Cluster Computing
https://doi.org/10.1007/s10586-018-1696-z

Deep neural network based hybrid approach for software defect prediction using software metrics

C. Manjula[1] · **Lilly Florence**[2]

Received: 21 September 2017 / Revised: 10 November 2017 / Accepted: 2 January 2018

Abstract
In the field of early prediction of software defects, various techniques have been developed such as data mining techniques, machine learning techniques. Still early prediction of defects is a challenging task which needs to be addressed and can be improved by getting higher classification rate of defect prediction. With the aim of addressing this issue, we introduce a hybrid approach by combining genetic algorithm (GA) for feature optimization with deep neural network (DNN) for classification. An improved version of GA is incorporated which includes a new technique for chromosome designing and fitness function computation. DNN technique is also improvised using adaptive auto-encoder which provides better representation of selected software features. The improved efficiency of the proposed hybrid approach due to deployment of optimization technique is demonstrated through case studies. An experimental study is carried out for software defect prediction by considering PROMISE dataset using MATLAB tool. In this study, we have used the proposed novel method for classification and defect prediction. Comparative study shows that the proposed approach of prediction of software defects performs better when compared with other techniques where 97.82% accuracy is obtained for KC1 dataset, 97.59% accuracy is obtained for CM1 dataset, 97.96% accuracy is obtained for PC3 dataset and 98.00% accuracy is obtained for PC4 dataset.

Keywords Software metrics · Software quality · Software defect prediction · Machine learning · Deep neural network · Genetic algorithm

1 Introduction

During the past two decades the demand for software for various applications is increasing rapidly. To meet the customer demand, huge amount of software applications are developed for corporate or daily use purpose. Due to the mass production of software applications, quality of software remains an unaddressed issue which gives unsatisfactory performance for industrial and individual applications. Hence, to address this issue software testing is introduced which helps to find the defects or bugs in the software application and tries to resolve it. Due to increasing demand by modern technology based industries and business applications, large number of software applications are developed each year but software quality remains an unnoticed issue during this development. In recent years, for business purpose and daily routine purpose software applications have become an essential part. In corporate organizations, a minor defect in software may lead to the loss to the industry and affect the satisfaction level of costumers. As a concern, software testing related issues becomes critical. Hence, an automated process of software testing is required which can result in performance improvement and implementation cost minimization. This issue can be mitigated if the causes of possible defects and general process of software development are known to the tester. This can not only help in reducing the overall cost of software development but also help in better planning and executing of software development projects.

According to IEEE standards, fault or bug is considered as 'inappropriate step, process or data definition in a computer program' [1]. In this article, software defects or bugs are interchangeable which represents error in the source code of the software [2]. Undesirable system behavior experienced

✉ C. Manjula
manjulaprasad@pes.edu

Lilly Florence
lilly_swamy@yahoo.co.in

[1] MCA Department, PESIT-BSC, Bangalore, Karnataka, India

[2] MCA Department, Adiyamman College of Engineering, Hosur, Tamil Nadu, India

Published online: 19 January 2018

by the users at any point in time is due to failure of the system on account of defects or faults.

In software industries, software development life cycle is known as the key contributor for software development. According to recent scenario, early defect prediction is demanded more and considered as a key challenging task by project managers [3]. Recent software development field includes complex problem domains, increasing requirement of software application, uncertainty in software performance and complex development process. In this process of software development, despite of careful documentation and well-organized process, some of the defects are unavoidable which result in performance degradation of software.

According to today's industrial development, various techniques are introduced for software defect minimization. But these techniques require more amounts of time, money and resources for appropriate analysis of software application. However, these efforts can help to analyze the causes of defect and improving the performance of software. Software performance is also known as software reliability. According to software reliability, software applications are tested in a given inconsistent environment and analyzed whether software application is capable to operate or not in the environment for a given particular time [4]. This technique depends on the estimation of software reliability probability [5]. In order to perform this task, software metrics are used and it is observed that more number of failure prediction require more quality improvement resources which becomes a challenging task. In the last few decades several software defect prediction models have been proposed using software metrics. These models can help in early prediction of defects and development of reliable software. Software metrics such as traditional software metrics, object oriented software metrics, process metrics [6,7] have been extensively used in most of the defect prediction models developed till date. In real-time scenario, organizations use Pareto analysis [8] for software quality measurement where software metrics are used along with the highest metric value for particular software application. This technique provides better performance but still it is not able to capture multiple defects resulting in performance degradation of software testing. The study which concludes the human evaluation is poor to identify the defects in software modules. In another scenario of software defect prediction, a statistical model is formulated using software metrics for predicting the software defects. These models are based on the regression or function-approximation problem analysis [9]. However, these techniques fail to provide efficient results. This is because, the architecture of each software is unique with different function combination, development team, and third-party components. This causes inappropriate outcome of the software defect prediction. Furthermore, due to the diversity in various terms, a "critical value" cannot be fixed for any of the software metrics and due to this parametric models such as linear model, Poisson regression and quadratic models etc. cannot be accepted for defect prediction and software analysis.

To address the complexity issue, non-parametric techniques are presented for software-defect prediction. These techniques include machine learning technique and computational intelligence for predicting the software defects. Various researches have been reported for software defect analysis using machine learning techniques. Machine learning encompasses a variety of tasks which process and model data. Many algorithms have been designed to support these tasks, each with its own requirements for data and differing levels of complexity. Examples include linear regression, K-nearest neighbours, decision tree, support vector machines, neural networks, deep learning (DL) etc. Gondra et al. [10] developed a machine learning technique using artificial neural network (ANN) model. In this work, values of software metrics and number of errors are considered for training the neural network. In this study, neural network is trained using the critical values of each software metrics and a sensitivity analysis is performed for performance measurement. In this context of software defect prediction, machine learning techniques attracted researchers due to its performance for imbalanced and uncertainty datasets. In [11], Thwin et al. presented a study about software defect prediction using neural network technique. This work provides the defect using two type of investigation. According to first type, total number of defect present in a class are presented whereas in second type number of lines changed in each class is predicted which helps to analyze the software architecture. Moreover, this model utilizes two type of neural network known as Ward neural network and general regression neural network (GRNN). Experimental study shows that GRNN obtains more accurate results compared to Ward neural network. Support vector machine [12] is also used for software defect prediction. However, use of support vector machine technique for software defect prediction is very limited due to accuracy perception. Hence, accuracy can be further improved by applying feature selection and optimization techniques. Some of the most promising optimizing techniques are known as ant-colony optimization [13] and genetic algorithm (GA) [14]. Moreover, clustering techniques such as fuzzy-clustering model [15], k-means clustering [16] are also employed, Dejaeger et al. [17] presented Bayesian classification based approach for software defect prediction using machine learning approach. Shuai et al. [18] developed a hybrid approach for software defect prediction using a combination of GA and support vector machine classification approach. This work shows that GA provides better performance in terms of optimization by reducing the error in the given input data and genetic-fuzzy approach, also shows significant performance. As discussed before, support vec-

tor machine (SVM) suffer from the classification accuracy issue.

Yang et al. [19] discussed a DL approach which provides better classification performance. In the field of software defect prediction, DL based schemes have provided significant performance. Hinton et al. [20] discussed deep belief network (DBN) for feature learning and training purpose. This technique considers semantic feature for software defect prediction using learning process. However, as discussed before, that hybrid approach can provide better performance by using optimization technique with classification. In this paper we propose a novel hybrid approach using a combination of deep neural network (DNN) technique for classification and GA for feature optimization process resulting in development an efficient optimization based classification scheme for software defect prediction.

The rest of the manuscript is organized as follows: Sect. 2 describes recent literature reviews in this field, Sect. 3 deals with proposed approach, experimental study is presented in Sects. 4 and 5 gives concluding remarks about proposed approach and its performance.

2 Literature review

This section presents a brief review of recent techniques for software defect prediction where various recent studies are considered which include software metrics based defect prediction, optimization schemes, machine learning technique and hybrid techniques (combination of optimization and machine learning). First, we discuss the techniques which are based on the software metrics and well-known in this field of software defect prediction. Chidamber et al. [21] discussed a software metrics suite for defect prediction. Similarly, Basili et al. [22] investigated object-oriented design metrics. The main aim of this study is to validate the software metrics and assessing whether these can be used for defect indicator or not for software defect prediction. In order to carry out this work, eight software applications are collected. This software is developed using sequential life cycle model using C++ programming language based on the object-oriented module.

For performance improvement, multivariate models are also introduced but due to diversity of software applications these models are not capable to provide efficient performance. Hence to address this issue, multivariate models are combined with software matrix based model which can perform the desired operation on different software applications [23].

Reliability of software application becomes an issue due to different coding styles and various other parameters. In order to measure the reliability, more information needs to be extracted from the source code. To address this problem, Gyimothy et al. [24] developed a new approach based on the object-oriented metrics analysis. Metrics are most important for developing the predictive model.

These techniques which are based on the software matrices are not capable to provide satisfactory results when a complex software application is given for analysis. According to the study presented in [23], proper selection of software matrix will lead to better prediction. However, the selection of matrix is a crucial task for huge applications. Another issue which affects the performance is known as uncertainty of software application architecture. To overcome these issues, clustering, classification and optimization techniques are discussed.

