Journal of Neonatal Surgery
ISSN(Online): 2226-0439
Vol. 14, Issue 1s (2025)
https://www.jneonatalsurg.com

OPEN ACCESS

A Novel Adam Hypertuned Artificial Neural Network Using Autoencoder Network Approach to Improve Software Defect Prediction Accuracy

S. Thenmozhi[1], Dr. PM. Shanthi[2]

[1]Research scholar, Department of computer science, J.J. College of Arts and Science (Autonomous), Affiliated to Bharathidasan University, Pudukkottai - 622422, Tamilnadu, India

[2]Assistant professor, Department of computer science. J.J. College of Arts and Science (Autonomous), Affiliated to Bharathidasan University, Pudukkottai - 622422, Tamilnadu, India

Cite this paper as: S. Thenmozhi, Dr. PM. Shanthi, (2025) A Novel Adam Hypertuned Artificial Neural Network Using Autoencoder Network Approach to Improve Software Defect Prediction Accuracy. *Journal of Neonatal Surgery*, 14 (1s), 437-453.

ABSTRACT

Defect prediction is a very active area in the software engineering field. It is crucial to bridge the gap between software engineering and data mining to ensure the programme's success. Predicting software flaws helps find faults in the code before testing is done. Cluster analysis, statistical approaches, mixed algorithms, neural network-based metrics, black box testing, white box testing, and machine learning are only some of the methods used to investigate the software effect area while trying to forecast defects in software. In order to improve the accuracy of deep learning classifiers for defects forecasting, this study makes a novel contribution by using feature selection for the first time. This research was conducted with the hope of enhancing the accuracy with which errors may be predicted in five NASA data sets: CM1, JM1, KC2, KC1, and PC1.Here initially the data was retrieved and processed using rounded mean regressor interpolation approach. Then for selecting feature information grain methodology was used. Dimensionality Component Analysis (DCA), Self-Regulating Component Analysis (SRCA), and Non-Negative Linear Matrix Factorization (NNLMF) were used to extract features from the recovered data . In order to improve upon previous techniques of defect prediction, we combine the factorization selection approach with the deep learning-based adam hypertuned ANN using autoencoder method. All of the tests were run in a python environment. This research shows that, in comparison to the currently used mechanisms, defect prediction accuracy may be increased by the application of feature selection.

Keywords: *Software Defect, deep learning, feature extraction, adam hypertuned ANN using autoencoder network*

1. INTRODUCTION

Software has become an important part of modern technology to the point where almost every business needs some kind of software [1]. But if software bugs are common, the main reason for making it is defeated, and it loses trust and dependability [2]. ally We live in a time when software is very important. Software that has a lot of bugs hurts its trustworthiness and dependability [3, 4]. Customers were unhappy with the programme because of this, which hurt the company's reputation [4, 5]. If faults are found early on in the software development process, resources can only be put towards the parts of the software that need them. As a result, software defect prediction (SDP) is one of the most important parts of the software industry's growth in making good software. Finding bugs in software early on will save time, money, and other resources for businesses. A lot of people are interested in machine learning (ML) models lately because they can help predict software departure [6]. This had to be done so that software goods could be delivered on time and the best use of the resources was made [7]. But software datasets that aren't fair and have a lot of dimensions have made it hard for ML-based models to work well [8]. When looking at binary problems [9], accuracy has been found to be a good way to compare models. However, if there are a lot more instances of one class than another, the accuracy of such a model won't work well. If there are 10 faults out of 100 cases and 90 instances do not have any defects, the evaluation results will be different when an ML-based model is learned on data that is not fair. Without a doubt, the findings will favour the middle-class group. In real life [10, 11], imbalance is a problem that comes up a lot. This method can be used to deal with big datasets that have a lot of traits that classifiers need to choose from before they can make predictions or put the datasets into groups [12]. This is used to find the most important factors that can greatly help in accurately predicting software flaws using different machine learning-based models. Using feature selection methods, you can get rid of the duplicates in a dataset. The classifiers will then get rid of the dataset's non-independent features [13]. The feature selection method will make it easier to get rid of factors that aren't needed or are used more than once in datasets that are used to predict software defects [14]. Picking the right features is a

big part of predicting software bugs. The way this classifier is used has a big effect on how well predictions are made, so it's important to use the right feature selection classifier to reduce the number of dimensions. In order to predict software bugs, this study suggests Dimensionality Component Analysis, Self-Regulating Component Analysis, and Non-Negative Linear Matrix Factorization. Before the feature extraction methods were used, the information grain-based feature selection was used to make the dataset less complicated. Finally, Adam's hypertuned ANN with an autoencoder network predictor was used on the software defect datasets to make predictions about software defects. Using feature selection to lessen the large dimensionality of the software fault dataset is one of the study's main achievements.

(ii) The uneven cases in the dataset were dealt with using the preprocessing method. (iii) Adam's hypertuned ANN with the autoencoder network method was used to correctly identify the flaws and predict software defects.

This article subsequent parts are organized as follows. In Section II, we provide relevant prior works; in Section III, we present the problem statement; in Section IV, we present the implemented proposed approach of software defect detection; and in Section V, we present the results and discussion . The article came to a conclusion in Section VI.

2. RELATED WORKS

“Machine learning (ML) models have been used by several researchers to address binary classification problems in diverse domains, such as rainfall prediction [15, 16], sentiment analysis [17–19], network intrusion detection [11, 20–22], and software defect prediction [1, 2, 4, 23–25]. This research examines many publications pertaining to the Social Development Paradigm (SDP). The investigation conducted by the authors in reference [26] included a comprehensive examination of machine learning-based models used to software defect prediction (SDP). The study utilised twelve meticulously curated datasets sourced from NASA to evaluate the effectiveness of several classifiers. The used algorithms include of Support Vector Machine (SVM), K-Nearest Neighbour (KNN), Naïve Bayes (NB), Radial Basis Function (RBF), Decision Tree (DT), Multi-Layer Perceptron (MLP), and Random Forest (RF). The performance criteria used in evaluating the analysed models include recall, accuracy, receiver operating characteristic (ROC), Matthews correlation coefficient (MCC), and F1-score. The experimental findings indicate that the RF classifier exhibited superior performance in comparison to other classifiers, with the SVM model ranking second. In a comparable study, the researchers in [27] used an ensemble classification model subsequent to using feature selection techniques to mitigate the inclusion of irrelevant characteristics in the dataset utilised for model evaluation. The suggested model was implemented in a binary dimension, using feature selection in conjunction with the classifier. Additionally, the model was applied without utilising the feature selection approach. The NASA datasets were used in conjunction with a range of performance indicators to assess the effectiveness of the suggested models. The outcomes of the model are contrasted with other cutting-edge used forecasting techniques. The findings demonstrated significant enhancements on some datasets; nevertheless, the model exhibited suboptimal performance when confronted with unbalanced class datasets due to inadequate resolution of the underlying problem by the suggested approach. In this study, the authors of reference [28] put forth six classifiers for Semantic Dependency Parsing (SDP) approaches. These classifiers use Principal Component Analysis (PCA) as a means of reducing the dimensionality of the features. The specific algorithms utilised in this study are Holographic Networks, Layered Neural Network, Logistic Regression (LR), and Discriminant Analysis. The performance parameters included in this study are the Misclassification Rate, Verification Cost, Predictive Validity, and Achieved Quality. The outcomes of the presented models demonstrated a perfect accuracy rate of 100%, particularly the model that used Principal Component Analysis (PCA) for reducing dimensionality, exhibiting no errors. In their study, the authors in reference [4] conducted an empirical evaluation of SDP models by using several ensemble approaches with boosting capabilities on three publicly available JAVA projects. The models are assessed using stable performance criteria such as Area Under the Curve (AUC), Balance, and G-Mean. The resampling techniques were used in the JAVA projects, using four ensemble classifiers. The use of resampling methods resulted in improved performance of classifiers in comparison to classifiers that did not employ resampling techniques. This finding demonstrates that the use of resampling techniques has a substantial influence on the performance of ensemble classifiers. Specifically, as compared to traditional boosting classifiers, the implementation of resampling techniques has led to significant improvements in the performance of the SDP models. Among the resampling algorithms used, RUSBoost had superior performance, followed by MSMOTEBoost, while SMOTEBoost demonstrated the least favourable outcomes. In a separate investigation conducted by [1], a model was proposed that utilises several ensemble learning techniques to forecast software module defects. The suggested system integrates a combination rule for ensemble models that incorporates both linear and non-linear approaches. The design and execution of the research used publicly accessible software defect datasets. The system under consideration shown a notable ability to accurately forecast software problems, yielding consistent and dependable outcomes across the various datasets used for performance assessment. The prediction at level l, denoted as Pred(l), was then used in the ensemble classifiers to assess the extent of the findings' comprehensiveness. The findings indicate that the average relative inaccuracy of the quantity of modules inside a dataset is either less than or equal to a specified threshold value, denoted as "l". The present research and assessment, using the metric, have substantiated the efficacy of the suggested approach in accurately predicting software defects. The performance of ensemble approaches shown improvement in comparison to the single fault prediction method for the prediction of software problems. The primary

