposed heuristic model. The goal was to minimize the MSE, which implies that the Root Mean Square Error (RMSE) should decrease with increasing generations as the model learns and approaches an optimal fitness value. Following genetic computing principles, the error should decrease over generations. The results obtained in this thesis, as shown in Figure 1, Figure 2, Figure 3, and Figure 4, confirm that the proposed heuristic-driven neuro-computing model (HNC-SDP) performs optimally in terms of software defect prediction, as expected. The decreasing trend of RMSE over generations demonstrates effective learning and convergence, leading to improved defect prediction accuracy.

4.1.3 Test Case-2 ANT Dataset

The proposed HNC-SDP model was also applied to the ANT1.7 PROMISE dataset, and the simulation results are presented below: Table 5 shows the confusion matrix of the ANT1.7 defect dataset before using the HNC-SDP model for defect prediction. Table 6 displays the confusion metrics obtained after applying the HNC-SDP model (referred to as the proposed model) for defect prediction.

Table.5 Confusion matrix for ANT data before prediction

	Normal	Fault/Defect
Normal	578	0
Fault/Defect	166	9

Table.6 Confusion Matrix for ANT Data After Prediction

	Normal	Fault/Defect
Normal	578	0
Fault/Defect	157	9

The statistical performance results obtained based on the confusion metrics from Table 6 are presented in Table 7. Additionally, Figure 6 illustrates the variation of Mean Squared Error (MSE) over the learning process. The results demonstrate the performance of the proposed HNC-SDP model on the ANT1.7 PROMISE dataset.

Table.7 Aggregate HNC-SDP Performance assessment for ANT data

Accuracy	Precision	F-Measure	Recall
0.8145	0.9343	0.8867	0.8438

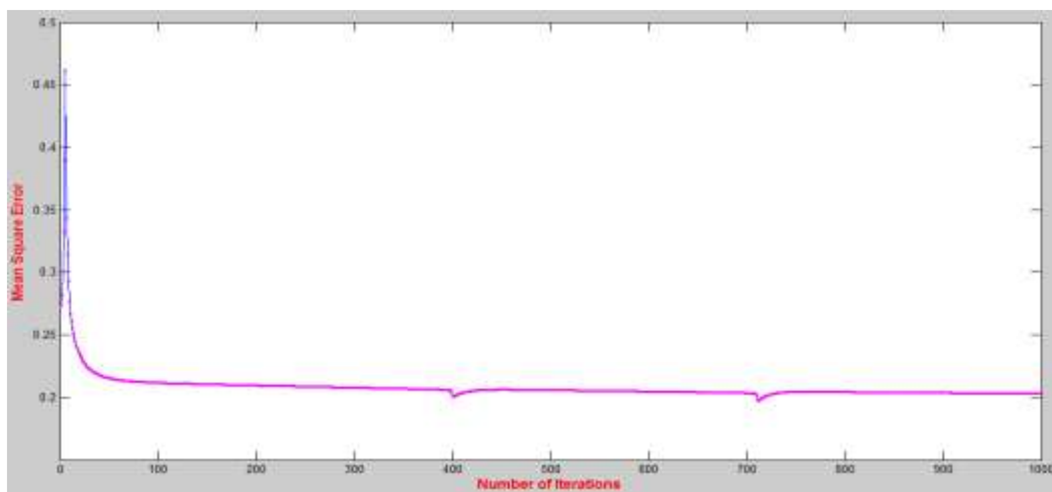

Fig.6 MSE variation for HNC-SDP model over ANT dataset

The results from Table 7 indicate that the proposed HNC-SDP model achieved an accuracy of 81.4%, precision of 93.43%, recall (sensitivity) of 84.3%, and F-Measure of 88.67% when applied to the ANT1.7 dataset. These performance metrics demonstrate the effectiveness of the proposed model in defect prediction and classification for the ANT1.7 dataset.

