A Hybrid Soft Computing Technique for Software Fault Prediction based on Optimal Feature Extraction and Classification

A.Balaram[1], and S.Vasundra[2]

[1] Ph.D. Scholar, Department of CSE, JNTUA University, Anantapur, Andhra Pradesh, India
balaram.balaram@gmail.com

[2] Professor, Department of CSE, JNTUA University, Anantapur, Andhra Pradesh, India
vasundras.cse@jntua.ac.in

Abstract

Software fault prediction is a method to compute fault in the software sections using software properties which helps to evaluate the quality of software in terms of cost and effort. Recently, several software fault detection techniques have been proposed to classifying faulty or non-faulty. However, for such a person, and most studies have shown the power of predictive errors in their own databases, the performance of the software is not consistent. In this paper, we propose a hybrid soft computing technique for SFP based on optimal feature extraction and classification (HST-SFP). First, we introduce the bat induced butterfly optimization (BBO) algorithm for optimal feature selection among multiple features which compute the most optimal features and remove unnecessary features. Second, we develop a layered recurrent neural network (L-RNN) based classifier for predict the software faults based on their features which enhance the detection accuracy. Finally, the proposed HST-SFP technique has the more effectiveness in some sophisticated technical terms that outperform databases of probability of detection, accuracy, probability of false alarms, precision, ROC, F measure and AUC.

Keywords:
software fault, software modules, layered recurrent neural network, hybrid soft computing, and optimal feature selection

1. Introduction

Software fault prediction (SFP) is typically used to predict faults in software components.The complexity of the software has already increased significantly in recent years, making it almost impossible to detect all failures before moving on. The global development of computer software led to advances in computer engineering, artificial intelligence, medicine, telecommunications, and image processing. If errors are later detected in the software development cycle, the cost of proper maintenance will increase significantly, so it is important to introduce software testing procedures to detect software errors in a timely manner [1]. Open source software systems are becoming more and more important these days. Many large companies invest in open source projects, most of which use this type of software. As a result, many of these projects have grown and developed rapidly [2]. However, the quality and reliability of coding needs to be explored, as open source software is usually developed outside of companies - mostly volunteers - and the development method used is quite different from the traditional methods used in business software development. Different types of code scales can be very useful for getting information about code quality and error correction [3][4]. Software testing activities play an important role in building reliable systems and utilizing key resources including time and cost. To increase the efficiency and effectiveness of the test operations, we should develop a model for predicting which files will have the highest density in the next version of the system [5][6]. These files allow testers to predict attempts, quickly detect errors, and give the rest of the system extra time to check. Accompanying this list of obvious results are high quality settings, low error settings, and programs. Misdiagnosis is an important part of sample research and has been the subject of many previous studies [7]. These studies typically develop error prediction models that allow programmers to focus on error code development measures, thereby improving software quality and improving resources. Many published models of miscalculations are complex and varied and do not provide a detailed picture of current errors [8][9].

Predicting software errors based on digital modeling techniques is attractive because the software system can select a certain number of models that accurately reflect the error generation process[10]. In addition, they have a unique feature in the number of these models, which allows a certain number of errors in a certain program module [11]. Numerical models can be used to identify program modules, i.e. incorrect and incorrect software. Identifying software modules that cause errors is an important process as it helps to identify modules that need to be restored or seriously tested [12]. This way can create more qualified software products. Software error forecasting is a type of software that uses significant forecast measurements and historical error data to predict future scale effects [13][14]. By considering software error forecasting methods, the project schedule can be planned more efficiently, especially for inspection and maintenance activities. Advantages of software error assessment: Improves the testing process and thus improves system quality. Using software error prediction at the design stage, along with class-level measurements, helps to select the best alternatives. SPF code will reduce the time and effort spent in the review process [15] [16]. However, SFF does not occur in the software development process because the practical use of SFF is impractical. The measurement database used for the final project is used to explore the machine-based method during the testing phase and the training phase [17].

Manuscript received May 5, 2022
Manuscript revised May 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.5.50

Therefore, various studies have reached counter-intuitive conclusions about the most basic and popular concepts of software engineering [18][19]. Such studies should address the importance of creating a comprehensive testing platform for the scientific engineering community. However, these warnings were clearly ignored. This indicates the lack of empirical research to evaluate the effectiveness of different software development and testing methods [20]. For further enhancement in SPF, a hybrid soft computing technique is proposed using optimal feature extraction and classification (HST-SFP).

The main contribution of proposed HST-SFP technique is given as follows:

- First, we introduce the bat induced butterfly optimization (BBO) algorithm for optimal feature selection among multiple features which compute the most optimal features and remove unnecessary features.
- Second, we develop a layered recurrent neural network (L-RNN) based classifier for predict the software faults based on their features which enhance the detection accuracy.
- Finally, the proposed HST-SFP technique has the more effectivenessin some sophisticated technical terms that outperform databases of probability of detection, accuracy, probability of false alarms, precision, ROC, F measure and AUC.

The rest of the article is organized as follows: Section 2 describes the recent works associated to SFP and corresponding techniques. And the problem methodology and system model of proposed HST-SFP technique is described in the section 3.Then, Sect. 4 gives the working function of proposed HST-SFP technique with the proper mathematical analysis. Then, the simulation results of the proposed and existing methods are in the section 5. Finally, the conclusion of the paper is explained in Sect. 6.

2. Related works

Rathore et.al [21] demonstrated a different group evaluation method which Use the number of errors and the linear combination rule and the non-linear combination rule total policies. This review is designed and featured for various software bugs. Data packets are collected from publicly accessible databases. They used L (Pred(l)) level predictions and an absolute scale to estimate the results. The evaluation of the dimensions of the 1-level analysis and results of the overall analysis confirm the effectiveness of the submitted system in estimating the number of errors. Kumar et al., [22] proposed a framework for identifying valid source code measurements with the aim of verifying source code measurements and reducing inappropriate performance and improving the performance of the incorrect rating model. They present the correlation analysis and the step-by-step multiplicative linear regression step wise option to find the correct source code functions for the computational error. The source code measurements obtained are considered as the input for creating an error assessment model using a neural network consisting of five different learning methods and three different group methods. The performance of the models can be evaluated using the proposed cost estimation framework to assess product defects. Arshad et al., [23] proposed the semi-supervised deep fuzzy c-mean (DFCM) clustering for software error detection, which is a set of semi-supervised DFCM clustering and feature summary methods. The classification evaluates the maximum integrity between the sample name attributes and the unnamed data, while the sample is practiced in a quiet database from two deep points of multiple groups and methods. Arshad et al., [24] proposed two steps in the technical pre-processing software technical empirical study of test error model data. In the initial phase, a new feature based on C-fish clustering (TFCM) semi-supervised extraction technology was proposed to create new features using names that enhance the intra-cluster class using multiple deep clusters of unnamed datasets. DFCM data pre-processing is processed by locating and embedding the required information in the data properties. The results show that the DFCM feature data extraction technology is equivalent to the experimental models of the test. Riaz et al., [25] proposed KNN sound filter, which includes two-step pre-processing of data and an easy-to-use KNN Easy Panel (RKEE) filter before implementation. The feature is inappropriate for the ranking algorithm to remove unwanted features. The second step is to cure KNN, which is hard to understand, when he filters it out by removing everything. Aziz et al., [26] proposed to explore how hereditary measures can help predict the impact of software bugs. The model structure uses artificial neural network (ANN) that uses accuracy, recall, accuracy, F1 measurements, and true negative rate (TNR) to measure performance. Comparisons and results show the acceptable contribution of hereditary measurements to SFP. The testing community can safely use traditional measurements to predict software errors.