contribution of the approach is in its ability to rapidly detect software flaws by effectively using testing resources. In their study, the authors introduced a model for SDP that encompasses four distinct stages: (i) feature selection, (ii) pre-processing, (iii) classification, and (iv) reflection of outcomes, as documented in reference [23]. The use of a feature selection model was employed to exclude unnecessary features from the dataset prior to the application of ensemble learning classifiers. The authors used the NASA MDP datasets that had been cleansed for the purpose of implementing the suggested model. Multiple performance criteria, including as accuracy, F1-score, Matthews correlation coefficient (MCC), and receiver operating characteristic (ROC), are used to evaluate the model. The model underwent testing on each dataset in order to compare and determine the greatest scores among the six datasets. A comparative analysis was conducted between the suggested model and ten other supervised classifiers. The search techniques and outcomes of the proposed system demonstrated superior performance in comparison to all other classifier approaches. In a scholarly investigation conducted by the authors [29], the use of Support Vector Machines (SVM) was presented as a means to forecast software errors. To develop the model, NASA datasets were employed. The suggested framework was evaluated in comparison to various models, including logistic regression (LR), k-nearest neighbours (K-NN), random forest (RF), naive Bayes (NB), radial basis function (RBF), and multilayer perceptron (MLP). The findings indicate that the suggested technique exhibited superior performance compared to certain categorization methods used for performance evaluation. The authors in reference [30] have shown the significance of feature selection in SDP systems by demonstrating that some parameters or features possess more relevance compared to others. The artificial neural network (ANN) is equipped with feature selection techniques to facilitate the deployment of the framework. The chosen characteristics are used in order to make predictions on the SDP via the implementation of ANN classifiers. The performance of the suggested technique was evaluated using the Gaussian Kernel Support Vector Machine (SVM) and the JM1 NASA dataset. Based on the outcomes obtained from the suggested model, it can be concluded that the Support Vector Machine (SVM) exhibited superior performance compared to the other model in the classification of defects in the twofold scenario. The authors in reference [31] used a hybrid approach that combined a Genetic Algorithm (GA) with a Deep Neural Network (DNN) for solving the Semi-Definite Programming (SDP) problem. This approach was evaluated using many datasets sourced from the PROMISE repository. The Hybrid Genetic Algorithm (GA) is used for the purpose of feature selection in order to identify the most effective characteristics for the model. Additionally, the Deep Neural Network (DNN) is utilised to carry out the prediction of the system. The outcomes of the suggested model shown superior performance compared to other methodologies used for evaluating the model's efficacy. Based on previous studies, it has been shown that the presence of unbalanced software data might impede the effectiveness of models, resulting in inaccurate interpretations of outcomes. Additionally, datasets with a large number of features can diminish the efficiency of ensemble methods. Hence, this research presents a hybrid model that encompasses feature selection, data normalizations, and XGBoost algorithm for the purpose of classifying software products and predicting the presence of flaws inside them.

3. PROPOSED WORK

This work examines the efficacy of the proposed learning classifiers in the context of software defect prediction, using benchmark datasets from NASA. Each dataset has many characteristics in addition to a predefined output class. The output or target class is determined by prediction using other accessible features. The characteristic that is being predicted is referred to as the dependent attribute, whereas the other attributes utilized to forecast the dependent attribute are referred to as independent attributes. The datasets used for this research consist of a dependent characteristic that is characterized by values of either "Y" or "N". The symbol "Y" denotes the presence of defects in a particular software instance or module, whereas the symbol "N" indicates the absence of defects. This study use a total of five datasets sourced from NASA for experimental purposes. The datasets include CM1, JM1, KC1, KC2, and PC1, as shown in Table I. Each dataset that has been chosen reflects a software system developed by NASA. These datasets consist of several metrics that are strongly associated with software quality. The suggested method architecture was illustrated in figure 1.