In the field of clustering, k-means clustering is known for its performance for uncertain datasets. By taking this into consideration, Bishnu et al. [25] developed a software prediction scheme using k-means clustering model. Further performance is investigated using Quad Tree k-means algorithm which helps to find the optimal clusters using an initial threshold value. The main advantage of this technique is that it is capable to achieve maximum gain and can be used for software fault prediction but selection of threshold need to be further optimized or needs to be fully automated according to the diverse software application scenarios. Based on clustering model, Yuan et al. [26] developed a new clustering model to reduce the clustering error and improve the clustering performance. This work mainly includes fuzzy-clustering with module-order modeling which is used for software quality prediction. Furthermore, the performance of prediction can be improved by applying optimization technique. Azar et al. [27] developed a scheme for software defect prediction using ant-colony optimization technique. This technique contains pre-build model for different software applications and applies adaptive technique for model consideration according to the new incoming data. This model uses ant-colony technique for adaption process. In [28] Chen et al. presented a similar approach for software scheduling using ant-colony optimization approach.

Optimization techniques obtained better results when compared with classification and software metrics. Time taken for decision making remains an unaddressed issue in optimization techniques. In order to achieve an optimal point, the techniques require more time. Hence, machine learning techniques are introduced for fault prediction in software application.

Byoung et al. [29] developed polynomial function-based neural network (pf-NN) classifiers (predictors) and used for software fault prediction. This technique is a combination of fuzzy C-means and genetic clustering techniques which helps to obtain nonlinear parameters for operation. A weighted cost function (objective function) is used for learning method and a standard receiver operating characteristics (ROC) analysis is used to analyze the performance of the system. Fuzzy

clustering (fuzzy C-means, FCM) is employed for development of premise layer of the rules while the corresponding consequences of the rules are formed by using some local polynomials. The learning algorithm for the pf-NNs is presented with provisions for handling imbalanced classes.

Clustering techniques fail to obtain better performance in terms of classification. In order to overcome this, machine learning technique are also used for software defect prediction. These machine learning techniques utilize support vector machine, Naïve Bayes classifier, decision tree, neural network and deep learning techniques. Elish et al. [30] developed software defect prediction scheme using SVM (support vector machine). This technique helps to find the solution for regression problems and classification. Based on SVM classification technique, Gray et al. [31] also presented software defect prediction where prior to classification input data is pre-processed to reduce the randomness. Recently, Rong et al. [32] introduced a new technique for software defect prediction. This technique uses SVM classification, for improvised performance optimization capacity of bat algorithm with centroid strategy (CBA).

Naïve Bayes classifier is also widely used for various classification studies. Shivaji et al. [33] analyzed Naïve Bayes classification technique and implemented for software defect prediction where feature selection module is also incorporated for efficient bug prediction. Similarly, Dejaeger et al. [17] discussed about naïve Bayes technique where it was concluded that Naive Bayes classifier formulates a training network where some arcs and nodes remain unexplored resulting in classification performance degradation. Santosh et al. [34] introduced software defect prediction model using decision tree classification and developed a recommendation system for software defect prediction. Recently, Yang et al. [35] developed deep learning based technique for software defect prediction. This study illustrates that DL technique plays an important role in machine learning and can be used for various classification fields. This technique provides better performance for software bug prediction.

In this section, a brief study is presented which includes recent works in the field of DNN in various applications. Kumudha and Venkatesan [36] proposed a prediction approach employing conventional radial basis function neural network (RBFNN) and a novel adaptive dimensional biogeography based optimization (ADBBO) model. The ADBBO based RBFNN model is tested using five datasets from the publicly available NASA data program repository. Wahono et al. [37] present neural network parameter optimization based on genetic algorithm for software defect prediction. A combination of GA and bagging technique has been used for improving the performance of the software defect prediction. Parameter optimization of neural network has been carried out using GA. Bagging technique has been employed to handle the class imbalance problem.

However, still these techniques suffer from various issues such as computational time, accuracy and complexity w.r.t defect prediction which can be further improved. To overcome all these issue, here we present a novel hybrid model which helps to reduce the time taken and provide better accuracy. In order to carry out this research, we use adaptive GA and DL approach for defect prediction. The complete process is presented in next section.

3 Proposed model

This section presents the proposed approach for software defect prediction. Proposed technique is divided into two sections. First, we apply feature optimization model using GA and in the second stage DNN is applied for classification.

3.1 Genetic algorithm

GA is known as a search algorithm by taking random patterns or probability distribution. The random distribution causes uncertainty scenario which affects the prediction performance of distribution. GA is based on the natural selection and recombination technique. During this process of natural selection and recombination, optimal solution of given random pattern is computed by applying manipulation in a population resulting in next population generation. In each population, good candidates dominate the performance hence the desired performance of any given model can be obtained. Generally, GA is capable of obtaining optimal solution for given problem. In our work our aim is software defect prediction which contains complex scenarios. In this type of complexity, GAs suffer from the issue of inefficient local optima convergence and require more time for computation of optimal solution [38]. This issue is responsible for degrading the performance of the system. Huang et al. [39] analyzed the causes of classification performance degradation and presented a new approach for classification rate improvisation by introducing a feature selection model using GA, according to this work, the features which has more impact on the data pattern will be selected as most optimal features and these features formulate non-linear optimization problem which can be solved by applying GA.There are some certain advantages of GA which motivate to use GA for optimization. These advantages are such as: (a) GA can support multi-objective function computation. (b) this algorithm performs parallel computation from a population point which helps to avoid the error caused due to local optimal solution unlike conventional optimization schemes. (c) according to this process, probabilistic selection rules are used whereas other schemes use deterministic rules.

In this work, we develop an improved genetic algorithm named as genetic algorithm for optimization (Fig. 1).

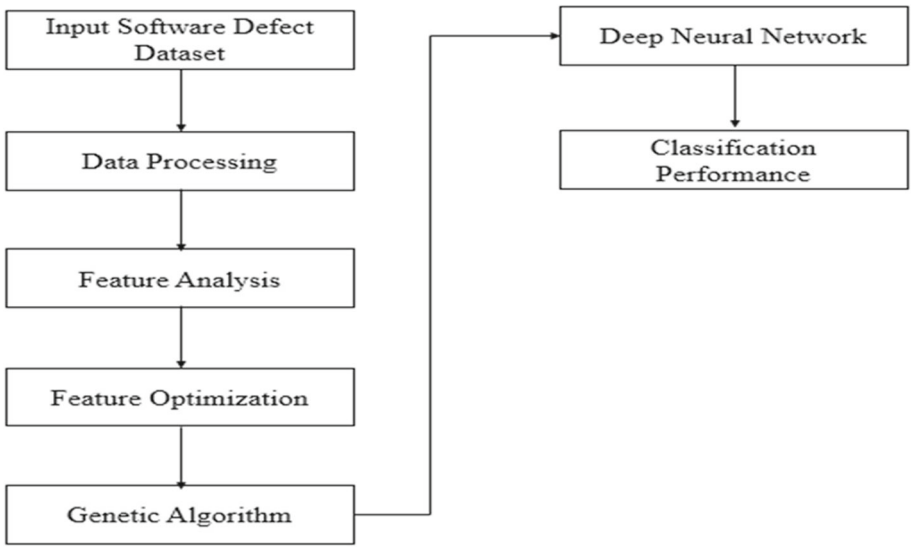


Fig. 1 Overall architecture of proposed model

Proposed approach divides total population into various sub-populations and computation is applied on each population. With the help of this, diverse population is created which increases the speed of computation, resulting in less time consumption for computation. In this work, proposed approach performs optimal feature selection for deep-learner algorithm for software defect prediction. Initially a random population is created which is divided into various sub-populations resulting in distinct evaluation for each population. Later, we apply different evolutionary scheme to obtain the best results. After specific number of iterations or population generation, worst solution in the given population is replaced by the best solution at the current iteration.

Proposed genetic algorithm consists of main stages such as designing the chromosome, fitness function formulation. Rest of the steps remain unchanged and follows conventional approach of GA.

3.2 Designing of chromosome

In order to design the chromosome, first of all we generate a feature sub-space for feature distribution. This feature sub-space is designed using kernel function which is expressed as given in Eq. (1).

$$Gaussian\ kernel: \ \mathcal{K}(a_i, a_j) = \exp(-\gamma \|a_i - a_j\|^2) \qquad (1)$$

where a denotes input data for training, $(a_i - a_j)$ denotes Euclidean distance and $\gamma = \frac{1}{2\sigma^2}$.

After finishing the kernel selection process, we apply chromosome designing which comprises of three components which are denoted by $\mathcal{C}$, γ and subset of input features. Here we use binary code ($\mathcal{B}$) to represent the chromosome as depicted in Fig. 2.

