Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 44 No. 8
August 2023

Deep Learning Based Multilayer D_Highway Memory Neural Network for Software Defect Prediction Using Software Metrics

S. Thenmozhi[1], Dr. P. M. Shanthi[2]

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

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

Abstract—As technology has progressed, new hardware and software needs have evolved. There has been a significant increase in demand for software across many different applications, which has coincided with the rise of the software industry. To expand the software business, producing and maintaining high-quality software is regarded to be the most crucial duty for every company. Software engineering is critical to the software industry's success in achieving this goal. In order to create software applications, computer code is used to accomplish the required goal. Software faults may cause defective software to be developed if these scripts include certain incorrect examples. In the realm of software engineering, predicting software defects is regarded as the most significant activity that can be utilized to ensure the quality of software. As a consequence of defect prediction findings, quality assurance teams may more efficiently allocate limited resources for testing software products by focusing their efforts on the source code that is most likely to have defects. Using defect prediction methods to help developers and speed up the time to market for more dependable software products will become more vital as software projects grow in size. The process of detecting and correcting flaws is one of the most time-consuming and expensive parts of embedded software development. Complex infrastructure, large scale, high costs and time constraints make it difficult to monitor and meet quality standards. A unique deep learning-based software fault prediction model has been shown in this study. A single spectrum stream flow filter was first used to remove data errors. The minkowski prewitt clustering approach was then utilized to group the software characteristics. Then, using the first-order binary - α sail fish optimization technique, we choose the best defective characteristics. Finally, a multilayer D highway memory neural network is used to identify different sorts of software error codes according on the severity of the problem. Using the PROMISE JM1 software defect prediction dataset in the python environment, the whole process was developed. Accuracy, AUC, precision, recall, and F1 score were used to evaluate the efficacy of the proposed methodology's performance.

Keywords— software defect, singular spectrum stream flow filter, minkowski prewitt clustering, first order binary α - sail fish optimization algorithm, multilayer D_highway memory neural network

I. INTRODUCTION

A rise in the demand for a wide range of applications has been seen during the last two decades. Many software applications are created for business or personal usage in order to satisfy the needs of the client. For both industrial and personal use, software quality remains an unresolved problem because of the mass manufacturing of software programs. For this reason, the introduction of software testing has been made possible by the introduction of testing tools that assist to detect and fix flaws or faults in software applications. As the need for contemporary technology-based industries and business applications grows, an ever-increasing number of software applications are built each year, yet software quality is mostly ignored throughout this process. Deep learning has made it possible for researchers to use deep learning to detect the software quality by identifying its defects and faults. As a result, we have developed an algorithm for the first order binary - α sail fish optimization and a multi-layer D highway memory neural network that can identify software flaws across the PROMISE repository using a hierarchical architecture of deep learning. Within project file-level software defect prediction is the topic of this study, which assesses prediction performance using metrics under both non-effort-aware and effort-aware

scenarios. Included in this article is the most significant contribution,

- To classify faults and continuously learn to improve the accuracy of deep learning.

- We first use α - sail fish optimization to analyze the source code and to isolate the faulty features

- In the classification part we use multilayer D-highway memory neural network for classifying the code faults.

The remainder of the document is structured as follows. The literature review is presented in section two, the issue statement is discussed in section three, and the suggested softaware classification model and its architecture are shown in section four. Sections five and six are dedicated to the implementation, results, and conclusion of this research, respectively.