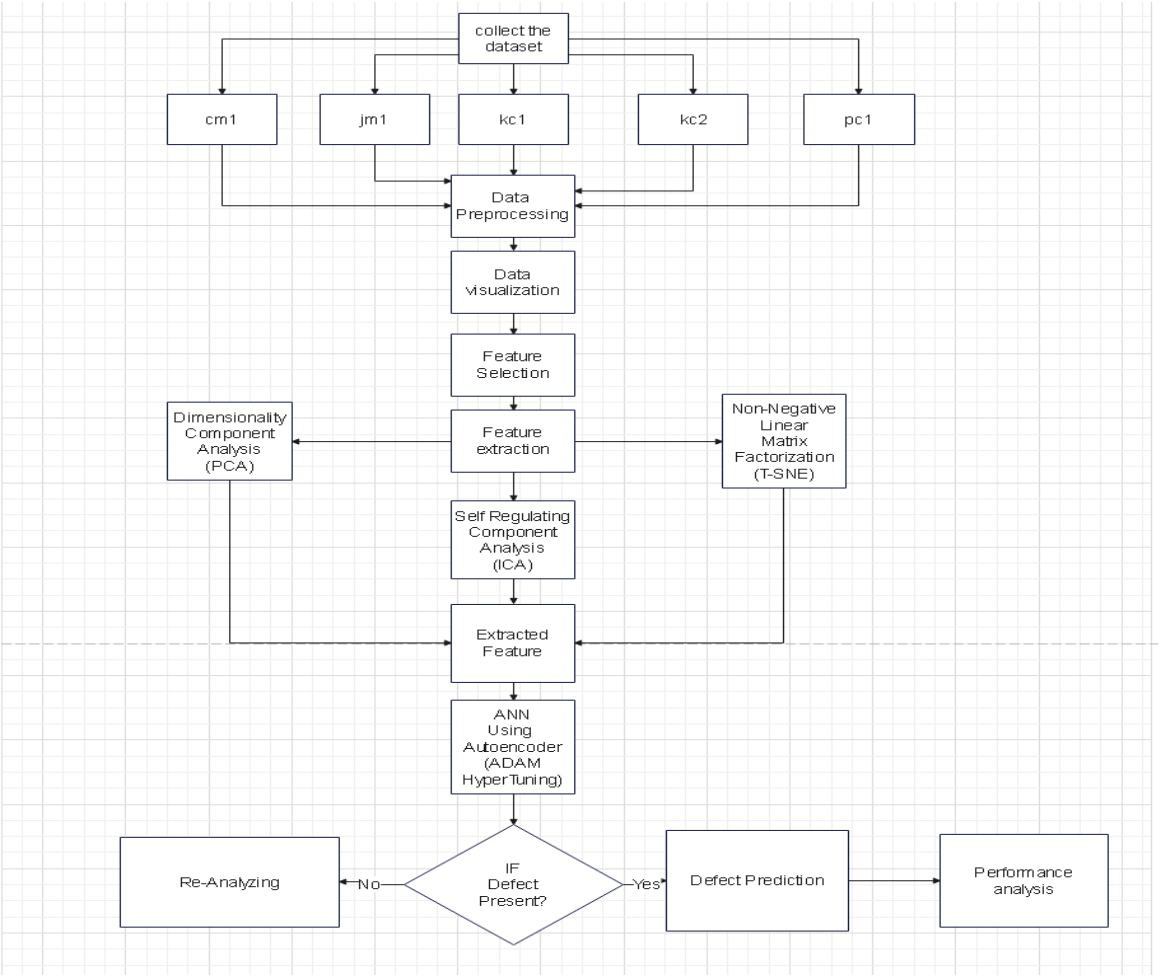

Figure 1 Schematic representation of the suggested methodology

Table 1 Description of the dataset

Dataset	Attributes	Modules	Defective	Non-Defective	Defective (\%)
CM1	38	327	42	285	12.8
JM1	22	7,720	1,612	6,108	20.8
KC1	22	1,162	294	868	25.3
KC2	40	194	36	158	18.5
PC1	38	679	55	624	8.1

3.1 Data preprocessing

In order to get suitable data for the proposed Software Development Process (SDP) framework, many pre-processing processes were performed on the dataset provided by NASA. The below procedures are undertaken to modify the dataset used for the purpose of this investigation:

(i) The elimination of duplicated instances refers to the identification and removal of software modules that possess identical class labels and software metric values, particularly in cases when these instances are associated with faulty labels. In the context of a practical issue, this scenario is very conceivable. Unfortunately, machine learners may have severe consequences when encountering multiple instances, which may lead to overoptimistic results if these instances are correctly categorised as part of the test data. Misclassifying these presentations as part of the test data might lead to excessively negative outcomes.

Moreover, this phenomenon might lead to a significant increase in the duration of the training process, hence impeding the enhancement of the models and classifiers' performance. Therefore, it is essential to eliminate duplicate instances inside the datasets.

(ii) Relocation of Misplaced Labels: It is common to encounter several instance values associated with different software metrics. Multiple missing values might occur for a given occurrence, particularly when data collection was conducted in a casual manner, due to many factors. It has become more common for instances to be unable to meet the input requirements of the proposed model due to the presence of missing data that need to be processed. In the present research, in the event of any missing value, it is deemed appropriate to substitute such value with the mean of the corresponding measure. For example, let us consider a scenario where both nt_{99} and $nt_{100}$are absent. In this case, we have a metric mt and its corresponding explanations $\{nt_1, \ldots\ldots, nt_{100}\}$. The two missing numbers may be substituted by using Equation (1).

$$nt_{99} = nt_{100} = \frac{1}{98}\sum_{i=1}^{98} nt \qquad (1)$$

(iii) Data Standardization: The values of different software metrics exhibit significant variations in magnitude, necessitating the use of data normalisation techniques to standardise these metrics. The research used the widely utilised min-max normalisation approach to convert all values and normalise the data within the range [0,1].

The metric x exhibits a one-to-one relationship with the maximum and minimum values, which are represented as max(y) and min(y) respectively. The computation of the value $\tilde{x}_i$for each value of $y_i$of metric y is possible.

$$\tilde{y}_i = \frac{y_i - \min(y)}{\max(y) - \min(y)} \qquad (2)$$

Subsequently, the raw data undergoes preprocessing. Data visualization refers to the systematic procedure of generating a visual depiction of data with the objective of enhancing comprehension of its underlying information. The evaluation of quality within visualization methods lacks a well-defined mathematical standard. The data is presented in a visually accessible style that highlights distinct bits of information.

3.2 Feature selection

The entropy-based selection approach known as Information Gain (IG) includes the computation of gain (y, A) from the output data that is categorised by feature A. The representation of the Information Gain (y,A) is as follows,

$$\text{gain}\ (y, A) = \text{entropy}\ (y) - \Sigma_{C \in \text{vals}\ (A)} \frac{y_c}{y} \text{entropy}\ (yc) \quad (3)$$

The variable (A) represents the range of potential values for characteristic A, with Yc denoting the subset of y in which A has a cumulative total of c. Moreover, Eq. (3) governs the calculation of the overall entropy of variable y, which is afterwards used for the purpose of data segregation, specifically with respect to feature A.

The determination of the threshold value may be done independently or by using a predetermined value of 0.05. The final feature's threshold value was obtained by calculating the average of each data frequency.

The procedure for assessing the variety of data groups entails the reduction of information value by means of calculating the average of associations, followed by the summation of the obtained outcomes. The technique used in this context is often referred to as standard deviation, which quantifies the extent to which the measured data deviates from the mean value. The present research utilises the data group as a measure of the informational significance of each characteristic inside a given dataset, as determined by the use of Equation (4).

$$S = \sqrt{\frac{d\sum_{i=1}^{d} x_i^2 - \left(\sum_{i=1}^{d} y_1\right)^2}{n(n-1)}} \qquad (4)$$

In the given context, S represents the standard deviation, x denotes the average value of the IG (Information Gain), xi signifies the rate of x to i, and n represents the total number of features used in the dataset.

3.3 Feature extraction

The total number of features may be specified as an input for three distinct feature selection methods, which are outlined below,

2.3.1 DCA

By using the DCA technique on X_{aug} , the DCs (Dynamic Connectivities) may be derived using the usual DCA process. The dimension of the resulting DC is$1 \times \left(N_{\text{input}} + N_{\text{class}}\right)$. To decrease the input dimension to$N_{\text{feature}} < N_{\text{input}}$, we choose for N_{feature} discriminant components (DCs) that effectively capture the N_{feature} biggest variances. Within the context of this discourse, the variable W is defined in the following manner.

$$W_{\text{Sort}} = \begin{bmatrix} DC_1^T & DC_2^T & \cdots & DC_{N_{\text{feature}}}^T \end{bmatrix} = \begin{bmatrix} W_{\text{input}} \\ W_{\text{class}} \end{bmatrix} \quad (5)$$

In this context, the superscript T denotes the transposition operation applied to a matrix. The size of DC_i is $1 \times (N_{\text{input}} + N_{\text{class}})$, whereas the dimension of W_{sort} is $(N_{\text{input}} + N_{\text{class}}) \times N_{\text{feature}}$. The terms W_{input} and W_{class} refer to the input and the class, respectively, within the context of the enhanced input.

The modified data on the feature space may be obtained by multiplying the W_{sort} with the augmented data X_{aug}

$$X_{\text{feature}} = [X \quad C(X)] \times \begin{bmatrix} W_{\text{input}} \\ W_{\text{class}} \end{bmatrix} = X \times W_{\text{input}} + C(X) \times W_{\text{class}} \quad (6)$$

The equation mentioned above cannot be applied to data instances with unknown class labels, since it relies on the presence of the class label C(X). Based on the observation that the factors of matrix W in the k-th column and j-th row (where k ranges from 1 to N_{input} and j ranges from 1 toN_{feature}) are much lower compared to the other factors of W, we may derive the following approximate equation for the calculation of X_{feature}.

$$X_{\text{feature}} \cong X \times W_{\text{input}} \quad (7)$$

The aforementioned equation has the capability to be used for any variable X, regardless of its unknown classification, in order to get the matching value inside the feature space.