For each parameter of chromosome, binary code is represented. For parameter $\mathcal{C}$, binary code is considered from $\mathcal{B}_C^1 \sim B_C^{n_C}$, for parameter γ, binary code is represented as $\mathcal{B}_\gamma^1 \sim B_\gamma^{n_\gamma}$ and binary code for input feature set is given as $\mathcal{B}_\mathcal{F}^1 \sim B_\mathcal{F}^{n_\mathcal{F}}$. Here, n_C, n_γ and $n_\mathcal{F}$ denotes the number of bits required for parameter $\mathcal{C}$, γ and feature subset. With the help of these parameters, chromosome is transformed into phenotype where true value of input bit string is computed as expressed in Eq. (2),

$$\mathcal{P} = \min_{\mathcal{P}} + \frac{(max_\mathcal{P} * d) - (min_\mathcal{P} * d)}{2^n - 1} \qquad (2)$$

Here, d decimal value of bit string, length of bit string is denoted by n, $max_\mathcal{P}$ is the maximum value of any given parameter and $min_\mathcal{P}$ is the minimum value of input feature parameter.

After finishing this stage, apply *population generation* where a random population is initiated. This population consists of base feature values which are obtained from the processed dataset. This generation is updated after a given specific number of iterations i.e., after each iteration a best population is generated and replaces existing population to reduce the optimization error. In order to update the population, we apply fitness function computation.

3.3 Fitness function computation

Fitness function computation is known as an important step of GA. Fitness function is used to evaluate the performance of each population and provides the best population among all the initiated population. In this work, fitness function is constructed based on the deep learner model and selected features. According to proposed strategy, higher value of fitness function is assigned to each population resulting in improved

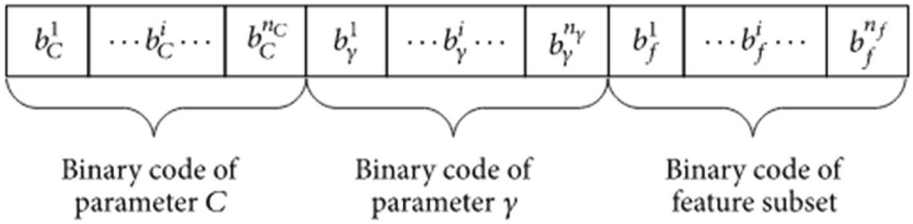


Fig. 2 Binary representation of chromosome

classification accuracy. Fitness function of this approach is given as expressed in Eq. (3),

$$Fit = \mathcal{W}_A \times Acc + \mathcal{W}_{\mathcal{F}} + \mathcal{W}_d v, \tag{3}$$

where $f = 1 - \frac{(\sum_{i=1}^{n_{\mathcal{F}}} \mathcal{F}_i)}{n_{\mathcal{F}}}$ and $v = 1 - \frac{(\sum_{i=1}^{l} d_i)}{l}$, $\mathcal{W}_A$ denotes the classification accuracy weight, $W_{\mathcal{F}}$ denotes weight of feature score, $\mathcal{W}_d$ denotes weight of deep-learner classification per iteration and l denotes number of sample. d denotes number of iteration and v is the neuron available at dth iteration.

In the next stages, GA follows conventional approach and the steps are mentioned below.

3.3.1 Genetic operator definitions

The main operators are as follows:

(a) *Selection* selection of individuals is performed for breeding to generate further population.
(b) *Crossover* here we have used single point crossover. According to single point cross over, a random point is selected which is known as crossover point. This helps to generate new solutions.
(c) *Mutation* this is the measurement for mutation based on the probability.

3.3.2 Parameters used for GA

Considered parameters are as follows:

Population size = 100
Generation = 50
Crossover probability = 0.7
Mutation probability = 0.4
Crossover technique = Random single point

After applying feature selection and optimization approach, these features are parsed to deep-neural network modeling for classification resulting in prediction of software defects. However, various recent models utilize DNN based scheme for feature selection or dimension reduction. Based on this assumption, Zhang et al. [40] introduced a new approach for feature selection for dimension reduction using multilayer network model where k-clustering technique is also required for efficient dimension reduction and further same scheme has to be utilize as classification scheme. This complete process of feature selection using neural network scheme becomes complex which may lead to performance degradation of the system whereas GA is used for solving the large scale non-linear optimization problems [41,42] which is formulated using software defect attributes.

3.4 Deep neural network

This section describes DNN technique for classification and software defect prediction. This technique is useful for learning the features which are effective and discriminative in nature. DNN can be applied for large amount of unlabeled data. Since our study is aimed at the software defect prediction where data acquisition is a difficult task it causes complexity in pattern learning. To overcome this issue, here we present an improved DNN with the help of sparse auto-encoders. According to proposed approach, to learn the feature pattern, adaptive auto-encoders are incorporated with de-noising model. Furthermore, these learned features are fed to neural network classifier. Complete process of proposed technique is described in following subsections. First of all, we discuss the designing of adaptive auto-encoders. However, neural networks also show significant performance in terms of classification but high-dimensional data and multi-objective function can not be addressed using neural network.

3.4.1 Adaptive auto-encoder

An auto-encoder is a technique which is known as a type of neural network which is symmetrical in thee nature. Auto encoders are used for feature learning for huge datasets. Auto-encoders provide significant performance by reducing the training error and loss function minimization.

Base architecture of auto-encoder is depicted in Fig. 3. In this figure, input layer, hidden layer and output layers are presented. At hidden layer, this auto-encoder performs pattern learning of features which are obtained from input layer. The main aim of auto encoder is to achieve perfect reconstruction of input data at output layer. The conventional encoders suffer fundamental issue that it copies input layer to hidden layer which doesn't provide significant information regarding input data. In this work, we present an extension of conventional auto-encoder which improves the performance of system.

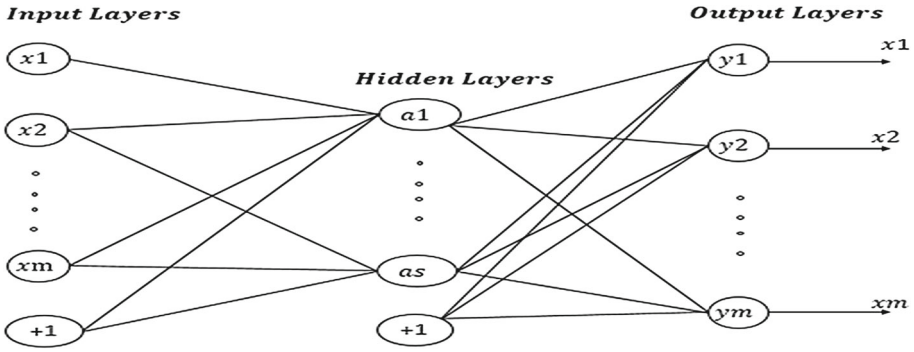


Fig. 3 Basic architecture of auto-encoder

Let us consider that input dataset $\mathcal{I}$ contain $\mathcal{N} \times \mathcal{M}$ dimension which is formulated as $\mathcal{X} = \{x(1), x(2), \ldots, x(i), \ldots x(\mathcal{N})\} \in R^M$. In this modeling, $\mathcal{M}$ is the length of dataset and $\mathcal{N}$ denotes number of samples considered for evaluation. This data matrix is considered as input matrix for proposed adaptive auto-encoder. In the next stage, neural network is formulated, similar to the architecture depicted in Fig. 3. In this process, activation function selection is important step which affects the performances. In our work, we select sigmoid activation function because of its significant nature for performance enhancement.

In this approach, our next aim is to perform feature learning and understanding the expression of feature on input data where input data $\mathcal{X}$ is known as unlabeled data. This expression is given as

$$h(x(i), \mathcal{W}, b) = \sigma(\mathcal{W}x(i) + b), \quad i = 1, \ldots \mathcal{N}) \tag{4}$$

And the output layer function is given as

$$Output\_layer = \sigma(\mathcal{W}^T h(x(i), \mathcal{W}, b) + c) \tag{5}$$

Furthermore, we introduce a term which mitigates the repetition of input feature by using an objective function. This term is applied on hidden layer and controls the active neurons at current iteration. Let us consider that $a_j(x)$ is the activation function of jth hidden layer. For software defect prediction model, we have considered back propagation technique.Back propagation is a technique where learned weights are propagating from outer nodes to inner nodes to reduce the learning error which can be obtained by computing the gradient of the network weights. This technique is applied on input feature set $\mathcal{X}$, where hidden layer can be expressed as given in Eq. (6).

$$a = sigmoid(\mathcal{W}\mathcal{X} + b), \tag{6}$$

where b denotes biases and weight of input feature set is given by $\mathcal{W}$. Using this assumption, the average weight of activation function can be given as:

$$\rho_j = \frac{1}{n}\sum_{i=1}^{n}[a_j(x(i))] \tag{7}$$

At the initial stage of training the DNN, average activation function is considered to be closer to zero because of inactivity of the neurons. In order to achieve this, we apply a penalty to the average activation function, if it diverges from the significant value of average activation function. This can be given as,

$$P_{penalty} = \sum_{j=1}^{s_2} KL(\rho \| \rho_j) \tag{8}$$

s_2 denotes total number of neurons available in hidden layer and $KL(.)$ is the Kullback–Leibler divergence (KL divergence). This divergence can be expressed as:

$$KL(\rho \| \rho_j) = \rho log\frac{\rho}{\rho_j} + (1-\rho) log\frac{1-\rho}{1-\rho_j} \tag{9}$$

If $\rho_j = \rho$ then $KL(\rho||\rho_j) = 0$ or else it sustains an increment which causes divergence, known as adaptive constant.