II. LITERATURE SURVEY

This section provides a short overview of modern strategies for predicting software defects, including software metrics-based defect prediction, optimization schemes, machine learning techniques, and hybrid techniques (combination of optimization and machine learning). This section begins with a discussion of well-established methods for predicting software defects based on metrics. One of the most widely used Machine Learning algorithms, Linear Classifier (LC) [1-15], was shown to perform better than the other algorithms tested. A NASA Promise data repository provided the data used in this investigation, which was quite comparable to the data they used in their own investigation. Using a 10-fold cross validation, the seven datasets are analyzed and classified. For the research in [2], the author used metrics extracted from the top-k metrics to forecast the likelihood of software defects. The PROMISE repository was used as a source of data for the study. It was inspired by this thought that they [3] employed dimensionality reduction to extract the dataset's top six features. exploring the root cause of SDLC faults and the numerous sorts of faults that may arise at different stages of SDLC. Clustering methods such as agglomerative clustering, COBWEB, k-means, density-based scan, expectation maximization, and furthest first were used. More specifically, [4] employed supervised-classification learning algorithms and unsupervised-clustering learning algorithms to anticipate defects more effectively. The k-means method was the fastest and most accurate of all unsupervised algorithms. Using datasets with a high number of dimensions to train a defect prediction model takes a long time, according to research [5.] New hybrid SDP models are employed to identify the metrics in this study. A variety of datasets and classification techniques, including SVM and ANN, were included. In order to compare the outcomes, AUC and MEWMA were used. [6] utilized data from NASA's repositories in this study. As a way of resolving issues with class imbalance in the SDP model, they came up with a new framework for analysis. It's clear from [7] that combining feature mapping and feature selection has a considerable impact on HFP models.. When it comes to feature optimization, [8] present a hybrid technique that combines genetic algorithm (GA) with deep neural network (DNN). Sample approaches and cost sensitive classifiers are used to assess [9] the performance of machine learning classifiers for software fault prediction on twelve unbalanced NASA dataset . [10] Java code defects are better predicted with the aid of Abstract Syntax Tree n-grams (AST n-grams). Defect prediction using SVMs may be improved utilizing an unique filtering approach (FILTER) presented in [11.] Linear, polynomial and radial basis function SVM-based classifiers are built using the suggested filtering approach and their performance is tested on five datasets. Seven-ensemble machine learning for SDP is proposed in [12]. Classification methods will be examined side by side to see how well they handle the data imbalance and large dimensions seen in defect datasets, as detailed in [13]. For determining whether or not a software module includes faults, [14] present a defect distribution prediction model (Deep belief network prediction model, DBNPM).. [15] For the WPDP model, they suggested an enhanced CNN model that was compared to current CNN models as well as an empirical investigation. [16] Their paper proposes a geodesic flow learning cross-project software fault prediction approach.. [17] offer Seml , a new framework for defect prediction that integrates word embedding and deep learning algorithms. To begin, they extract a token sequence from the abstract syntax tree of each program source file. They then use a mapping table trained with an unsupervised word embedding model to translate

each token in the sequence to a real-valued vector. Finally, they establish a Long Short Term Memory (LSTM) network using the vector sequences and their labels (defective or non-defective) [18]. The enormous quantity of unlabeled data in the kernel space, as well as the limited labeled defect data, [19] may be fully used by CKSDL. CKSDL also takes into account the misclassification costs while learning a lexicon. Bootstrap aggregating, an ensemble learning approach for software fault prediction in object-oriented modules, has been suggested [20].

As of from the existing research because the performance of each feature in defect prediction might vary from software to software, it is difficult to identify the appropriate ratios of two types of features in the suitable feature combination. The following prediction method may be harmed by adding noise simply by concatenating two different types of information. An innovative approach is proposed to assure the extraction of each characteristic and uses a classification technique to automatically identify the defective software.

III. PROPOSED METHODOLOGY

To address the existing issue, a deep learning-based optimization method is provided for predicting software flaws.

Fig.1 Schematic representation of the implemented methodology

A.Dataset

The data set used for the experimentation was JM1/software defect predictions dataset obtained from Kaggle (https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction). The dataset belongs to PROMISE data set. JM1 is written in "C" and is a real-time predictive ground systemThe dataset contains 746 projects with 20 independent features and a dependent feature with two classifications.

B. Preprocessing

."Normalization" is the process through which data values are turned into something that can be more easily understood and used.

Data preparation may benefit from the use of the single spectrum stream flow filter (SSSFF). I is a constant term in the SSSFF, which is combined with the reference value Z $_{ref}$ and extra terms representing polynomial trends of degree $u = 1, \dots, b$ (c^u) as well as a residual term in a linear combination (w).

$$Z = I + Z_{ref}.v + \sum_{u=1}^{b} s_u c^u + w \quad (1)$$

The variance in data intensity may be used to estimate the variation in data.

$$Y_{ij} = \sum_{i=0}^{\infty} (-1)^i \binom{n}{i} m_i \times xy_{ij}(Z) \times e_{ij,} \quad (2)$$

Data variation is represented by y, x_{ij} the errorly constants are m, n, e, the hash function is represented by i j, and the normalization factor is represented by Z.

Where,

$$Z = log\ x, \Omega = log\ Z, \mu = \log m, p(\Omega) = \log r\,(Z), \varepsilon = \log e$$

$$y_{ij} = \mu_i + p_{ij}(\Omega) + \varepsilon_{ij.}$$

The randomness of the input error,

$$p_{ij} = \sum_s \mu_{is} \left(\Omega_{sj} - \langle \Omega_{s.} \rangle\right) \quad (3)$$

Where the average $\langle p_{ij} \rangle$ is taken over the samples j=1....M,i.e. $\langle \Omega_{s.} \rangle \equiv \frac{1}{M} \sum_j \Omega_{sj}$.

Each value in the dataset is used to estimate the coefficients l, v, and s u, while the residual term w is used to account for the constant variance. It is thus possible to represent the SSSFF-corrected data this way:

$$Z_{corr} = \frac{Z-l-\sum_{u=1}^{b} s_u c^u}{v} = Z_{ref} + \frac{w}{v} \quad (4)$$

Error interference, data variation replication, and other issues are all handled by direct expansions to the SSSFF technique.