2.3.2 SRCA

The SRCA method is a computer methodology used for nonlinear feature extraction. It involves subdividing a signal into its statistically independent components, often known as ICs. The study employs SRCA as a method for lowering the dimensionality of the original features and extracting the essential independent characteristics from the provided signal. The ICs of a dataset are determined by the execution of the subsequent mathematical calculation.

1. Let us consider a scenario where there are n linear vectors that are obtained from a combination of $N_1, N_2, \ldots, N_k$, each of which represents k observations. Likewise, the source vector x is constructed using the elements $x_1, x_2, \ldots, x_k$. The weighted matrix, designated as W, is represented by components $a_{i,j}$. The mixing model may be expressed as follows.

$$N = Wx \quad (8)$$

The model may also be written with the segments of the matrix W represented by the symbols a_j.

$$N = \sum_{i=1}^{n} a_i x_i \quad (9)$$

SRCA is the name of the factual model represented by Eq. (9). SRCA's status as a generative model is therefore confirmed. This means that it shows how the things that viewers see are made via a process of combining various elements of x_i.

2. Then, by evaluating the matrix W, we can obtain the IC defined by its inverse, say A.

$$N = Ax \quad (10)$$

where W and A are complementary opposites of one another.

In this study, the characteristics are retrieved in order to carry out the SRCA calculations.

2.3.3 NNLMF

In the context of a vector $\bar{v}$ containing N high-dimensional points $z_1, z_2, \cdots z_n$, the calculation of the Euclidean distances between any two points z_i and z_j inside the vector $\bar{v}$ is used to derive a conditional probability $P_{j|i}$. This conditional probability serves as a measure of resemblance between the point z_i and the point z_j. To clarify, the conditional probability $P_{j|i}$denotes the chance that the point z_i would choose z_j as its neighbour, assuming that the probability density of the features follows a normal distribution (Gaussian) and is centred at the point z_i. Therefore, the conditional probability exhibits an increase when considering neighbouring data points, however for data points that are far apart, $P_{j|i}$becomes almost negligible. Mathematically, the conditional probability $p_{j|i}$ may be denoted as such,

$$p_{j|i} = \frac{\exp\left(-\|z_i - z_j\|_2 / 2\sigma_i^{\ 2}\right)}{\sum_{k \neq i} \exp\left(-\|z_i - z_j\|^2 / 2\sigma_i^{\ 2}\right)} \quad (11)$$

where σ_i is the mean of the Gaussian distribution centred at the positionx_i.

The conditional probability of a point being next to itself is 0 since the method focuses on representing pairwise similarities $P_{i|i} = 0$.

A conditional probability, $q_{j|i}$ may be derived to characterise the similarities between the map feature points y_i and y_j, which are the low-dimensional analogues of the high-dimensional z_i and z_j, respectively.

$$q_{j|i} = \frac{\exp\left(-\|z_i - z_j\|_2\right)}{\sum_{k \neq i} \exp\left(-\|z_i - z_j\|^2\right)} \quad (12)$$

Since this method is solely interested in simulating pairwise similarities, the conditional probability $q_{i|i}$is similarly zero ($q_{i|i}$=0).

In order to obtain a low-dimensional representation of the data that minimises the discrepancies between $p_{j|i}$ and $q_{j|i}$, the dimensionality reduction mapping is performed. The gradient descent technique is used frequently in NNLMF to achieve this for a certain cost function C, such that

$$C = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{i|i}}{q_{i|i}} \quad (13)$$

The Kullback-Leibler divergence function of $P_i \parallel Q_i$is denoted by $KL(P_i \parallel Q_i)$. The Kullback-Leibler divergence, denoted as $KL(P_i \parallel Q_i)$, is the distance between two discrete probability distributions P_i and Q_i, and it is defined as,

$$KL(P_i \parallel Q_i) = -\sum_{x \in X} P_i(z) \log\left(\frac{Q_i(z)}{P_i(z)}\right) \quad (14)$$

This is equivalent to

$$KL(P_i \parallel Q_i) = \sum_{x \in X} P_i(z) \log\left(\frac{P_i(z)}{Q_i(z)}\right) \quad (15)$$

The expectation of the logarithmic difference between P_i and $Q_i$is given by Equation (5) above. Any random continuous variable x in P_i and $Q_i$may be treated in the same way.

$$KL(P_i \parallel Q_i) = \int_{-\infty}^{\infty} p_{j|i}(z) \log\left(\frac{p_{j|i}(z)}{q_{j|i}(z)}\right) dz \quad (16)$$

where $p_{j|i}$ and $q_{j|i}$are the distributions of likelihoods of P_i and Q_i.

The Kullback-Leibler divergence function may be rewritten as when P_i and $Q_i$are evaluated over continuous sets X and P.

$$KL(P_i \parallel Q_i) = \int_z \log\left(\frac{dP_i}{dQ_i}\right) dP_i \quad (17)$$

where $\frac{dP_i}{dQ_i}$ in (7) is a derivative of Radon and Nikolayevich Nikodym of P_i with respect to Q_i.

By using the chain rule for factorization, we can rewrite (17) as

$$KL(P_i \parallel Q_i) = \int_X \log\left(\frac{dP_i}{dQ_i}\right) \frac{dP_i}{dQ_i} dQ_i \quad (18)$$

It is commonly agreed that the entropy of P_i with respect to $Q_i$is given by the preceding equation.

If we have two absolutely continuous probability densities $p_{j|i}$and $q_{j|i}$such that$p_{j|i} = \frac{dP_i}{d\mu}$ and $q_{j|i} = \frac{dQ_i}{d\mu}$, then the Kullback-Leibler divergence from Q_i to $P_i$is given by for every measure on the set z.

$$KL(P_i \parallel Q_i) = \int_z \log\left(\frac{p_{j|i}}{q_{j|i}}\right) d\mu \quad (19)$$

Recursively using a gradient descent algorithm, the following form is used to minimise the cost function in (19):

$$\frac{\delta C}{\delta y_i} = 2 \sum_j \left(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}\right)\left(z_i - z_j\right) \quad (20)$$

z_i and z_j are located at different coordinates on a map.

In order to update the map's coordinates at each iteration, the factorised gradient is added to an exponentially decaying sum of prior gradients. The specified formula controls this update

$$z^t = z^{(t-1)} + \beta \frac{\delta C}{\delta y_i} + \alpha(t)\left(z^{(t-1)} - z^{(t-2)}\right) \quad (21)$$

where z^t represents the gradient value at iteration t is denoted as β, the learning rate is represented by α, and α(t) is a substantial momentum factor that is included into the gradient to enhance the identification of local minima.

It is crucial to note that the computational complexity of Non-Negative Matrix Factorization is of the order O(N^2). Nevertheless, the application of the cost function (Equation (3)) of NNLMF is constrained due to its quadratic scaling with respect to the number of objects N. Consequently, it is only suitable for datasets containing a small number of input objects, typically in the range of a few thousand. As the size of datasets increases, it is anticipated that the learning process would experience a decrease in speed, while the memory needs will see an increase. Due to recent advancements in high-end computer technology and its increased affordability, the execution of NNLMF on extensive datasets may now be accomplished within a matter of minutes.

d. Defect prediction

The features obtained from the three different methodologies are separately given as a input to this optimized classifier. The current study is centred on the proposition of an adam hypertuned ANN with an autoencoder. The outputs of the routes are combined and then fed into another pathway, which utilises a distinct set of network layers, in order to get the ultimate output.

