International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

Optimized Deeplearning Algorithm for Software Defects Prediction

[*1]Anju A.J, [2]J.E. Judith
[*1]Research Scholar, Computer Science, and Engineering, Noorul Islam Centre for Higher Education, Kumaracoil, India.
[*1]Corresponding Author Email: ajanju1@gmail.com
[2]Associate Professor, Computer Science and Engineering, Noorul Islam Centre for Higher Education, Kumaracoil, India.
Email: judithjegan@gmail.com

Abstract -Accurate software defect prediction (SDP) helps to enhance the quality of the software by identifying potential flaws early in the development process. However, existing approaches face challenges in achieving reliable predictions. To address this, a novel approach is proposed that combines a two-tier-deep learning framework. The proposed work includes four major phases:(a) pre-processing, (b) Dimensionality reduction, (c) Feature Extraction and (d) Two-fold deep learning-based SDP. The collected raw data is initially pre-processed using a data cleaning approach (handling null values and missing data) and a Decimal scaling normalisation approach. The dimensions of the pre-processed data are reduced using the newly developed Incremental Covariance Principal Component Analysis (ICPCA), and this approach aids in solving the "curse of dimensionality" issue. Then, onto the dimensionally reduced data, the feature extraction is performed using statistical features (standard deviation, skewness, variance, and kurtosis), Mutual information (MI), and Conditional entropy (CE). From the extracted features, the relevant ones are selected using the new Euclidean Distance with Mean Absolute Deviation (ED-MAD). Finally, the SDP (decision making) is carried out using the optimized Two-Fold Deep Learning Framework (O-TFDLF), which encapsulates the RBFN and optimized MLP, respectively. The weight of MLP is fine-tuned using the new Levy Flight Cat Mouse Optimisation (LCMO) method to improve the model's prediction accuracy. The final detected outcome (forecasting the presence/ absence of defect) is acquired from optimized MLP. The implementation has been performed using the MATLAB software. By using certain performance metrics such as Sensitivity, Accuracy, Precision, Specificity and MSE the proposed model's performance is compared to that of existing models. The accuracy achieved for the proposed model is 93.37%.

Keywords: Software defect prediction; Incremental Covariance Principal Component Analysis (ICPCA); Euclidean Distance with Mean Absolute Deviation (ED-MAD); Levy Flight Cat Mouse Optimization (LCMO); Two- Fold Deep Learning Framework (TFDLF)

Nomenclature

Abbreviation	Description
WPDP	Within Project Defect Prediction
TSE	Two-Stage Ensemble
SDP	Software defect prediction
SDAEs	Stacked Denoising Autoencoders
LSTM	Long Short-Term Memory
LDFP	Learning Deep Feature Representation
DPs	Deep Representations
DP-ARNN	Defect Prediction via Attention-Based Recurrent Neural Network
DL	Deep Learning
CNN	Convolutional Neural Networks
ASTs	Abstract Syntax Trees

I. INTRODUCTION

Software flaws are mistakes made during the development of the software that can cause failure, faults, collapse, and even jeopardise the security of people and their property. Software dependability, understandability, availability, maintainability, and effectiveness are all significantly impacted by software bugs [1]. Even carefully applied software requires laborious bug-free software development because hidden bugs are frequently present. A significant challenge in software engineering is the

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
***Article Received**: 01 May 2023 **Revised:** 28 June 2023 **Accepted:** 24 July 2023*

development of software bug prediction models that can identify faulty modules early on. Predicting software error is a crucial step in the development of software. This is due to the fact that identifying buggy modules before the software is deployed increases user satisfaction and enhances overall software performance. Additionally, foreseeing software bugs early enhances software adaptation to various environments and boosts resource utilization [2, 3]. SDP [4] is only possible with past data obtained during the implementation of similar or identical software projects, or with design metrics gathered during the software development design phase. [5].

SDP is a current area of study for software repository mining [6]. By anticipating potential defective program modules, the methods of SDP can be used to allocate resources for the assurance of software quality more effectively [7, 8]. As a result, the minimal testing resources can be used more wisely to test the specified modules. After mining and analysing software warehouse, SDP can train models, and these models can then be used to distinguish between non-defective and defective modules in a project. The level of detail of modules for gathering can be set to file, method, or even code change depending on the developer usage scenario [9]. Defects resulting from that software have a prominent effect on businesses and people's lives as it continues to play a essential role in every aspect of our society. However, the complexity and the size of the software codebase significantly expands, and finding errors in code grows more and more challenging. The topic of defect prediction is a active research in software engineering due to its significance and difficulties. Significant research has gone into creating tools and predictive models that allow software testers and engineers to quickly identify the most likely faulty areas of a software codebase [10, 11].

Software engineering is one of many fields where DL has been applied since 2012 DL made its debut in the SDP field in 2015, and since then, its use has increased. Numerous academics have looked into the application of DL to the prediction of software defects up to this point [12, 13]. The need for software for various applications has been growing quickly over the last 20 years. Numerous software applications are created for daily or business use in order to satisfy customer demand. Because of the number of productions of software programmes, software quality remains an unresolved issue, resulting in poor functionality for both commercial and personal applications. Software testing was developed as a result to address this problem by helping to identify and attempt to fix any flaws or bugs in the software application [14, 15].

The major outcome of this research is:

- To introduce a new Incremental Covariance Principal Component Analysis (ICPCA) model for resolving the “curse of dimensionality” issue.
- To Select the optimal Features using the new Euclidean Distance with Mean Absolute Deviation (ED-MAD).
- To design a new O-TFDLF for accurate decision making regarding the forecasting of presence/ absence of defects. The TFDLF encapsulates the RBFN and optimized MLP, respectively.
- To enrich the prediction accuracy of the model, the weight of MLP is fine-tuned using the new LCMO.

This article's remaining sections are structured as follows: The literature studies conducted in the prediction of employee absenteeism are discussed in Section 2. Section 3 explains the proposed employee absenteeism prediction model. Section 4 describes the findings obtained using the projected model, and Section 5 concludes up this research.

II. LITERATURE REVIEW

Qiao et al. [16] proposed a mechanism for forecasting the occurrence of software faults in 2020. First, we pre-processed a publicly available dataset by conducting data normalization and log transformation. Data modelling was done to prepare the data input for the DL model. Third, submit the modelled data to a deep neural network-based model designed to forecast the number of flaws. I have put the proposed approach to the test on two well-known datasets. The findings of the study revealed that the proposed strategy was reliable and could outperform new methods.

In 2019, Liang *et al.* [17] suggested Seml, an innovative structure for defect prediction that combined word embedding and DL techniques. In particular, a token sequence was extracted from the abstract syntax tree of each program source file. The next step was to use a mapping table learned with an unsupervised word embedding model to convert each token in the sequence to a real-valued vector. Finally, an LSTM network was constructed using the vector sequences and their labels. The LSTM model could predict defects and automatically learn the program's semantic information.

In 2018, Tong *et al.* [18] put forth a new SDP strategy called SDAEsTSE that makes use of ensemble learning and SDAEs to create the proposed TSE. The DL phase and the TSE phase were the main components of the methodology. The class imbalance issue was then addressed using a novel ensemble learning strategy called TSE after first use SDAEs to extract the DPs from the conventional software metrics.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

IN 2019, Xu *et al.* [19] have suggested a new framework called LDFR based on the SDP defect data. To address the imbalance issue, a deep neural network with a new hybrid loss function composed of a triplet loss and a weighted cross-entropy loss was used to develop a more discriminative feature representation of the defect data. Conducted extensive experiments on a benchmark dataset with 27 defect data using three conventional and three effort-aware indicators to assess the efficacy of the proposed LDFR framework.

In 2023, Giray *et al.* [20] have conducted a comprehensive evaluation of the literature of the available SDP techniques using DL to understand the state-of-the-art. To found articles, used a thorough procedure supported by snowballing and searched several scientific databases. As a result of a multiple-assessor quality assessment step with clear criteria, we chose the articles to be considered for analysis. Totalling 102 high-quality primary studies, the research was eventually included.

In 2021, Nevendra *et al.* [21] have put forth a method for using improved CNNs to find software defects in modules. The goal of the research involves an improved DL technique to identify defective instances. The tests were based on WPDP, which employs K-fold cross-validation. On 19 open-source software defect datasets, the suggested approach was assessed using various evaluation metrics.