This can be determined by using a cost function which is given as,

$$C_{adaptive}(\mathcal{W}, b) = \mathcal{C}(\mathcal{W}, b) + \beta \sum_{j=1}^{S2} KL(\rho||\rho_j) \tag{10}$$

β is the weight of penalty applied using Kullback–Leibler divergence (KL divergence) model. Identification of important parameters $\mathcal{W}$ and b is the important task because cost function and these two parameters are directly proportional to each other which can affect the system performance. Hence, in order to solve this, we formulate an optimization problem and the outcome of this problem is to minimize the $C_{adaptive}(\mathcal{W}, b)$. This optimization problem can be solved by using back-propagation approach by updating $\mathcal{W}$ and b iteratively. This can be written as:

$$W_{ij}(l) = W_{ij}(l) - \varepsilon \frac{\partial}{\partial \mathcal{W}_{ij}(l)} C_{adaptive}(\mathcal{W}, b)$$

And

$$b_i(l) = b_i(l) - \varepsilon \frac{\partial}{\partial b_i(l)} C_{adaptive}(\mathcal{W}, b) \tag{11}$$

ε denotes the learning rate of the DNN model.

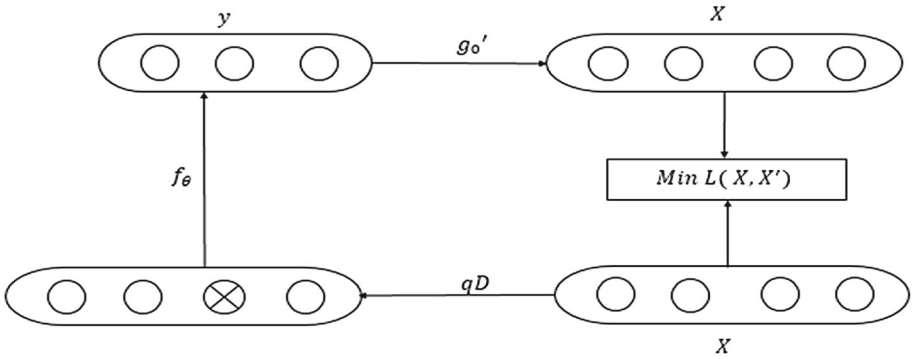


Fig. 4 Denoise auto-encoder architecture

3.4.2 Denoising for adaptive auto-encoder

In this section we discuss about the de noising technique for adaptive auto-encoder. According to this technique partially corrupted input pattern is added to the input data pattern for initializing the deep architectures. This technique is capable to provide better representation of feature with the help of de noise coding [43].

Figure 4 shows the architecture which is used for denoising for Adaptive auto-encoder.

Initially, input data is given in corrupted form which is denoted by $\mathcal{X}$ using a stochastic data mapping i.e. $\mathcal{X} \sim qD(\mathcal{X}|\mathcal{X})$ where qD denotes the data distribution. This corrupted pattern then passed to the y. This joint distribution can be expressed as:

$$q^0(\mathcal{X}, \mathcal{X}, y) = q^0(\mathcal{X}) qD(\mathcal{X}|\mathcal{X}) \partial_{f_{0(x)}}(y) \tag{12}$$

$$\partial_{f_{0(x)}}(y) \text{ is set to 0 if } f_0(x) \neq y \cdot q^0(\mathcal{X}) \tag{13}$$

which denotes the empirical distribution of input training dataset. The reconstruction of input corrupted pattern is the objective function, which can be expressed as:

$$\arg^{min}_{\theta,\theta'} E_{q^0(\mathcal{X},\mathcal{X})}[L(\mathcal{X}, \mathcal{X}')] \tag{14}$$

$\mathcal{X}'$ is the reconstructed pattern and $L(\mathcal{X}, \mathcal{X}'))$ denotes the error in reconstruction. This error can be minimized using Eq. (13).

During this process of data training, if small dataset is considered for evaluation then a well-known "over fitting" error occurs which degrades the performance of training resulting in lower classification accuracy. To overcome this issue, here we consider Eq. (6) and set the output of neurons to the zero which will not take part in the propagation step. This technique helps to reduce the error in feature extraction or selection and improves the classification accuracy. Moreover, auto-encoder is capable to learn the pattern of unlabeled data and can help to normalize the features for robust feature representation [44].

The complete process of proposed approach is as follows

Input: Software defect database
Output: Defect prediction accuracy and classification performance

Step 1: Read software defect data $D \leftarrow Input\ (Software\_Defect\_DB)$
Step 2: Apply data pre-processing
Step 3: Apply data segregation based on the labels as given in original DB i.e. $Defective\_DB \rightarrow Y\ (Yes)\ and\ NonDefective \rightarrow N\ (No)$
Step 4: Feature analysis i.e. $Total\ Number\ of\ Feature \leftarrow DB(1 : end - 1, end)$ and $Total\ class \leftarrow unique\ (DB\ (:, 1 : end))$
Step 5: Initialize proposed hybrid algorithm
Step 6: Generate initial configuration of deep neural network and genetic algorithm parameters $Population_{size}, Crossover_{probability}, Mutation_{probability}$
Step 6: Random weight ($\mathcal{W}$) and fitness function assignment
Step 7: Apply feature selection
  Begin $C = total\ number\ of\ chromosomes$
  Initial population generation $P^n = \left\{p_1^n, p_2^n, \ldots, p_C^n\right\}$
  Fitness computation $Fit = \mathcal{W}_A \times Acc + \mathcal{W}_{\mathcal{F}} Fit + \mathcal{W}_d v$
Step 8: Generate new population P_{new}
Step 9: Randomly select two new chromosomes as c_p^{new} *and* c_q^{new} from P_{new}
Step 10: apply crossover functionality to obtain new chromosomes with the help of C_p as cross probability
Step 11: apply mutation functionally with M_p mutation probability
Step 12: compute fitness of $Fitness\left(c_p^{new}\right)$ and $Fitness\left(c_q^{new}\right)$
Step 13: Insert c_p^{new} and c_q^{new} in the P_{new}
Step 14: Pick best chromosomes from $P^{n-1}\ and\ P^{new}$ to formulate P^n
Step 15: obtain optimal weights and best fit values as feature selection $\mathcal{X}_{best}$
Step 16: Apply feature learning using weights and threshold parameters as $\sigma\left(\mathcal{W}^T h\ (x\ (i)\ , \mathcal{W}, b) + c\right)$
Step 17: apply hidden layer computation and KL divergence computation to minimize the weight errors $C_{adaptive}\ (\mathcal{W}, b) = W_{ij}\ (l) - \varepsilon \frac{\partial}{\partial \mathcal{W}_{ij}(l)} C_{adaptive}\ (\mathcal{W}, b)$
Step 18: apply denoising adaptive autoencoders to reconstruct the corrupted pattern using joint distribution $q^0\ (\mathcal{X}, \mathcal{X}, y) = q^0\ (\mathcal{X})\ qD\ (\mathcal{X}|\mathcal{X})\ \partial_{f_{0(x)}}\ (y)$
Step 19: formulate objective function as $\arg^{min}_{\theta,\theta'} E_{q^0(\mathcal{X},\mathcal{X})}\left[L\left(\mathcal{X}, \mathcal{X}'\right)\right]$
Step 20: Error minimization
Step 21: Performance evaluation in terms of classification accuracy and other statistical parameters.

Table 1 PROMISE software defect prediction dataset details

Name of dataset	Total number of modules	Defective measurement (%)	Software language	LOC	Number of defects
KC1	2109	15.5	C++	43,000	325
KC2	522	20.5	C++	18,000	105
CM1	496	9.7	C	20,000	48
PC1	1107	6.9	C	40,000	76
JM1	8779	19.35	C	10,885	2106

4 Experimental study

In this section, we describe the experimental studies which are carried out for software defect prediction using proposed hybrid learning approach. In this paper, a hybrid approach deploying GA and DNN classification techniques is employed for prediction of software defects using MATLAB tool. For this study, we have considered PROMISE software defect dataset repository [45].

4.1 Dataset details

In this repository, KC1, KC2, CM1, PC1, JM1 etc., datasets are present which are used for applying software defect prediction. Since, various studies are presented using this dataset we can present a comparative analysis to show the significance of the proposed hybrid approach. Table 1 gives a brief description of PROMISE software defect datasets which are used in this study. This table includes information such as module number, coding language, percentage defect, line of code and total number of defect present in the module.

These datasets contain various feature sets which are called as attributes. These attributes and details of attributes are given in Table 2.

As discussed before, GA is also applied for feature selection and reduction. Selected features with the help of GA are given in Table 3. These features or attributes are known as the selected features which are further given to the deep neural network for pattern learning and analysis.

Table 2 PROMISE software defect prediction attribute details

Attribute name	Description of attribute
LOC	Counts total number of line in the module
Iv(g)	Design complexity analysis (McCabe)
Ev(g)	McCabe essential complexity
N	Number of operators present in the software module
v(g)	Cyclomatic complexity measurement (McCabe)
D	Measurement difficulty
B	Estimation of effort
L	Program length
V	Volume
I	Intelligence in measurement
E	Measurement effort
Locomment	Line of comments in software module
Loblank	Total number of blank lines in the module
uniq_op	Total number of unique operators
uniq_opnd	Total number of unique operand
T	Time estimator
Branchcount	Total number of branch in the software module
total_op	Total number of operators
Total_opnd	Total number of operators
Locodeandcomment	Total number of line of code and comments
Defects	Information regarding defect whether defect is present or not

4.2 Performance measurement parameters

For performance measurement, a well-known technique is used. This technique is called as confusion matrix analysis. According to this matrix, actual class and predicted class are stored to obtain the classification results. A sample representation of confusion matrix is given in Table 4.