The use of autoencoders in the hyperparameter tuning of ANNs is grounded on the widespread adoption of ANNs as a prominent architecture for data processing. Typically, this neural network model takes into account the characteristics derived from the dataset. The proposed architecture utilises a dataset containing features denoted as $f_k(k = 1,2,3,4 \dots \dots)$, where k represents the index of the network layer.

Every layer consists of a set of neurons and an activation function. The characteristics are convolved using several network layers, referred to as convolutional pathways, and the resulting output of these convolutional routes is then concatenated to get the final resultant output.

A network consisting of several layers, with a certain number of layers denoted as "l", and an encoder function σ_l

Step 1: To ascertain the output of the Hidden layer, it is necessary to determine its corresponding values $h_l = \sigma_l(W_l{}^T h_{l-1} + b_l)$ and the network $\hat{z} = hl$

Step 2: Compute the gradient $\delta = \frac{\partial \varepsilon(z_i \hat{z_l})}{\partial_y}$

for i ⟵ l to 0 do

Regarding the computation of the current layer gradient:

$$\frac{\partial \varepsilon(z,\hat{z})}{\partial W_l} = \frac{\partial \varepsilon(z_i \tilde{z_l})}{\partial h_1}\frac{\partial h_1}{\partial W_1} = \delta \frac{\partial h_1}{\partial W_1}$$
$$\frac{\partial \varepsilon(z,\hat{z})}{\partial b_l} = \frac{\partial \varepsilon(z_i (\tilde{z_l})}{\partial h_1}\frac{\partial h_1}{\partial b_1} = \delta \frac{\partial h_1}{\partial b_1} \quad (22)$$

Apply gradient descent using $\frac{\partial \varepsilon(z,\tilde{z})}{\partial w_l} = \frac{\partial \varepsilon(z,\tilde{z})}{\partial b_l}$

Step 3: The gradient is propagated backwards to the lowest layer.

$$\delta \longleftarrow \frac{\partial \varepsilon(z_i \tilde{z_l})}{\partial h_1}\frac{\partial h_1}{\partial h_{l-1}} = \delta \frac{\partial h_1}{\partial h_{l-1}} \quad (23)$$

Step 4: In the above experimental setup, the hyperparameters of the model A are fine-tuned for each batch of z_i, with the corresponding target variable y_i.

$$\hat{z_l} = Z(\theta, x);$$
$$\theta = \theta - \eta \cdot \frac{1}{M}\sum_{i=1}^{M} \frac{\partial \varepsilon(z_i \hat{z_l})}{\partial \theta} \quad (24)$$

Table 2 Hyperparameters of the suggested framework

Hyperparameters	
"Number of layers	16
Activation function	ReLU

Optimizer	Adam
Batch size	32,64,128
Learning rate	0.00010
Loss function	Mean_squared_error"

The Adam divisive algorithm is a well-known adaptive learning rate optimisation approach. The optimisation technique under consideration was especially devised for the purpose of enhancing deep learning methodologies. The algorithm's main use lies in its capacity to ascertain personalised adaptive learning rates for a diverse array of factors. The nomenclature of the method is derived from its procedure of adaptive moment estimation. In a deep neural network, the learning rate for each weight is improved separately by using first- and second-moment gradient estimations. The concepts of mean and variance may be seen as the first and second moments, respectively. The Adam optimisation approach employs exponentially weighted moving averages to estimate the moments inside each batch throughout each iteration. The selection rule for the divisive parameter value of the Adam optimizer may be used to elucidate certain mathematical calculations. This rule is outlined as follows:

$$E = \{R_1, R_2, R_3, \ldots., R_n\} \quad (25)$$

The tuning rule is defined as follows

$$V_j(E_i) = \begin{cases} z_{j,i} & (1 \le j \le m, 1 \le i \le n) \\ & (1 \le j \le m, 1 \le i \le n) \end{cases} \quad (26)$$

The last stage of the procedure is modifying the learning rate for each parameter by using moving averages. To determine the weighted value update, we may use Equation (27).

$$U_t = U_{t-1} - \alpha \frac{\overline{N_t}}{\sqrt{\overline{U_t}+\epsilon}} \quad (27)$$

The default value for α, denoted as the learning rate or step size, is set to 10^(-8) in the context where t represents the number of iterations and w represents the tuned value weight.

Equation (28) illustrates the methodology for selecting the hyper-tuned value attributes.

$$Y_n(e^{j\Omega}) = \sum_{m=-\infty}^{\infty} Y[N]U[n-m]e^{-j\Omega m} \quad (28)$$

The overall temporal complexity of the proposed framework is evaluated by analysing several factors, such as the number of layers, the depth of each layer, and the spatial extent of the output map. This evaluation is expressed mathematically in Equation 29.

$$O\left(\sum_{k=1}^{d} n_{k-1} \cdot s_k^2 \cdot n_k \cdot m_k^2\right) \quad (29)$$

where

"k = index of a layer

d = depth of the layer

n_k = number of parameters in the kth layer

n_{k-1} = number of input channels of the k^{th} layer

s_k = length / spatial size of the tuned value

m_k = spatial size of the output map"

The training time for each input is typically three times longer than the testing time, as measured for both forward and backward propagation. During the forward propagation phase, the performance of the proposed framework is evaluated by the computation of the loss using the cross-entropy method. On the other hand, the backpropagation phase involves the update of kernels and weights.

The updating of kernels and weights is performed based on the loss value. Hence, the procedure is used to iterate through the trainable parameters (kernels and weights) in order to minimise the loss. The process involves the use of a gradient loss function, which guides the model in the direction of the steepest rise rate. Additionally, various hyperparameters, such as the

Adam optimizer, are selected prior to commencing the actual training process. Additionally, it is advised to use fixed input and output layers in the proposed network, with an emphasis on using Fully Convolutional Networks (FCN) instead of Fully Connected (FC) layers.

To predict the probability of defect,

$$\begin{aligned} j_t &= \sigma\left(Kh_{t,Bi-MDSPN} + b\right) \\ H_t &= \text{ReLU}\left(\tilde{J}h_{t,Bi-MDSPN_W} + \tilde{b}\right) \\ C_t &= 1 - G_t \\ O_t &= k_t G_t + h_{t,Bi-MDSPN_W} C_t \end{aligned} \quad (30)$$

The weight matrices G, as well as the bias vectors b and $\tilde{b}$,, are shown. The activation function is often represented by the acronym "ReLU." The degree of output transformation is regulated by the encoder and decoder layers of G_t. The resultant vector is represented by the symbol O_t. The software problem may now be accurately identified.

4. PERFORMANCE ANALYSIS

This study utilises publicly accessible benchmark datasets from the NASA promise repository to examine the enhancement of defect prediction accuracy with the use of feature selection.