In 2019, Fan *et al.* [22] put forth a framework known as DP-ARNN. To be more precise, DP-ARNN first extracts vectors by parsing the ASTs of programs. Then it used word embedding and dictionary mapping to encrypt the vectors that make up the DP-ARNN's inputs. It could then automatically pick up syntactic and semantic features after that. It also makes use of the attention mechanism to produce additional important features for precise defect prediction.

In 2019, Ramay *et al.* [23] have suggested an automatic deep neural network approach for predicting bug reports' severity. For text preprocessing of bug reports, used natural language processing techniques. The second step involve calculation and assign an emotion score to each bug report. For every pre-processed bug report, create a vector. Fourth, send the created vector and each bug report's emotion score to a classifier built with a deep neural network to predict the severity. On the basis of bug report history data, assessed the suggested approach. The cross-product findings show that the suggested method beats the most recent methods.

A. Research Gaps

The research gaps table (Table 1) provides a summary of the existing approaches reviewed in the literature. Each study's author, aim, and identified research gaps are presented. The research gaps highlight the limitations or areas that have not been adequately addressed in previous studies, which create opportunities for further research. Qu et al. [2] analysed the effectiveness of network embedding techniques for predicting software bugs but found that these techniques were not evaluated in their study, suggesting a research gap in the evaluation of network embedding techniques. Hammouri et al. [3] focused on predicting software bugs using machine learning (ML) but identified a research gap in the absence of additional software metrics incorporated into the learning process, indicating the need for considering a broader range of metrics. Zhang et al. [4] explored semi-supervised learning for SDP using label propagation. They discovered, however, that the performance of defect prediction models declined, indicating a research gap in enhancing the performance of defect prediction models. Dam et al. [11] developed an algorithm for predicting software defects based on deep trees, but they did not involve programming languages and web applications in their study, indicating a research gap in considering these specific domains. Qiao et al. [16] focused on predicting software defects using DL. However, they did not investigate the number of predicted defects in software modules, highlighting a research gap in exploring the prediction accuracy of defect counts. Liang et al. [17] proposed a semantic LSTM model for predicting software defects but found that more program semantic information was not recorded, indicating a research gap in capturing and utilizing richer semantic information. Giray et al. [20] utilized DL to predict software defects but identified a research gap in the lack of advancement in creating new, comprehensive DL methods that can automatically capture richer representations and features from diverse sources. Nevendra et al. [21] employed DL for the prediction of software defects and found research gaps in reducing time and developing more effective DL models, emphasizing the need for more efficient and powerful approaches. Fan et al. [22] used an attention-based recurrent neural network (RNN) for SDP but did not implement some programming languages, suggesting a research gap in considering a wider range of programming languages in the prediction process.

These identified research gaps provide valuable insights into the areas that have not been fully explored or addressed in previous studies. They serve as a basis for justifying the need for the current research and contribute to the overall novelty and significance of the proposed methodology. The research gaps identified in the existing works is manifested in Table 1.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
***Article Received**: 01 May 2023 **Revised:** 28 June 2023 **Accepted:** 24 July 2023*

TABLE **1:** REVIEW ON THE EXISTING APPROACHES

Author	Aim	Research Gaps
Qu *et al.* [2]	Analysing the effectiveness of network embedding techniques for predicting software bugs	Network Embedding Techniques is not evaluated.
Hammouri *et al.* [3]	Prediction of Software Bugs Using ML	There is no addition of more software metrics in the learning process.
Zhang *et al.* [4]	Semi-supervised learning for SDP using label propagation	Performance in defect prediction has not improved.
Dam *et al.* [11]	An algorithm for predicting software defects based on deep trees	Programming Languages & Web Applications are not involved.
Qiao *et al.* [16]	Prediction of Software Defects Using DL	The number of predicted defects in software modules is not looked into.
Liang *et al.* [17]	A Semantic LSTM Model for Predicting Software Defects	More programme semantic information is not recorded.
Giray *et al.* [20]	Using DL to Predict Software Defects	There is no advancement in the creation of new, all-encompassing DL methods that automatically capture richer representations and features from diverse sources.
Nevendra *et al.* [21]	DL for Prediction of Software Defects	Time is not reduced. There is no development of more effective DL models.
Fan *et al.* [22]	Prediction of Software Defects Using Attention-Based RNN	Some programming languages is not implemented.

III. SOFTWARE DEFECT PREDICTION VIA OPTIMIZED TWO- FOLD DEEP LEARNING FRAMEWORK (TFDLF)

Recently, ML techniques are being highly applied for automated SDP. These approaches require higher computation time and require manually extracted features. DL approaches enable practitioners to automatically extract and learn from more complicated and high-dimensional data. Therefore, in this research work, a novel two-fold-deep learning-based SDP model is introduced. The proposed work includes the following phases: "(a) pre-processing, (b) Dimensionality reduction, (c) Feature Extraction and (d) Two-fold deep learning-based Software defect prediction". The prediction model is shown in figure 1.

Step 1: The acquired raw data is pre-processed using the Data Cleaning (Missing Data Removal (MDR)) and Decimal scaling normalisation techniques.

Step 2: The dimensions of the pre-processed data are decreased using the recently created Incremental Covariance Principal Component Analysis (ICPCA), which assists in the resolution of the "curse of dimensionality".

Step 3: Feature extraction is then conducted on the dimensionally reduced data using statistical features (standard deviation, skewness, variance, and kurtosis), Mutual information (MI), and Conditional entropy (CE).

Step 4: The appropriate features are chosen from the retrieved features using the new Euclidean Distance with Mean Absolute Deviation (ED-MAD).

Step 5: Finally, SDP is performed using the new Optimized Two-Fold Deep Learning Framework (O-TFDLF). The TFDLF encapsulates the RBFN and optimized MLP, respectively. To enhance the prediction accuracy of the model, the weight of MLP will be fine-tuned using the new (LCMO). The proposed LCMO model is the extended version of the standard Cat Mouse optimization Algorithm (CMBO). The final outcome regarding the predicted outcomes is acquired from optimized MLP.

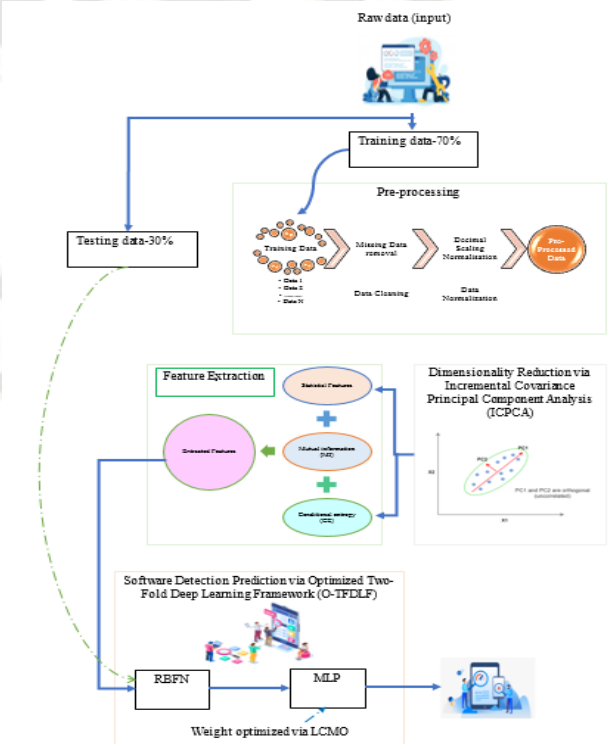


Figure 1: Prediction Model

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
***Article Received**: 01 May 2023 **Revised:** 28 June 2023 **Accepted:** 24 July 2023*

A. Pre-processing

Initially, the collected raw data is pre-processed via data cleaning and Decimal scaling normalization approach. Since datasets frequently values, contain missing noise, and noticeable changes in the size of the features, data pre-processing is frequently done before training ML models. The following pre-processing steps have been used for the IBM HR dataset.

a) Data cleaning - Missing Data Removal (MDR)

The term "missing data" refers to information that is not recorded for a variable for a particular observation. Missing data lowers the analysis's statistical power, which might skew the conclusions' validity. To prevent bias when dealing with missing data at random, relevant data may be eliminated. If there aren't enough observations to perform a reliable analysis, data removal may not be the best solution. In some cases, it may be vital to keep a watch on specific things.

b) Decimal scaling normalization

The effectiveness and simplicity of the mining process may both be enhanced by pre-processing. The decimal point of attribute B values is relocated during normalisation using the decimal scaling approach. The amount of decimal points that are shifted is determined by the highest absolute value of B. The following expression (Eq. (1) converts a value of B to P':

$$P' = P/10^i \quad (1)$$

where i is the smallest integer such that Max (P') <1.