C. Data clustering

Minkowski Prewitt clustering was used to group the data in this section. These notations will be explained: crisp, Minkowski Prewitt, α cut of a Minkowski, support of a Minkowski, and the core (sets are usually denoted by upper case letters, and their members by lower case letters).

The binary membership function of a crisp set L in the discourse universe Y means that it has no uncertainty. As a collection of ordered pairs, it is known as

$$L = \{(z, \phi_L(z)) | z \in Y\} \quad (5)$$

"where$\phi_L(z)$is the binary membership function:$\phi_L(z)$ = 1 if z∈L, and $\phi_L(x)$ = 0 if z ∉L. $\phi_L(z) \in \{0, 1\}$".

A minkowski set $L^{\sim}$ there is uncertainty in the minkowski membership function of Y in the discourse world. As a collection of ordered pairs, it is known as:

$$L^{\sim} = \{(z, \mu_{L^{\sim}}(z)) | z \in Y\} \quad (6)$$

where$\mu_{L^{\sim}}$ zero to one degrees of minkowski membership is the function that returns zero to one degrees $\mu_{L^{\sim}}(z) \in [0, 1]$.

What a Minkowski set L has to do with speech The items of Y that have membership values in L equal to 1 are included in Y, which is a compact set.:

$$core(L^{\sim}) = \{z \in Y | \mu_{L^{\sim}}(z) = 1\} \quad (7)$$

"A minkowski set L in the universe of discourse is supported It's possible to create a set called Y that includes all of the items of Y that have nonzero membership values in L":

$$supp(L^{\sim}) = \{z \in Y | \mu_{L^{\sim}}(z) > 0\}$$
(8)

"An α − cut a minkowski set $L^{\sim}$is a crisp set $L_{\alpha}^{\sim}$that containsall the elements in Ythat have membership values in L greater than or equal to α, that is" :

$$L_{\alpha}^{\sim} = \{z \in Y | \mu_{L^{\sim}}(z) \geq \alpha\} \quad (9)$$

Let $Z = \{z_1, \dots, z_v, \dots, z_b\}$be a set of b , and$C = \{c_1, \dots, c_v, \dots, c_x\}$ in two-dimensional feature space consist of x centroids. Zinto x clusters are partitioned by the Minkowski Prewitt by reducing the following objective function

$$H = \sum_{h=1}^{b} \sum_{u=1}^{x} (y_{uh})^n \|z_h - c_u\|^2 \quad (10)$$

"where1 ≤ n≤ ∞ is the prewitt fuzzifier, $c_u$is the u^{th}centroid corresponding to cluster β_u, y_{uh}∈[0, 1] is the minkowski membership of the pattern $z_h$to cluster β_u, and ||.||is the distance norm such that",

$$c_u = \frac{1}{b_u} \sum_{h=1}^{b} (y_{uh})^n z_h \text{where} b_u = \sum_{h=1}^{b} (y_{uh})^n z_h$$
(11)

and

$$y_{uh} = \frac{1}{\sum_{j=1}^{x} \left(\frac{s_{uh}}{s_{jh}}\right)^{\frac{2}{n-1}}} \text{where} s_{uh}^2 = \|z_h - c_u\|^2$$
(12)

The first step in MP is to randomly choose xobjects to serve as the centers (means) of the x clusters. Z h's membership is determined by the Euclidean distance (Eq.) 12 between it and the centroids . To determine the cluster centroids, we must first find out which objects are members of each cluster using Eq (11). if the previous iteration's centroids matched the current iteration's produced centroids, the procedure ends.

To be added to the crisp set, information must fulfill three criteria: (1) be part of the support of the minkowski set, (2) be in a 4-neighbourhood, and (3) reduce the distance between the minkowski set and the crisp set, including the new data, at each iteration of this method.

Data groupings d and c are indistinguishable in terms of the Minkowski distance $X_j$at iteration jis:

$$s_n(X_j, d) = \sqrt[n]{\sum_{u=1}^{length(X_j)} |X_j(u) - d(u)|^n} \quad (13)$$

After adding a data $o_u$to the crisp set, the Minkowski distance between d and the new crisp set $X_{new}\left(X_{j\cup\{o_u\}}\right)$becomes:

$$s_n(X_{new}, d) = \sqrt[n]{\sum_{u=1}^{length(X_j)} |X_{new}(u) - d(u)|^n} \quad (14)$$

Eq. (14) can be rewritten as:

$$s_n(X_{new}, d) = \left(|X_{new}(o_u) d(o_u)|^n + \sum_{u=1;u\neq o_u}^{length(X_j)} |X_{new}(u) - d(u)|^n\right)^{1/n}$$
(15)

“The difference between $X_j$and $X_{new}$is only in the element at location o_u. Thus, in order to include $o_u$in X_j, $\left|X_j(o_u) - d(o_u)\right|^n$would be subtracted from $|X_{new}(o_u) - d(o_u)|^n$, and the condition $h \neq o_u$in the summation of Eq. (15) would be removed”.