Further inheritance is not desirable as this may lead to software errors. Li et al., [27] have proposed a three-way network is a heavyweight network that integrates developer contributions, block chains, and employee relationships with developers and explores their integrated impact on program quality. File network errors have four dimensions to predict net bag volumes of partially blown software. In addition, existing network software is considered to be one of the best ways to better predict error. The contributions prevent the developer and also the developer from creating collaborations that actually predict two networks or intelligent software errors. Bal et al., [28] proposed explored useful learning methods for precting how many bugs the software has, for example the intensive learning machine (ELM). A specific unbalanced learning model used 26 open source collateral software data sets, three prediction scenarios, internal output, interpolation, and programming. They conducted tests to predict the number of errors. Aziz et al., [29] proposed to collecting, organizing, categorizing, and investigating published fault prediction.The findings include 78 public databases containing 54 inheritance sizes and various combinations of 10 inheritance sizes, 60% of system size usage and personal database usage, and numerous studies using machine learning approaches.

Table 1 Research gap summary

Ref.	Dataset used	Classifier used	Parameter improved
21	Number of faults data	Linear regression based combination rule (LRCR)	Error rate
22	Software quality	ANN	Accuracy
23	classification model	DFCM clustering	Accuracy
24	Software quality	DFCM clustering	Accuracy
25	NASA and Eclipse	KNN rule	Accuracy
26	CK metric dataset	N-ANN	TNR and TFR
27	Software quality	Tri-relation network	F measure
28	Number of software faults	Weighted regularization extreme learning machine	Accuracy
29	Software metrics	SVM	Accuracy
30	Imbalanced data problem	Enhanced binary moth flame	Accuracy
31	NASA dataset	ML-CNN	TNR of CM1
32	Software defect	Machine learning algorithm	Precision

Tumaret al., [30] proposed Enhanced Binary Moth Flame Optimization (EBMFO) with Adaptive Synthetic Sampling (ADASYN) to predict SF. Here, BMFO is used for the packaging feature of choice and ADASYN on the other hand, database upgraded databases were launched and prayed for the unbalanced. This paper describes the possibility of a continuous binary version coming from two different groups, by changing the functions to the recommended EPFMFO version. Al Qasem et al., [31] proposed two deep learning algorithms MLPs and CNN to tackle the issue which may affect the performance of both methods. Haouari et al., [32] evaluated 8 immune systems for software deficiency treatment in three different definitions, so selected 41 databases associated with 11 Java programs. The results of the Friedman and Nemenyipostgog tests, no algorithm have been studied with a call size better than Immuno-1 and Immuno 99. Wilcoxon test suggest that studies dealing with internal project deficiencies should evaluate their modeling conditions.

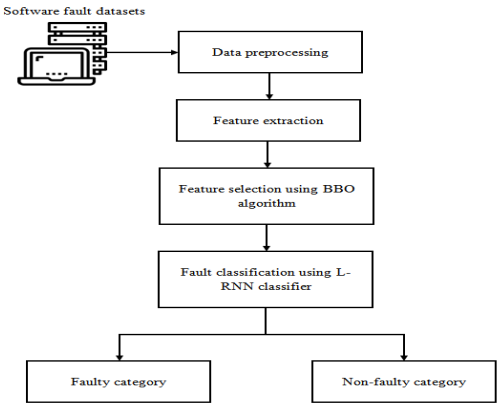
3. Problem methodology and System architecture

3.1 Research Gap

Yucalar et al. [33] developed defect predictors for software quality engineers. It reduces software modules and software effort required to better detect prediction errors. This study uses the group algorithm to evaluate the effectiveness of software error predictions. Proven results based on FM and AUC operations can be used to predict group algorithms, especially cyclic forest, with highly developed gaps. The ROF-MLP combination does not perform better than the original ROF-J48 strategy, and incorrect alarm speeds can be asset. False alarm speeds for VOT (MLP, NB and LR) decreased by 1.2% compared to VOT (J48, NB and LR) and increased by 2.7% in AUC. Due to the large amount of data available from the mining software history collections, it is possible that the learning algorithm may have erroneous features (measurements) that are misleading and reduce its effectiveness. From [21]-[33], several SFP models have since been proposed for troubleshooting multi-object problems that predict and localize errors. These dimensions directly affect the quality of much software. Measurements of physical properties such as heredity, polymorphism, and synthesis can be used to assess mispronunciation. Many researchers have studied the use of object-based measurements in predicting software errors. Recent studies have shown that machine learning techniques are designed to accurately predict such defects. Meanwhile, the way in which many independent learners are connected has proven in many ways that individual models are better. To solve above problems, we proposed HST-SFP technique which enhance the detection accuracy and reduce FPR, FNR. The main objectives of proposed HST-SFP technique are list as follows: to study and analyze the several SFP techniques, preaching to improve the performance and reliability of software fault appearance, to study and analyze the several features for effective prediction, propose novel optimal feature selection technique to improve the prediction accuracy, introduce soft computing based classifier to reduce FPR and FNR and evaluate proposed soft computing based classifier with different benchmarks.

Fig. 1 System architecture of proposed HST-SFP technique

4. Hybrid soft computing technique for software fault prediction (HST-SFP)



4.1 Feature selection using bat induced butterfly optimization (BBO)

Feature selection is a preliminary process used to improve product quality. FS is considered to be an integrated set of optimizations aimed at finding the optimal subset of properties in the original database that accurately reflects the original data. There are two main stages in a typical FS process: (i) finding the minimum reduction and (ii) evaluating the selected characteristics. The main challenge is to find out if the best FS still exists about the properties of the original data. Providentially, FS is considered a search unit that represents a subset of the attribute at each point of the search point. For this, we applied a bat induced butterfly optimization (BBO) for selecting the optimal feature and for removing unwanted data.