The dimensions of the pre-processed data are reduced using the newly developed Incremental Covariance Principal Component Analysis (ICPCA), and this approach aids in solving the “curse of dimensionality” issue.

B. Dimensionality reduction using ICPCA

The performance of ML models is frequently improved using feature selection and dimensionality reduction techniques. The PCA is a well-liked approach to data analysis and a technique for unsupervised linear feature extraction. PCA is frequently employed in tasks including dimensionality reduction, feature selection, and lossy data compression. Because PCA integrates comparable characteristics as a result of variance, data from a high-dimensional space might be reduced to a low-dimensional one by using this method. PCA requires the entire dataset to be stored in memory for computation, which can be challenging for large datasets. In contrast, ICPCA addresses this issue by processing the dataset in incremental batches, reducing the memory requirements significantly. As a result, IPCA can reduce the volume of data and the number of data features, preventing model overfitting.

- In ICPCA, the eigenvectors of the matrix are generated once the covariance matrix of the feature vector has been determined. Due to the greatest eigenvectors' eigenvalues, the feature vector acquires a new decreased dimensionality. The most important components of the data were maintained rather than some of them being lost in order to preserve the variance. Prior to use the feature dimension reduction using the PCA technique, data pre-processing must be carried out since it is required for the next phases. Apply mean normalisation or feature scaling in a manner akin to supervised learning techniques depending on the training set with N dimension is represented as $a^{(1)}, a^{(2)}, a^{(3)} \ldots a^{(N)}$ that are used in Eq. (2) which calculates the mean of each attribute.

$$\mu_x = \frac{1}{N}\sum_{k=1}^{N} a_x^{(k)} \quad (2)$$

If various features have varying means, scale them so that they are within a similar range. To make sure that each of the a_x variables has a mean value of exactly zero, then replace each one with an $a_x - \mu_x$ value. The scaling procedure of the j^{th} element is explained by supervised learning's Eqn. (3), where s_x is the j^{th} feature value of $|max - mean|$ or static deviation.

$$a_x^{(k)} = \frac{a_x^{(k)} - \mu_x}{S_x} \quad (3)$$

To reduce the feature's dimension from N to m (where m<N) and N-dimensional spatial definition of the surface, it is necessary to first calculate the predicted data's inaccuracy on the m-dimensional vector. The computational verification of the evaluation of these m vectors: $v^1, v^2, v^3 \ldots\ldots, v^m$ and the projected points: $vy^1, y^2, y^3 \ldots\ldots, y^N$ on these vectors is challenging and outside the purview of this study. Eqn. (20) is used to calculate the covariance matrix. The $a^{(k)}$ vector has N×1 dimension, while $(a^{(k)})^T$ has 1×N dimensions, resulting in a covariance matrix with N ×N dimensions. The covariance matrix's eigenvalues and eigenvectors, which stand for the feature vectors' new magnitude and associated directions in the modified vector space, are next calculated. While working with the covariance matrix, the eigenvalues provide a quantitative measure of all the vectors' variance. When an eigenvector includes high valued eigenvectors, it signifies the dataset's variance is high and the eigenvector contains a variety of significant dataset-related information. Conversely, eigenvectors with tiny eigenvalues contain very little data about the dataset.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

$$Covariance_{matrix} = \frac{1}{N}\sum_{i=1}^{N} a^{(k)} \times (a^{(k)})^{T} \quad (4)$$

Since $w^{(q)}$ is the eigenvector at q^{th} stage, $a^{(k)}$ *is* the covariance matrix, it is possible to assign a score to the q^{th} full principal component of a data vector, $a^{(k)}$, in the transformed coordinates, using the formula $t_c^{(q)} = a^{(k)}\, w^{(q)}$. Since the whole vector decomposition of the PCA is defined as A which can be written as T = $A \times W$. The primary function of the PCA technique is to enhance the covariance matrix called Incremental Covariance Principal Component Analysis (ICPCA) and rebuild the original covariance matrix into a low-dimensional matrix while retaining the majority of its data. The first step is to determine each column vector in the original covariance matrix.

$$\|\boldsymbol{b_k}\|_2 = \sqrt{\sum_{k=1}^{N} |\boldsymbol{C_{jk}}|^2} \quad \textbf{(5)}$$

- Create a new matrix B by combining the top k biggest column vectors in the resulting norm. To create the low-dimensional matrix C, do QR decomposition on the new matrix B. Singular value decomposition should be applied to the C matrix. After dimensionality reduction, sort the generated singular value representations according to relevance, toss out the irrelevant eigenvectors, and store the data set's eigenvalues.

- The next step is to select k eigenvalues from these N eigenvectors by increasing the variance of the retained actual data and reducing the total square reconstruction error. The "Cumulative Explained Variance," which is the subject of the following computation, is the sum of all variances found across the top m main components. Next, decide at what point the eigenvalues are judged valuable and the rest are discarded as irrelevant qualities.
- The dimensionality reduced feature vectors connected to each feature track are first standardised to a mean of zero and a variance of one before choosing a random subset to generate a codebook. Each input feature vector is then calculated using the top closest vectors from the codebook, resulting in a fixed number of vectors. The counts of these assigned vectors, when added together over all feature tracks in a sample recording, constitute a representation known as a histogram that is then subjected to prediction.

- Singular value decomposition is applied to the C matrix. After dimensionality reduction, arrange the generated singular value representations according to relevance, toss out the unnecessary eigenvectors, and save the original data set's eigenvalues. In the IPCA technique, the data is projected onto a lower-dimensional space using just the most significant singular vectors, and the centre data is decomposed into singular values for linear dimensionality reduction. From the dimensionally reduced data, the features like Geometric mean (GM), Mutual information (MI), and Conditional entropy (CE) are extracted.

C. Feature Extraction

In the feature extraction phase, we utilized statistical features such as standard deviation, skewness, variance, and kurtosis, along with Mutual Information (MI) and Conditional Entropy (CE).

a) Statistical features

(1) Mean: The term mean is defined as the total number of items divided by the total number of elements in a collection. The mean calculation provides with a complete knowledge of the complete collection of data. Consequently, the mean formula is calculated as per Eq. (6) and Eq. (7)

$$Mean = \frac{Sum\ of\ all\ the\ elements}{Number\ of\ elements} \quad (6)$$

$$\bar{y} = \frac{\sum y}{z} \quad (7)$$

Where, $\bar{y}$ = mean value, y = Items given, z = Total number of items

The significance of mean resides in its capacity to sum up the entire dataset in a single value.

(2) Standard Deviation: A measurement that demonstrates the degree of deviation from the mean is the standard deviation. When data points are close to the mean, there is little variation, whereas when data points are scattered from mean, there is a lot difference. The standard deviation governs the amount of deviation from the mean. The standard deviation, which is the most widely used measure of dispersion, is based on all data. Therefore, the value of the standard deviation can change if even one number does. It is distinct of origin and scale. In some difficult statistical problems, it is also beneficial.

$$SD\ (\sigma) = \sqrt{\frac{\sum(x_i - \mu)^2}{N}} \quad (8)$$

(3) Skewness: A measure of a distribution's symmetry is its skewness. Actually, calling it a measure of asymmetry would be more appropriate. A typical normal distribution is completely symmetrical and has zero skew as shown in Eq. (9)

$$skewness = \frac{3(Mean - Median)}{StandardDeviation} \quad (9)$$

(4) Variance: The dispersion of a data set's data points from its mean is referred to as variation, and it is calculated as the average squared departure from the population mean for each data point. By appending the squared deviations of all the data points and dividing by the total number of data points in the data set, one can obtain the formula for a variance.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

$$\sigma^2 = \frac{\sum(Y_i-\mu)^2}{n} \quad (10)$$

Here Y_i is the i^{th} data point in the dataset, μ defines the mean population and n is the number of population data points.