“As the singleton datao_u is added to X_{new}, $X_{new}(o_u)$ =1, and the corresponding location in $X_j(o_u)$ = 0. By substitutingthose values, we get”:

$$s_n^{\sim}(X_{new}, d) = \left(|1 - d(o_u)|^n - |-d(o_u)|^n + \sum_{u=1}^{length(X_j)} \left|X_j(u) - d(u)\right|^n\right)^{1/n} \quad (16)$$

It can be seen from Equation (16) that the value of d(o u)increases, and the distance value decreases, when d(ou) indicates the degree of membership of some data, which at the same time represents the characteristic of that data. In other words, data with the greatest degree of membership are those that are selected in order to keep the minkowski set as close to the crisp set as possible. Each iteration of the algorithm will eliminate some data from the crisp set.

The Minkowski distance between d and the new crisp set X new is increased once a data ou is removed from the crisp set.

$$s_n(X_{new}, d) = \sqrt[n]{\sum_{u=1}^{length(X_j)} |X_{new}(u) - d(u)|^n} \quad (17)$$

Following the previous procedure we conclude:

$$s_n^{\sim}(X_{new}, d) = \left(|-d(o_u)|^n - |1 - d(o_u)|^n + \sum_{u=1}^{length(X_j)} \left|X_j(u) - d(u)\right|^n\right)^{1/n}$$
(18)

It can be noticed from Eq. (18) that as the value of$d(o_u)$decreases. The distance value decreases.

As a result, the data with the lowest degree of membership will be eliminated from clustering in order to reduce the gap between the minkowski set and the crisp set.

D. Feature extraction

Then for extracting the faulty code features first order binary α - sail fish optimization (B α – SFO) algorithm is used. Group hunting is a fascinating example of social behavior in a variety of organisms, including insects, fish, birds, and mammals. Predators kill their prey with less effort while hunting in groups than when hunting alone. Prowlers attack prey in groups with little or no coordination of assault, but in more sophisticated groups they employ distinct roles to herd and capture animals... This is one of the most complicated group hunting methods, in which assaults are alternated.

There are a total of S features or dimensions in this feature set and each class has an associated label, which we'll call X = $\{x_1, \ldots, x_k\}, D = \{d_1, d_2, \ldots. d_S\}$, where S is how many features or dimensions there are in the feature set. A subset A = $\{A_1, \ldots, A_n\}$, where $m < S$, A $\subset D$ has a lower classification error rate than any other subset of the same size or any suitable subset of A is found through an optimization procedure. To put it another way, DA is an optimization issue that can only be solved using binary numbers 0, 1. If a feature has been chosen, it will have a value of 1, and otherwise, it will have a value of 0, in this example. The number of features in the original dataset determines the size of this vector. To handle continuous optimization problems when the solution consists of actual values, the first order binary - sail fish optimization technique is presented In order to convert the typical algorithm's continuous search space to a binary one, we employ a transfer function. Equation 19 represents the alpha transfer function,.

$$R(z)=\alpha\frac{1}{1= w^{-z}} \quad (19)$$

The current location of the Sailfish will now be updated using the probability values given by Equation 20.

$$Z^s(r) = \begin{cases} 1 \ if \ ebs \ < R(Z^s(r)) \\ 0 \ if \ ebs \ \geq R(Z^s(r)) \end{cases}$$
(20)

This is a multi-objective, two-pronged task: (I) to maximize classification accuracy (ie, the maximization issue), and (v) to pick the smallest number of features possible (ie, the reduction problem) (i.e. minimization problem). In this case, the two goals are in direct conflict. We looked at the categorization mistake rate to get rid of this inconsistency. The FS issue is transformed into a single-objective problem using Equation 21.

$$\downarrow Fitness = \alpha\gamma(A) + (1 - \alpha)\frac{|A|}{S}$$
(21)

"where A represents the selected feature subset, $|A|$ represents cardinality of the selected feature subset or number of selected features, γ (A) represents classification error rate of A, S is the original dimension of the dataset and α ∈ [0, 1] represents weight".

Sailfish and sardine populations are randomly initialized, and encircling strategies following hyper-sphere neighborhood are used to take care of the exploration now. Using the sardine population and migration around the best sailfish and sardine, the exploitation is dealt with. Algorithm exploration and exploitation capabilities are balanced by LRJ, a parameter. A good search and exploration of the search space is needed to locate the optimal feature subset. This is where the DA approach comes into play. In this way, we've improved VAD's exploration and exploitation capabilities via the use of LGX. In the worst-case scenario, the LβVAD analysis shows that the time complexity is maximum number of iterations, B_{aes} is the number of sardines, $r_{fitness}$ is the time required to calculate the fitness value of a specific agent using a given classifier, and S is the dataset dimension. Finally, it is possible to extract the problematic characteristics. The retrieved features may be used as a classification input.