This confusion matrix helps us to compute total accuracy, precision, specificity, sensitivity and F-measure of the proposed approach.

Accuracy is a measurement of rate of correct classification which is denoted by *Acc*. It is computed by taking the ratio of correct prediction and total number of prediction. It can be expressed as:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{15}$$

Another parameter is known as sensitivity analysis of the model. This is the measurement of true positive rate which can be computed by identifying the correctly classified non-defective modules. This can be expressed as

$$Sensitivity = \frac{TP}{TP + FN} \tag{16}$$

Table 3 Selected features using genetic algorithm

Attribute name	Description of attribute
LOC	Counts total number of line in the module
Iv(g)	Design complexity analysis (McCabe)
Ev(g)	McCabe essential complexity
N	Number of operators present in the software module
v(g)	Cyclomatic complexity measurement (McCabe)
D	Measurement difficulty
B	Estimation of effort
L	Program length
I	Intelligence in measurement
E	Measurement effort
uniq_op	Total number of unique operators
uniq_opnd	Total number of unique operand
Branchcount	Total number of branch in the software module
Locodeandcomment	Total number of line of code and comments

Table 4 Confusion matrix

Actual class	Predicted class	
	Non defective	Defective
Non defective	False negative (FN)	True positive (TP)
Defective	True negative (TN)	False positive (FP)

Next parameter is computed as true negative rate which shows the measurement of correct classified defective software modules and can be expressed as:

$$Specificity = \frac{TN}{TN + FP} \tag{17}$$

Then, we compute precision of the proposed approach. It is computed by taking the ratio of True Positive and (True and False) positives.

$$P = \frac{TP}{TP + FP} \tag{18}$$

Finally, F-measure is computed which is the mean of precision and sensitivity performance. It is expressed as:

$$F = \frac{2 * P * Sensitivity}{P + Sensitivity} \tag{19}$$

4.3 Performance analysis and discussion of results

This section describes the experimental study carried out for software defect prediction using PROMISE dataset. This study is carried out for 4 datasets KC1, CM1, PC3 and PC4.The performance with data sets KC1 and CM1 are studied both without optimization (Experimental scenario 1) and with optimization (Experimental scenarios 2–3) and the results are compared. Furthermore, the studies with data sets PC3 and PC4 are carried out only with optimization (Experimental scenarios 4–5) using the proposed hybrid method and the results are discussed.

Table 5 Confusion matrix for KC1 dataset using DNN (feature selection and optimization is not considered)

Actual class	Predicted class	
	Defective	Non-defective
Defective	280	40
Non defective	104	1685

4.3.1 Experimental scenario 1

First, we conducted experiments where GA is not considered for feature selection and optimization. These experiments are conducted for KC1 and CM1 using only DNN. Performance of software defect prediction without GA is evaluated in terms of confusion matrix and statistical performance analysis (Table 5).

Another parameter for performance evaluation, are given in Table 6.

The above experiment produces the software defect prediction accuracy of 93.17 for KC1 dataset using DNN.

Similarly, we have evaluated the performance for CM1 dataset and the obtained results are given in Tables 7 and 8.

Another parameter for performance evaluation, are given in Table 7.

The above experiment produces the software defect prediction accuracy of 92.97 for CM1 dataset using DNN.

Table 6 Statistical performance analysis for KC1 dataset using DNN (feature selection and optimization is not considered)

Precision	Sensitivity	Specificity	Recall	F-score	Accuracy
0.875	0.976	0.875	0.729	0.79	93.17
0.72	0.87	0.82	0.76	0.81	

Table 7 Confusion matrix for CM1 dataset using DNN (feature selection and optimization is not considered)

Actual class	Predicted class	
	Defective	Non-defective
Defective	57	19
Non defective	16	406

Table 8 Statistical performance analysis for CM1 dataset using DNN (feature selection and optimization is not considered)

Precision	Sensitivity	Specificity	Recall	F-score	Accuracy
0.75	0.960	0.75	0.76406	0.79	92.97
0.74	0.91	0.79	0.83	0.84	

Table 9 Confusion matrix for KC1 dataset using proposed hybrid classifier

Actual class	Predicted class	
	Defective	Non-defective
Defective	309	17
Non defective	29	1754

4.3.2 Experimental scenario 2

Next, in this stage, we compute the performance of proposed GA and DNN hybrid approach, for software defect prediction for KC1 data set. Confusion matrix using proposed approach is depicted in Table 9.

Another parameter for performance evaluation, are given in Table 10.

ROC curve and precision–recall (P–R) curve analysis is depicted in Figs. 5 and 6 respectively.

Above given Fig. 5 shows ROC analysis for software defect prediction. This experiment is carried out on KC1 data using proposed hybrid approach. Figure shows a significant performance in terms of true positive rate. Similarly Fig. 6 shows P–R analysis which shows a significant performance of proposed approach in terms of higher precision value.

Complete comparative study for KC1 dataset is presented in Table 11 which shows improved classification accuracy for KC1using proposed hybrid approach.

Table 11 shows comparative study for KC1 dataset. From the table it is evident that the proposed approach obtains better performance when compared with other state-of-art classification techniques.

In Fig. 7, we present a graphical performance analysis for software defect prediction. This figure shows a comparative analysis of decision tree, KNN, SVM, etc., and proposed hybrid classification. From the graph it is clear that the present approach using hybrid technique gives better accuracy when compared to other techniques.

Another comparative pictorial representation of accuracy is given in Fig. 8 which shows that proposed hybrid approach gives better prediction results when compared with other state-of-art techniques.

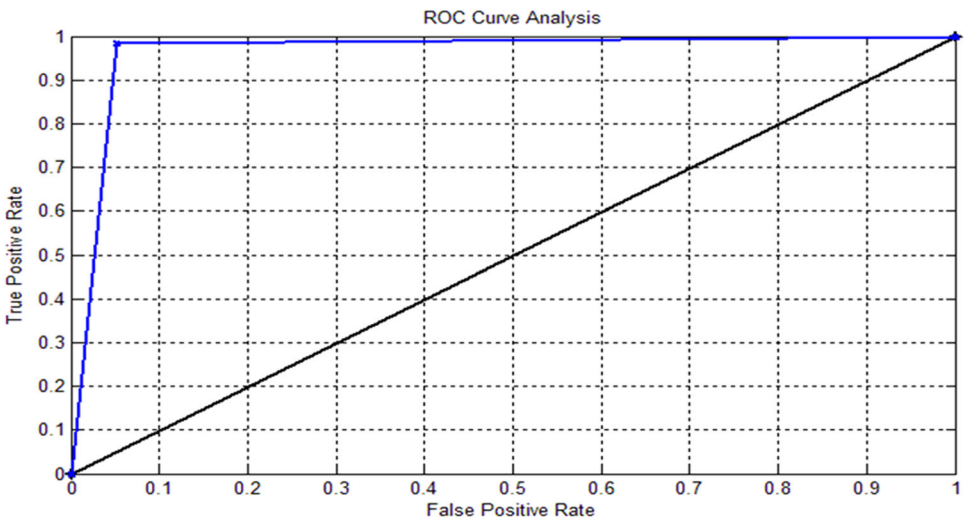
This experimental study shows that proposed approach is capable to obtain better classification results. It produced the accuracy of 97.82%. This scheme provides reliable and significant performance which can be used for software defect prediction model.

When it is compared with the Experimental scenario 1 of software defect prediction using DNN without optimization with KC1 dataset, the present approach gives better accuracy results.