(5) Kurtosis: The probability distribution of signals is reflected in kurtosis. As per Eq. (11), the definition of kurtosis is displayed.

$$V = \frac{X\{(y-\mu)^4\}}{[X\{(y-\mu)^2\}]^2} = \frac{\mu_4}{\sigma^4}, \quad (11)$$

where $X\{\cdot\}$ is the expectation operator, is the expectation mean of $y(t)$, and σ is the anticipated standard deviation. As per Eq. (11), kurtosis is defined as the fourth central moment divided by the variance's square. Some definitions of kurtosis deduct 3 from the calculated value since the kurtosis of the normal distribution is 3.

(6) Geometric Mean (GM): The GM is an average that sums all the data points and establishes the number's root. A group of n integers must be used to get the nth root of each integer's product. Use this descriptive statistic to sum up your data.

$$GM = \sqrt{\frac{T_p \times T_n}{(T_p+F_n)\times(T_n+F_p)}} \quad (12)$$

b) MI

According to the definition of the MI between two random variables A and B:

$$I(A;B) = H(A) - H(B|A) \quad (13)$$

Inferentially, the MI between A and B indicates the decrease in B's uncertainty following the observation of A and vice-versa. I (A; B) = I (B; A), indicating that the MI is symmetric.

c) CE

The CE, an idea from information theory, quantifies how much information is needed to explain a random variable's outcome when the value of another random variable is known.

In addition, the empirical conditional entropy of the features C_e for dataset D_s is denoted by the following formula.

$$H(D_s|C_e) = -\sum_{j=1}^{N} P(K_j, C_e) \log_2 P(K_j|C_e) \quad (14)$$

The following is the expression for $P(K_l|C_e)$ in the formula above:

$$P(K_l|C_e) = \frac{\sum_j^{|K_l|} x_{j,k}^{(l)}}{\sum_{j,k,l} x_{j,k}^{(l)}} \quad (15)$$

Also $P(K_j|C_e) is\ expressed\ as$

$$P(K_l|C_e) = \frac{\sum_j^{|K_l|} x_{j,k}^{(l)}}{\sum_{j,l} x_{j,k}^{(l)}} \quad (16)$$

From the extracted features, the relevant ones are selected using the new Euclidean Distance with Mean Absolute Deviation (ED-MAD).

D. Feature Selection using ED-MAD model

The ED-MAD of a dataset is defined as the typical distance between each data point and the mean. It provides an understanding of a dataset's degree of variability.

The ED-MAD is calculated as follows.

Step 1: Calculate the geometric mean (proposed).

Step 2: Use Euclidian distances(proposed) to determine how distant each data point is from the mean. Such are referred to as absolute deviations.

The Euclidean distance can be defined as per Eq. (17).

$$E_d = \sqrt{(A-B)^T(A-B)} = \sqrt{\sum_{i=1}^{N}(A_i - B_i)^2} \quad (17)$$

Where A and B are the two different classes. When the datasets are grouped together in compact spaces, Euclidean distance yields great results.

Step 3: Combine those deviations.

Step 4: Subtract the total from the quantity of data points.

It is usually better to follow these steps in the example below to understand about mean absolute deviation, but here is a more formal method to put the stages in a formula (Eq. (18).

$$MAD = \frac{\sum|P_i-\bar{P}|}{n} \quad (18)$$

Where $\bar{P}\ is\ the$ the data set's average value, P_i is the collection of data values, where n is the total number of data values.

E. Prediction using Optimized Two- Fold Deep Learning Framework (O-TFDLF)

Finally, the SDP carried out using the O-TFDLF, which encapsulates the RBFN and optimized MLP, respectively. Utilizing the TFDLF model, which combines RBFN and MLP, the SDP is carried out. In order to get greater prediction performance than what can be achieved with only one learning algorithm, ensemble model called TFDLF attempt to build a set of numerous learning algorithms. To enhance the detection accuracy of the model, the weight of MLP is fine-tuned using the new (LCMO).

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

a) Optimized Multilayer Perceptron

An ANN known as the MLP has many layers of nodes. The output signals are produced by computing the activation from the sum of the inputs, and the linked nodes have weights associated with them. Its design is made up of an input layer that transmits the input vector to the network's subsequent levels. The inputs and outputs of the MLP are denoted by the phrases "input vectors" and "output vectors," which are single vectors. An MLP also contains an additional hidden layer or layers in addition to the output layer. MLPs are completely linked, meaning that each node is connected to every other node in the layer above and below.

The first processing components of MLP are set up in a one-directional order beforehand. These networks' three matching layer types—input, hidden, and output—communicate with one another in order for information to evolve. A MLP network with a single hidden layer is depicted in Figure 1. The weighting values for the networks between these layers range between [-1, 1]. Every node of the MLP is capable of performing the summation and activation operations. Based on the summing function shown in Eq. (19), it is possible to determine the product of input values, weight values, and bias values. The architecture of optimized MLP is shown in Fig.2.

$$T_s = \sum_{n=1}^{L} h_{nm} I_N + B_N \qquad (19)$$

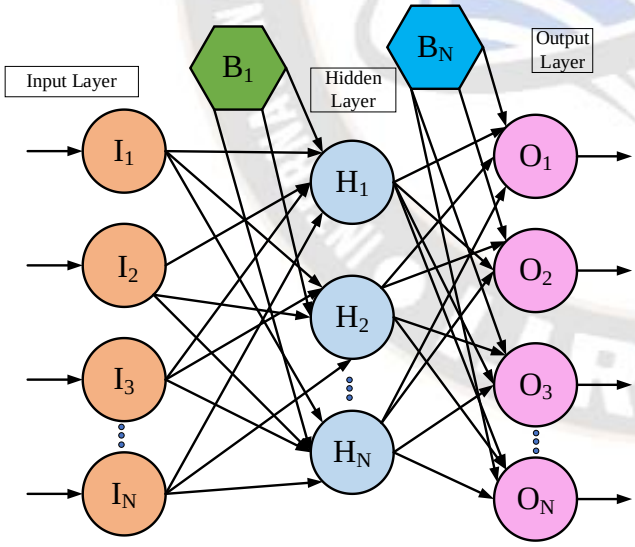


Figure 2: optimized MLP

where L stands for the overall number of inputs, I_N represents the input variable I_m is a bias value, and h_{nm} shows the connection weight. The conclusion of the Eqn. (14) is used to trigger an activation function in the following step. The MLP supports a number of activation strategies, the most popular of which, according to the literature, is S-shaped sigmoid function. Based on Eq. (20), one can compute this function. To enhance the prediction accuracy of the model, the weight of MLP is optimized using thew new LCMO model.

$$f_m(a) = \frac{1}{1+e^{-T_s}} \qquad (20)$$

As a result, Eq. (21) is used to produce the neuron m's final output:

$$b_n = f_m\left(\sum_{n=1}^{L} h_{nm} I_N + B_N\right) \qquad (21)$$

The learning process is started once the ANN's final structure has been built in order to adjust and evolve the network's weighting vectors. To approximate the findings and reduce the network's overall inaccuracy, these weighting vectors should be modified. The MLP's effectiveness and capacity for handling various situations are significantly impacted by the computationally difficult learning (training) stage of the ANN.

b) Levy Flight Cat Mouse Optimization (LCMO)

The LCMO is a population-based algorithm that took design cues from how a mouse would naturally flee from a cat assault and find safety. In the recommended method, cats and mice are split into two groups as the search agents. The recommended strategy involves updating the population twice. Cats advance toward mice in the first phase of the simulation, and mice flee to safe havens in the second phase to avoid being killed. As per Eq. (22), a matrix called population matrix is used to determine the algorithm's population.

$$Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_i \\ \vdots \\ Y_N \end{bmatrix}_{N\times m} = \begin{bmatrix} Y_{1,1} \ldots Y_{1,d} \ldots Y_{1,m} \\ \vdots \\ Y_{i,1} \ldots Y_{i,d} \ldots Y_{i,m} \\ \vdots \\ Y_{N,1} \ldots Y_{N,d} \ldots Y_{N,m} \end{bmatrix}_{N\times m} \qquad (22)$$

where Y define LCMO population of matrix, Y_i represents i^{th} search agent, $Y_{i,d}$ represents value of d^{th} variable discovered by i^{th} search, N represents size of population, and m define the problem variable number. As per Eq. (8), a vector is used to indicate obtained values for the objective function.