E. Classification Using the MDHMNN model, a kernel may evaluate a local patch in input sequence and extract dependency among residues in the small patch. Correlations between more distant residues in the input sequence may be accounted for by stacking convolutional layers on top of each other. More convolutional layers enable the extraction of long-range relationships between residues at greater distances in a model with more convolutional layers. Highway models, on the other hand, will eventually lose the local contexts that lower levels were able to capture. MD-HMNN, a revolutionary approach proposed in this study, aims to tackle this issue. Any two neighboring convolutional layers of a multi-layer CNN may be connected using the MD-interconnector. HMNN's Convolutional layers provide complexity to MD-HMNN, allowing it to extract not just long-range interdependencies, but also the local contexts derived from lower levels via highway. In Fig. 4 the MD-HMNN frame is seen. Input, multi-scale CNN with highway, and output sections are all shown in Figure 4 of MD-HMNN OAA. $z_u \in E^n$ is the feature vector as a concatenation of sequence features and evolutionary information for each u_{th} residue in feature**.**

"Thus a feature of length K is encoded as a K × n matrix $z_{1:K} = [z_u,\ z_2, \ldots, z_K]^R$, where K and m denote the length of feature and the number features used to encode residues, respectively. In this study, m equals to 42. In order to keep the output of convolutional layer have the same height with the input, we need to pad ⌊g/2⌋ and ⌊(g − 1)/2⌋ m-dimensional zero vectors to the head and the tail of the input $z_{1:K}$, respectively, where g is the length of convolutional kernels in the convolutional layer. The second section contains two parts: multi-scale CNN and highway, where the multi-scale CNN contains n convolutional layers. In the (r − $1)^{th}$ layer, the convolution operation of the Jth kernel $Q_j^{r-1} \in E^{g\times n}$ executed on feature fragment $Z_{u:u+g-1}$ is expressed as"

$$x_{j,u}^{r-1} = d^{r-1}(Q_j^{r-1} \cdot Z_{u:u+g-1} + v_j^{r-1}$$
(22)

"where g is the length of convolution kernel v_j^{r-1} is the bias of the Jth kernel, d is activation function and $Z_{u:u+g-1}$ denotes the feature fragment $z_u, z_{u+1}, z_{u+2}, \ldots, z_{u+g-1}$. Through executing convolution operation of the Jth kernel on all fragments with length g of the padded input, we get a novel feature vector"

$$x_j^{r-1} = [x_{j,1}^{r-1}, x_{j,2}^{r-1}, x_{j,3}^{r-1}, \dots, x_{j,k}^{r-1}]^R \quad (23)$$

The convolutional layer has n kernels, therefore we can generate n new feature vectors. We may create a novel feature matrix by concatenating the s new feature vectors K × s

$$x^{r-1} = [x_1^{r-1}, x_2^{r-1}, x_3^{r-1}, \dots, x_k^{r-1}] \quad (24)$$

The following convolutional layer takes these new feature matrices as input. If there are b convolutional layers and θ_r is used to denote the kernels and the bias of the rth convolutional layer, then the output of the bth convolutional layer is"

$$x^b = d_{\theta_b}^b\left(d_{\theta_{b-1}}^{b-1}(\cdots d_{\theta_1}^1(z_{1:K}))\right) \quad (25)$$

Finally, the output of the bth prediction is made using a convolutional layer as input to a fully connected softmax layer $t_u = argmax(Q \cdot x_u^n + v)$ (26)

where q and v is the weight and bias of the fully connected softmax layer, respectively. x_u^n is the feature vector of the uth outputted by the bth convolutional layer and t_u is its predicted secondary structure. When it comes to software defect prediction, MD-HMNN uses a highway network and a multilayer CNN to extract both local contexts and long-range connections. There are three access points to each convolutional layer except the final one in MD-HMNN. The output and convolution kernels of the next layer are each served by two separate accesses, one from the current layer and the other from the next. The other is a weight that is used to define the proportion of information from the roadway that is included. This means that the output x^r of the rth convolutional layer is the weighted sum of the information given by roadway from the previous layer and the information outputted by the convolution kernels of the current layer

$$m^r = \delta(q^m x^{r-1}) \quad (27)$$

$$x^r = (1 - m_r) \times d_{\theta_r}^r(x^{r-1}) + m_r \times x^{r-1} \quad (28)$$

"where δ(.) is alpha function, m_r is the weight of the highway and $d_{\theta_r}^r(.)$ is the convolution operation of current convolutional layer. So the output of the rth convolutional layer contains two portion: information from the (r − 1)th highway convolutional layer and the convolution kernels of the current layer's convolutional layer Finally the defected software was classified."

