Vol. 06, No. 1 (2024) 363-372, doi: 10.24874/PES.SI.24.02.019

Proceedings on Engineering Sciences

www.pesjournal.net

COMPUTERIZED SOFTWARE QUALITY EVALUATION WITH NOVEL ARTIFICIAL INTELLIGENCE APPROACH

Dhyan Chandra Yadav[1]
Yaduvir Singh
Arvind Kumar Pandey
A Kannagi

Received 17.11.2023.
Received in revised form 17.01.2024.
Accepted 26.01.2024.
UDC – 004.8

Keywords:

Artificial Intelligence (AI), Software Quality, Software quality assurance (SQA), Software fault prediction (SFP), Software Quality prediction.

A B S T R A C T

Software quality assurance has grown in importance in the fast-paced world of software development. One of trickiest parts of creating and maintaining software is predicting how well it will perform. The term "computer evaluation" refers to use of advanced AI techniques in software quality assurance, replacing human evaluations and paving the way for a new era in software evaluation. We proposed Hybrid Elephant herding optimized Conditional Long short-term memory (HEHO-CLSTM) to estimate Software Quality Prediction. Software quality prediction and assurance has grown in importance in ever-changing world of software development. Software quality prediction encompasses a wide range of activities aimed at improving the quality of software systems via the use of data-driven approaches for prediction, evaluation and enhancement. We have collected Software Defects data and we feature extracted the attributes using linear discriminant Analysis (LDA). The suggested system improves the accuracy, AUC and Buggy instance compared with the current methods.

1. INTRODUCTION

Software quality assurance (SQA) is an approach for overseeing and controlling the production of software to ensure that it meets the specifications while keeping costs down. The use of software testing, software failure prediction and formal code inspections are the possibilities. Software fault forecasting is an effort to maximize the efficiency with which limited SQA resources are used by predicting the problem-proneness of program components (Rathore and Kumar (2019)). Finding and resolving bugs is one of the most time-consuming and expensive aspects of developing embedded software. In automotive embedded systems quality monitoring along with fulfillment presents significant challenges due to infrastructure complexity, scale, cost and time constraints. It is essential to maintain the highest standards of quality and dependability (Thota et al., (2020)). Several methods for predicting defects have been developed during the past couple of decades in an attempt to improve software quality. Machine learning is used more often. It is possible to classify these strategies as either supervised (requiring labels, whether they are right or not) or unsupervised (not requiring labels). Most models of prediction need some human intervention. Software

[1]Corresponding author: Dhyan Chandra Yadav
Email: dc9532105114@gmail.com

defect prediction (SDP) models need defect categorization labels for training, but these might be hard in reality. There has been a lot of interest in Unsupervised Defect Prediction (UnSDP) models recently (Li et al., (2020)). The term electromagnetic interference (EMI) is used to describe the unintentional transfer of energy between two circuits or systems, whether that energy is radiated through space or conducted through grounding, power or signal conductor. It's not a new problem for electronics to have EMI interference. When (analog) electronics were first developed, creating side effects was a significant worry (Herbold (2019)).

The quality and reliability of software are under increasing strain as processing power rises with the amount and complexity of software. Companies must invest time and money into hiring quality assurance staff to inspect their software for bugs (Zhu et al., (2021)). But to uncover as many defects as possible, software testing takes a lot of time to complete different test cases, making it unfeasible to run the test for the whole project when resources are limited and deadlines are short. SDP has been proposed to speed up the discovery of faulty code by the assurance team while reducing the time and money spent on software testing (Deng et al., (2020)). Studying mining software repositories (MSR) mining has grown in popularity as a method for finding valuable data on software systems and projects. Central to model-driven software development is the construction of prediction models, which in turn requires massive amounts of labeled data. While studies on the repercussions of inaccurate labels are lacking, it is nevertheless essential to consider marking accuracy while creating a predictive model. The present research investigates the consequences of mislabeled cases for prediction, which is necessary since locating SBRs in a giant bug library is critical for reducing computer product safety risks (Wu et al., (2021)).

The last decade has seen a shift in emphasis towards software-based systems, with the quality of the software itself seen as the most critical factor in the success of the system. Due to the high volume of produced application software, poor-quality software needs to be made for public and private usage. During the development phase, businesses use defect prediction design models to aid in fault prediction, effort estimation, software reliability testing, hazard analysis and other similar tasks (Prabha and Shivakumar (2020)). Developers must adhere to functional and non-functional quality standards in software development. Poor software quality is caused by the absence of non-functional quality requirements, which in turn increases the complexity and effort required for maintenance and evolution owing to the programmed design's inherent weakness. The phrase code smells is used to characterize poor software implementation architecture (Mhawish and Gupta (2020)). The objective of the research team behind this project set out to discover and use a new AI-based computerized software quality assessment methodology. Through the implementation of novel approaches, with an emphasis on cutting-edge AI techniques, this project intends to improve software quality assurance. The aim of this study is to determine how well the proposed technique, Hybrid Elephant herding optimized Conditional Long short-term memory (HEHO-CLSTM), can accurately predict software quality. A more effective and trustworthy software quality assurance procedure is the end aim and current approaches are a starting point.