The fitness function of this research work is minimization of the error (RMSE). This is mathematically shown in Eq. (23). The weight of MLP is the input (solution) fed as input to LCMO. This weight function is delineating to the position of the solutions Y.

$$Fit = \min(RMSE) \qquad (23)$$

$$V = \begin{bmatrix} V_1 \\ \vdots \\ V_i \\ \vdots \\ V_N \end{bmatrix}_{N\times m} \qquad (24)$$

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
***Article Received**: 01 May 2023 **Revised:** 28 June 2023 **Accepted:** 24 July 2023*

Here V_i defines value of the goal function for the first search agent and V is an objective vector. The sorted goal function and the sorted population matrix are calculated as per Eq. (25) and Eq. (26), respectively.

$$Y^s = \begin{bmatrix} Y_1^s \\ \vdots \\ Y_i^s \\ \vdots \\ Y_N^s \end{bmatrix}_{N\times m} = \begin{bmatrix} Y_{1,1}^s \dots Y_{1,d}^s \dots Y_{1,m}^s \\ \vdots \\ Y_{i,1}^s \dots Y_{i,d}^s \dots Y_{i,m}^s \\ \vdots \\ Y_{N,1}^s \dots Y_{N,d}^s \dots Y_{N,m}^s \end{bmatrix}_{N\times m} \quad (25)$$

$$V^s = \begin{bmatrix} V_1^s & \min(V) \\ \vdots & \\ V_N^s & \max(V) \end{bmatrix}_{N\times 1} \quad (26)$$

here Y^s defines depending on value of objective, sorted population matrix, Y_i^s defines sorted matrix with member, $Y_{i,d}^s$ explains the dth problem variable value derived from the population-sorted matrix by ith agent, and V^s is objective function sorted vector. Two populations of mice and cats make up the proposed LCMO's population matrix. In LCMO, It is assumed that the mouse population contains half of the population members who produced higher values for the objective function, and the cat population contains the other half of the population members who produced lower values for the objective function. As per Eq. (27) and Eq. (28), determine the numbers of cats and mice.

$$Mi = \begin{bmatrix} Mi_1 = Y_1^s \\ \vdots \\ Mi_i = Y_i^s \\ \vdots \\ Mi_{N_m} = Y_{N_m}^s \end{bmatrix}_{N_m\times m} = \begin{bmatrix} Y_{1,1}^s \dots Y_{1,d}^s \dots Y_{1,m}^s \\ \vdots \\ Y_{i,1}^s \dots Y_{i,d}^s \dots Y_{i,m}^s \\ \vdots \\ Y_{N_m,1}^s \dots Y_{N_m,d}^s \dots Y_{N_m,m}^s \end{bmatrix}_{N_m\times m} \quad (27)$$

$$Ca = \begin{bmatrix} Ca_1 = Y_{N_m+1}^s \\ \vdots \\ Ca_j = Y_{N_m+j}^s \\ \vdots \\ Ca_{N_{ca}} = Y_{N_m+N_{ca}}^s \end{bmatrix}_{N_{ca}\times m} = \begin{bmatrix} Y_{N_m+1,1}^s \dots Y_{N_m+1,d}^s \dots Y_{N_m+1,m}^s \\ \vdots \\ Y_{N_m+j,1}^s \dots Y_{N_m+j,d}^s \dots Y_{N_m+j,m}^s \\ \vdots \\ Y_{N_m+N_{ca},1}^s \dots Y_{N_m+N_{ca},d}^s \dots Y_{N_m+N_{ca},m}^s \end{bmatrix}_{N_m\times m} \quad (28)$$

where Mi is the cat population matrix, Ca is the mouse population matrix, N_{ca} is the cat population matrix, Mi_i is the jth mouse, and Ca_j is the ith cat. Cats' position change is first simulated based on their natural behaviour and movement toward mice in updating the search parameters. Eq. (29) to Eq. (31) are used to describe the portion of the planned LCMO update analytically.

$$Ca_j^{new}: Ca_{j,d}^{new} = Ca_{j,d} + r \times \left(mi_{k,d} - I \times C_{j,d}\right) \& j = 1: N_{ca}, d = 1: mi, k \epsilon 1: N_{mi}, \quad (29)$$

$$I = round\ (1 + rand), \quad (30)$$

$$Ca_j = \begin{cases} Ca_j^{new}, |V_j^{ca,new} < V_j^{ca} \\ Ca_j, |else \end{cases} \quad (31)$$

Here, Ca_j^{new} is the jth cat's new status, $Ca_{j,d}^{new}$ is the jth cat new value of dth variable, r - random number between [0,1], $mi_{k,d}$ - kth mouse's dth dimension, and $V_j^{ca,new}$ - depending on the jth cat's new status, objective function value. Escaping mice to safe havens is represented in the second stage of the proposed LCMO. In LCMO, it is presumed that each mouse has a random haven, and the mice seek solace there. By patterning the placements of various algorithmic elements, the havens are positioned in the search space at random. Eq. (32) to Eq. (34) are used in a mathematical model to represent this step of updating the positions of mice. To increase the convergence rate, the proposed levy flight is added in the second stage.

$$Ha_i: ha_{i,d} = y_{l,d} \& i = 1: N_{mi}, d = 1: mi, l \epsilon 1: N \quad (32)$$

$$Mi_i^{new}: mi_{i,d}^{new} = mi_{i,d} + r \times \left(ha_{i,d} - I \times mi_{i,d}\right) sign\left(F_i^{mi} - F_i^{Ha}\right) \& i = 1: N_{mi}, d = 1: mi * Levy(\beta) \quad (33)$$

$$Mi_i = \begin{cases} Mi_i^{new}, |V_i^{mi,new} < V_i^{mi} \\ Mi_i, |else \end{cases} \quad (34)$$

In this case, Ha_i is the ith mouse safe haven and F_i^{Ha} - value of objective function. The ith mouse's new status is Mi_i^{new}, and objective function value - $V_i^{mi,new}$. The haven location is chosen at random within the search space as per the Eq. (11) to Eq. (18), until the stop condition is satisfied, the algorithm iterates. Optimization techniques may be stopped after a certain number of iterations or when an acceptable error between solutions obtained in subsequent rounds is defined. Furthermore, after running for a specific amount of time, the programme might be terminated. The best acquired optimum solution is provided by the LCMO following the completion of iterations and full application of the algorithm to the optimization problem.

c) RBFN

The Multilayer Perceptron is typically mentioned when people discuss neural networks or "Artificial Neural Networks" (MLP). An MLP's neurons take the weighted sum of their input values into consideration. In other words, after multiplying each input value by a coefficient, the results are summed together. Simple linear classifiers can be created by a single MLP neuron, while complicated non-linear classifiers may be created by connecting these neurons together into a network.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

Compared to the MLP, the RBFN method is easier to understand. By comparing input instances to examples from the training set, an RBFN conducts categorization. One sample from the training set serves as the "prototype" that each RBFN neuron stores. When classifying a new input, each neuron computes the Euclidean distance between the input and its prototype. The architecture of RBFN is shown in Fig.3.

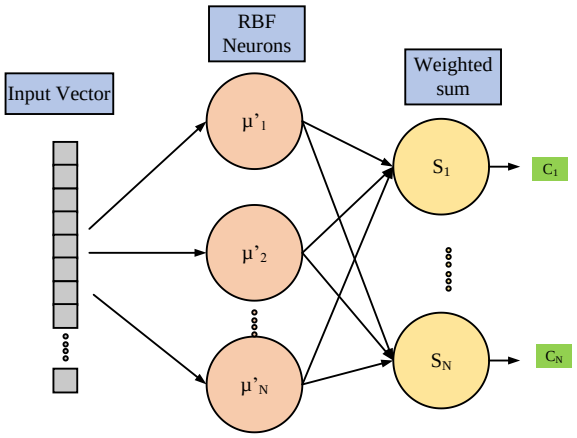


Figure 3: Architecture of RBFN

The accompanying diagram depicts the usual design of an RBF network. It has an input vector, an output layer with one node for each type or class of data, and an RBF neuron layer.

- **The Input Vector**

The input vector is being attempted to be categorized by the n-dimensional vectors. Each RBF neuron is shown the whole input vector.

- **The RBF Neurons**

The RBF neuron is carrying a "prototype" vector, which is one of the vectors from the training set. Each RBF neuron compares the input vector to its prototype and returns a value between 0 and 1 indicating the degree of similarity. Whenever the value of the input is equal to the prototype, the RBF neuron's output also equals 1. The reaction decreases exponentially approaches zero as input and prototype go further apart. The diagram of network architecture shows that the RBF neuron's response takes the form of a bell curve. Sometimes the term "activation" value—which refers to the neuron's response value—is used. Given that it represents the value located in the middle of the bell curve, the prototype vector is also sometimes referred to as the neuron's "centre.".