4.1.4 Test Case-3 JEDIT Dataset

The confusion matrix for the JEDIT dataset before the execution of HNC-SDP is presented in Table 8. After applying the HNC-SDP model, the corresponding confusion matrix is shown in Table 9. These matrices provide valuable insights into the classification performance of the proposed model for the JEDIT dataset.

Table.8 Confusion matrix for JEDIT data prior to prediction

	Normal	Fault/Defect
Normal	481	0
Fault/Defect	11	0

Table.9 Confusion matrix for JEDIT data later of prediction

	Normal	Fault/Defect
Normal	481	0
Fault/Defect	10	1

The statistical results for the proposed HNC-SDP model execution on the JEDIT dataset are presented in Table 10. The overall performance of the model for the JEDIT data is assessed in terms of accuracy, precision, recall, and F-Measure. The results indicate that the accuracy achieved by the HNC-SDP model on the JEDIT dataset is approximately 98%. Additionally, the precision, recall, and F-Measure values are found to be 100%, 100%, and 98.97% respectively. Moreover, the MSE variation analysis also demonstrates superior learning, leading to optimal performance as shown in Table 10.

Table.10 Cumulative HNC-SDP Performance evaluation for JEDIT data

Accuracy	Precision	Recall	F-Measure
0.9799	1	1	0.9897

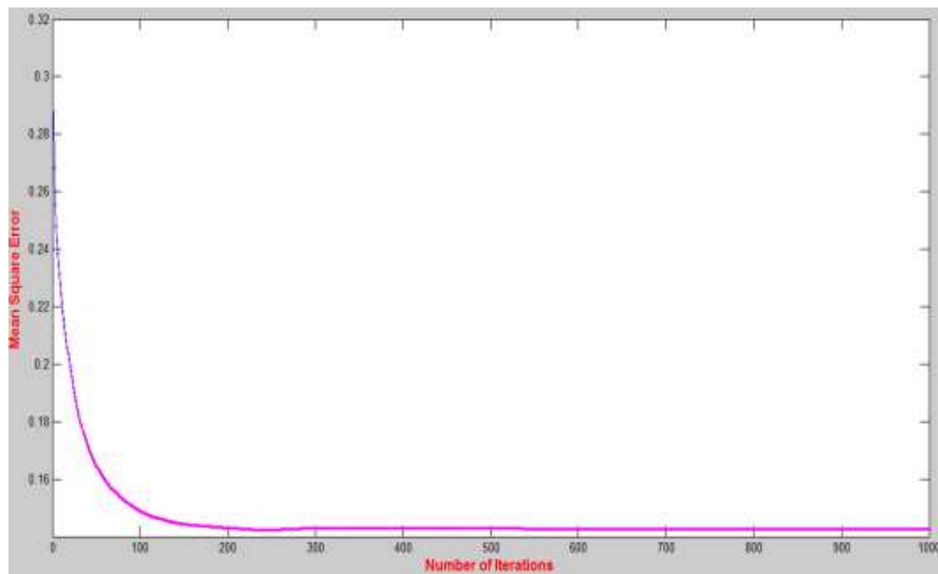

Fig.7 MSE variation for HNC-SDP model over JEDIT dataset

4.1.5 Test Case-4 CAMEL Dataset

Table 11 displays the confusion matrix for the CAMEL dataset before the execution of HNC-SDP. On the other hand, Table 12 represents the confusion matrix obtained after the execution of HNC-SDP for software defect prediction on the CAMEL dataset.

Table.11 Confusion matrix for Camel data prior to prediction

	NON-FAULTY	FAULTY
Normal	777	0
Fault/Defect	188	0

Table.12 Confusion matrix for Camel data later prediction

	NON-FAULTY	FAULTY
Normal	770	7
Fault/Defect	172	16

Table.13 Cumulative HNC-SDP Performance evaluation for Camel data

Accuracy	Precision	Recall	F-Measure
0.8114	1	0.8102	0.8952

Upon observing the results (Table 13) for the CAMEL dataset, it is evident that the suggested HNC-SDP model demonstrates higher precision, although the accuracy of 81% indicates a high level of non-linearity and potentially lower overall performance. However, the higher values of F-measure (89.5%) and recall (81.2%) indicate that the proposed SDP model can still deliver reliable performance even with high non-linear inputs or training data.