The objective of this section is to evaluate the effectiveness of various categorization approaches. The sample input and the data visualization process was done that was illuystrated in figure 2 and 3

```
cm1.dropna()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	
2	24.0	5.0	1.0	3.0	63.0	309.13	0.11	9.50	32.54	2936.77	...	1	0	6	0	1
3	20.0	4.0	4.0	2.0	47.0	215.49	0.06	16.00	13.47	3447.89	...	0	0	3	0	1
4	24.0	6.0	6.0	2.0	72.0	346.13	0.06	17.33	19.97	5999.58	...	0	0	3	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
493	47.0	3.0	1.0	3.0	256.0	1563.78	0.04	28.00	55.85	43785.90	...	2	13	2	0	2
494	24.0	4.0	3.0	3.0	107.0	587.63	0.05	19.13	30.72	11241.58	...	1	7	4	0	2
495	82.0	11.0	3.0	10.0	475.0	3155.83	0.02	44.71	70.59	141084.24	...	9	59	35	0	3
496	10.0	2.0	1.0	1.0	32.0	150.41	0.15	6.50	23.14	977.69	...	1	12	4	0	1
497	28.0	6.0	5.0	5.0	104.0	564.33	0.06	16.09	35.08	9078.38	...	2	7	0	0	2

498 rows × 22 columns

Figure 2 Sample data input from NASA dataset

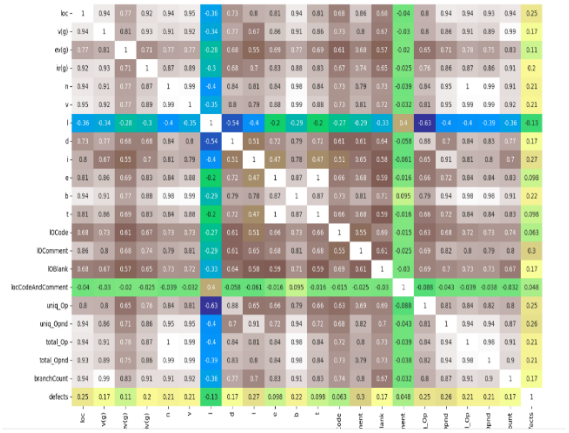

Figure 3 Process of data visualization

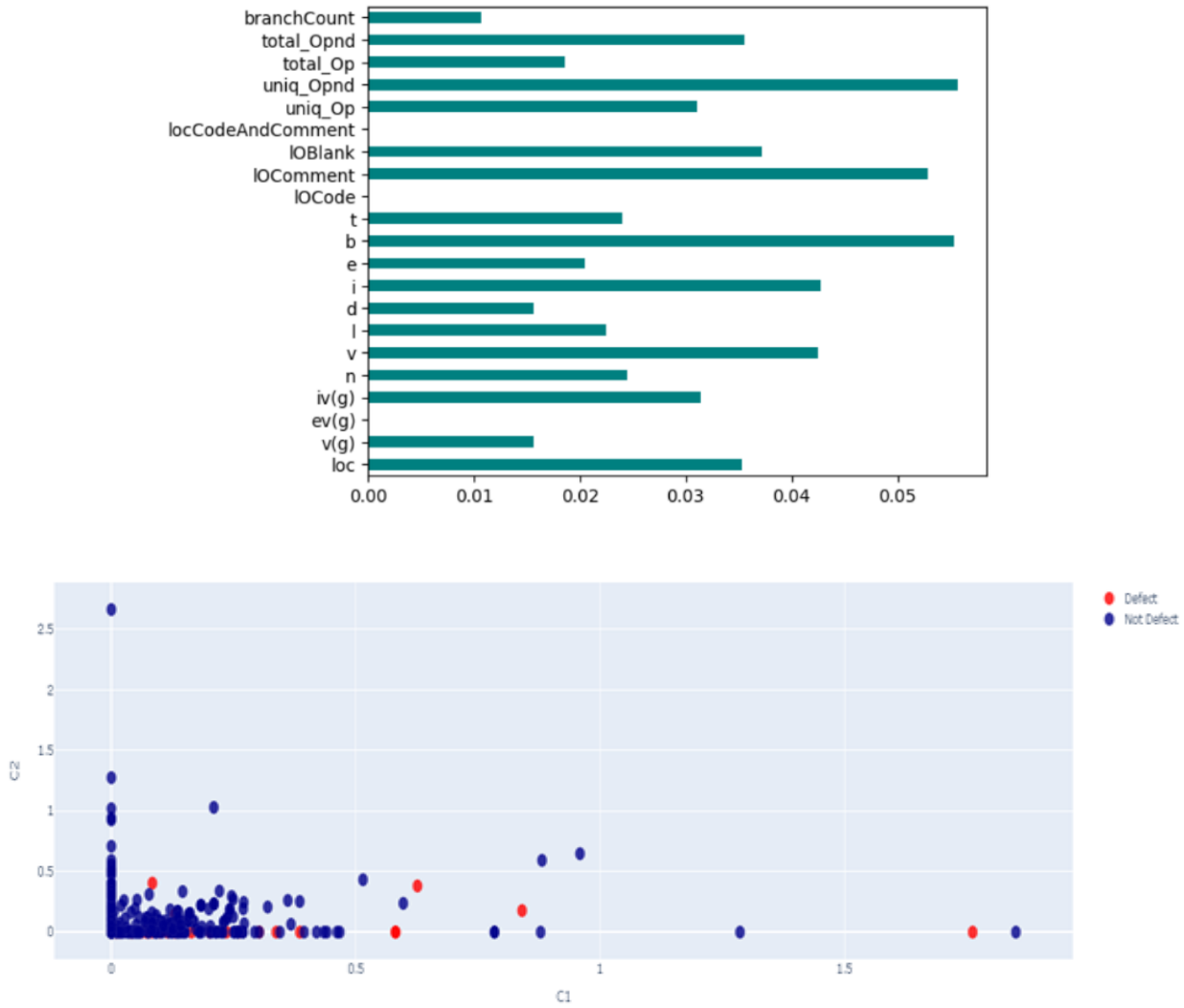

Figure 4 Classified output

Depend upon the features selected the defect module was classified by using the suggested methodology as depicted in figure 4.

The performance of the system is assessed and appraised using a range of metrics derived from the confusion matrix. A confusion matrix is composed of the following parameters:

A true positive (TP) refers to instances that are both truly positive and correctly labelled as positive.

False positives (FPs) refer to instances that are labelled as positive, although being genuinely negative.

False negatives (FN) refer to instances that are really positive but are incorrectly labelled as negative.

True Negative (TN) refers to events that are really negative and are correctly classed as negative.

The evaluation of classification algorithms is conducted using the following measures: The evaluation metrics often used in information retrieval and learning algorithm tasks include precision, recall, F-measure, and accuracy.

Precision may be defined as the proportion of True Positive modules in relation to the total number of modules that are categorised as positive.

$$\text{Precision} = \frac{TP}{(TP+FP)} \qquad (31)$$

Recall may be defined as the proportion of True Positive modules in relation to the overall number of modules that are really positive.

$$\text{Re call} = \frac{TP}{(TP+FN)} \quad (32)$$

F-measure provides the average of Precision & Recall .

$$\text{F-measure} = \frac{\text{Precision} * \text{Recall} * 2}{(\text{Precision} + \text{Recall})} \quad (33)$$

Accuracy refers to the degree to which a forecast aligns with the actual outcome, reflecting the level of precision in the prediction. .

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (34)$$

```
1.056791772000004
[[120  11]
 [ 17   2]]
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       131
           1       0.15      0.11      0.12        19

    accuracy                           0.81       150
   macro avg       0.51      0.51      0.51       150
weighted avg       0.78      0.81      0.80       150
```

Figure 5 .CM1 dataset Classification report

```
10.242347178999978
[[3067  233]
 [ 501  161]]
              precision    recall  f1-score   support

           0       0.86      0.93      0.89      3300
           1       0.41      0.24      0.30       662

    accuracy                           0.81      3962
   macro avg       0.63      0.59      0.60      3962
weighted avg       0.78      0.81      0.79      3962
```

Figure 6 .JM1 dataset Classification report

```
1.8007313609999755
[[490  45]
 [ 67  31]]
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       535
           1       0.41      0.32      0.36        98

    accuracy                           0.82       633
   macro avg       0.64      0.62      0.63       633
weighted avg       0.81      0.82      0.81       633
```

Figure 7 KC1 dataset Classification report