Table 10 Statistical performance analysis for KC1 dataset using proposed hybrid classifier

Precision	Sensitivity	Specificity	Recall	F-score	Accuracy
0.914201183	0.947852761	0.983735278	0.947852761	0.930722892	97.819
0.990400903	0.983735278	0.947852761	0.983735278	0.987056837	

Fig. 5 ROC curve analysis



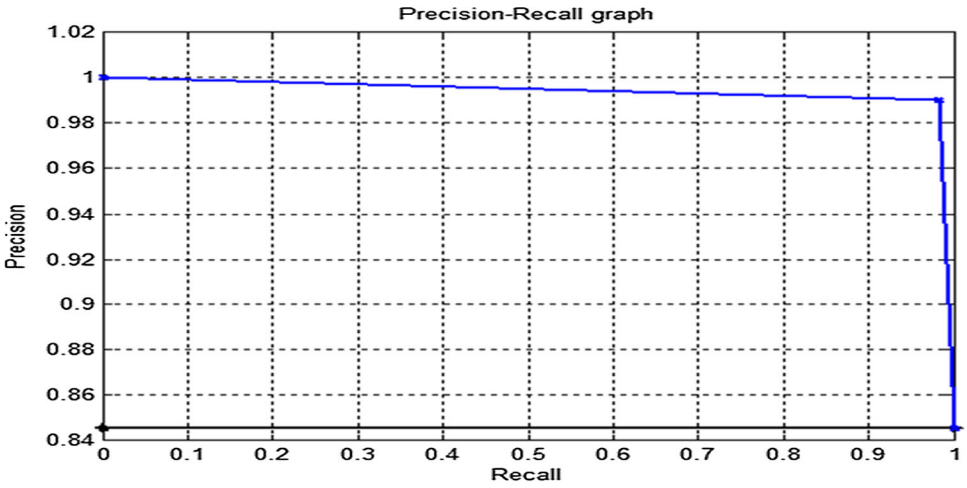


Fig. 6 Precision–recall analysis

Table 11 Performance comparison for KC1 dataset using proposed hybrid classifier

NASA dataset	Techniques	Sensitivity	Specificity	FPR	Accuracy %
KC1	Naïve Bayes [46]	74.33	76.85	35.71	65.87
	Random Forest [46]	72.54	75.89	37.91	67.99
	C4.5 Miner [46]	76.42	75.64	34.05	68.01
	Immunos [46]	78.05	72.91	36.92	63.55
	ANN-ABC [46]	79	77	33	69
	Hybrid self-organizing map [47]	80.92	80.94	35.67	78.43
	SVM [13]	81.37	81.27	28.96	79.24
	Majority vote [13]	82.65	85.62	30.98	79.66
	AntMiner+ [13]	84.29	84.99	26.11	80.51
	ADBBO-RBFNN [36]	85.67	87.95	20.24	84.96
	NN GAPO + B	–	–	–	79.4
	Decision Tree	83.34	94.1	87.78	86.35
	KNN	89.31	91.24	92.56	91.33
	Proposed hybrid approach	90.22	93.2	97.59	97.82

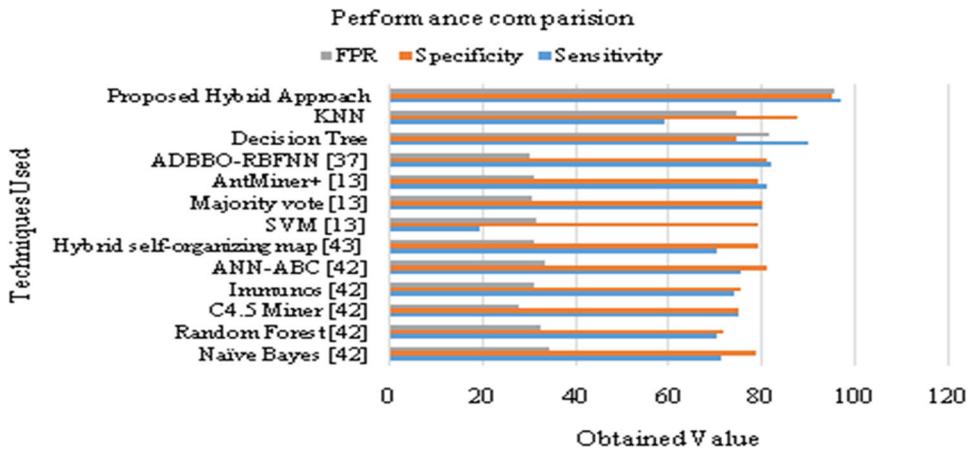


Fig. 7 Comparative performance comparison (obtained values in %)

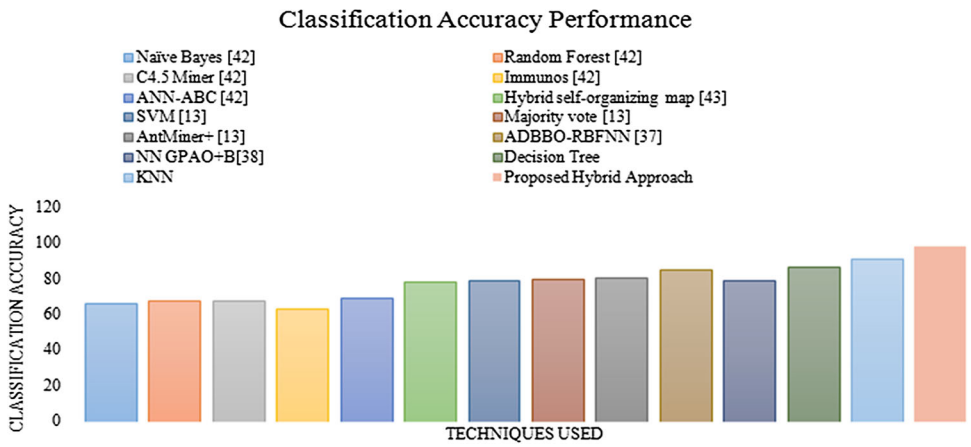


Fig. 8 Classification accuracy performance

Table 12 Confusion matrix for CM1 dataset using proposed hybrid classifier

Actual class	Predicted class	
	Defective	Non-defective
Defective	49	0
Non defective	12	437

4.3.3 Experimental scenario 3

In this section we present experimental study on CM1dataset using GA and DNN hybrid approach for software defect prediction. Complete experiment is carried out similar to Experimental scenario 2 (Table 12).

Another parameter for performance evaluation, are given in Table 13.

Table 14 shows comparative study for CM1 dataset. From the table it is evident that the proposed approach obtains better performance when compared with other state-of-art classification techniques.

This experimental study shows that proposed approach is capable to obtain better classification results. It produced the Accuracy of 97.59%. This scheme provides reliable and significant performance which can be used for software defect prediction model.

When it is compared with the Experimental scenario 1 of software defect prediction using DNN without optimization

Table 13 Statistical performance analysis for CM1 dataset using proposed hybrid classifier

Precision	Sensitivity	Specificity	Recall	F-score	Accuracy
0.8032787	0.9732739	0.9732739	0.9732739	0.8909091	97.59
0.9016393	0.9866371	0.9866371	0.9866371	0.9386825	

Table 14 Performance comparison for CM1 dataset using proposed hybrid classifier

NASA dataset	Techniques	Sensitivity	Specificity	FPR	Accuracy
CM1	Naïve Bayes [46]	71.03	78.65	34.09	64.57
	Random Forest [46]	70.09	71.29	32.17	60.98
	C4.5 Miner [46]	74.91	74.66	27.68	66.71
	Immunos [46]	73.65	75.02	30.99	66.03
	ANN-ABC [46]	75	81	33	68
	Hybrid self-organizing map [47]	70.12	78.96	30.65	72.37
	SVM [13]	18.97	79.08	31.27	78.69
	Majority vote [13]	79.8	80	30.46	77.01
	Ant Miner+ [13]	80.65	78.88	30.9	73.43
	ADBBO-RBFNN [36]	81.92	80.96	29.71	82.57
	NN GAPO + B [37]	–	–	–	74.4
	Decision Tree	89.62	74.23	81.2	73.49
	KNN	58.88	87.21	74.12	89.19
	Proposed hybrid approach	96.54	94.78	95.23	97.59

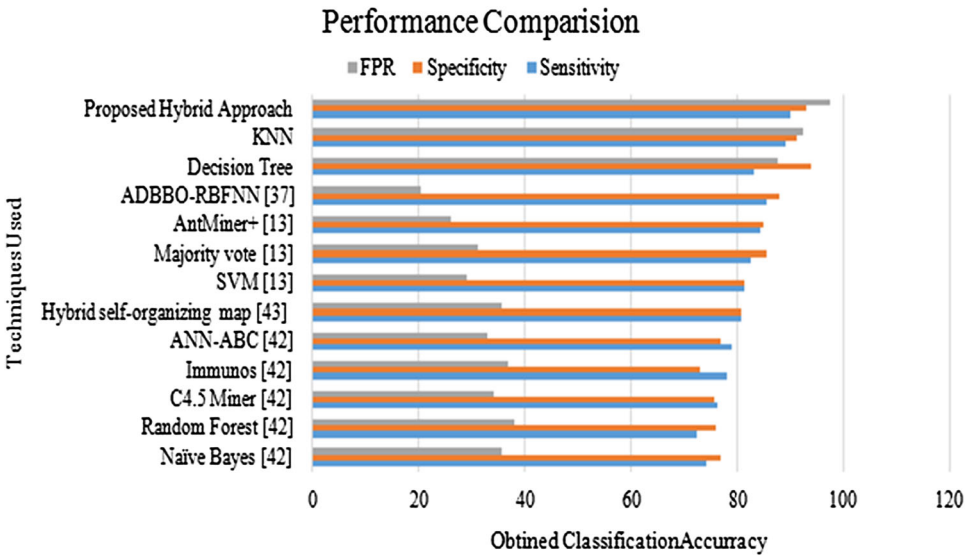


Fig. 9 Comparative performance comparison (obtained values in %)

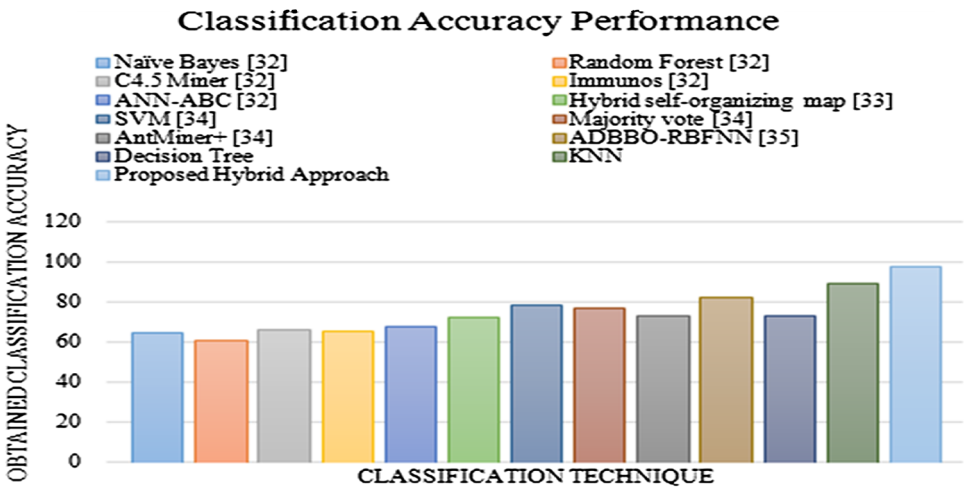


Fig. 10 Classification accuracy comparison

with CM1 dataset, the present approach gives better accuracy results

Furthermore, in Fig. 9 a graphical comparative study is presented for the proposed approach.