It is interesting to note that unlike in Figure 5, Figure 6, and Figure 7 where the error proneness decreased over iterations, the error (MSE) in Figure 8 for the CAMEL dataset shows fluctuations due to the non-uniform error distribution. Despite this, the proposed HNC-SDP model demonstrates robustness by quickly stabilizing the performance with reduced error and progressing towards the optimal fitness value within a few iterations.

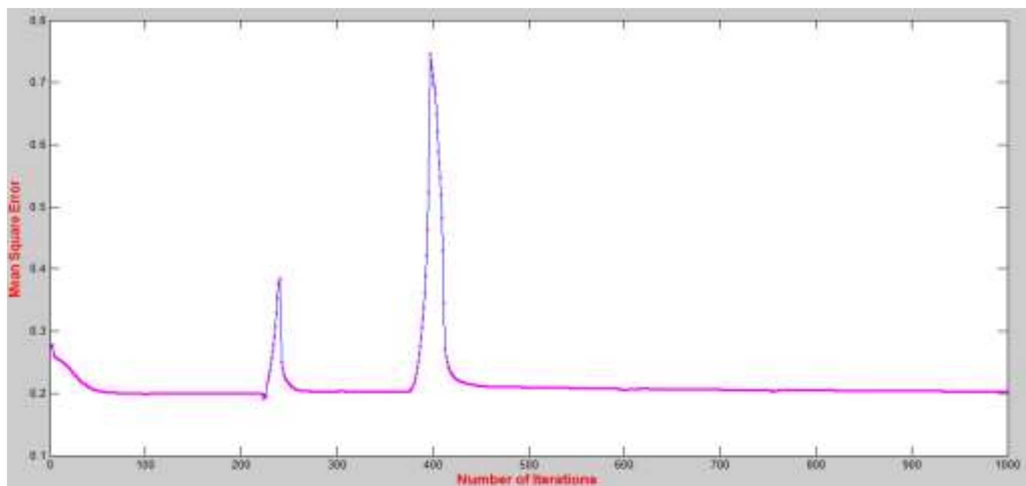

Fig.8 MSE variation for HNC-SDP model over CAMEL dataset

The swift convergence of any machine learning model can lead to superior accuracy and avoid saturation. In this study, the HNC-SDP model was developed as a heuristic-driven neuro-computing model to achieve prediction results before encountering local minima and convergence issues typically faced by conventional neuro-computing models. The employed heuristic model, an improved genetic algorithm, ensured that the fitness value of candidate chromosomes or sub-solutions increased after each generation. The fitness value in HNC-SDP was defined as the inverse of MSE ($F_i = 1/E_i$), where E_i represents the fitness value of a chromosome. The reduction in MSE indicated better solutions and improved learning.

The implementation of the proposed heuristic model, as observed in Figure 5 to Figure 8, effectively prevented local minima and convergence problems. Additionally, the use of SMOTE sampling followed by Min-Max normalization helped address over-fitting and data imbalance concerns. These factors collectively contributed to better learning and superior results in terms of accuracy, precision, recall, and F-Measure (as shown in Table 4, Table 7, Table 10, and Table 13). The overall performance summary of the proposed HNC-SDP model over different input datasets is presented in Table 14.