- **The Output Nodes**

A collection of nodes, one for each category are attempting to categorize, produce the network's output. For the corresponding category, each output node calculates a type of score. A classification decision is normally made by allocating the input to the category with the greatest score.

The score is calculated by adding the activity levels from all RBF neurons in a weighted manner. By "weighted sum," that means each output node assigns each RBF neuron a weight value and multiplies the activity of each neuron by this weight before adding it to the overall response.

In the hidden layer of the RBFN neural network, each node has a centroid attached to it. It determines the distance between the centroid of the node and P for each of the input vectors, P = (P_1, P_2,..., P_n). The output of the unit is thereafter determined as a nonlinear function of this distance. In the nodes of the output layer, the output of the hidden nodes is finally merged and given weight. The response function of each output node may be determined in the scenario of R' input nodes and m output nodes as per Eq. (35).

$$\sum_{j=1}^{N} H_j \times K' \left(\frac{P-a_j}{\sigma_j}\right) = \sum_{j=1}^{N} H_j \times d \left(\frac{\|P-a_j\|}{\sigma_j}\right) \quad (35)$$

where P is an input vector and N is the total number of hidden units; The weights connecting the j^{th} hidden-layer unit to the output nodes are designated $H_j$as; K' represents a radially symmetric kernel function; j is the ith kernel node's smoothing factor; a_j is its centroid factor; and, d[0,1) ∈R is the activation function.

With respect to weights, each output node is different since they are each used to calculate the score for a separate category. RBF neurons that fall into this category are normally given a positive weight by the output node, while the others receive a negative weight.

- **RBF Neuron Activation Function**

A measure of how comparable the input and its prototype vector are computed by each RBF neuron (taken from the training set). The output is nearer to 1 for input vectors that are more like the prototype. Although there are alternative possibilities, the Gaussian distribution-based similarity function is the most popular. The equation for a one-dimensional Gaussian is shown in Eq. (36).

$$g(a) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{(a-\mu)^2}{2\sigma^2}\right)} \quad (36)$$

Where the input value is represented as a, the mean value is μ, and the standard deviation is σ .

A somewhat modified formula is used to represent the RBF neuron activation function and is usually represented as per Eq. (37).

$$\varphi(a) = e^{-\beta\|a-\mu\|^2} \quad (37)$$

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
***Article Received**: 01 May 2023 **Revised:** 28 June 2023 **Accepted:** 24 July 2023*

The Gaussian distribution's mean is referred to as μ. In this instance, the bell curve's centre is occupied by the prototype vector.

IV.RESULT AND DISCUSSION

The proposed work has been implemented in MATLAB. The data for analysis has been collected from PROMISE. The performance of the proposed model in this part is evaluated using metrics like sensitivity, accuracy, F-score, specificity, MCC, recall, NPV, FPR, and FNR, respectively.

A. Performance metrics

Below are the performance measures and their calculation algorithms.

- **Sensitivity: To** determine the sensitivity value, just divide the total positives by the percentage of genuine positive predictions.

$$Sensitivity = \frac{TP}{TP+FN} \quad (38)$$

- **Specificity:** The number of predicted negative outcomes is precisely divided by the total number of negatives to calculate specificity.

$$Specificit\ y\ = \frac{TN}{TN+FP} \quad (39)$$

- **Accuracy:** The accuracy is calculated as the proportion of correctly sorted data to all other data in the log. The level of accuracy is defined as,

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \quad (40)$$

- **Precision:** Precision is the depiction of the complete number of authentic samples that are properly taken into account throughout the classification process by using the full number of samples utilised in the classification procedure.

$$Precision = \frac{TP}{TP+FP} \quad (41)$$

- **Recall:** Recall rate is determined by estimating how many real samples are taken into account overall when classifying data using all samples drawn from the same categories in the training data.

$$Recall = \frac{TP}{TP+FN} \quad (42)$$

- **F- Score:** The harmonic mean of recall rate and accuracy is the definition of the F-score.

$$F_{Score} = \frac{2\ Precision \times Recall}{Precision+Recall} \quad (43)$$

- **NPV:** A diagnostic test or other quantitative metric's efficiency is described by NPV.

$$NPV = \frac{TN}{TN+FN} \quad (44)$$

- **MCC:** The MCC, a two-by-two binary variable association measure, is depicted below,

$$MCC = \frac{(TP \times TN - FP \times FN)}{\sqrt{(TP+FN)(TN+FP)(TN+FN)(TP+FP)}} \quad (45)$$

- **FPR:** The ratio of the number of negative events to the number of negative events that are mistakenly broken down into positive events gives the FPR value.

$$FPR = \frac{FP}{FP+TN} \quad (46)$$

- **FNR:** The false-negative rate, commonly referred to as the "miss rate," is the probability that a real positive may go undetected by the test.

$$FNR = \frac{FN}{FN+TP} \quad (47)$$

B. Classifier Performance Analysis

The proposed approach's (TFDLF)) effectiveness has been investigated, and the outcomes have been analysed with those of existing techniques like the Artificial Neural Network (ANN), K-Nearest Neighbour (KNN), Deep Belief Network (DBN), random forest (RF), Support Vector Machine (SVM), and RNN. Table 2 explains the classifier performance with highest accuracy by optimized RNN.

TABLE 2**:** OVERALL PERFORMANCE ANALYSIS: CLASSIFIER PERFORMANCE (AT LEARNING RATE=70)

Performance metrics	ANN	DBN	SVM	KNN	RF	RNN	TFDLF
Accuracy	0.87542	0.80502	0.86318	0.81114	0.88766	0.84481	0.93442
Precision	0.94276	0.88447	0.93795	0.88163	0.94775	0.82711	0.95389
Sensitivity	0.88239	0.82255	0.87078	0.83034	0.89432	0.85336	0.92883
Specificity	0.91391	0.86791	0.90516	0.87351	0.92265	0.91892	0.94888
F-Measure	0.89389	0.82837	0.88186	0.83695	0.90561	0.79732	0.92908
MCC	0.83281	0.72877	0.81729	0.73455	0.84835	0.75047	0.89536
NPV	0.91391	0.86791	0.90516	0.87351	0.92265	0.91891	0.91888
FPR	0.09622	0.14221	0.10495	0.13661	0.08745	0.09118	0.05121
FNR	0.12771	0.18755	0.13932	0.17976	0.11611	0.15674	0.07127

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

As per table I, the effectiveness of the proposed method is evaluated by using different methods and metrics including accuracy, recall, sensitivity, specificity, false-positive-ratio (FPR), false-negative-ratio (FNR), and precision. Analysis of results shows that projected model, which uses an optimised RNN model, has the highest accuracy level, at 93.4%. The projected model clearly recorded the lowest FPR and FNR, 0.051 and 0.071, respectively, which is the least value, after analysing the obtained results. Figure 2, explains the graphical representation of the classifiers.

C. Algorithmic Analysis

As per Table 3, the algorithmic performance of the overall analysis is performed. The performance of the suggested strategy is contrasted with that of widely used algorithms like Honey Badger Algorithm (HBA), Cat Swarm Optimization (CSO), Grey Wolf Optimizer (GWO), Genetic Algorithm (GA), Grasshopper Optimization Algorithm (GOA), Squirrel Search Algorithm (SSA). The projected model, which makes use of the LCMO model as a suggestion, has the highest accuracy level at 95.4%, according to the results of the analysisThe lowest FPR and FNR are clearly recorded in the projected model, which are 0.041 and 0.061, and that is employed as the least value, after analysing the acquired results. The algorithmic analysis of the proposed as well as state-of-art models is shown in Fig.4.