The first change is that we use a certain frequency and sound instead of a different frequency g_j . In BBO, each bat is

determined by its position y_j^T , velocity U_j^T . The new solutions y_j^T and velocities U_j^T at time step T are given by

$$U_j^T = U_j^{T-1} + (y_j^T - y_*)g \tag{1}$$

$$y_j^T = y_j^{T-1} + U_j^T \tag{2}$$

The global best solution is referred as y∗. In this g is equal to 0.5. To increase demographic diversity the search performance is improved by Eq. (3)

$$Y_{NEW} = y_{s1}^T + G(Y_{s2}^T - Y_{s3}^T) \tag{3}$$

where G is the mutation weight factor, while s_1 , s_2 , s_3 are evenly divided into random numbers. The migration process can be expressed as follows:

$$y_{j,z}^{T+1} = y_{s1,z}^T \tag{4}$$

where $y_{j,z}^{T+1}$ zth denotes an element of y_j at generation T+1 it gives the position of King Butterfly *i*. Similarly, $y_{s1,z}^T$ indicates the *z*th newly formed stage of the monarch butterfly s_1 . *T* is the number of the current generation. Monarch butterfly s_1 is approximately selected from the sub-population. Here, *s* can be calculated as

$$s = Rand * Peri \tag{5}$$

Peri indicates immigration period. Rand is a random number obtained as a result of consolidated distribution. Or rather, if s>q, the kth element in the butterfly is the newly formed king

$$y_{j,z}^{T+1} = y_{s2,z}^T \tag{6}$$

where $y_{j,z}^{T+1}$ the newly formed phase of the monarch butterfly is the return element s_2 . Monarch butterfly r_2 is approximately selected from the sub-population. If the generated probable number q is less than or equal to q for all components of the monarch butterfly, it can be updated as follows:

$$y_{j,z}^{T+1} = y_{Best,z}^T \tag{7}$$

where $y_{j,z}^{T+1}$ zth denotes an element of y_i at generation T+1 gives the position of King Butterfly j. Similarly, $y_{Best,z}^T$ zth denotes an element of y_{Best} that is Best King Butterfly in Land 1 and Land 2. T is the number of the current generation. Or rather, if larger than the Rand P, it can be upgraded

$$y_{j,z}^{T+1} = y_{s3,z}^T \tag{8}$$

where $y_{s3,z}^T$ and zth denotes an element of y_{s3} . In this case, if it is Rand >BAR, it can be updated as follows

$$y_{i,z}^{T+1} = y_{i,z}^{T+1} + \alpha \times (dy_z - 0.5) \tag{9}$$

where it indicates butterfly adjustment speed. dy is the according to the monarch butterfly i Levy calculate this by flight.

$$dy = Levy(y_i^T) \tag{10}$$

In Eq. (9), α is the expectation factor is given as Eq. (11)

$$\alpha = R_{Max} / T^2 \tag{11}$$

The working function of algorithm 1 represents the function of the BBO.

Algorithm 1 Optimal feature selection using bat induced butterfly optimization

Input	: Velocity
Output	: Weight factor
1	Initialize the parameters
2	Compute the new solutions $U_j^T = U_j^{T-1} + (y_j^T - y_*)g$
3	Improve the performance using $Y_{NEW} = y_{s1}^T + G(Y_{s2}^T - Y_{s3}^T)$
4	Compute the migration process using $y_{j,z}^{T+1} = y_{s1,z}^T$
5	Determine the new population using $y_{j,z}^{T+1} = y_{s2,z}^T$
6	Upgrade the position of the butterfly
7	Calculate the levy flight using $dy = Levy(y_i^T)$
8	End

4.2 Software fault prediction using LRNN classifier

After feature selection, the data are classified based on optimal features. The features are classified using LRNN which enhances the detection accuracy. The LRNN is a pre-defined number for the number of iterations of the sample, for updated estimates, which can be repeated using experimental data. When the algorithm gets the results of the best value and maximum number of repetitions, then the algorithm is stopped. Otherwise, it starts a new iteration. Simple network weight repeats and propagates through the layer of rusty passengers and in the previous position to implement recurring extra weight additions, V,

$$x_i(T) = g(NET_i(T)) \tag{12}$$

$$NET_i(T) = \sum_j^m y_j(T)u_{ij} + \theta_i \tag{13}$$

where n denotes the number of 'state' nodes and for l output nodes the i and g are hidden, and J indicates for input nodes. In the feeder network, the input vector, y, is applied using a weight layer U. The output of the network is determined by the condition and the output weight sets Z,

$$x_i(T) = g(NET_i(T)) \tag{14}$$

$$NET_i(T) = \sum_j^m y_j(T)u_{ij} + \sum_h^n x_h(T-1)v_{ih} + \theta_i \tag{15}$$

The g represents an output function. Everywhere training LRNN good historical reason 50 processes have been successfully applied to an image in complex domain lion, and software running exit prediction degree. LRNN can solve all problems in all the right ways with weight dynamic memory LRNN. It is time to copy the data stored in the LRNN. Basically, in the form of a learning process, the changes above the LRNN apply to a link or link-feeding feedback. In addition, the school neurological network is similar to the standard practice of LRNN, but with a slight twist. This requires a calculation of the output, not just the current time hierarchy, but one step per season. For this reason, the output takes place through the lymph nodes or any other or repetitive response of the network. He memorized the feedback node values from the previous level. So the new input output data depends on the current and previous ones. Here, LRNN time T. An input sequence is given as $K = (K_1,...,K_T)$, the fixed LRNN is calculated as a hidden vector array $Q = (Q_1,...,Q_T)$ and vector sequence output is $X = (X_1,...,X_T)$ by below equations,

$$Q_T = g(Z_{hK}K_T + Z_{QQ}Q_{T-1} + a_Q) \tag{16}$$

$$x_T = g(Z_{xh}Q_T + a_x) \tag{17}$$

Here, an activation function (sigmoid function) is denoted as g().As a result, it is necessary to determine the required output and cost function E of the network.