Table.14 Summary of Intra-model performance assessment

Dataset	Accuracy (%)	Precision (%)	F-Measure (%)	Recall (%)
IVY1.7	88.35	99.36	93.80	88.83
ANT1.8	81.45	93.43	88.67	84.38
JEDIT	97.99	100.00	100.00	98.97
CAMEL	81.14	100.00	81.02	89.52

4.1.6 Inter-Model Assessment

To assess the relative efficacy of the proposed HNC-SDP machine learning model, a comparison was made with the classical machine learning algorithm, namely Artificial Neural Network (ANN). The purpose of this comparison was to determine whether the inclusion of the proposed heuristic-driven neuro-computing concept (HNC-SDP) led to superior results compared to traditional ANN.

The simulation of both the proposed HNC-SDP model and ANN was performed on four different input datasets: ANT, IVY, JEDIT, and CAMEL. The relative performance comparison is illustrated in Figure 9, Figure 10, Figure 11, and Figure 12. The data indicates that the average fault prediction accuracy of the proposed HNC-SDP model is 98%, while the ANN-based SDP models achieve an average accuracy of 75.48%. This demonstrates that the proposed HNC-SDP model outperforms the conventional ANN model.

Furthermore, the proposed HNC-SDP model exhibits defect prediction accuracy of 98%, precision of 100%, F-measure of 98.9%, and recall efficiency of 100%. These high-performance parameters validate the robustness of the proposed HNC-SDP model for software defect prediction purposes. Overall, the results indicate that the proposed model achieves superior performance compared to existing ANN models for software defect prediction.