TABLE 3: ALGORITHMIC PERFORMANCE ANALYSIS (AT LEARNING RATE=70)

Performance metrics	HBA	GA	SSA	GWO	GOA	CSO	LCMO
Accuracy	0. 80709	0. 88323	0. 84590	0. 87104	0. 85886	0. 80099	0.95442
Precision	0. 87722	0. 94301	0. 82297	0. 93805	0. 93326	0. 88005	0.93389
Sensitivity	0. 82619	0. 88953	0. 84909	0. 87798	0. 86642	0. 81844	0.94883
Specificity	0. 86914	0. 91803	0. 91431	0. 90933	0. 90063	0. 86357	0.92888
F-Measure	0. 83277	0. 90107	0. 79333	0. 88942	0. 87745	0. 82423	0.94908
MCC	0. 73088	0. 84411	0. 74672	0. 82865	0. 81321	0. 72512	0.91536
NPV	0. 86914	0. 91803	0. 91431	0. 90933	0. 90063	0. 86357	0.93888
FPR	0. 13591	0. 08701	0. 09074	0. 09571	0. 10442	0. 14148	0.04121
FNR	0. 17886	0. 11552	0. 15596	0. 12708	0. 13863	0. 18661	0.06127

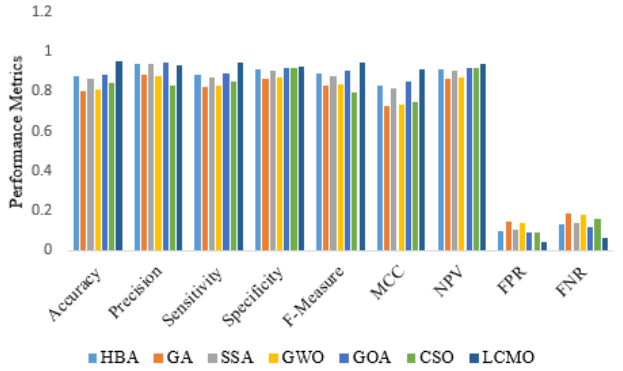


Figure 4: Algorithmic analysis of graphical representation

Figure 4, explains the graphical representation of the algorithmic analysis. All of the aforementioned findings make it clear that the suggested technique is superior to the existing methods.

D. Analysis on Feature Selection

The analysis on feature selection techniques for SDP reveals interesting insights into their impact on the accuracy of the prediction models. The results acquired are manifested in Fig. 5. Four different techniques were evaluated, and their performance was compared against a baseline model without any feature selection. The results demonstrate that the choice of feature selection technique has a significant influence on the accuracy of the SDP. When no feature selection was applied, the accuracy varied between 0.78 and 0.86, indicating that using all available features without any filtering or selection leads to inconsistent results. The application of a Genetic Algorithm (GA) for feature selection showed improvement compared to the baseline model. The accuracy ranged from 0.844 to 0.89, suggesting that the GA-based approach helped identify more relevant features, resulting in better prediction performance. Using the Mean Absolute Deviation (MAD) as a feature selection technique yielded mixed results. The accuracy ranged from 0.79 to 0.89,

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

indicating that MAD alone may not effectively capture the most important features for SDP. However, the proposed Euclidean Distance with Mean Absolute Deviation (ED-MAD) feature selection technique consistently outperformed the other approaches. The accuracy ranged from 0.91 to an impressive 0.967, demonstrating the effectiveness of ED-MAD in selecting highly relevant features. By identifying and including only the most informative features, the ED-MAD technique significantly improved the accuracy of the SDP models. Overall, this analysis emphasizes the importance of feature selection in SDP. The results highlight that the proposed ED-MAD technique outperforms other methods, consistently achieving high accuracy. By selecting the most relevant features, the ED-MAD approach enhances the performance and reliability of SDP models, contributing to enhanced software quality and early identification of potential defects in the development process.

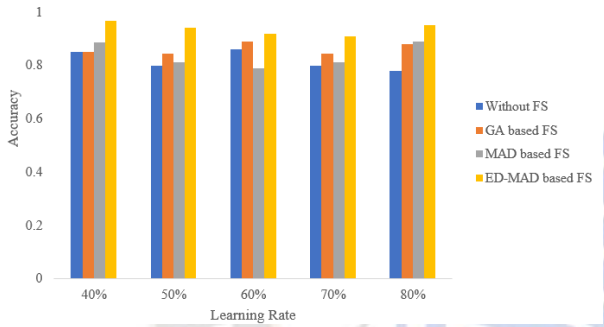


Figure 5: Analysis on Feature Selection Approaches

E. Analysis on Dimensionality reduction

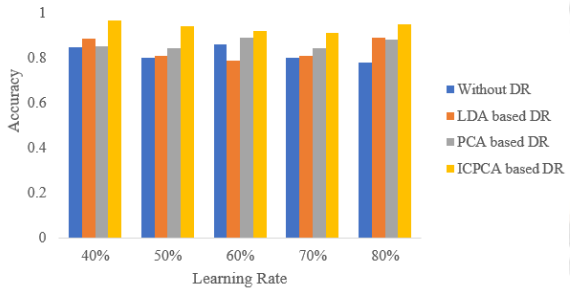


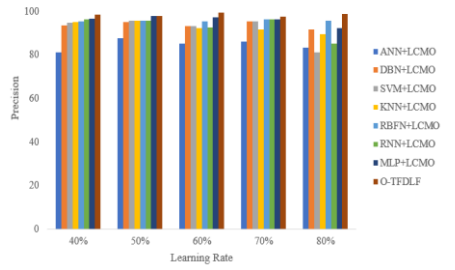
Figure 6: Analysis on Dimensionality Reduction Approaches

The analysis on dimensionality reduction techniques for SDP provides valuable insights into their impact on the accuracy of prediction models. The results acquired are manifested in Fig. 6. Four different techniques were evaluated, and their performance was compared against a baseline model without any dimensionality reduction. The results reveal that the choice of dimensionality reduction technique has a significant influence on the accuracy of SDP. When no dimensionality reduction was applied, the accuracy ranged from 0.78 to 0.86, indicating that using all available dimensions without any reduction can lead to inconsistent results. Applying Linear Discriminant Analysis (LDA) for dimensionality reduction showed improvement compared to the baseline model. The accuracy ranged from 0.79 to 0.89, suggesting that LDA-based approach helped capture the discriminative information in the data and reduced the dimensionality effectively. Principal Component Analysis (PCA) based dimensionality reduction yielded mixed results. The accuracy ranged from 0.844 to 0.89, indicating that PCA alone may not be sufficient to capture the most informative dimensions for SDP. However, the proposed Incremental Covariance Principal Component Analysis (ICPCA) technique consistently outperformed the other approaches. The accuracy ranged from 0.91 to an impressive 0.967, demonstrating the effectiveness of ICPCA in reducing dimensionality while preserving the most relevant information. By extracting the most discriminative dimensions, the ICPCA technique significantly improved the accuracy of SDP models. Overall, this analysis emphasizes the importance of dimensionality reduction in SDP The results highlight that the proposed ICPCA technique outperforms other methods, consistently achieving high accuracy. By reducing the dimensionality while retaining the most informative features, the ICPCA approach enhances the performance and reliability of SDP models, contributing to improved software quality and early detection of potential defects in the development process.

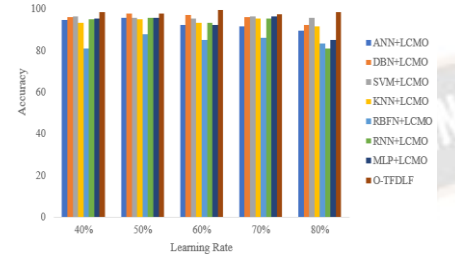
F. Overall Performance Analysis

The proposed model, O-TFDLF, consistently exhibits the highest values in terms of accuracy, MCC, and sensitivity across different data percentages. The results acquired are manifested in Fig. 7. This indicates that O-TFDLF performs exceptionally well in correctly classifying both positive and negative cases, capturing underlying patterns in the data, and providing an overall accurate prediction. The high performance of O-TFDLF can be attributed to its unique combination of techniques, such as DL and feature selection through LCMO. These techniques enhance the model's ability to extract meaningful features and optimize its predictive capabilities. The consistently superior performance of O-TFDLF underscores its effectiveness in addressing the problem at hand and makes it a strong candidate for accurate predictions in real-world scenarios.