In Fig. 9, we present a graphical performance analysis for software defect prediction. This figure shows a comparative analysis of decision tree, KNN, SVM, etc., and proposed hybrid classification. From the graph it is clear that the present approach using hybrid technique gives better accuracy when compared to other techniques.

Finally, classification accuracy comparison is presented in Fig. 10 in pictorial format.

For further performance analysis, in Fig. 11 P–R curve (precision and recall) performance is analyzed where precision rate and recall rate are compared to show the precise performance of proposed approach.

Similarly, Fig. 12 receiver operating characteristic (ROC) analysis is also carried out for CM1 NASA database by considering false positive rate and true positive rate where proposed approach shows better performance.

This experimental study shows that proposed approach is capable to obtain better classification results. This scheme provides reliable and significant performance which can be used for software defect prediction model.

4.3.4 Experimental scenario 4

In this section we conducted experiment for PC3 using GA and DNN hybrid approach for software defect prediction. Similar set of computations are applied here as discussed for previous software defect prediction experiments. Confusion matrix for this scenario is presented in Table 15.

For this experiment, other statistical performance outcomes are given in Table 16.

This experimental study shows that proposed approach is capable to obtain better classification results. It produced the accuracy of 97.96%. This scheme provides reliable and sig-

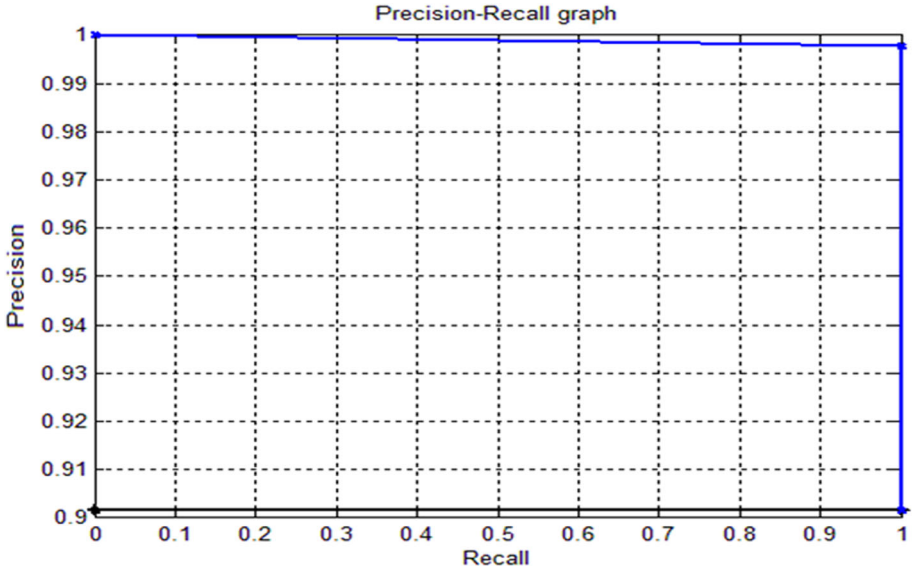


Fig. 11 Precision and recall analysis

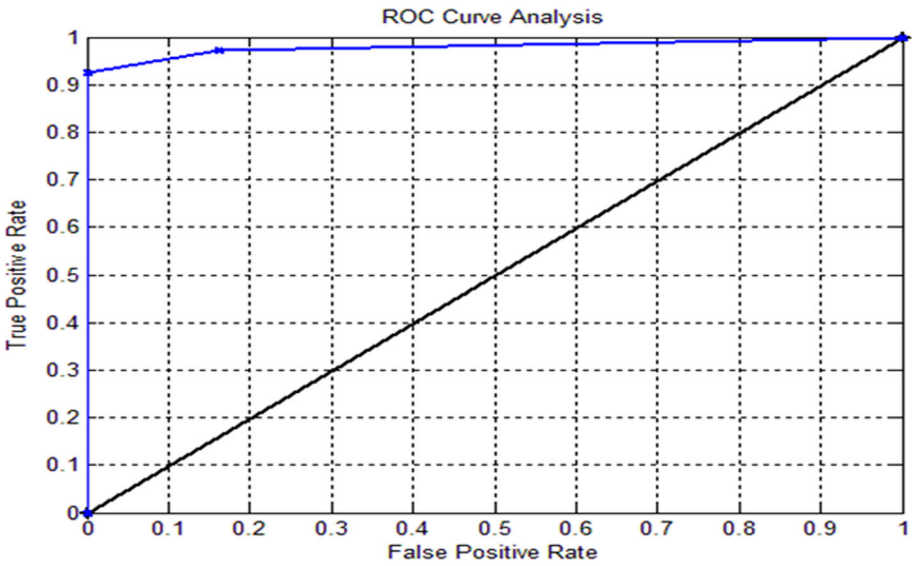


Fig. 12 ROC curve analysis

Table 15 Performance evaluation for PC3 dataset using proposed hybrid classifier

Actual class	Predicted class	
	Defective	Non-defective
Defective	985	0
Non defective	23	117

Table 16 Statistical performance analysis for PC3 dataset using proposed hybrid classifier

Precision	Sensitivity	Specificity	Recall	F-score	Accuracy
0.97	0.83	1	1	0.98	97.96
1	1	0.835	0.83	0.91	

nificant performance which can be used for software defect prediction model.

4.3.5 Experimental scenario 5

This section presents experimental study for PC4 datasets using GA and DNN hybrid approach for software defect prediction. Here, we measure the performance in terms of classification accuracy by computing confusion matrix and other statistical performance parameters as given in Table 17.

For this experiment, other statistical performance outcomes are given in Table 18.

This experimental study shows that proposed approach is capable to obtain better classification results. It produced the accuracy of 98.0%. This scheme provides reliable and sig-

Table 17 Confusion matrix for PC4 dataset using proposed hybrid classifier

Actual class	Predicted class	
	Defective	Non-defective
Defective	1220	1
Non defective	27	151

Table 18 Performance evaluation for PC4 dataset using proposed hybrid classifier

Precision	Sensitivity	Specificity	Recall	F-score	Accuracy
0.97	0.99	0.843	0.99	0.97	98.0
99	0.84	0.99	0.84	0.90	

nificant performance which can be used for software defect prediction model.

5 Conclusion

Early detection and prediction of software defects plays very important role in the software industry in terms of quality measurement. To address this issue of detecting software bugs, various techniques have been developed by researchers in the past. In this field, machine learning based approaches have been considered most promising technique due to learning mechanism of classifiers. In this work, we have introduced a new hybrid approach for software defect prediction which is a combination of Genetic Algorithm and DNN Classification scheme. Proposed approach is implemented on benchmark datasets which are obtained from PROMISE repository. The improved efficiency of the proposed hybrid approach due to deployment of optimization technique is demonstrated. Performance of proposed hybrid approach is compared with existing classification schemes such as Naïve Bayes, SVM, Decision Tree, KNN, etc. from published literature. Comparison is carried out in terms of classification accuracy, sensitivity, specificity, precision, recall. According to experimental analysis, improved performance is reported by using proposed approach. Classification accuracy of proposed GA and DNN hybrid approach, is obtained as 97.82% on KC1 dataset, 97.59% on CM1 dataset, 97.96% for PC3 dataset and 98.0% for PC4 dataset which is far better than existing techniques.