```
1.3021623450000561
[[106  13]
 [ 16  22]]
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       119
           1       0.63      0.58      0.60        38

    accuracy                           0.82       157
   macro avg       0.75      0.73      0.74       157
weighted avg       0.81      0.82      0.81       157
```

Figure 8 KC2 dataset Classification report

```
1.814995052999997
[[294  13]
 [ 17   9]]
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       307
           1       0.41      0.35      0.38        26

    accuracy                           0.91       333
   macro avg       0.68      0.65      0.66       333
weighted avg       0.90      0.91      0.91       333
```

Figure 9 PC1 dataset Classification report

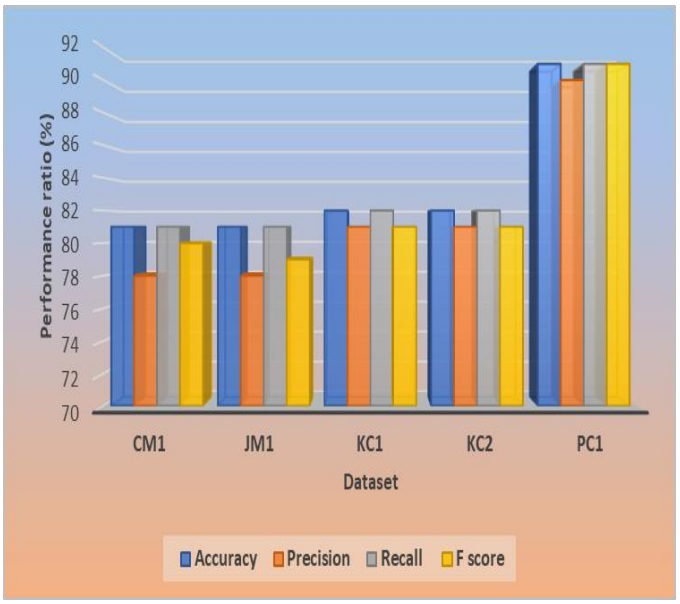

Figure 10 Performance analysis of the suggested methodology

The classification performance outcome ratio of the suggested methodology over 5 different datasets was demonstrated in figure 5-10

Table 3: CM1 dataset comparative performance analysis

Methodology	Precision	Recall	F-measure	Accuracy
LM	0.9110	0.91130	0.9113	91.13150

BFG	0.8831	0.8777	0.880400	88.07340
CG	0.8634	0.9083	0.8852	88.22630
Proposed	78	81	80	81

Table 3 presents the findings obtained from the CM1 dataset. The results indicate that the suggested approach achieved a comparable performance to existing training methods, as shown by an Accuracy score of 81%.

Table 4: JM1 dataset comparative performance analysis

Methodology	Precision	Recall	F-measure	Accuracy
LM	0.79980	0.8056	0.80270	80.1943
BFG	0.79080	0.8132	0.80180	79.9028
CG	0.80220	0.7929	0.79750	79.8705
Proposed	0.78	0.810	0.79	0.81

Table 4 presents the findings obtained from the analysis of the JM1 dataset. The results indicate that the suggested approach achieved a comparable performance to existing training methods, as shown by an Accuracy score of 81%.

Table 5: KC1 dataset comparative performance analysis

Methodology	Precision	Recall	F-measure	Accuracy
LM	0.80430	0.8029	0.80360	80.37870
BFG	0.78010	0.7909	0.78550	78.3993
CG	0.7873	0.7771	0.7822	78.3563
Proposed	0.81	0.82	0.81	0.82

The outcomes for the KC1 dataset are shown in Tab 5. With an Accuracy score of 82%, it is clear that the suggested technique performed similarly to previous training methods.

Table 6: KC2 dataset comparative performance analysis

Methodology	Precision	Recall	F-measure	Accuracy
LM	0.94330	0.94330	0.9433	94.3299
BFG	0.8670	0.907200	0.8866	88.40210
CG	0.828100	0.8196	0.8238	82.4742
Proposed	0.810	0.82	0.81	0.82

The KC2 dataset's final findings are shown in Tab 6. With an Accuracy score of 82%, it is clear that the suggested technique performed similarly to previous training methods.

Table 7: PC1 dataset comparative performance analysis

Methodology	Precision	Recall	F-measure	Accuracy
LM	0.93810	0.93810	0.93810	93.8144
BFG	0.93230	0.93230	0.93230	93.2253

CG	0.93230	0.93230	0.93230	93.2253
Proposed	90	91	91	91

Table 7 presents the findings pertaining to the PC1 dataset. The results indicate that the suggested approach achieved a comparable performance to existing training methods, as shown by an Accuracy score of 91%.

Table 8: Evaluation of the suggested method's accuracy in comparison to existing classification strategies [36]

Dataset	MLP	NB	SVM	RBF	kStar	kNN	PART	OneR	RF	DT	Proposed
CM1	86.7347	82.65310	90.81630	90.81630	77.5510	77.5510	90.8163	85.714300	89.7959	77.55100	81
JM1	80.35410	79.83590	79.18830	80.3972	75.9931	73.96370	79.49050	77.158090	80.18130	79.10190	81
KC1	77.3639	74.2120	75.35820	78.7966	72.2063	69.341	76.5043	73.35240	77.937	75.6447	82
KC2	82.75860	81.03450	82.75860	77.58620	75.86210	75.86210	79.31030	82.75860	77.58620	75.86210	82
PC1	96.56860	89.70590	95.0980	94.60780	86.27450	92.64710	93.13730	94.60780	96.07840	93.13730	91

Based on the findings shown in Table 8, the proposed approach demonstrates superior performance by achieving a much higher level of efficiency compared to other currently used mechanisms". GAK385YA51

5. CONCLUSION

The present work introduces a novel model for the anticipation of software errors that are likely to occur inside software modules. The timely identification of software module defects may significantly mitigate the expenses associated with the software development life cycle. The feature selection method in the proposed system involves the use of Dimensionality Component Analysis, Self-Regulating Component Analysis, and Non-Negative Linear Matrix Factorization techniques. These techniques are employed to identify and discard unnecessary characteristics that do not contribute significantly to the classification process of the model. The use of data normalization was employed in order to mitigate the issue of excessive dimensionality within the datasets utilised, hence enhancing the performance of the suggested model. Adam used a hyper-tuned artificial neural network (ANN) that employed an autoencoder network to forecast the software metrics module using four datasets provided by NASA. The evaluation of the model's performance on the five datasets revealed that the adam hypertuned artificial neural network (ANN), which used an autoencoder network with Non-Negative Linear Matrix Factorization for feature selection, had superior results specifically on the PC1 dataset. The findings of the proposed framework demonstrate that the inclusion of feature selection significantly enhances the performance of the prediction model in comparison to the model that lacks feature selection. However, it is important for future research to continue optimising different classifiers by using a comprehensive range of features. This will allow for a wider selection of variations to be considered for machine learning-based models. The use of Deep learning models using feature selection techniques is expected to enhance the accuracy of software module fault classification..

REFERENCES

[1] Rathore, S.S., Kumar, S.: Towards an ensemble based system for predicting the number of software faults. Expert Syst. Appl. 82, 357–382 (2017)

[2] Laradji, I.H., Alshayeb, M., Ghouti, L.: Software defect prediction using ensemble learning on selected features. Inf. Softw. Technol. 58, 388–402 (2015)

[3] Abisoye, O.A., Akanji, O.S., Abisoye, B.O., Awotunde, J.: Slow hypertext transfer protocol mitigation model in software defined networks. In: 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy, ICDABI 2020, 9325601 (2020)

[4] Malhotra, R., Jain, J.: Handling imbalanced data using ensemble learning in software defect prediction. In: 2020

10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 300–304. IEEE (2020)