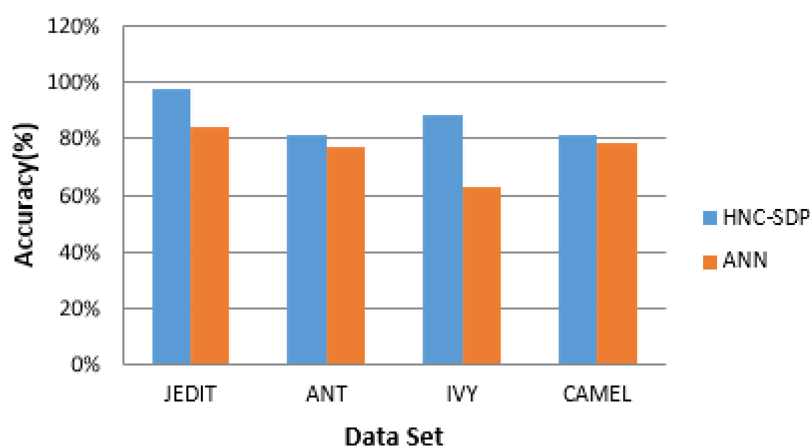

Fig.10 Comparison of the accuracy of HNC-SDP and ANN

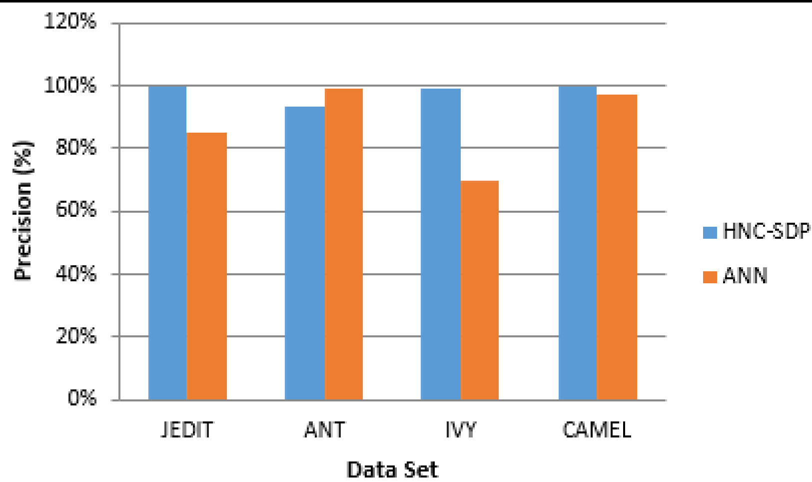

Fig.11 Comparison of the Precision of HNC-SDP and ANN

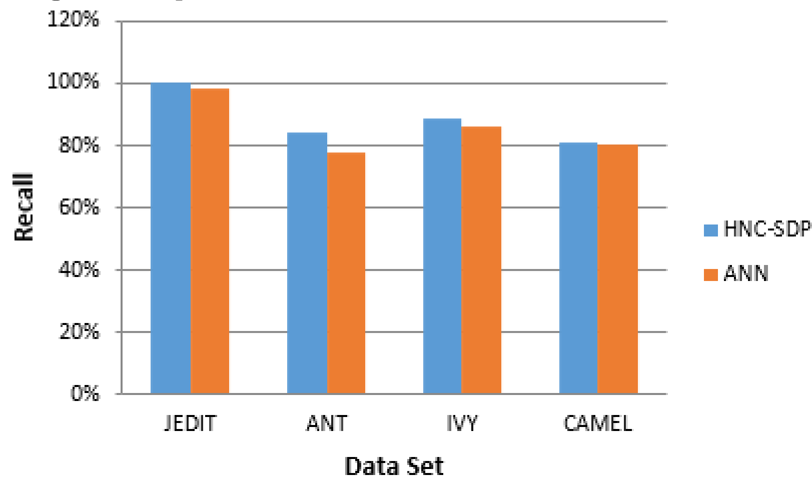

Fig.12 Comparison of the Recall of HNC-SDP and ANN

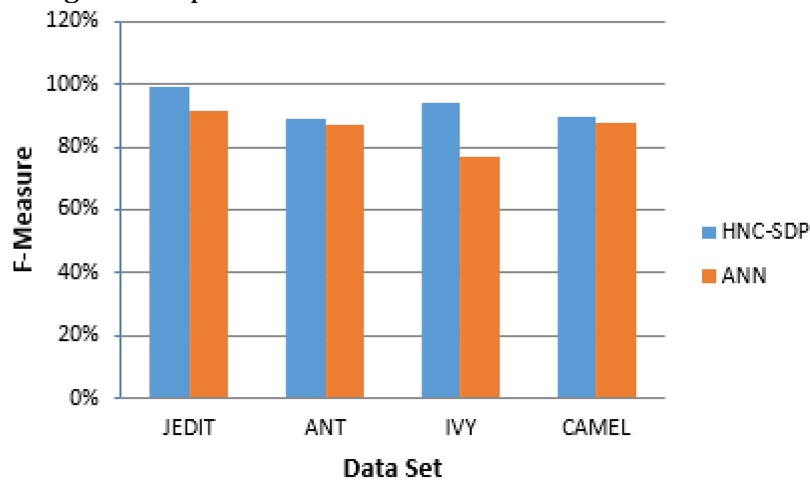

Fig.13 Comparison of the F-Measure of HNC-SDP and ANN

The F-measure, being the harmonic mean of precision and recall, provides a balanced assessment of the model's performance in defect prediction. The higher F-measure value in the proposed HNC-SDP model indicates better overall performance compared to the ANN algorithm. Additionally, the higher recall in the proposed HNC-SDP model signifies that it is more sensitive and capable of correctly identifying a higher proportion of true positive instances compared to the ANN-based SDP model.

To further evaluate the effectiveness of the proposed HNC-SDP model in comparison to other state-of-the-art methods, a qualitative study approach was employed. Various literature discussing machine learning-based software defect prediction (SDP) approaches were reviewed, and their corresponding performances were examined. The relative performance analysis between the proposed HNC-SDP model and other existing approaches is summarized in Table 15. This comparison demonstrates that the proposed HNC-SDP model outperforms other existing methods in terms of accuracy, precision, recall, and F-measure, reaffirming its superiority in software defect prediction.