Algorithm 2 Fault prediction using LRNN classifier

Input	: Number of nodes and vectors
Output	: Cost function

1 Initialize the values for the input

2 Determine the weight of the network

$$NET_i(T) = \sum_{j}^{m} y_j(T) u_{ij} + \theta_i$$

3 Calculate the output function of the network

$$NET_i(T) = \sum_{j}^{m} y_j(T) u_{ij} + \sum_{h}^{n} x_h(T-1) v_{ih} + \theta_i$$

4 Calculate the fixed LRNN fir hidden vectors,

$$Q_T = g\left(Z_{hK} K_T + Z_{QQ} Q_{T-1} + a_Q\right)$$

5 Compute the general cost function

6 Compute the cost function of the network,

$$E = \sum_{T} \sum_{j=1}^{3} \eta_j(T) \left[\zeta_j(T) - o_j(T)\right]^2$$

7 End

The waveform flow rate function in the package of training is defined as the amount of the output and errors in the output of the E form:

$$E = \sum_{T} \sum_{j=1}^{3} \eta_j(T) \left[\zeta_j(T) - o_j(T)\right]^2 \qquad (18)$$

For j^{th} neuron, the expected output is indicated as ζ_j and the weighting coefficient of learning error is represented as η_j. Then, LRNN real output is referred as o_j. The cost function is

$$E = \sum_{T} \sum_{j=1}^{3} \eta_j(T) \left[\zeta_j(T) - o_j(T)\right]^2 + (1-\gamma) \sum_{j=1}^{n} \sum_{i=1}^{m} z_{ji}^2 \qquad (19)$$

The regulatory parameter γ controls how well the execution of the penalty period takes effect. The regulation of the parameter γ controls the operation of how much they own for the period he owns the sentence. The reduction is depending on the slope according to the formula,

$$\frac{\partial E}{\partial z_{qp}} = 2\gamma \sum_{T} \sum_{j=1}^{3} \eta_j(T) \left[\zeta_j(T) - o_j(T)\right] \frac{\partial O_j(T)}{\partial Z_{qp}} + 2(1-\gamma) z_{qp} \qquad (20)$$

The algorithm 2 represents classification of software prediction using LRNN.

5. Results and Discussion

In this section, we evaluate the performance of proposed hybrid soft computing technique for software fault prediction (HST-SFP). The proposed HST-SFP technique is implemented using Spyder (Python 3.7) with different libraries. The computer runs Windows 10, has 2 GB of RAM and an Intel i3 core processor. We used the 10-fold-cross validation through all experiments. We compare the performance of proposed LRNN classifier with existing state-of-art ensemble classifiers are AdaboostM1 (E1), Logic Boost (E2), Multiboost AB (E3), Bagging (E4), Random Forest (E5), Dagging (E6), Rotation Forest (E7), Stacking (E8), Multi scheme (E9) and Voting (E10). Because we often have public databases for the use of Turkish, white and workers' products, NASA has introduced programs, software programs, processes and applications from the open source applications of Apache. The description of dataset is given in Table 1. The performance of proposed LRNN classifier is evaluate through frequently used performance metrics are PD, PF, accuracy (A), precession (P), F-measure (F) and AUC. The detailed description of performance metrics define as follows:

$$PD = \frac{T^p}{T^p + F^n} \qquad (21)$$

$$PF = \frac{F^p}{F^p + T^n} \qquad (22)$$

$$A = \frac{T^p + T^n}{T^p + T^n + F^p + F^n} \quad (23)$$

$$P = \frac{T^p}{T^p + F^p} \quad (24)$$

$$F = \frac{2 \times PD \times P}{PD + P} \quad (25)$$

where T^p, T^n, F^p and F^n defines the True Positive, True Negative, False Positive and FalseNegative.

Table 1 Dataset description

Project name	Number of attributes	Number of modules	Number of groups
ar5	29	36	1
ar6	29	101	1
cm1	21	498	2
jm1	21	10885	2
kc1	95	145	2
mc2	40	161	2
Pc1	21	1109	2
Pc3	21	1563	2
Interface	23	27	3
ivy2_0	23	352	3
jedit_4_3	23	538	3
lucene 2 4	23	341	3
Serapion	23	47	3
tomcat	23	855	3
workflow	21	427	3

Table 2 Probability of detection comparison of proposed and existing techniques

Datasets	Existing ensample classifiers										Proposed classifier
	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	LRNN
ar5	82.4	79.1	68.1	81.0	66.9	91.8	89.7	84.2	69.0	85.9	88.7
ar6	80.6	83.1	78.3	83.8	92.4	54.7	84.9	68.6	92.9	86.5	88.3
cm1	85.8	85.4	85.5	85.2	87.6	89.7	85.9	62.6	86.6	70.0	87.9
jm1	75.6	76.1	73.2	78.2	76.5	84.9	84.2	70.2	83.0	68.8	91.4
kc1	93.6	94.1	91.8	94.1	84.9	78.4	68.6	86.9	86.0	72.5	90.3
mc2	66.7	69.0	54.7	66.9	76.2	83.4	62.6	64.1	75.2	88.9	89.6
Pc1	91.4	92.9	89.7	92.4	66.4	33.6	70.2	78.6	68.1	61.5	88.9
Pc3	86.4	86.6	84.9	87.6	70.0	44.6	86.9	89.7	80.4	80.5	90.1
interconfc	85.9	83.0	78.4	76.5	86.7	74.9	64.1	84.9	66.9	84.2	90.9
ivy2_0	84.5	86.0	83.4	84.9	64.1	86.7	78.6	85.9	92.4	68.6	91.6
jedit_4_3	77.6	75.2	33.6	76.2	79.6	34.8	91.8	84.2	87.6	67.6	91.3
luccne_2_4	65.8	68.1	44.6	66.4	81.0	79.1	54.7	68.6	76.5	70.2	90.4
serapion	80.2	80.4	74.9	70.0	83.8	83.1	89.7	62.6	84.9	86.9	90.5
tomcat	88.6	87.9	86.7	86.7	85.2	85.4	84.9	70.2	76.2	64.1	88.9
workflow	66.6	66.1	34.8	64.1	78.2	76.1	78.4	86.9	66.4	54.7	90.9
Average	80.4	80.8	70.8	79.6	78.6	72.0	78.3	76.5	79.4	73.7	89.9

5.1 Comparative analysis for Probability of detection

Table 2 describes the probability of detection comparison of proposed LRNN predictors and existing state-of-art ensemble predictors are AdaboostM1 (E1), Logic Boost (E2), Multiboost AB (E3), Bagging (E4), Random Forest (E5), Dagging (E6), Rotation Forest (E7), Stacking (E8), Multi scheme (E9) and Voting (E10). From the Table, we observe that the detection probability of proposed LRNN predictor is very high compare to the existing ensemble predictors. Fig. 2 shows the average detection probability comparison of proposed and existing predictors. It is represents the average detection probability of projected LRNN predictor is 10.5%, 10.1%, 21.2%, 11.4%, 12.5%, 19.9%, 12.9%, 14.9%, 18% and 12.8% higher than the existing state-of-art ensemble predictors are E1, E2, E3, E4, E5, E6, E7, E8, E9 and E10 respectively.