1.1 Contributions of the study

- This research solves the problems of software performance prediction, which is a major step forward for software quality assurance. A groundbreaking development in software quality prediction, Hybrid Elephant herding optimized Conditional Long short-term memory (HEHO-CLSTM) employs state-of-the-art AI techniques.
- Accuracy, area under the curve (AUC) and the detection of instances that include bugs are improved by the use of data-driven methodologies, notably linear discriminant analysis (LDA).
- This forward-thinking work highlights the growing relevance of software quality prediction in the ever-changing environment of software development. It highlights the need of using modern approaches to guarantee the dependability and performance of software systems.

In the section 2 of the paper, we combine a thorough literature review for background and insight. Section 3 provides a deeper dive into the approach. In Section 4, we present an in-depth evaluation and of the results. Section 5 discussions and in the section 6, the relevance of the conclusion is discussed in depth.

2. RELATED WORKS

Massoudi et al., (2021) analysed that they use five open-source datasets that can be found in the Promise Data Repository. The prediction of software defects was an essential component in ensuring the quality of programming. Methods of deep learning can be used for the purposes above. To carry out this comparison analysis, they make use of five open-source datasets that can be found in the Promise Data Repository. The prediction of software defects was an essential component in ensuring the quality of programming. Methods based on deep learning can be used for the reasons listed above. Wu et al., (2021) described the high-impact bug report (HBR) prediction, which has seen a number of machine learning-based techniques put out in recent years. Supervised computer learning was the basis for the majority of them. It can be challenging to get sufficient quantities of labeled data, which were fundamental to the actual application of machine learning. Issues discovered during software development and maintenance was documented in bug reports.

An HBR explains a problem that, if it arises after deployment, might result in significant harm. To guarantee the quality of the software, it was essential to locate HBRs in the bug repository as soon as was practical. Radu (2019) evaluated a number of scholars and practitioners who have developed different estimating strategies in recent years. Since there were a lot of unknown factors in the software development process, some projects continue to fail because the budget and timeline were not adequately forecasted. Since many businesses had embraced agile approaches, the success rate of software projects has grown. The primary cause of the failure has switched from the creation and comprehension of the criteria to erroneous effort estimate as a result of their adaptability and ongoing customer contact.

Farid et al., (2021) discussed the significant efforts made by the software industry to raise the quality of software inside businesses. Developers and white box testers save time and effort by identifying flaws sooner with the aid of proactive software defect prediction. Code complexity, lines of code and other conventional source code characteristics were the focus of traditional software fault prediction models. Khuat and Le (2020) assessed the maintainability of source code from three distinct projects in collaboration with qualified quality analysts. Next, using code metrics, they trained machine learning algorithms to forecast how programmed classes would be evaluated for human maintainability. Tools for static code analysis were the standard for controlling and monitoring the quality of a software system. These tools provide a multitude of metrics, which the developers must evaluate to get insight into the true quality of the product. Pascarella et al., (2019) determined if each code patch that a developer submits has a software flaw or not. The technique has the benefit of being quick and simple to monitor. The worst problem was that the data set category imbalance has an impact on Just-in-Time software's forecasting reliability. Aziz et al., (2019) examined the extent to which inheritance metrics aid in the prediction of software fault proneness. Software bugs can range from minor irritations to catastrophic errors. Software fault prediction (SFP) research from recent times suggests that to facilitate testing, defects should be predicted before deployment. Object-oriented programming was more intricate than procedural languages, which include many dimensions and inheritance as a critical component. Rizwan et al., (2019) examined the performance metrics of 14 commonly used, non-graphic classifiers utilized in software failure prediction (SFP) investigations. Evaluating the software's quality was a crucial and challenging task. SFP models have been employed for this purpose. But deciding which model to use and which among many models was the best depends on the performance metrics. Kurniawan et al., (2021) discussed forecasting water quality parameters, including dissolved oxygen (DO), in the watershed system. For effective management of water resources, accurate water quality prediction was essential. The data on water quality parameters was non-stationary, unpredictable and nonlinear; to get over these issues, a hybrid model that employed synchrosqueezed wavelet transform (SWT) was used to denoise the original data.