Table.15 Various SDP strategies are compared in terms of performance

Reference	Machine Learning Techniques	Accuracy (%)	Precision (%)	F-Measure (%)
[35]	LLE-SVM	81.1	82.5	80.4
[35]	SVM	69.4	68.1	69.7
[36]	SVM	55.3	88.0	83.2
[37]	Natural Gas	94.2	-	-
[37]	Symbolic Regression	89.50	-	-
[37]	RBP-NN	80.0	-	-
[36]	LP	86.6	86.6	87.4
[36]	Naive Based	85.6	83.1	83.9
[38]	CPSO	69.2	67.6	-
[39]	T-SVM	75.8	84.1	80.9
[38]	GANN	73.4	81.6	-
[38]	AdaBoost	79.1	82.3	-
[40]	Random Forest	91.4	-	-
[41]	k-NN	91.8	-	-
[41]	C4.5	88.3	-	-
[41]	J 48	90.9	-	-
[41]	Levenberg-Marquardt	88.0	-	-
[38]	NNEP-Evolutionary	88.8	81.2	-
[42]	PSO	78.7	-	-
[37]	PSO-NN	97.7	-	-
Proposed	HNC-SDP	97.9	1	98.9

Based on the results presented in Table.15, it is evident that the proposed HNC-SDP model outperforms all other existing methods in software defect prediction. The superior performance of the proposed model demonstrates its robustness and effectiveness in handling various SDP tasks.

5. Future Work

The authors of the study found that certain intrinsic features, such as inheritance and polymorphism, were not sufficient as standalone features for effective classification in large-scale software. Instead, they identified CK metrics as a potential approach for

automatic software defect prediction using machine learning techniques.

To optimize the performance of the LM-ANN, the authors chose to apply a GA algorithm instead of traditional methods. It is important to note that many existing heuristic-based models typically use a pareto combination of crossover and mutation probabilities, such as 0.8 and 0.2 or 0.6 and 0.4. However, this approach can lead to a significant increase in the search space with each generation, leading to issues with convergence and local minima, which can adversely affect the overall model performance. Similarly, like other machine learning models, the challenge of avoiding local minima and convergence persists due to the high number of weight estimations required for the 17 input features

6. CONCLUSION

The growing demand for reliable and secure software systems has highlighted the need for effective software defect prediction (SDP) methods. However, traditional manual fault assessment methods for large and complex software architectures have proven to be challenging and resource-intensive. To address these issues, machine learning models have been proposed for SDP, but they often face limitations such as class imbalance, local minima, and convergence problems.

In this dissertation, we developed a novel and robust heuristic-driven neuro-computing model, called HNC-SDP, for software defect prediction. Leveraging the Levenberg Marquardt Neural Network (LM-ANN), the proposed model exhibited adaptive learning, making it suitable for non-linear feature learning from defect data. To overcome local minima and convergence issues associated with high weight estimation for 17 input features, we introduced an improved genetic algorithm as a heuristic model to assist in weight estimation and update during learning. The integration of feature engineering techniques, such as resampling and Min-Max normalization, further enhanced the model's performance by addressing class imbalance and over-fitting problems.

Through extensive case studies on four different defect datasets, including JEDIT, IVY, CAMEL, and ANT, the HNC-SDP model demonstrated its superiority over traditional neural networks and other state-of-the-art machine learning methods in terms of accuracy, precision, recall, and F-Measure. With accuracy reaching up to 98% and significant improvements in performance metrics, the proposed HNC-SDP model emerged as a highly effective solution for real-time SDP tasks.

In conclusion, the HNC-SDP model offers a robust and efficient approach to software defect prediction, providing higher accuracy and better generalization compared to classical machine learning algorithms. By combining adaptive learning, feature engineering, and heuristic-driven optimization, the proposed model shows great potential in addressing the challenges of SDP in complex and diverse software architectures. This research contributes to advancing the field of software defect prediction and offers valuable insights for building reliable and secure software systems.

References

[1] Q. Li and H. J. I. A. Pham, "A generalized software reliability growth model with consideration of the uncertainty of operating environments," vol. 7, pp. 84253-84267, 2019.