5.2 Comparative analysis for Probability of false alarm

Table 3 describes the probability of false alarm comparison of proposed LRNN predictors and existing state-of-art ensemble predictors are AdaboostM1 (E1), Logic Boost (E2), Multiboost AB (E3), Bagging (E4), Random Forest (E5), Dagging (E6), Rotation Forest (E7), Stacking (E8), Multi scheme (E9) and Voting (E10). From the Table, we observe that the probability of false alarm of proposed LRNN predictor is very high compare to the existing ensemble predictors. Fig. 3 shows the average probability of false alarm comparison of proposed and existing predictors. It is clearly depicts the average probability of false alarm of proposed LRNN predictor is 11.49%, 11.4%, 21.1%, 21.8%, 1 0.74% , 11.9%, 10.24%, 11.23%, 21.19% and 11.8% higher than the existing state-of-art ensemble predictors are E1, E2, E3, E4, E5, E6, E7, E8, E9 and E10 respectively.

5.3 Comparative analysis for Accuracy

Table 4 describes the accuracy comparison of proposed LRNN predictors and existing state-of-art ensemble predictors are AdaboostM1 (E1), Logic Boost (E2), Multiboost AB (E3), Bagging (E4), Random Forest (E5), Dagging (E6), Rotation Forest (E7), Stacking (E8), Multi scheme (E9) and Voting (E10). From the Table, we observe that the accuracy of proposed LRNN predictor is very high compare to the existing ensemble predictors. Fig. 4 shows the average accuracy comparison of proposed and existing predictors. It is denotes the average accuracy of specified LRNN predictor are 16.28%, 10.19%, 19.07%, 19.4%, 18.05%, 16.64%, 19.4%, 11.1%, 13.5% and 20.8% higher than the existing state-of-art ensemble predictors are E1, E2, E3, E4, E5, E6, E7, E8, E9 and E10 respectively.

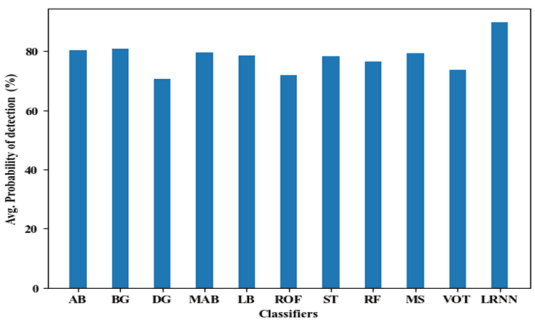


Fig. 2 Comparison probability of detection for proposed and existing detectors

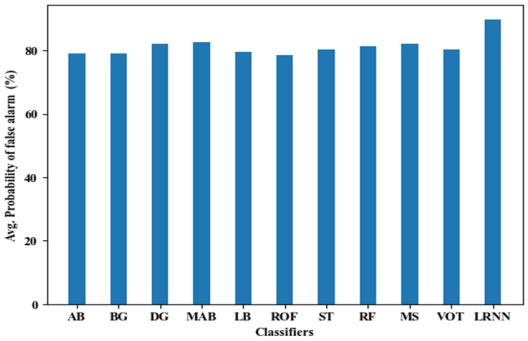


Fig. 3 Comparison probability of false alarm for proposed and existing detectors

Table 3Probability of false alarm comparison of proposed and existing techniques

Datasets	Existing ensample classifiers										Proposed classifier
	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	LRNN
ar5	72.8	64.1	77.1	87.6	68.1	62.6	86.4	89.7	87.6	72.8	69.3
ar6	78.6	78.6	91.8	76.5	78.3	70.2	85.9	84.9	76.5	78.3	71.8
cm1	91.8	91.8	78.0	84.9	85.5	86.9	84.5	78.6	84.9	85.2	70.9
jm1	54.7	91.8	90.8	72.8	73.2	64.1	72.6	91.8	80.6	77.1	70.1
kc1	89.7	66.9	91.8	78.3	91.8	78.6	65.8	54.7	85.8	91.8	68.9
mc2	84.9	92.4	66.9	85.2	66.9	91.8	80.2	89.7	75.6	78.0	68.8
Pc1	78.4	87.6	92.4	77.1	92.4	54.7	88.6	84.9	93.6	90.8	69.3
Pc3	66.9	76.5	87.6	91.8	87.6	89.7	66.6	78.4	66.7	89.7	69.9
intercafe	92.4	86.9	76.5	78.0	76.5	84.9	82.4	87.6	91.4	84.9	70.1
ivy2_0	86.4	64.1	78.6	90.8	84.9	78.4	80.6	76.5	89.7	66.7	68.3
jedit_4_3	85.9	78.6	91.8	91.8	76.2	82.4	85.8	84.9	84.9	91.4	69.4
lucene_2_4	84.5	91.8	54.7	66.9	70.2	80.6	75.6	82.4	87.6	86.4	69.8
serapion	72.6	54.7	89.7	92.4	86.9	85.8	93.6	80.6	76.5	74.3	70.8
tomcat	65.8	66.9	84.9	87.6	64.1	75.6	66.7	85.8	84.9	71.0	70.3
workflow	80.2	92.4	78.4	76.5	92.1	93.6	91.4	72.9	69.8	69.3	69.9
Average	79.0	79.0	82	82.5	79.6	78.6	80.4	81.2	82	80.2	69.8

Table 4Accuracy comparison of proposed and existing techniques

Datasets	Existing ensample classifiers										**Proposed classifier**
	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	**LRNN**
ar5	85.2	81.0	69.3	66.9	64.1	72.8	66.7	78.9	70.7	61.3	**97.5**
ar6	77.1	83.8	72.8	92.4	78.6	78.3	91.4	85.2	50.7	65.5	**97.3**
cm1	91.8	85.2	78.3	87.6	91.8	85.2	86.4	77.1	64.4	68.9	**97.6**
jm1	78.0	78.2	85.2	76.5	91.8	77.1	72.8	91.8	63.8	62.5	**98.8**
kc1	19.8	72.8	77.1	84.9	66.9	91.8	78.3	78.0	75.3	56.6	**97.9**
mc2	72.8	78.3	91.8	76.2	72.8	75.3	85.2	65.3	83.8	81.9	**98.7**
Pc1	78.3	85.2	78.0	66.4	78.3	66.9	77.1	72.8	85.2	68.8	**96.9**
Pc3	85.2	77.1	90.8	85.2	85.2	92.4	91.8	78.3	78.2	91.4	**97.1**
intercafe	77.1	91.8	64.1	77.1	77.1	87.6	78.0	85.2	72.8	86.4	**96.5**
ivy2_0	91.8	85.2	78.6	91.8	91.8	76.5	90.8	77.1	64.1	81.9	**97.3**
jedit_4_3	78.0	77.1	91.8	78.0	78.0	84.9	64.1	91.8	81.9	68.8	**96.2**
lucene_2_4	90.8	91.8	91.8	81.0	66.7	76.2	78.6	91.8	68.8	69.4	**96.4**
serapion	66.7	78.0	66.9	83.8	91.4	66.4	91.8	78.0	69.4	75.1	**97.4**
tomcat	91.4	71.6	91.8	85.2	86.4	73.9	91.8	91.8	75.1	78.0	**97.3**
workflow	86.4	85.3	78.0	78.2	72.8	69.3	66.9	91.8	54.1	81.0	**97.5**
Average	78.0	81.4	80.4	80.7	79.5	78.3	80.7	82.3	70.5	73.1	**97.4**