3. METHODOLOGY

By using state-of-the-art AI techniques, An Innovative AI-powered system for Evaluating Software Quality completely revamps conventional quality assessment methods. To evaluate software quality metrics in a more thorough and precise manner, this novel method makes use of state-of-the-art algorithms and machine learning models. This approach improves software dependability by using AI not only to find and fix bugs but also to anticipate as well as to prevent future problems. The incorporation of new methods guarantees flexibility to changing software environments, which is a giant step forward in the quest for effective and efficient quality assurance in the domain of computerized systems**.**

3.1 Dataset

A feasible, research-based repository calledEvidence-Based Software Portfolio Management (EBSPM) enables software organizations in actively enhancing their software delivery portfolio. To encourage innovation in software development inside an organization, the EBSPM attempts to benchmark, evaluate, and appraise related applications based on of costs, duration, features, and defect counts**.**

3.2 Feature extraction using linear discriminate Analysis (LDA)

After the data was acquired using LDA, the data was retrieved from it. Fisher established LDA as one of the first methods of discriminate analysis. Each class's probability distribution is assumed to be Gaussian (standard) in this procedure. In addition to the normalcy hypothesis, define a priori probabilities by the LDA. $\pi_i$every single J class. The learning set can calculate this probability, for instance, as,M_i/MFor all types set is equal. The second strategy is used in this study. The Bayes rule allocates each sample to the group with the most excellent posterior probability. The class j that generates the most negligible value of each component is indicated and C_i is based on the assumptions above in Eq. (1).

$$C_J = \left(X_i - \mu_j\right)^t \Sigma^{-1}\left(x_i - \mu_j\right) + log|\Sigma| - 2log(\pi_i) \quad (1)$$

Where $\mu_j S$ represents the class means, while the group refers to the variance-covariance matrix shared by all types. Mahalanobis distances are subjected to this requirement if the prior probability for each class is the same. The mean and covariance matrices must be estimated using the data. The group typically means x_j is employed to establish the standards. The expected covariance matrix is calculated using the following value in Eq. (2):

$$T = \sum_{j=1}^{j} \frac{(N_j - 1)T_i}{(N - J)} \tag{2}$$

Where T_i is the class$i$ empirical variance-covariance matrix, the critical drawback of LDA's is that it needs a covariance matrix with good conditioning. This indicates that the approach is inapplicable, not when there are more than a few variables or when the variables are strongly linked.

3.3 Computerized Software using Hybrid Elephant herding optimized Conditional Long short-term memory (HEHO-CLSTM)

An advanced algorithm for data analysis and prediction, the HEHO-CLSTM is novel and complex. This state-of-the-art model integrates the best features of conditional long short-term memory networks with those of hybrid elephant herding methods. With these components, HEHO-CLSTM hopes to improve forecast accuracy and flexibility on various datasets. The programme is able to traverse complicated data landscapes because of the one-of-a-kind combination of modern neural network architecture with the principles of elephant herding. For sectors that need accurate forecasts, HEHO-CLSTM is an excellent option since it is at the cutting edge of predictive analytics**.**

3.3.1 Conditional Long short-term memory (CLSTM)

The suggested architecture relies on CLSTMs to translate video frame descriptions into textual narration. Conditional Long Short-Term Memory (CLSTM) RNNs are optimized to prevent RNNs from developing long-term dependence. A CLSTM is made up of a memory cell and three major gates that regulate data input, output and accumulation. When it comes to time series, natural language processing, voice recognition and other tasks that include sequential patterns, CLSTMs shine. Based on the way data moves through the network, a mathematical model of CLSTM can be stated as follows in Eq.s (3-7):

$$e_s = \sigma\left(X_e\left[g_{t-1,}W_s\right] + C_e\right) \tag{3}$$

$$J_s = \sigma\left(X_j\left[g_{s-1,}w_s\right] + c_e\right) \tag{4}$$

$$\widehat{d}_s = tang\left(X_d\left[g_{s-1,}w_s\right] + c_b\right) \tag{5}$$

$$D_s = e_s{}^\circ d_{s-1} + j_s{}^\circ \hat{d}_s \tag{6}$$

$$h_t = o_t{}^\circ tang(c_t) \tag{7}$$

The product is the location where the sigmoid nonlinearity function and (°) intersect. More robust networks can be created by stacking and temporally linking the fundamental CLSTM unit. These networks have been put to use to address a variety of time-series challenges a shown in Figure 1.