[2] S. Martínez-Fernández *et al.*, "Continuously assessing and improving software quality with software analytics tools: a case study," vol. 7, pp. 68219-68239, 2019.

[3] M. Mijač and Z. Stapić, "Reusability metrics of software components: survey," in *26th Central European Conference on Information and Intelligent Systems (CECIIS 2015)*, 2015: Faculty of Organization and Informatics Varazdin.

[4] M. Lafi, J. W. Botros, H. Kafaween, A. B. Al-Dasoqi, and A. Al-Tamimi, "Code smells analysis mechanisms, detection issues, and effect on software maintainability," in *2019 IEEE Jordan International Joint Conference on*

Electrical Engineering and Information Technology (JEEIT), 2019, pp. 663-666: IEEE.

[5] H. Liu, Q. Liu, Z. Niu, and Y. J. I. T. o. S. E. Liu, "Dynamic and automatic feedback-based threshold adaptation for code smell detection," vol. 42, no. 6, pp. 544-558, 2015.

[6] A. Baabad, H. B. Zulzalil, and S. B. J. I. A. Baharom, "Software architecture degradation in open source software: A systematic literature review," vol. 8, pp. 173681-173709, 2020.

[7] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. J. I. T. o. S. E. Oliveto, "Toward a smell-aware bug prediction model," vol. 45, no. 2, pp. 194-218, 2017.

[8] J. L. B. Justo, N. M. Araujo, and A. G. J. I. L. A. T. Garcia, "Software reuse and continuous software development: A systematic mapping study," vol. 16, no. 5, pp. 1539-1546, 2018.

[9] C. Diwaker *et al.*, "A new model for predicting component-based software reliability using soft computing," vol. 7, pp. 147191-147203, 2019.

[10] M.-C. Chiang, C.-Y. Huang, C.-Y. Wu, and C.-Y. J. I. T. o. R. Tsai, "Analysis of a fault-tolerant framework for reliability prediction of service-oriented architecture systems," vol. 70, no. 1, pp. 13-48, 2020.

[11] S. Maggo, C. J. I. J. o. I. T. Gupta, and C. Science, "A machine learning based efficient software reusability prediction model for java based object oriented software," vol. 6, no. 1, pp. 1-12, 2014.

[12] N. E. Fenton and M. J. I. T. o. s. e. Neil, "A critique of software defect prediction models," vol. 25, no. 5, pp. 675-689, 1999.

[13] N. Padhy, R. Singh, and S. C. J. C. C. Satapathy, "Enhanced evolutionary computing based artificial intelligence model for web-solutions software reusability estimation," vol. 22, no. Suppl 4, pp. 9787-9804, 2019.

[14] Y. Liu, T. M. Khoshgoftaar, and N. J. I. T. o. S. E. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," vol. 36, no. 6, pp. 852-864, 2010.

[15] Q. Song, M. Shepperd, M. Cartwright, and C. J. I. T. o. s. e. Mair, "Software defect association mining and defect correction effort prediction," vol. 32, no. 2, pp. 69-82, 2006.

[16] S. Lessmann, B. Baesens, C. Mues, and S. J. I. t. o. s. e. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," vol. 34, no. 4, pp. 485-496, 2008.

[17] J. C. Munson and T. M. J. I. T. o. s. E. Khoshgoftaar, "The detection of fault-prone programs," vol. 18, no. 5, p. 423, 1992.

[18] X. Huang, J. R. J. P. e. Jensen, and r. sensing, "A machine-learning approach to automated knowledge-base building for remote sensing image analysis with GIS data," vol. 63, no. 10, pp. 1185-1193, 1997.

[19] N. Ohlsson, "Quality improvement by identification of fault-prone modules using software design metrics," in *Proceedings: International Conference on Software Quality, 1996*, 1996, pp. 1-13.