5.4 Comparative analysis for Precession

Table 5 describes the precession comparison of proposed LRNN predictors and existing state-of-art ensemble predictors are AdaboostM1 (E1), Logic Boost (E2), Multiboost AB (E3), Bagging (E4), Random Forest (E5), Dagging (E6), Rotation Forest (E7), Stacking (E8), Multi scheme (E9) and Voting (E10). From the Table, we observe that the precession of proposed LRNN predictor is very high compare to the existing ensemble predictors. Fig. 5 shows the average precession comparison of proposed and existing predictors. It describes the average precession of projected LRNN predictor is 18.68%, 30.03%, 25.58%, 19.68%, 17.61%, 13.68%, 50.83%, 17.57%, 26.69% and 17.13% higher than the existing state-of-art ensemble predictors are E1, E2, E3, E4, E5, E6, E7, E8, E9 and E10 respectively.

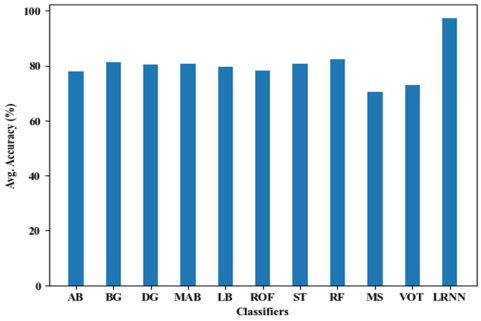


Fig. 4 Comparison accuracy for proposed and existing detectors

5.5 Comparative analysis for F-measure

Table 6 describes the F-measure comparison of proposed LRNN predictors and existing state-of-art ensemble predictors are AdaboostM1 (E1), Logic Boost (E2), Multiboost AB (E3), Bagging (E4), Random Forest (E5), Dagging (E6), Rotation Forest (E7), Stacking (E8), Multi scheme (E9) and Voting (E10). From the Table, we observe that the precession of proposed LRNN F-measure is very high compare to the existing ensemble predictors. Fig. 6 shows the average F-measure comparison of proposed and existing predictors. It describes the average precession of projected LRNN predictor is 12.5%, 15.35%, 10.90%, 14.57%, 10.56%, 10.12%, 21.46%, 11.54%, 12.2% and 10.45% higher than the existing state-of-art ensemble predictors are E1, E2, E3, E4, E5, E6, E7, E8, E9 and E10 respectively.

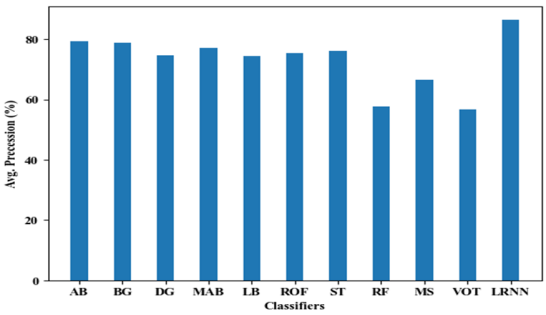


Fig. 5 Comparison precession of detection for proposed and existing detectors

5.6 Comparative analysis for Area under curve

Table 7 describes the area under curve comparison of proposed LRNN predictors and existing state-of-art ensemble predictors are AdaboostM1 (E1), Logic Boost (E2), Multiboost AB (E3), Bagging (E4), Random Forest (E5), Dagging (E6), Rotation Forest (E7), Stacking (E8), Multi scheme (E9) and Voting (E10). From the Table, we observe that the area under curve of proposed LRNN F-measure is very high compare to the existing ensemble predictors. Fig. 6 shows the average area under curve comparison of proposed and existing predictors. It describes the average precession of projected LRNN predictor is 11.67%, 12.34%, 17.04%, 14.12%, 17.13%, 16.12%, 15.46%, 36.9%, 26.06% and 36.72% higher than the existing state-of-art

ensemble predictors are E1, E2, E3, E4, E5, E6, E7, E8, E9 and E10 respectively.

Table 5Precession comparison of proposed and existing techniques

Datasets	Existing ensample classifiers										Proposed classifier
	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	LRNN
ar5	85.3	66.9	80.4	85.9	80.5	72.5	85.9	67.4	76.7	49.1	95.9
ar6	67.3	92.4	59.3	82.7	70.5	88.9	86.5	65.5	49.1	39.1	95.6
cm1	77.1	87.6	72.4	65.2	81.8	61.5	70.0	84.3	39.1	48.0	96.3
jm1	72.9	76.5	71.3	74.2	78.7	80.5	68.8	81.9	48.0	48.6	96.4
kc1	87.0	84.9	67.3	82.0	63.2	71.3	72.5	76.1	48.6	49.9	96.9
mc2	74.40	85.3	77.1	67.3	74.0	67.4	88.9	76.7	49.9	80.5	97.1
Pc1	85.9	67.3	72.9	77.1	67.4	85.3	61.5	49.1	67.3	71.3	95.8
Pc3	82.7	77.1	87.0	72.9	65.5	67.3	80.5	39.1	77.1	67.4	96.3
intercafe	65.2	72.9	74.40	87.0	84.3	77.1	71.3	48.0	72.9	85.3	96.5
ivy2_0	74.2	87.0	85.9	74.4	81.9	72.9	67.4	48.6	87.0	76.7	96.7
jedit_4_3	82.0	74.40	82.7	85.9	76.1	87.0	65.5	49.9	74.4	49.1	97.3
lucene_2_4	93.6	86.0	65.2	82.7	76.7	79.3	84.3	17.4	85.9	39.1	97.2
serapion	66.7	75.2	81.8	65.2	67.4	85.9	81.9	49.7	82.7	48.0	96.5
tomcat	91.4	68.1	78.7	74.2	65.5	56.3	76.1	49.1	65.2	48.6	96.4
workflow	86.4	80.4	63.2	82.0	84.3	78.2	79.2	48.7	73.9	49.9	97.0
Average	79.4	78.8	74.6	77.2	74.5	75.4	76.0	56.7	66.5	56.7	96.5