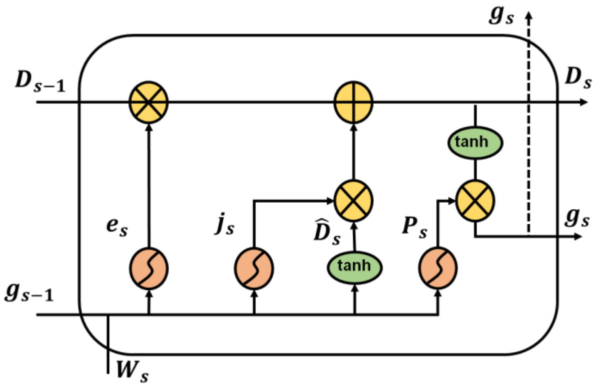

Figure 1. Architecture of LSTM

3.3.2 Hybrid Elephant Herding Optimization (HEHO)

An innovative strategy for the protection and management of elephant populations is HEHO. This ground-breaking approach uses data analytics, drones and GPS monitoring to combine old-school herding methods with modern technologies. Improved elephant herding efficiency, less human-wildlife conflict and more sustainable cohabitation are the goals of HEHO's hybrid approach. The welfare of elephants is the priority of this innovative project, which aims to balance the impact of elephants on local ecosystems and communities. HEHO exemplifies the power of combining traditional knowledge with contemporary approaches to tackle the intricate problems of animal preservation.

3.3.2.1 Operator for updating clans

As it is typical among elephants, a matriarch leads each group. Accordingly, supreme dj changes the way every elephant stands dj. To find the clan, use Eq.s (8-10) dj elephant i.

$$w_{new.dj,i} = y_{dj,i} + b \times (w_{best,dj} - y_{dj,i}) \times q \tag{8}$$

Where, $w_{new.dj,i}$ and $y_{dj,i}$ are the elephant's current and previous positions i in clan dj, respectively. $w_{best,dj}$ Is matriarch, dj symbolizes the most superior elephant in the family. A ∈ [0, 1] denotes a factor of scale, r ∈ [0, 1]. Eq. (9) can be used for each tribe to choose the best elephant.

$$w_{new.dj,i} = \beta \times w_{center,dj} \tag{9}$$

The impact of the $w_{center,dj}$ on the new individual $w_{new.dj,i}$ is determined by β, a factor that falls in the interval [0, 1]. $w_{center,dj}$represents the central member of clan dj. Eq. (3) can be used to compute it in the dth dimension.

$$w_{center,dj} = \frac{1}{m_{wj}} \times \sum_{i=1}^{m_{dk}} w_{dj,i,c} \tag{10}$$

Where, m_{wj} is the number of elephants in clan ci and 1 ≤ d ≤ D. Eq. (3) can be used to update $w_{center,dj}$, which is the centre of clan dj and $w_{dj,i,c}$ which one is the elephant's dthdimension $w_{dj,i,c}$.

3.3.2.2 The separation of Operator

The process by which a male elephant departs from his family can be represented as a separation operator for use in optimization algorithms. According to Eq. (11), the elephant with the lowest fitness in each generation is responsible for implementing the separation operator.

$$w_{worst,dj} = w_{min} + (w_{max} - w_{min} + 1) \times rand \quad (11)$$

Where w_{min} denotes the individual's lower limit and w_{max} represents the individual's upper bound. The weakest member of clan ci is indicated by$w_{worst,dj}$. The random variable Rand [0, 1] ranges from 0 to 1. The mainframe of EHO is summarized as described by the operator responsible for clan updates and the operator accountable for separating. The following flow diagram shows this. The maximum generation is MaxGen. Here are the fundamental stages of the EHO, as shown in Algorithm 1. Figure 2 displays the matching flow diagram.

Algorithm 1: Hybrid Elephant herding optimization (HEHO)

```
(1)   Start
(2)   Beginning. Define the starting point for iterations $H = 1$; to begin with the group $O$ without prior planning; determine the highest possible output $MaxGen$.
(3)   While failure to meet the stopping requirement do
(4)   Arrange the population in descending order of fitness.
(5)   For every group $dj$do
(6)   For elephant $i$ in the clan $dj$do
(7)   Produce $w_{new.dj,i}$ =and inform $y_{dj,i}$by Eq. (8).
(8)   If $y_{dj,i} = w_{best,dj}$ then
(9)   Produce $w_{new.dj,i}$and inform $y_{dj,i}$by Eq. (9).
(10)  Finish if
(11)  Finish for
(12)  Finish for
(13)  For all clans $dj$ do
(14)  Take over from the worst person $dj$ by Eq. (11).
(15)  Finish for
(16)  Assess every elephant based on its location.
(17)  $S = S + 1$.
(18)  finish while
(19)  Finish.
```