References

1. IEEE Standard Glossary of Software Engineering Terminology: In: IEEE Std 610.12-1990, 31 December 1990, pp. 1–84 (1990)
2. Ouriques, J.F.S., Cartaxo, E.G., Machado, P.D.L., Neto, F.G.O., Coutinho, A.E.V.B.: On the use of fault abstractions for assessing system test case prioritization techniques. In: Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing (SAST). ACM, New York, Article 7 (2016). https://doi.org/10.1145/2993288.2993295
3. Benediktsson, O., Dalcher, D., Thorbergsson, H.: Comparison of software development life cycles: a multiproject experiment. IEE Proc. Softw. **153**(3), 87–101 (2006)
4. Hassan, M. M., Afzal, W., Blom, M., Lindström, B., Andler, S. F., Eldh, S.: Testability and software robustness: a systematic literature review. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, Funchal, pp. 341–348 (2015)
5. Tomaszewski, P., Håkansson, J., Grahn, H., Lundberg, L.: Statistical models vs. expert estimation for fault prediction in modified code—an industrial case study. J. Syst. Softw. **80**, 1227–1238 (2007)
6. Catal, C., Diri, B.: A systematic review of software fault predictions studies. Expert Syst. Appl. **36**(4), 7346–7354 (2009)
7. El Emam, K., Benlarbi, S., Goel, N., Rai, S.N.: The confounding effect of class size on the validity of object-oriented metrics. IEEE Trans. Softw. Eng. **27**, 630–650 (2001)
8. Gittens, M., Kim, Y., Godwin, D.: The vital few versus the trivial many: examining the Pareto principle for software. In: 29th Annual International Computer Software and Applications Conference (COMPSAC'05). **2**, 179–185 (2005)
9. Khoshgoftaar, T.M., Gao, K.: Count models for software quality estimation. IEEE Trans. Rel. **56**, 212–222 (2007)
10. Gondra, I.: Applying machine learning to software fault-proneness prediction. J. Syst. Softw. **81**(2), 186–195 (2008). https://doi.org/10.1016/j.jss.2007.05.035
11. Thwin, M.M.T., Quah, T.-S.: Application of neural networks for software quality prediction using object-oriented metrics. J. Syst. Softw. **76**, 147–156 (2005)
12. Bo, Y., Xiang, L.: A study on software reliability prediction based on support vector machines. In: 2007 IEEE International Conference on Industrial Engineering and Engineering Management, pp. 1176–1180 (2007)
13. Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., Haesen, R.: Mining software repositories for comprehensible software fault prediction models. J. Syst. Softw. **81**, 823–839 (2008)
14. Espejo, P.G., Ventura, S., Herrera, F.: A survey on the application of genetic programming to classification. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **40**(2), 121–144 (2010)
15. Dick, S., Meeks, A., Last, M., Bunke, H., Kandel, A.: Data mining in software metrics databases. Fuzzy Sets Syst. **145**, 81–110 (2004)
16. Seliya, N., Khoshgoftaar, T.M.: Software quality analysis of unlabeled program modules with semisupervised clustering. IEEE Trans. Syst. Man Cybern. Part A **37**, 201–211 (2007)
17. Dejaeger, K., Verbraken, T., Baesens, B.: Toward comprehensible software fault prediction models using Bayesian network classifiers. IEEE Trans. Softw. Eng. **39**(2), 237–257 (2013)
18. Shuai, B., Li, H., Li, M., Zhang, Q., Tang, C.: Software defect prediction using dynamic support vector machine. In: 2013 Ninth International Conference on Computational Intelligence and Security, Leshan, pp. 260–263 (2013)
19. Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J.: Deep learning for just-in-time defect prediction. In: 2015 IEEE International Conference on Software Quality, Reliability and Security, Vancouver, BC, pp. 17–26 (2015)
20. Hinton, G.E., Osindero, S., Teh, Y.-W.: A fastlearning algorithm for deep belief nets. Neural Comput. **18**(7), 1527–1554 (2006)
21. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)

22. Basili, V.R., Briand, L.C., Melo, W.L.: A validation of object-oriented design metrics as quality indicators. IEEE Trans. Softw. Eng. **22**(10), 751–761 (1996)
23. Denaro, G., Pezze, M.: An empirical evaluation of fault-proneness models. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), Orlando, FL, USA, pp. 241–251 (2002)
24. Gyimothy, T., Ferenc, R., Siket, I.: Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. Softw. Eng. **31**(10), 897–910 (2005)
25. Bishnu, P.S., Bhattacherjee, V.: Software fault prediction using Quad Tree-based K-means clustering algorithm. IEEE Trans. Knowl. Data Eng. **24**(6), 1146–1150 (2012)
26. Yuan, X., Khoshgoftaar, T.M., Allen, E.B., Ganesan, K.: An application of fuzzy clustering to software quality prediction. In: Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, Richardson, TX, pp. 85–90 (2000)
27. Azar, D., Vybihal, J.: An ant colony optimization algorithm to improve software quality prediction models: case of class stability. Inf. Softw. Technol. **53**(4), 388–393 (2011)
28. Chen, W.-N., Zhang, J.: Ant colony optimization for software project scheduling and staffing with an event-based scheduler. IEEE Trans. Softw. Eng. **39**(1), 1–17 (2013)
29. Park, B.-J., Oh, S.-K., Pedrycz, W.: The design of polynomial function-based neural network predictors for detection of software defects. Inf. Sci. **229**(20), 40–57 (2013)
30. Elish, K.O., Elish, M.O.: Predicting defect-prone software modules using support vector machines. J. Syst. Softw. **81**(5), 649–660 (2008)
31. Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B.: Using the support vector machine as a classification method for software defect prediction with static code metrics. In: Engineering Applications of Neural Networks, pp. 223–234. Springer, Berlin (2009)
32. Rong, X., Li, F., Cui, Z.: A model for software defect prediction using support vector machine based on CBA. Int. J. Intell. Syst. Technol. Appl. **15**(1), 19–34 (2016)
33. Shivaji, S., James Whitehead, E., Akella, R., Kim, S.: Reducing features to improve code change-based bug prediction. IEEE Trans. Softw. Eng. **39**(4), 552–569 (2013)
34. Rathore, S.S., Kumar, S.: A decision tree logic based recommendation system to select software fault prediction techniques. Computing **99**(3), 255–285 (2017)
35. Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J.: Deep learning for just-in-time defect prediction. In: Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS '15). IEEE Computer Society, Washington, DC, USA, pp. 17–26 (2015)
36. Kumudha, P., Venkatesan, R.: Cost-sensitive radial basis function neural network classifier for software defect prediction. Sci. World J. **2016**, Article ID 2401496 (2015)
37. Wahono, R.S., Herman, N.S., Ahmad, S.: Neural network parameter optimization based on genetic algorithm for software defect prediction. Adv. Sci. Lett. **20**, 1951–1955 (2014)
38. Suzuki, M., Tsuruta, S., Knauf, R.: Structural diversity for genetic algorithms and its use for creating individuals. In: IEEE Congress on Evolutionary Computation, Cancun, pp. 783–788 (2013)
39. Huang, C.L., Wang, C.J.: A GA-based feature selection and parameters optimization for support vector machines. Expert Syst. Appl. **31**(2), 231–240 (2006)
40. Zhang, X.L.: Nonlinear dimensionality reduction of data by deep distributed random samplings. In: Asian Conference on Machine Learning, February, pp. 221–233 (2015)
41. Gallagher, S., Kerry, M.: Genetic algorithms: a powerful tool for large-scale nonlinear optimization problems. Comput. Geosci. **20**(7), 1229–1236 (1994)
42. Rajan, C., Shanthi, N.: Genetic based optimization for multicast routing algorithm for Manet'. Sadhana Acad. Proc. Eng. Sci. **40**(7), 2341–2352 (2015)
43. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning, pp. 1096–1103 (2008)
44. Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T.S., Yan, S.: Sparse representation for computer vision and pattern recognition. Proc. IEEE **98**, 1031–1044 (2010)
45. Software Defect Dataset: PROMISE REPOSITORY. http://promise.site.uottawa.ca/SERepository/datasets-page.html
46. Arar, O.F., Ayan, K.: Software defect prediction using cost sensitive neural network. Appl. Soft Comput. J. **33**, 263–277 (2015)
47. Abaei, G., Selamat, A., Fujita, H.: An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. Knowl. Based Syst. **74**, 28–39 (2015)

C. Manjula from Bangalore, Karnataka India. Has completed Bachelor of Science, Master of Computer Applications and MPhil in Computer Science. Having 17 years of Teaching experience and 5 years of Industry experience. Currently working as an Associate Professor at PES Institute of Technology Bangalore South Campus, Bangalore, Karnataka, India. Has 5 publications in National/ International Journals. Has organized more than 10 workshops and seminars.

Lilly Florence from Hosur, Tamil Nadu has completed her Bachelor Degree in Mathematics and Master degree MCA, M.Tech. (IT) and Doctorate in Computer Science. She has 17 years of teaching experience. She is good in teaching all Programming Languages. Prof. Lilly Florence has published 20 research papers in National and International Journals, also she has published 24 Papers in various National and International Conferences. She is an author of three text books namely, Operating Systems, Computer Graphics and Multimedia and Computer Architecture and Organization. Prof. Lilly has organized more than 20 workshops, seminars for various groups of audience. She has visited more than 20 colleges as a Technical Resource Person. She has received grants from DRDO, DST, ISRO, etc to organize FDP and seminars. She is acting as a Computer Society of India Student Branch Counselor for Adhiyamaan College of Engineering. In Research, she is a recognized supervisor of Periyar University and Bharathiyar University. She has produced one Ph.D. Scholar and currently she is guiding 6 Ph.D. scholars. Dr. Lilly has undertaken 2 research projects funded by Department of Science and Technology for Rs. 23.00 lakhs. Prof. Lilly is a life member of Computer Society of India and ISTE. Also she is a BOS member of MCA board.