Table 6F-measure comparison of proposed and existing techniques

Datasets	Existing ensample classifiers										Proposed classifier
	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	LRNN
ar5	80.1	72.8	88.3	77.8	82.4	79.1	68.1	81.0	70.1	86.4	74.2
ar6	76.5	78.3	80.6	79.4	80.6	83.1	78.3	83.8	81.0	82.4	85.6
cm1	85.5	85.2	85.3	85.3	85.8	85.4	85.5	85.2	86.3	85.0	83.4
jm1	72.7	77.1	72.3	70	75.6	76.1	73.2	78.2	77.0	76.7	78.9
kc1	92.0	91.8	83.6	93.6	93.6	94.1	91.8	94.1	90.4	91.7	85.4
mc2	65.5	78.0	68.8	69.9	66.7	69.0	54.7	66.9	63.4	71.6	73.4
Pc1	89.7	90.8	98.7	89.8	91.4	92.9	89.7	92.4	91.1	91.3	84.8
Pc3	84.9	85.5	84.3	84.9	86.4	86.6	84.9	87.6	86.9	87.8	85.5
intercafe	85.9	78.4	78.4	83.0	85.9	83.0	78.4	76.5	85.9	85.9	86.9
ivy2_0	84.2	83.3	86.0	83.5	84.5	86.0	83.4	84.9	85.4	86.5	88.5
jedit_4_3	68.6	74.1	69.1	68.2	72.6	75.2	33.6	76.2	72.5	70.0	87.8
lucene_2_4	62.6	44.6	69.6	63.9	65.8	68.1	44.6	66.4	69.6	68.8	85.6
serapion	70.2	71.1	83.9	80.0	80.2	80.4	74.9	70.0	83.2	72.5	89.3
tomcat	86.9	86.7	88.1	86.7	88.6	87.9	86.7	86.7	87.6	88.9	89.7
workflow	64.1	46.4	63.1	61.5	66.6	66.4	34.8	64.1	53.5	61.5	87.3
Average	78.6	76.1	80.1	76.8	80.4	80.8	70.6	79.6	78.9	80.5	82.9

Table 7AUC comparison of proposed and existing techniques

Datasets	Existing ensample classifiers										Proposed classifier
	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	LRNN
ar5	80.8	79.5	84.4	79.0	80.4	85.3	36.6	77.7	70.7	78.6	77.2
ar6	58.8	68.7	69.3	49.9	59.3	67.3	40.2	63.9	50.7	65.6	72.4
cm1	70.0	73.4	55.1	71.2	72.4	77.1	49.0	72.36	64.4	74.1	73.6
jm1	71.0	77.4	51.7	68.9	71.3	72.9	49.1	71.8	63.8	72.1	84.9
kc1	75.5	71.7	79.1	68.8	67.4	87.0	39.1	81.2	75.3	75.7	86.6
mc2	59.1	69.1	68.1	63.4	65.5	74.40	48.0	69.9	61.3	75.8	86.9
Pc1	80.3	82.7	56.9	74.8	84.3	85.9	48.6	81.3	65.5	84.2	87.8
Pc3	78.2	82.0	54.8	79.3	81.9	82.7	49.9	77.7	68.9	79.2	85.54
intercafe	76.1	17.4	55.4	70.7	76.1	65.2	17.4	65.8	62.5	56.5	83.6
ivy2_0	87.3	49.9	67.6	80.7	76.7	74.2	49.7	76.9	56.6	76.5	84.9
jedit_4_3	80.1	81.9	76.7	79.3	80.5	82.0	49.1	83.3	81.9	80.6	85.9
lucene_2_4	66.4	50.2	74.5	68.5	70.5	76.7	48.7	74.2	68.8	86.6	86.7
serapion	81.8	42.3	83.5	88.7	81.8	84.9	41.4	78.7	69.4	73.7	83.9
tomcat	77.7	50.1	62.6	70.9	78.7	81.1	48.6	78.1	75.1	81.0	84.8
workflow	64.3	50.8	64.2	64.5	63.2	68.7	47.6	62.4	54.1	66.5	76.9
Average	73.1	62.9	66.9	72.2	74.0	77.6	44.2	74.1	65.9	74.5	86.7

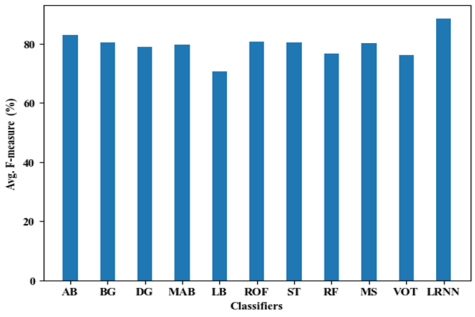


Fig. 6 Comparison F-measure for proposed and existing detectors

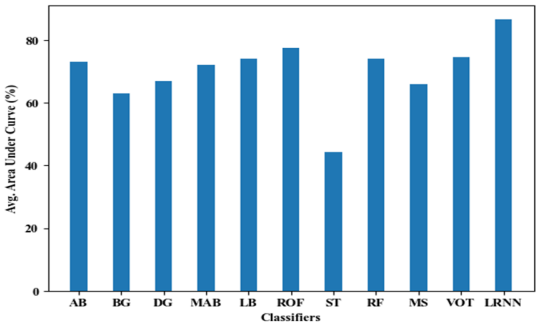


Fig. 7 Comparison AUC for proposed and existing detectors

6. Conclusion

Background: We have proposed a hybrid soft computing technique for SFP based on optimal feature extraction and classification (HST-SFP). **Contributions:** A bat induced butterfly optimization (BBO) algorithm for optimal feature selection among multiple features which compute the most optimal features and remove unnecessary features. A Layered Recurrent Neural Network (L-RNN) based classifier is used to predict the software faults based on their features which enhance the detection accuracy. **Findings:** The average detection probability of proposed LRNN detector is 14.42% higher than the existing state-of-art detectors. The average probability of false alarm of proposed LRNN detector is 14.49% higher than the existing state-of-art detectors. The average accuracy of proposed LRNN detector is 16.54% higher than the existing state-of-art detectors. The average precession of proposed LRNN detector is 23.76% higher than the existing state-of-art detectors. The average F-measure of proposed LRNN detector is 12.97% higher than the existing state-of-art detectors. The average precession of proposed LRNN detector is 20.35% higher than the existing state-of-art detectors. **Summary:** From simulation results, we observe the proposed HST-SFP technique has the more effectiveness exceeds any sophisticated technologies for databases in terms of Probability of Detection, Probability of False Alarms, Accuracy, Precision, F- Measure, and AUC.

Acknowledgment

Authors thank, Dr.C,Shoba bindu (Director ,R&D) JNT University Anantapuramu for providing a assistance to establish working environment in the lab to carry out my present research.

References

[1] Dejaeger, K., Verbraken, T. and Baesens, B., 2012. Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering*, *39*(2), pp.237-257.

[2] Gyimóthy, T., Ferenc, R. and Siket, I., 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering*, *31*(10), pp.897-910.