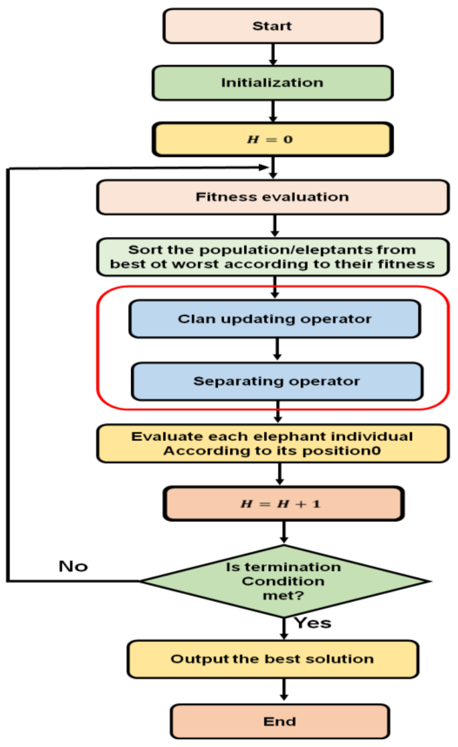
Start
Initialization
H = 0
Fitness evaluation
Sort the population/eleptants from best ot worst according to their fitness
Clan updating operator
Separating operator
Evaluate each elephant individual According to its position0
H = H + 1
Is termination Condition met?
No
Yes
Output the best solution
End

Figure 2. The EHO algorithm's flowchart

3.3.2.3 Examination of the Complexity of Algorithms

The algorithm's computational complexity is examined according to the steps of the EHO algorithm. Assume that MO is the population size and C is the dimension. Sorting the population in step (4) according to individuals' fitness is complicated in time of $P(MO)$, as it is evident. Given the intricacy of time $P(MO \times C)$, run the clan-updating user that can access all clans dj in steps (5)–(12). Perform the separation operator for each clan dj in steps (13)–(15) with a time complexity of $P(MO)$. Evaluate eachelephant based on its location in stage (16), the difficulty of which is time-related to $P(MO)$. $P(S \times MO \times C)$Is theoveralltime complexity required to accomplish this task in elephant herding optimization. After removing the low-order components, the EHO algorithm's overall time complexity is $P(S \times MO \times C)$, which is connected to S, MO and C, according to the findings given above.

In this study, Intel® Core i9 CPUs running Windows 11 and a laptop with 8.00 GB of RAM are employed in conjunction with the Python platform to access data. The relationship between the fault-proneness of software modules and the static code metrics, most notably McCabe's complexity metrics, is investigated and presented in this part. The section examines and explains the link. For the purpose of carrying out the analysis, a confusion matrix, area under the curve (AUC) and accuracy requirements are used. This section discusses the findings of the research as well as the results of empirical assessments.

4. RESULT

4.1 Accuracy Measure

Classifier efficiency is measured using the Accuracy metric. Table 1 summarizes the results of 30 different classifiers in terms of the Accuracy metric. Specifically, the HEHO-CLSTM model demonstrated its efficacy in the particular environment with impressive accuracy percentages of 90.97%, 91.35%, 90.60% and 95.30% for the dataset of CM1, KC1, KC3 and PC2, respectively. As seen in Figure 3, the Accuracy metric was assessed using the graphical depiction.

Table 1. Comparison of Accuracy measure

Methods	Accuracy (%)			
	CM1	KC1	KC3	PC2
ANN	0.8996	0.8482	0.8371	0.9296
NB	0.8735	0.8307	0.8409	0.9017
DT	0.8494	0.8373	0.8065	0.9089
KNN	0.8614	0.275	0.8084	0.7682
HEHO-CLSTM (Proposed)	0.9097	0.9135	0.906	0.953

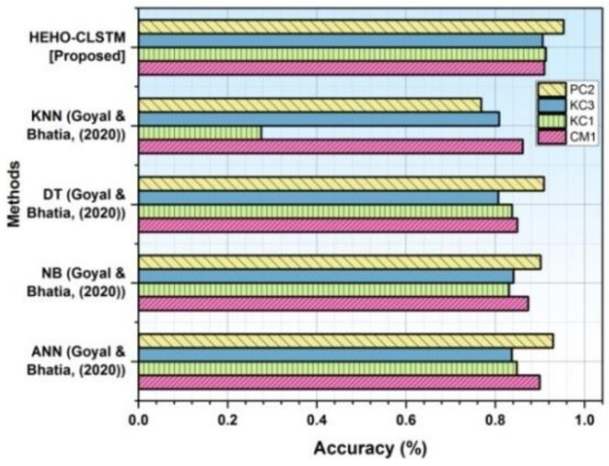

Figure 3. Outcomes of Accuracy measure

4.2 Buggy instance

When comparing the number of modules that were recognized as having bugs to the number of modules that were correctly identified as clean, it is essential to conduct the comparison using a logical manner. Incorrectly identifying a clean instance as a problem is linked with a number of costs, while the costs associated with missing a bug instance are far higher. Table 2 and Figure 4 displays the results of an analysis of the proportion of buggy modules that were accurately predicted in light of this perspective on avoiding the omission of "risky" modules. Equal to this percentage is the predictor's sensitivity. In particular, for CM1, KC1, KC3 and PC2, the HEHO-CLSTM model showed buggy