[5] Awotunde, J.B., Ayo, F.E., Ogundokun, R.O., Matiluko, O.E., Adeniyi, E.A.: Investigating the roles of effective communication among stakeholders in collaborative software development projects. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2020, 12254 LNCS, pp. 311–319 (2020)

[6] Awotunde, J.B., Folorunso, S.O., Bhoi, A.K., Adebayo, P.O., Ijaz, M.F.: Disease diagnosis system for IoT-based wearable body sensors with machine learning algorithm. Intelligent Systems Reference Library 2021(209), 201–222 (2021)

[7] Awotunde, J.B., Misra, S.: Feature extraction and artificial intelligence-based intrusion detection model for a secure internet of things networks. Lecture Notes Data Eng. .ications Technol. 2022(109), 21–44 (2022)

[8] Behera, R.K., Shukla, S., Rath, S.K., Misra, S.: Software reliability assessment using machine learning technique. In: Gervasi, O., et al. (eds.) Computational Science and Its Applications – ICCSA 2018: 18th International Conference, Melbourne, VIC, Australia, July 2-5, 2018, Proceedings, Part V, pp. 403–411. Springer International Publishing, Cham (2018). https:// doi.org/10.1007/978-3-319-95174-4_32

[9] Chicco, D., Jurman, G.: The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics 21(1), 1–13 (2020)

[10] Shukla, S., Behera, R.K., Misra, S., Rath, S.K.: Software reliability assessment using deep learning technique. In: Chakraverty, S., Goel, A., Misra, S. (eds.) Towards Extensible and Adaptable Methods in Computing, pp. 57–68. Springer, Singapore (2018). https://doi.org/10. 1007/978-981-13-2348-5_5

[11] Awotunde, J.B., Chakraborty, C., Adeniyi, A.E.: Intrusion detection in industrial internet of things network-based on deep learning model with rule-based feature selection. Wirel. Commun. Mob. Comput. 2021(2021), 7154587 (2021)

[12] Ogundokun, R.O., Awotunde, J.B., Sadiku, P., Adeniyi, E.A., Abiodun, M., Dauda, O.I.: An enhanced intrusion detection system using particle swarm optimization feature extraction technique. Procedia Computer Science 193, 504–512 (2021)

[13] Jagdhuber, R., Lang, M., Stenzl, A., Neuhaus, J., Rahnenführer, J.: Cost-Constrained feature selection in binary classification: adaptations for greedy forward selection and genetic algorithms. BMC Bioinformatics 21(1), 1–21 (2020)

[14] Kumari, A., Behera, R.K., Sahoo, B., Sahoo, S.P.: Prediction of link evolution using community detection in social network. Computing, 1–22 (2022)

[15] Mishra, N., Soni, H.K., Sharma, S., Upadhyay, A.K.: Development and analysis of artificial neural network models for rainfall prediction by using time-series data. International Journal of Intelligent Systems Applications, 10(1) (2018)

[16] Zhang, X., Mohanty, S.N., Parida, A.K., Pani, S.K., Dong, B., Cheng, X.: Annual and nonmonsoon rainfall prediction modelling using SVR-MLP: an empirical study from Odisha. IEEE Access 8, 30223–30233 (2020)

[17] Jagdale, R.S., Shirsat, V.S., Deshmukh, S.N.: Sentiment analysis on product reviews using machine learning techniques. In: Mallick, P.K., Balas, V.E., Bhoi, A.K., Zobaa, A.F. (eds.) Cognitive Informatics and Soft Computing. AISC, vol. 768, pp. 639–647. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-0617-4_61

[18] Hassonah, M.A., Al-Sayyed, R., Rodan, A., Ala'M, A.Z., Aljarah, I., Faris, H.: An efficient hybrid filter and evolutionary wrapper approach for sentiment analysis of various topics on Twitter. Knowledge-Based Syst.192, 105353 (2020)

[19] Rehman, A.U., Malik, A.K., Raza, B., Ali, W.: A hybrid CNN-LSTM model for improving accuracy of movie reviews sentiment analysis. Multimedia Tools and Applications 78(18), 26597–26613 (2019)

[20] Awotunde, J.B., Abiodun, K.M., Adeniyi, E.A., Folorunso, S.O., Jimoh, R.G.: A deep learning-based intrusion detection technique for a secured IoMT system. Communications in Computer and Information Science, 2022, 1547 CCIS, pp. 50–62 (2021)

[21] Verma, A., Ranga, V.: Machine learning based intrusion detection systems for IoT applications. Wireless Pers. Commun. 111(4), 2287–2310 (2020)

[22] Amouri, A., Alaparthy, V.T., Morgera, S.D.: A machine learning based intrusion detection system for mobile Internet of Things. Sensors 20(2), 461 (2020)

[23] Matloob, F., Aftab, S., Iqbal, A.: A framework for software defect prediction using feature selection and ensemble learning techniques. International Journal of Modern Education Computer Sci. 11(12) (2019)

[24] Yalçıner, B., Özde¸s, M.: Software defect estimation using machine learning algorithms. In: 2019 4th International Conference on Computer Science and Engineering (UBMK), pp. 487– 491. IEEE (2019)

[25] Arar, Ö.F., Ayan, K.: Software defect prediction using cost-sensitive neural network. Appl. Soft Comput. 33, 263–277 (2015)

[26] Iqbal, A., et al.: Performance analysis of machine learning techniques on software defect prediction using NASA datasets. Int. J. Adv. Comput. Sci. Appl 10(5), 300–308 (2019)

[27] Iqbal, A., Aftab, S., Ullah, I., Bashir, M.S., Saeed, M.A.: A feature selection based ensemble classification framework for software defect prediction. Int. J. Modern Education Comput. Sci. 11(9), 54 (2019)

[28] Lanubile, F., Lonigro, A., Vissagio, G.: Comparing models for identifying fault-prone software components. In: SEKE, pp. 312–319 (1995)

[29] Elish, K.O., Elish, M.O.: Predicting defect-prone software modules using support vector machines. J. Syst. Softw. 81(5), 649–660 (2008)

[30] Gondra, I.: Applying machine learning to software fault-proneness prediction. J. Syst. Softw. 81(2), 186–195 (2008)

[31] Manjula, C., Florence, L.: Deep neural network based hybrid approach for software defect prediction using software metrics. Clust. Comput. 22(4), 9847–9863 (2018). https://doi.org/ 10.1007/s10586-018-1696-z

[32] Witten, I.H., Frank, E.: Data mining: practical machine learning tools and techniques with Java implementations. ACM SIGMOD Rec. 31(1), 76–77 (2002)

[33] Dai, H., Hwang, H.G., Tseng, V.S.: Convolutional neural network based automatic screening tool for cardiovascular diseases using different intervals of ECG signals. Comput. Methods Programs Biomed. 203, 106035 (2021)

[34] 34. Awotunde, J.B., et al.: An improved machine learnings diagnosis technique for COVID19 pandemic using chest X-ray images. Communications in Computer and Information Science, 2021, 1455 CCIS, pp. 319–330 (2021)

[35] Daoud, M. S., Aftab, S., Ahmad, M., Khan, M. A., Iqbal, A., Abbas, S., ... & Ihnaini, B. (2022). Machine learning empowered software defect prediction system.

[36] A. Iqbal, S. Aftab, U. Ali, Z. Nawaz, L. Sana et al., “Performance analysis of machine learning techniques on software defect prediction using nasa datasets,” International Journal of Advanced Computer Science and Applications, vol. 10, no. 5, pp. 300–308, 2019.