IV. Proposed Methodology

Experiments for predicting software defects using the suggested technique are described in this section. In this research, a method for predicting software problems using a Python tool makes use of optimization and MD-HMCNN classification approaches. Because of this investigation, we looked at the PROMISE JM1 database of software defect datasets.

2(a)

click to expand output; double click to hide output

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_C
							30	1.30	1.30	1.30	...	2	2	2	2	1.2	1.2	1
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	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	17	36	1
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	17	135	3
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	11	16	[illegible]

5 rows × 22 columns

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	
count	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	1.088500e+04	10885.00
mean	42.016178	6.348590	3.401047	4.001599	114.389738	673.758017	0.135335	14.177237	29.439544	3.683637e+04	0.22
std	76.593332	13.019695	6.771869	9.116889	249.502091	1938.856196	0.160538	18.709900	34.418313	4.343678e+05	0.64
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.00
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000	3.000000	11.860000	1.619400e+02	0.02
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000	9.090000	21.930000	2.031020e+03	0.07
75%	46.000000	7.000000	3.000000	4.000000	119.000000	621.480000	0.160000	18.900000	36.780000	1.141643e+04	0.21
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000	418.200000	569.780000	3.107978e+07	26.95

2(b)

Fig. 2(a,b) Sample input and the standardized output

Figure 2 illustrated the recommended JM1 software defect prediction dataset's parameter, which was shown in the table. It offers explanations of the datasets' various parameters, such as the number of incidents modules, , and the number of metrics.The size of the dataset should be of (10885, 22)

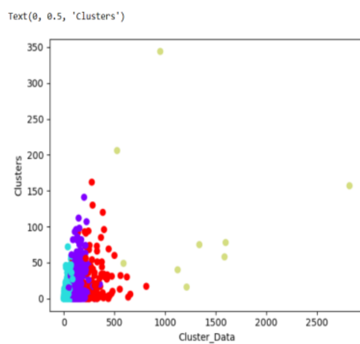

Fig.3 Clustered output

Three clustered output showing the variation between different features involved in the dataset.

Serial	Probability	Importance	Rule	Defects
1	1.000	0.592	Vg = 68.3 - 218, LO Comment = 22.9 - 46.7	TRUE
2	1.000	0.592	Branch Count >= 137.9, LO Comment = 22.9 - 46.7	TRUE
3	1.000	0.588	LO Code And Comment >= 84.4, Branch Count >= 137.9	TRUE
4	1.000	0.582	Evg >= 48.3, LO Comment = 22.9 - 46.7	TRUE
5	0.917	0.567	Ivg >= 37.3, LO Comment = 4.1 - 22.9	TRUE
6	0.889	0.564	Evg >= 48.3, Branch Count >= 137.9	TRUE
7	0.853	0.558	Branch Count >= 137.9, L < 0.03	TRUE
8	0.900	0.554	Vg = 68.3 - 218, Uniq Op = 23.5 - 32.7	TRUE
9	0.900	0.554	Ivg >= 37.3, LO Comment = 22.9 - 46.7	TRUE
10	0.900	0.554	LO Code And Comment >= 84.4, LO Comment >= 46.7	TRUE
11	0.867	0.550	Evg >= 48.3, Vg = 68.3 - 218	TRUE
12	0.833	0.549	Branch Count >= 137.9	TRUE
13	0.850	0.548	Evg >= 48.3, Ivg >= 37.3	TRUE
14	0.889	0.546	LO Code And Comment >= 84.4, Ivg >= 37.3	TRUE
15	0.889	0.546	Branch Count >= 137.9, Uniq Op = 23.5 - 32.7	TRUE
16	0.889	0.546	Branch Count >= 137.9, LO Comment = 4.1 - 22.9	TRUE
17	0.846	0.537	Uniq Op >= 32.7, LO Comment = 22.9 - 46.7	TRUE
18	0.875	0.536	Branch Count >= 137.9, Uniq Op = 15.6 - 23.5	TRUE
19	0.875	0.536	Ivg >= 37.3, LO Code And Comment = 43.8 - 84.4	TRUE
20	0.875	0.536	Vg = 68.3 - 218, Uniq Op = 15.6 - 23.5	TRUE
1	0.775	0.290	Evg < 18.0	FALSE
2	0.400	-0.275	Evg = 18.0 - 25.6, Uniq Op = 15.6 - 23.5	FALSE
3	0.400	-0.275	LO Code And Comment = 43.8 - 84.4, Uniq Op = 23.5 - 32.7	FALSE
4	0.400	-0.275	Evg = 18.0 - 25.6, LO Code And Comment = 5.1 - 17.4	FALSE
5	0.400	-0.275	LO Comment >= 46.7, Uniq Op = 23.5 - 32.7	FALSE
6	0.407	-0.275	Branch Count = 37.2 - 84.1, L = 0.03 - 0.06	FALSE
7	0.407	-0.274	Evg = 18.0 - 25.6, Vg = 20.7 - 68.3	FALSE
8	0.400	-0.273	Ivg >= 37.3, LO Comment < 4.1	FALSE
9	0.407	-0.272	Branch Count = 37.2 - 84.1, Uniq Op < 8.0	FALSE
10	0.411	-0.270	Evg = 18.0 - 25.6, Ivg = 5.6 - 37.3	FALSE
11	0.409	-0.270	LO Code And Comment = 43.8 - 84.4, Evg < 18.0	FALSE
12	0.411	-0.269	LO Code And Comment = 43.8 - 84.4	FALSE
13	0.413	-0.269	LO Comment = 22.9 - 46.7, Ivg = 5.6 - 37.3	FALSE
14	0.409	-0.265	LO Comment >= 46.7, Uniq Op = 15.6 - 23.5	FALSE
15	0.417	-0.264	LO Comment = 22.9 - 46.7, L < 0.03	FALSE
16	0.424	-0.264	Vg = 20.7 - 68.3, Ivg = 5.6 - 37.3	FALSE
17	0.419	-0.263	Vg = 20.7 - 68.3, LO Comment = 4.1 - 22.9	FALSE
18	0.412	-0.261	LO Code And Comment = 43.8 - 84.4, L = 0.03 - 0.06	FALSE
19	0.412	-0.261	Vg = 20.7 - 68.3, Branch Count = 9.8 - 37.2	FALSE
20	0.420	-0.260	LO Comment >= 46.7	FALSE