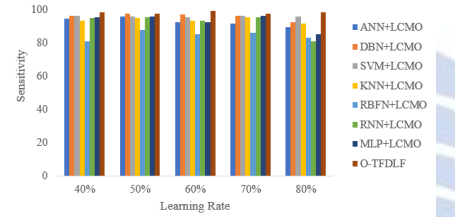
International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
***Article Received**: 01 May 2023 **Revised:** 28 June 2023 **Accepted:** 24 July 2023*



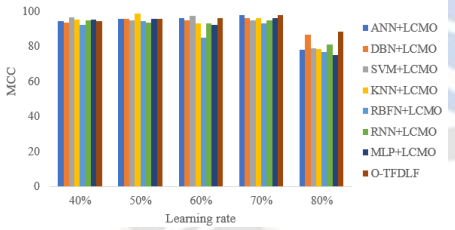
Precision



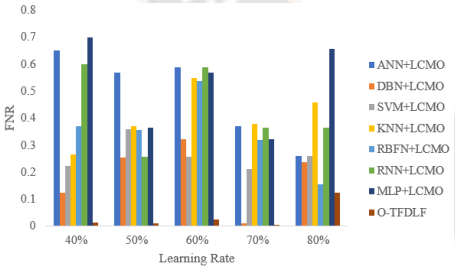
Accuracy



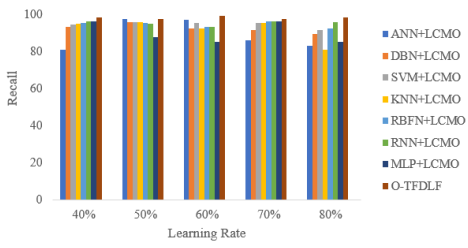
Sensitivity



MCC



(e)FNR



(f) Recall

Figure 7: overall Performance Analysis

V.CONCLUSION

In the present study, we proposed a novel approach that aimed to enhance software quality by accurately predicting software defects at an early stage in the development process. Existing approaches encountered challenges in achieving reliable predictions, necessitating the need for a new solution. Our work focused on addressing these challenges and providing a significant contribution to the field. The significance of our work lay in the successful combination of a two-tier DL framework for SDP. This approach not only improved the accuracy of defect identification but also enabled the early detection of potential issues, leading to more efficient software development and improved overall quality. Throughout our research, we conducted a comprehensive investigation comprising four major phases: pre-processing, dimensionality reduction, feature extraction, and two-fold deep learning-based defect prediction. In the pre-processing phase, we employed data cleaning techniques to handle null values and missing data, ensuring the integrity of the dataset. Additionally, we applied Decimal scaling normalization to normalize the data for subsequent analysis. To address the issue of high-dimensional data and the associated "curse of dimensionality," we introduced the Incremental ICPCA method. This technique effectively reduced the dimensionality of the data, allowing for more efficient and accurate defect prediction. In the feature extraction phase, we utilized statistical features such as standard deviation, skewness, variance, and kurtosis, along with MI and CE. These features provided valuable insights into the data and played a crucial role in the subsequent defect prediction process. One of the significant contributions of our work was the development of the ED-MAD method for feature selection. This technique ensured that only relevant features were considered, eliminating noise and improving the overall accuracy of defect prediction. Furthermore, our O-TFDLF, which incorporated the RBFN and an optimized MLP, demonstrated enhanced performance in defect prediction. The fine-tuning of MLP weights using the LCMO algorithm further improved the accuracy of our model. Our work was implemented using MATLAB software, and we conducted a comprehensive evaluation of the proposed model's performance. We compared our approach against existing models, considering various performance metrics such as accuracy, sensitivity, precision, specificity, and Mean Squared Error (MSE). The achieved accuracy of 93.37% demonstrated the significance and effectiveness of our proposed model in accurately predicting software defects. In conclusion, our study made a significant contribution to the field of SDP by introducing a novel approach that combined a two-tier DL framework. The successful integration of pre-

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

processing, dimensionality reduction, feature extraction, and defect prediction phases provided accurate and reliable predictions, leading to improved software quality. The significance of our work lies in its potential to enable early defect identification, resulting in more efficient software development processes and enhanced overall software quality.

REFERENCE

[1]. X. Cai, Y. Niu, S. Geng, J. Zhang, Z. Cui, J. Li, and J. Chen, "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," Concurrency and Computation: Practice and Experience, 32(5), 2020, p.e5478.

[2]. Y. Qu, and H. Yin, "Evaluating network embedding techniques' performances in software bug prediction," Empirical Software Engineering, 26, 2021, pp.1-44.

[3]. A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," International journal of advanced computer science and applications, 9(2), 2018.

[4]. Z.W. Zhang, X.Y. Jing, and T.J. Wang, "Label propagation based semi-supervised learning for software defect prediction". Automated Software Engineering, 24, 2017, pp.47-69.

[5]. D.L. Gupta, and K. Saxena, "Software bug prediction using object-oriented metrics," Sādhanā, 42, 2017, pp.655-669.

[6]. Sunanda, P. ., Janardhanan, K. A. ., Gupta, R. ., Tannady, H. ., Shrivastava, N. K. ., & Sharma, T. K. . (2023). Distributed Hashing Based Group Management Scheme for the Peer-to-Peer Trust Model. International Journal of Intelligent Systems and Applications in Engineering, 11(3s), 08–13. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/2525

[7]. Z. Yuan, X. Chen, Z. Cui, and Y. Mu, "ALTRA: Cross-project software defect prediction via active learning and tradaboost," IEEE Access, 8, 2020, pp.30037-30049.

[8]. X. Chen, Y. Mu, K. Liu, Z. Cui, and C. Ni, "Revisiting heterogeneous defect prediction methods: How far are we?," Information and Software Technology, 130, 2021, p.106441.

[9]. X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction," Information and Software Technology, 93, 2018, pp.1-13.

[10]. D. Chen, X. Chen, H. Li, J. Xie, and Y. Mu, "Deepcpdp: Deep learning based cross-project defect prediction," IEEE Access, 7, 2019, pp.184832-184848.

[11]. Z. Wan, X. Xia, A.E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, expectations, and challenges in defect prediction," IEEE Transactions on Software Engineering, 46(11), 2018, pp.1241-1266.

[12]. H.K. Dam, T. Pham, S.W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.J. Kim, "A deep tree-based model for software defect prediction," arXiv preprint arXiv:1802.00921, 2018.

[13]. X, Li, H. Jiang, Z. Ren, G. Li, and J. Zhang, "Deep learning in software engineering," arXiv preprint arXiv:1805.04825, 2018.

[14]. C. Pan, M. Lu, B. Xu, and H. Gao, "An improved CNN model for within-project software defect prediction," Applied Sciences, 9(10), 2019, p.2138.

[15]. Mr. Vaishali Sarangpure. (2014). CUP and DISC OPTIC Segmentation Using Optimized Superpixel Classification for Glaucoma Screening. International Journal of New Practices in Management and Engineering, 3(03), 07 - 11. Retrieved from http://ijnpme.org/index.php/IJNPME/article/view/30

[16]. C. Manjula, and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," Cluster Computing, 22(Suppl 4), 2019, pp.9847-9863.

[17]. Dhablia, A. (2021). Integrated Sentimental Analysis with Machine Learning Model to Evaluate the Review of Viewers. Machine Learning Applications in Engineering Education and Management, 1(2), 07–12. Retrieved from http://yashikajournals.com/index.php/mlaeem/article/view/12

[18]. D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," arXiv preprint arXiv:2301.08653, 2023.

[19]. L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," Neurocomputing, 385, 2020, pp.100-110.

[20]. Pérez, C., Pérez, L., González, A., Gonzalez, L., & Ólafur, S. Personalized Learning Paths in Engineering Education: A Machine Learning Perspective. Kuwait Journal of Machine Learning, 1(1). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/107

[21]. H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic LSTM model for software defect prediction," IEEE Access, 7, 2019, pp.83812-83824.

[22]. H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," Information and Software Technology, 96, 2018, pp.94-111.

[23]. Z. Xu, S. Li, J. Xu, J. Liu, X. Luo, Y. Zhang, T. Zhang, J. Keung, and Y. Tang, "LDFR: Learning deep feature representation for software defect prediction," Journal of Systems and Software, 158, 2019, p.110402.

[24]. G. Giray, K.E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, On the use of deep learning in software defect prediction. Journal of Systems and Software, 195, 2023, p.111537.

[25]. M. Nevendra, and P. Singh, "Software defect prediction using deep learning," Acta Polytechnica Hungarica, 18(10), 2021, pp.173-189.

[26]. G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," Scientific Programming, 2019.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9s
DOI: https://doi.org/10.17762/ijritcc.v11i9s.7409
Article Received: 01 May 2023 Revised: 28 June 2023 Accepted: 24 July 2023

[27]. W.Y. Ramay, Q. Umer, X.C. Yin, C. Zhu, and I. Illahi, Deep neural network-based severity prediction of bug reports. IEEE Access, 7, 2019, pp.46846-46857.