[20] J. C. R. D. Rodríguez, R. Ruiz, and J. S. Aguilar-Ruiz, "Searching for rules to find defective modules in unbalanced data sets," *International Conf.in Symposium Search on Based Software Engineering,* pp. 89–92, 2009.

[21] C. Catal and B. Diri, "Software defect prediction using artificial immune recognition system," in *Proceedings of the 25th conference on IASTED international multi-conference: software engineering*, 2007, pp. 285-290: ACTA Press Anaheim.

[22] J. Wang, B. Shen, and Y. Chen, "Compressed C4. 5 models for software defect prediction," in *2012 12th International Conference on quality software*, 2012, pp. 13-16: IEEE.

[23] Y. Chen, X.-h. Shen, P. Du, and B. Ge, "Research on software defect prediction based on data mining," in *2010 The 2nd International Conference on Computer*

and Automation Engineering (ICCAE), 2010, vol. 1, pp. 563-567: IEEE.

[24] B. Li, B. Shen, J. Wang, Y. Chen, T. Zhang, and J. Wang, "A scenario-based approach to predicting software defects using compressed C4. 5 model," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 406-415: IEEE.

[25] T. J. M. s. Marwala and s. processing, "Probabilistic fault identification using vibration data and neural networks," vol. 15, no. 6, pp. 1109-1128, 2001.

[26] S. R. Chidamber and C. F. J. I. T. o. s. e. Kemerer, "A metrics suite for object oriented design," vol. 20, no. 6, pp. 476-493, 1994.

[27] R. Subramanyam and M. S. J. I. T. o. s. e. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," vol. 29, no. 4, pp. 297-310, 2003.

[28] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 521-530.

[29] T. Systä, *Static and dynamic reverse engineering techniques for Java software systems*. Tampere University Press, 2000.

[30] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, 2006, pp. 18-27.

[31] T. J. Ostrand, E. J. Weyuker, and R. M. J. I. T. o. S. E. Bell, "Predicting the location and number of faults in large software systems," vol. 31, no. 4, pp. 340-355, 2005.

[32] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Proceedings sixth international software metrics symposium (Cat. No. PR00403)*, 1999, pp. 242-249: IEEE.

[33] F. Rosenblatt, "Principles of Neurodynamics Spartan," *New York* 1962.

[34] A. K. J. Hertz, and R. G. Palmer, "Introduction to the Theory of Neural Computation," *Boulder, CO 80309-0430, USA CRC Press,* pp. 6-11, 1991.

[35] C. Shan, B. Chen, C. Hu, J. Xue, and N. Li, "Software defect prediction model based on LLE and SVM," 2014.

[36] G. Y. Y. Xia, X. Jiang, and Y. Yang, "A new metrics selection method for software defect prediction," *IEEE International Conf. on Progress in Informatics and Computing,* pp. 433–436, 2014,.

[37] A. Shrivastava, V. J. I. J. o. C. E. Shrivastava, and Technology, "A hybrid model of soft computing technique for software fault prediction," vol. 4, no. 4, pp. 2511-2518, 2014.

[38] R. Malhotra, N. Pritam, and Y. Singh, "On the applicability of evolutionary computation for software defect prediction," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2014, pp. 2249-2257: IEEE.

[39] A. C. a. S. Dhall, "Software defect prediction using supervised learning algorithm and unsupervised learning algorithm," *Turkish Journal of Physiotherapy and Rehabilitation,* vol. 12, pp. 2429– 2436, 2013.

[40] M. M. Askari, V. K. J. I. J. o. S. E. Bardsiri, and I. Applications, "Software defect prediction using a high performance neural network," vol. 8, no. 12, pp. 177-188, 2014.

[41] M. Singh and D. S. J. I. J. o. C. A. Salaria, "Software defect prediction tool based on neural network," vol. 70, no. 22, 2013.

[42] R. Verma and A. Gupta, "Software defect prediction using two level data pre-processing," in *2012 International Conference on Recent Advances in Computing and Software Systems*, 2012, pp. 311-317: IEEE.