Fig. 4 Simulated faults output

The simulated software fault prediction of the suggested algorithm was demonstrated using a sample that was illustrated in figure 4.

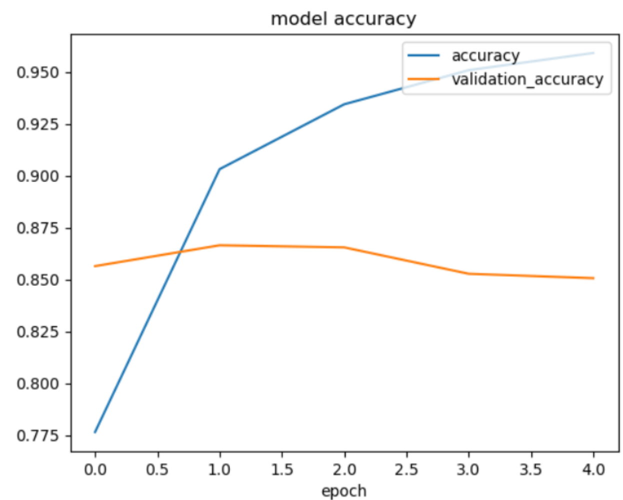

Fig. 5(a) Epoch Vs. accuracy

As of from figure 5(a) training and validation accuracy was calculated. Here high level of training and testing accuracy was obtained shows the efficiency of the mechanism.

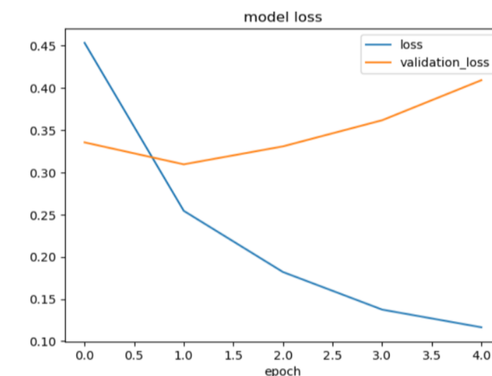

Fig. 5(b) Epoch Vs. Loss

Figure 65(b)shows the validation and training losses. The proposed model's training loss was not excessive in this case. Our model is underfitted if our training

and validation losses are almost identical. But here we increase the number of layers or the number of neurons in each layer in order to limit the amount of training loss.To prove the efficiency of the suggested technique it can be compared with the existing mechanisms [21,22]. The accuracy, precision, recall, and F1score of the proposed system are used to assess its performance.

TABLE 1 Performance evaluation of the suggested methodology

Methodology	Precision	Recall (%)	F score (%)	Accuracy(%)
NB [22]	53.7	22.6	31.8	79.8
KSTAR[22]	83	87	85.3	75.9
KNN[22]	82.9	84.6	83.7	73.9
DT[22]	49	26	34.8	79.4
Proposed	86	85	85	95.9

TABLE 2 Comparartive performance analysis

Source	Algorithm%	*F*-measure%	Accuracy%
[21]	RF	0.76	77.00%
	DS	0.71	71.00%
	SVM	0.71	69.00%
[21]	Naive Bayes	0.89	81.43%
	MLP	0.90	89.97%
	SVM	0.90	81.73%
	RBF	0.90	81.61%
[21]	J48	N/A	79.81%
[21]	RBF	0.87	84.87
This research	MD-HMCNN	0.85	95.9

As of from the result obtained from the table 1,2 it was revealed that the suggested methodology express satisfied results than other existing mechanisms.