[3] Ostrand, T.J., Weyuker, E.J. and Bell, R.M., 2005. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, *31*(4), pp.340-355.

[4] Hall, T., Beecham, S., Bowes, D., Gray, D. and Counsell, S., 2011. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, *38*(6), pp.1276-1304.

[5] Moeyersoms, J., de Fortuny, E.J., Dejaeger, K., Baesens, B. and Martens, D., 2015. Comprehensible software fault and effort prediction: A data mining approach. *Journal of Systems and Software*, *100*, pp.80-90.

[6] Jin, C. and Jin, S.W., 2015. Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization. *Applied Soft Computing*, *35*, pp.717-725.

[7] Catal, C. and Diri, B., 2009. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences, 179(8), pp.1040-1058.

[8] Malhotra, R., 2015. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, *27*, pp.504-518

[9] Mahajan, R., Gupta, S.K. and Bedi, R.K., 2015. Design of software fault prediction model using BR technique. *Procedia Computer Science*, *46*, pp.849-858.

[10] Rathore, S.S. and Kumar, S., 2015. Predicting number of faults in software system using genetic programming. *Procedia Computer Science*, *62*, pp.303-311.

[11] Arar, Ö.F. and Ayan, K., 2016. Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies. *Expert Systems with Applications*, *61*, pp.106-121.

[12] Chatterjee, S. and Roy, A., 2014. Web software fault prediction under fuzzy environment using MODULO-M multivariate overlapping fuzzy clustering algorithm and newly proposed revised prediction algorithm. *Applied Soft Computing*, *22*, pp.372-396.

[13] Vandecruys, O., Martens, D.,

[14] Baesens, B., Mues, C., De Backer, M. and Haesen, R., 2008. Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and software*, *81*(5), pp.823-839.

[15] Binkley, D., Feild, H., Lawrie, D. and Pighin, M., 2009. Increasing diversity: Natural language measures for software fault prediction. *Journal of Systems and Software*, *82*(11), pp.1793-1803.

[16] Hu, Q.P., Xie, M., Ng, S.H. and Levitin, G., 2007. Robust recurrent neural network modeling for software fault detection and correction prediction. *Reliability Engineering & System Safety*, *92*(3), pp.332-340.

[17] Zhao, Y., Yang, Y., Lu, H., Zhou, Y., Song, Q. and Xu, B., 2015. An empirical analysis of package-modularization metrics: Implications for software fault-proneness. *Information and Software Technology*, *57*, pp.186-203.

[18] Gao, K. and Khoshgoftaar, T.M., 2007. A comprehensive empirical study of count models for software fault prediction. IEEE Transactions on Reliability, 56(2), pp.223-236.

[19] Erturk, E. and Sezer, E.A., 2015. A comparison of some soft computing methods for software fault prediction. *Expert systems with applications*, *42*(4), pp.1872-1879.

[20] Erturk, E. and Sezer, E.A., 2016. Iterative software fault prediction with a hybrid approach. *Applied Soft Computing*, *49*, pp.1020-1033.

[21] Fenton, N.E. and Ohlsson, N., 2000. Quantitative analysis of faults and failures in a complex software system. IEEE Transactions on Software engineering, 26(8), pp.797-814.

[22] Rathore, S.S. and Kumar, S., 2017. Towards an ensemble based system for predicting the number of software faults. *Expert Systems with Applications*, *82*, pp.357-382.

[23] Kumar, L., Misra, S. and Rath, S.K., 2017. An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. *Computer standards & interfaces*, *53*, pp.1-32.

[24] Arshad, A., Riaz, S., Jiao, L. and Murthy, A., 2018. Semi-supervised deep fuzzy c-mean clustering for software fault prediction. *IEEE Access*, *6*, pp.25675-25685.

[25] Arshad, A., Riaz, S., Jiao, L. and Murthy, A., 2018. The empirical study of semi-supervised deep fuzzy c-mean clustering for software fault prediction. *IEEE Access*, *6*, pp.47047-47061.

[26] Riaz, S., Arshad, A. and Jiao, L., 2018. Rough noise-filtered easy ensemble for software fault prediction. *Ieee Access*, *6*, pp.46886-46899.

[27] Aziz, S.R., Khan, T. and Nadeem, A., 2019. Experimental validation of inheritance Metrics' impact on software fault prediction. *IEEE Access*, *7*, pp.85262-85275.

[28] Li, Y., Wong, W.E., Lee, S.Y. and Wotawa, F., 2019. Using Tri-Relation Networks for Effective Software Fault-Proneness Prediction. *IEEE Access*, *7*, pp.63066-63080.

[29] Bal, P.R. and Kumar, S., 2020. WR-ELM: Weighted Regularization Extreme Learning Machine for Imbalance Learning in Software Fault Prediction. *IEEE Transactions on Reliability*, *69*(4), pp.1355-1375.

[30] Aziz, S.R., Khan, T.A. and Nadeem, A., 2020. Efficacy of Inheritance Aspect in Software Fault Prediction—A Survey Paper. *IEEE Access*, *8*, pp.170548-170567.

[31] Tumar, I., Hassouneh, Y., Turabieh, H. and Thaher, T., 2020. Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction. *IEEE Access*, *8*, pp.8041-8055.

[32] Al Qasem, O., Akour, M. and Alenezi, M., 2020. The influence of deep learning algorithms factors in software fault prediction. *IEEE Access*, *8*, pp.63945-63960.

[33] Haouari, A.T., Souici-Meslati, L., Atil, F. and Meslati, D., 2020. Empirical comparison and evaluation of Artificial Immune Systems in inter-release software fault prediction. *Applied Soft Computing*, *96*, p.106686.

[34] Yucalar, F., Ozcift, A., Borandag, E. and Kilinc, D., 2020. Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability. Engineering Science and Technology, an International Journal, 23(4), pp.938-950.

Mr. A. Balaram working as Associate Professor in the Dept. of Computer Science and Engineering, CMR Institute of Technology, Hyderabad. He Obtained his M. Tech from JNTUA University and B. Tech from JNTUH University India. Pursuing Ph.D. in JNTUA University Anantapur. He has more than 15 years of teaching experience He has published 22 international Journals, 9 conferences and 1 book chapter. And having two patents. His research interests are Software Engineering, Network Security and Cryptography, Machine Learning, Cloud Computing.

Dr.S.Vasundra, Professor of Department of Computer Science and Engineering and NSS Coordinator, JNT University, Anantapuramu.She obtained her M.Techand Ph.D.degree from JNTUA University and B.E degree from Gulbarga University. She has published more than 59 international journals, 21 conferences, and 1 textbook. And also having three Patents. Her research interests include Mobile Ad hoc Networks, Computer Networks, and Big Data, data mining, cloud computing.