Table 2. Comparison of Buggy instance

Methods	Buggy Instance (%)			
	CM1	KC1	KC3	PC2
DT	20.4	27.9	47.66	15.58
KNN	24.48	64.41	48.59	41.55
ANN	8.16	13.19	29.9	2.59
NB	22.44	26.68	43.05	18.18
HEHO-CLSTM [Proposed]	30.15	67.15	50.29	45.09

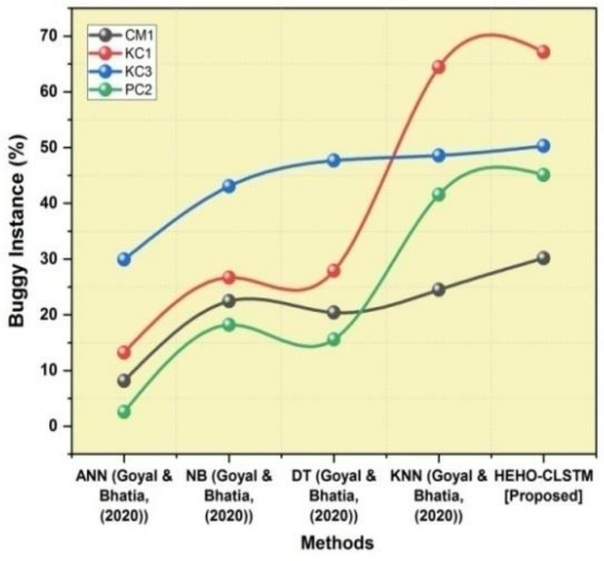

Figure 4. Outcomes Buggy instance

4.3 Area under the curve (AUC) Measure

The area under the ROC curve is the next performance metric. The proximity of the AUC value to '1' was measured. If the AUC is 1, then the classifier is accurate in its predictions. The AUC for each of the 30 classifiers is shown in Table 3. The ANN classifier achieves a maximum AUC of 0.8315 developed with the KC2 dataset. Figure 5 displays the AUC for classifiers arranged by dataset; The AUC percentages of 0.7598, 0.8095, 0.8560 and 0.7598 for CM1, KC1, KC3 and PC2, respectively, indicate the efficacy of the HEHO-CLSTM model in differentiating between positive and negative occurrences in the specified classification tasks.

Table 3. Comparison of AUC Measure

Methods	AUC			
	CM1	KC1	KC3	PC2
DT	0.5289	0.6828	0.7104	0.5863
KNN	0.5868	0.5741	0.6887	0.605
ANN	0.7286	0.7878	0.8315	0.7187
NB	0.6592	0.7442	0.7816	0.6053
HEHO-CLSTM [Proposed]	0.7598	0.8095	0.856	0.7598