V. CONCLUSION

Software fault prediction was improved using the Bα – SFO and the MDHMNN in this study. Improved accuracy may be achieved by using MDHMNN to choose the optimum parameters for this model. Tests on a variety of software failure prediction data sets were conducted using MDHMNN. These data sets were made available via the PROMISE JM1 repository. Performance criteria such as F1 score, accuracy, precision, and recall were used to assess the effectiveness of the recommended approach. Bα – SFO and the MDHMNN were the most effective in comparison to present methods. Data sets and performance measures were all beaten by this algorithm by obtaining high range of the accuracy(95.9%). A new method for reducing the algorithm's computational costs might be discovered in future research.

REFERENCES

[1] P. D. Singh and A. Chug, "Software defect prediction analysis using machine learning algorithms," in *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*, 2017, pp. 775-781.

[2] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology,* vol. 59, pp. 170-190, 2015.

[3] B. Grishma and C. Anjali, "Software root cause prediction using clustering techniques: A review," in *2015 Global Conference on Communication Technologies (GCCT)*, 2015, pp. 511-515.

[4] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: A survey," *Expert Systems with Applications,* vol. 172, p. 114595, 2021.

[5] S. Huda, S. Alyahya, M. M. Ali, S. Ahmad, J. Abawajy, H. Al-Dossari*, et al.*, "A framework for software defect prediction and metric selection," *IEEE access,* vol. 6, pp. 2844-2858, 2017.

[6] S. Choirunnisa, B. Meidyani, and S. Rochimah, "Software Defect Prediction using Oversampling Algorithm: A-SUWO," in *2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS)*, 2018, pp. 337-341.

[7] R. Arora and A. Kaur, "Heterogeneous Fault Prediction Using Feature Selection and Supervised Learning Algorithms," *Vietnam Journal of Computer Science,* pp. 1-24, 2022.

[8] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing,* vol. 22, pp. 9847-9863, 2019.

[9] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," *Neurocomputing,* vol. 343, pp. 120-140, 2019.

[10] T. Shippey, D. Bowes, and T. Hall, "Automatically identifying code features for software defect prediction: Using AST N-grams," *Information and Software Technology,* vol. 106, pp. 142-160, 2019.

[11] S. Goyal, "Effective software defect prediction using support vector machines (SVMs)," *International Journal of System Assurance Engineering and Management,* vol. 13, pp. 681-696, 2022.

[12] Y. K. Saheed, O. Longe, U. A. Baba, S. Rakshit, and N. R. Vajjhala, "An Ensemble Learning Approach for Software Defect Prediction in Developing Quality Software Product," in *International Conference on Advances in Computing and Data Sciences*, 2021, pp. 317-326.

[13] S. Mehta and K. S. Patnaik, "Stacking based ensemble learning for improved software defect prediction," in *Proceeding of Fifth International onference on Microelectronics, Computing and Communication Systems*, 2021, pp. 167-178.

[14] H. WEI, C. SHAN, C. HU, Y. ZHANG, and X. YU, "Software defect prediction via deep belief network," *Chinese Journal of Electronics,* vol. 28, pp. 925-932, 2019.

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

[16] W. Liu, B. Wang, and W. Wang, "Deep Learning Software Defect Prediction Methods for Cloud Environments Research," *Scientific Programming,* vol. 2021, 2021.

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

[18] J. Sun, Y.-m. Ji, S. Liu, and F. Wu, "Cost-Sensitive and Sparse Ladder Network for Software Defect Prediction," *IEICE TRANSACTIONS on Information and Systems,* vol. 103, pp. 1177-1180, 2020.

[19] F. Wu, X.-Y. Jing, Y. Sun, J. Sun, L. Huang, F. Cui*, et al.*, "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Transactions on Reliability,* vol. 67, pp. 581-597, 2018.

[20] P. Suresh Kumar, H. S. Behera, J. Nayak, and B. Naik, "Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature," *Innovations in Systems and Software Engineering,* vol. 17, pp. 355-379, 2021.

[21] Hussain, F., Abbas, S. G., Shah, G. A., Pires, I. M., Fayyaz, U. U., Shahzad, F., ... & Zdravevski, E. (2021). A framework for malicious traffic detection in IoT healthcare environment. Sensors, 21(9), 3025.

[22] Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., & Husen, A. (2019). Performance analysis of machine learning techniques on software defect prediction using NASA datasets. International Journal of Advanced Computer Science and Applications, 10(5).