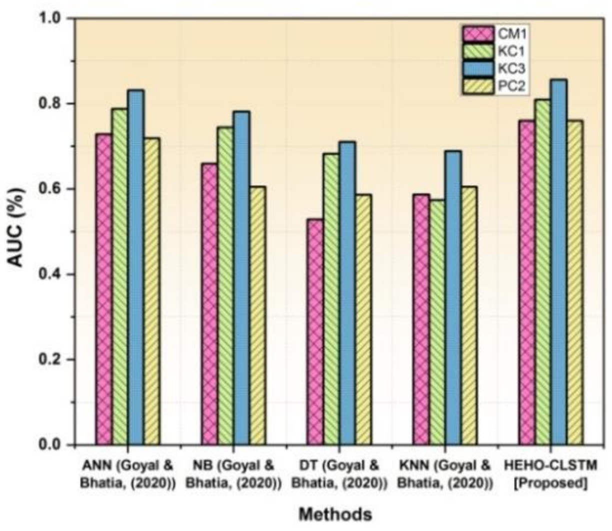

Figure 5. Outcomes of AUC Measure

5. DISCUSSION

According to (Goyal S & Bhatia P.K (2020)), there are a few restrictions that are related to ANN. They function in a "black box," which means that the intricate links that are found inside the network are impossible to understand or explain. This is one of the most significant downsides of these systems. Because of this lack of transparency, ANNs cannot be as trustworthy in critical applications where it is essential to have a solid grasp of the decision-making process, such as in the healthcare or financial sectors. An excellent example of such an application is here. In addition, ANN could need a significant amount of labeled data to be trained efficiently. This would make them computationally expensive and unsuitable for tasks that entail a limited dataset size. Another cause for concern is the danger of over fitting, which can occur when the model performs well on the data that it was trained on but poorly on data that it has never seen before. This demonstrates the need to find regularization strategies and do comprehensive model validation. The NB (Goyal S & Bhatia P.K (2020)) approach has a number of drawbacks, one of which is that it assumes the independence of the characteristics that are evaluated, which cannot be the case in some instances that occur in real life. The assumption that the qualities are independent of one another is made by the model, which is based on the class label. This can result in performance that could be better when dealing with data that indicates correlation. When presented with intricate relationships in the data or when attributes have significant interactions with one another, NB has a propensity to provide poor results. This is because of the previous point. This simplistic assumption can limit the model's ability to capture complex interactions, which would result in the model's less effective performance in some challenging tasks. Despite the fact that the assumption is straightforward and can be calculated with relative ease, this is the situation that unfolds. Over fitting is an issue that can occur when utilizing DT (Goyal S & Bhatia P.K (2020)), which is troublesome when dealing with big datasets. This problem can appear while working with trees. It is difficult for them to generalize to new circumstances when they become too complicated and unique to the training data. Additionally, trees are subject to noise in the data, which can result in several trees having virtually the same prediction value. This can be a difficult situation to deal with. In addition, there are instances in which alternative machine learning algorithms can perform better than them in terms of the accuracy of their predictions and the durability of their models. Furthermore, these algorithms can need help collecting complex correlations in the data. KNN (Goyal S & Bhatia P.K (2020)) has a number of drawbacks, one of which is that it is sensitive to the distance metric that is used, as well as the curse of dimensionality. The relevance of the distance between data points is diminished as the number of characteristics increases, which results in a dramatic decline in performance. With large datasets, KNN could be more efficient since it compares each query instance to the training instances and calculates the distance between them. This is true when dealing with large datasets. In light of this, it is possible that KNN is not suited for large-scale applications that are used in the real world because of its prohibitive processing needs. In order to solve this issue, they made use of the Intelligent HOI-CLSTM that is incorporated into LSTM. This tool improves the performance of the model by increasing its ability to adapt to new and diverse inputs, which results in an improved accuracy, quicker convergence and overall better outcomes.

6. CONCLUSION

In this study, there is a significant amount of potential for development in the field of automated software quality evaluation that makes use of a breakthrough artificial intelligence approach. Artificial intelligence (AI) approaches that are innovative increase the accuracy, efficiency and flexibility of software quality evaluation. Using this strategy, evaluation is strengthened and automated, which results in improved software development operations. Interpretability and data bases are two issues that need to be addressed to ensure that advanced AI algorithms for software quality assessment are used in a manner that is reliable and ethical. To evaluate the effectiveness of software prediction algorithms, this research compares four distinct classifiers to five software defect datasets taken from the repository. The datasets were used to analyze software defects. In order to provide an empirical assessment of machine learning techniques the objective of this project is to provide such an assessment for the purpose of software quality prediction. Through the course of this investigation, they formulate the task of software quality prediction as a two-class classification issue, which can be carried out with the assistance of machine learning techniques. Static code metrics are used in the process of developing prediction models, which are assessed using the AUC and accuracy metrics. The following are the outcomes of the MATLAB experiments and discourse: A total of thirty quality predictors were developed, trained and assessed by using five distinct machine learning algorithms on five datasets taken from the repository, one of which was the data pertaining to defect prediction. It has been discovered that static code metrics are, in fact, an adequate indication of the quality of software that will be developed in the future. The effectiveness of the created classifiers is evaluated and a comparison is carried out with the aid of the metrics along with charts that are required. AUC percentages of 0.7598, 0.8095, 0.8560 and 0.7598 for CM1, KC1, KC3 as well as PC2 indicate the model's exceptional discriminative ability. The HEHO-CLSTM model showed impressive accuracy, successfully identifying software problems with percentages of 30.15%, 67.15%, 50.29% and 45.09%. When it comes to future work on automated software quality evaluation utilizing a one-of-a-kind AI approach, one of the most important goals should be to enhance the adaptability of the AI models so that they can adjust to changing software environments and industry standards. Increasing the flexibility and accuracy of the evaluation system can be accomplished by including feedback loops from end-users and mechanisms for continuous learning. Furthermore, to conduct assessments that are efficient and focused, it would be beneficial to study the potential of automating the process of identifying and ranking the importance of software quality criteria according to the specific requirements of a particular project.

References:

Aziz, S. R., Khan, T., &Nadeem, A. (2019). Experimental validation of inheritance metrics' impact on software fault prediction. IEEE Access, 7, 85262-85275. doi: https://doi.org/10.1109/ACCESS.2019.2924040.

Deng, J., Lu, L., &Qiu, S. (2020). Software defect prediction via LSTM. IET software, 14(4), 443-450. https://doi.org/10.1049/iet-sen.2019.0149.

Farid, A. B., Fathy, E. M., Eldin, A. S., &Abd-Elmegid, L. A. (2021). Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory Bi-LSTM). PeerJComputer Science, 7, e739. https://doi.org/10.7717/peerj-cs.739.

Goyal, S., & Bhatia, P. K. (2020). Comparison of machine learning techniques for software quality prediction. International Journal of Knowledge and Systems Science (IJKSS), 11(2), 20-40. https://doi.org/10.4018/IJKSS.2020040102.

Herbold, S. (2019). On the costs and profit of software defect prediction. IEEE Transactions on Software Engineering, 47(11), 2617-2631. https://doi.org/10.1109/TSE.2019.2957794.

Khuat, T. T., & Le, M. H. (2020). Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems. SN Computer Science, 1(2), 108. https://doi.org/10.1109/ICPC.2019.00043.

Kurniawan, I., Hayder, G., & Mustafa, H. M. (2021). Predicting water quality parameters in a complex riversystem. Journal of Ecological Engineering, 22(1). https://doi.org/10.12911/22998993/129579.

Li, N., Shepperd, M., &Guo, Y. (2020). A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology, 122, 106287. https://doi.org/10.1016/j.infsof.2020.106287.

Massoudi, M., Jain, N. K., & Bansal, P. (2021, February). Software defect prediction using dimensionality reduction and deep learning. In 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV) (pp. 884-893). IEEE. https://doi.org/10.1109/ICICV50876.2021.9388622.

Mhawish, M. Y., & Gupta, M. (2020). Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics. Journal of Computer Science and Technology, 35, 1428-1445. https://doi.org/10.1007/s11390-020-0323-7.

Pascarella, L., Palomba, F., &Bacchelli, A. (2019). Fine-grained just-in-time defect prediction. Journal of Systems and Software, 150, 22-36. https://doi.org/10.1016/j.jss.2018.12.001.

Prabha, C. L., &Shivakumar, N. (2020, June). Software defect prediction using machine learning techniques. In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184) (pp. 728-733). IEEE. https://doi.org/10.1109/ICOEI48184.2020.9142909.

Radu, L. D. (2019, July). Effort Prediction in Agile Software Development with Bayesian Networks. In ICSOFT (pp.238-245) https://doi.org/.0000-0002-1463-8369.

Rathore, S. S., & Kumar, S. (2019). A study on software fault prediction techniques. Artificial Intelligence Review, 51, 255-327. https://doi.org/10.1007/s10462-017-9563-5.

Rizwan, M., Nadeem, A., &Sindhu, M. A. (2019). Analyses of classifier's performance measures used in software fault prediction studies. IEEE Access, 7, 82764-82775. https://doi.org/10.1109/ACCESS.2019.2923821.

Thota, M. K., Shajin, F. H., & Rajesh, P. (2020). Survey on software defect prediction techniques. International Journal of Applied Science and Engineering, 17(4), 331-344. 10.6703/IJASE.202012_17(4).331.

Wu, X., Zheng, W., Chen, X., Zhao, Y., Yu, T., & Mu, D. (2021). Improving high-impact bug report prediction with combination of interactive machine learning and active learning. Information and Software Technology, 133, 106530. https://doi.org/10.1016/j.infsof.2021.106530.

Wu, X., Zheng, W., Xia, X., & Lo, D. (2021). Data quality matters: A case study on data label correctness for security bug report prediction. IEEE Transactions on Software Engineering, 48(7), 2541-2556. https://doi.org/10.1109/TSE.2021.3063727.

Zhu, M., Sun, Z., Chen, T., & Lee, C. (2021). Low cost exoskeleton manipulator using bidirectional triboelectric Sensors enhanced multiple degree of freedom sensory system. Nature communications, 12(1), 2692. https://doi.org/10.1038/s41467-021-23020-3.

Dhyan Chandra Yadav
Maharishi University of Information Technology, Uttar Pradesh, India
dc9532105114@gmail.com
ORCID 0000-0003-0084-0360

Yaduvir Singh
Noida Institute of Engineering & Technology, Greater Noida, Uttar Pradesh, India
yaduyash@niet.co.in
ORCID 0000-0002-2552-4797

Arvind Kumar Pandey
Arka Jain University, Jamshedpur, Jharkhand, India,
dr.arvind@arkajainuniversity.ac.in
ORCID 0000-0001-5294-0190

A. Kannagi
Jain (Deemed to be University), Bangalore, Karnataka, India
a.kannagi@jainuniversity.ac.in
ORCID 0000-0003-3810-7500